Technical Report

UCAM-CL-TR-878
ISSN 1476-2986

Number 878

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Efficient multivariate statistical techniques for extracting secrets from electronic devices

Marios O. Choudary

September 2015

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted July 2014 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Darwin College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986
ISBN 978-0-901224-10-1
DOI *https://doi.org/10.48456/tr-878*

Summary

In 2002, Suresh Chari, Rao Josyula and Pankaj Rohatgi presented a very powerful method, known as the *Template Attack*, to infer secret values processed by a microcontroller, by analysing its power-supply current, generally known as its *side-channel leakage*. This attack uses a profiling step to compute the parameters of a multivariate normal distribution from the leakage of a training device, and an attack step in which these parameters are used to infer a secret value (e.g. cryptographic key) from the leakage of a target device. This has important implications for many industries, such as pay-TV or banking, that use a microcontroller executing a cryptographic algorithm to authenticate their customers.

In this thesis, I describe efficient implementations of this template attack, that can push its limits further, by using efficient multivariate statistical analysis techniques. Firstly, I show that, using a linear discriminant score, we can avoid some numerical obstacles, and use a large number of leakage samples to improve the attack, while also drastically decreasing its computation time. I evaluate my implementations on an 8-bit microcontroller, using different compression methods, including Principal Component Analysis (PCA) and Fisher's Linear Discriminant Analysis (LDA), and I provide guidance for the choice of attack algorithm. My results show that we can determine almost perfectly an 8-bit target value, even when this value is manipulated by a single LOAD instruction.

Secondly, I show that variability caused by the use of either different devices or different acquisition campaigns can have a strong impact on the performance of these attacks. Using four different Atmel XMEGA 256 A3U 8-bit devices, I explore several variants of the template attack to compensate for this variability, and I show that, by adapting PCA and LDA to this context, we can reduce the entropy of an unknown 8-bit value to below 1.5 bits, even when using one device for profiling and another one for the attack.

Then, using factor analysis, I identify the main factors that contribute to the correlation between leakage samples, and analyse the influence of this correlation on template attacks. I show that, in some cases, by estimating the covariance matrix only from these main factors, we can improve the template attack. Furthermore, I show how to use factor analysis in order to generate arbitrary correlation matrices for the simulation of leakage traces that are similar to the real leakage.

Finally, I show how to implement PCA and LDA efficiently with the stochastic model presented by Schindler et al. in 2005, resulting in the most effective kind of profiled attack. Using these implementations, I demonstrate a profiled attack on a 16-bit target.

Acknowledgments

I thank God for all His help throughout my research and my wife Daniela for her constant love and care. I thank also the Christian Orthodox Parishes of *Saint Ephraim the Syrian* and *Saint John the Evangelist* in Cambridge, for providing us with a family during our time in Cambridge, as well as my parents for their support at all times.

I also thank my supervisor Markus Kuhn, for accepting me as his student in my early times as a PhD student, for teaching me how to do research, for taking the time to teach me how to write academic papers and for all his continuous patience, as I was eager to finish this thesis.

I thank also Ross Anderson and Frank Stajano, who helped me coming to Cambridge, and then helped me throughout my MPhil and PhD.

I thank Sergei Skorobogatov as well, for many helpful comments and practical assistance in the development of hardware tools.

Special thanks to Google, for awarding me the Google European Fellowship in Mobile Security, which has financed my PhD work.

Thanks also to my college Darwin, for providing an enjoyable student environment, some funding, and a wonderful library, from which I could marvel at the river, the trees, the ducks, and the many other beautiful things that God has made.

I am also grateful to Dan Boneh, Andrew Ng, and Dan H. Johnson, for their excellent Coursera classes on cryptography, machine learning and electrical engineering, respectively.

Furthermore, I also thank my previous tutors from the University Politehnica of Bucharest, and from the Institut de Recherche en Informatique de Toulouse, for their help in my previous years.

Finally, I also thank all my colleagues, including Andrew Lewis, David Chisnall, Dongting Yu, Jonathan Anderson, Jonathan Woodruff, Joseph Bonneau, Laurent Simon, Mike Bond, Richard Clayton, Robert Watson, Rubin Xu, Saar Drimer, Sharad Kumar, Steven Murdoch, Timothy Goh, and the many others that I had the honour of meeting.

Contents

Glossary of symbols

x Leakage sample.

$\mathbf{x}$ Leakage vector (trace).

X Random variable.

$\mathbf{X}$ Leakage matrix or vector of random variables.

$\mathbf{A}'$ Transpose of vector or matrix $\mathbf{A}$.

r Symbol used with raw (uncompressed) traces.

d Distance.

μ Real mean.

$\boldsymbol{\mu}$ Real mean vector.

σ Real variance.

s Sample variance.

r Sample correlation.

$\boldsymbol{\Sigma}$ Real covariance matrix.

$\mathbf{S}$ Sample covariance matrix.

$\mathbf{R}$ Sample correlation matrix.

$\mathbf{D}$ Sample deviation matrix.

$\mathbf{U}$ Matrix of eigenvectors.

$\mathbf{s}$ Signal vector.

$\mathbf{e}$ Eigenvector.

λ Eigenvalue.

$\boldsymbol{\tau}$ Treatment vector.

f Probability density function (pdf).

p Probability mass function (pmf).

$\mathcal{N}$ Normal pdf.

χ^2 Chi-square pdf.

$\mathbf{F}$ Vector of factors for factor analysis.

$\mathbf{l}$ Vector of factor loadings.

$\mathbf{L}$ Matrix of factor loadings.

$\mathbf{E}$ Vector of specific factors.

$\mathbf{\Psi}$ Covariance of specific factors.

k Candidate value for side-channel attacks.

$k\star$ Target value for side-channel attacks.

i Leakage trace index.

j Leakage sample index.

n_{p} Number of profiling traces per candidate value.

n_{a} Number of attack traces per target value.

N Number of trials or total number of profiling traces.

$\mathcal{S}$ Candidate set.

$\mathcal{S}_s$ Candidate subset.

m Number of leakage samples with or without compression.

m^{r} Total number of leakage samples (raw trace).

$\bar{\mathbf{x}}_k$ Template mean vector.

$\mathbf{S}_k$ Template covariance matrix.

$\mathbf{S}_{\mathrm{pooled}}$ Template pooled covariance matrix.

$\mathbf{B}$ *Treatment* matrix.

$\mathbf{W}$ *Within* matrix.

g Guessing entropy.

$\mathbf{a}$ Fisher coefficient.

$\mathbf{A}$ Matrix of Fisher coefficients.

d_{LINEAR} Linear discriminant.

1ppc Selection of one sample (point) per clock cycle.

3ppc Selection of three samples (points) per clock cycle.

20ppc Selection of twenty samples (points) per clock cycle.

allap Selection of all samples above 95-th percentile of samples with highest SNR.

PCA Principal Component Analysis.

LDA Fisher's Linear Discriminant Analysis.

HW Hamming Weight function.

HD Hamming Distance function.

δ Deterministic part in stochastic model.

$\mathbf{d}$ Deterministic vector in stochastic model.

ρ Noise part in stochastic model.

$\mathbf{r}$ Noise vector in stochastic model.

g Base function in stochastic model.

β Coefficient in stochastic model.

$\mathbf{V}$ Matrix of coefficients for stochastic model.

$\mathcal{F}_9$ Set of 9 base functions for linear 8-bit model.

$\mathcal{F}_{17}$ Set of 17 base functions for linear 16-bit model.

$\hat{\mathbf{x}}_\mathbf{k}$ Mean vector estimated from stochastic model.

$\hat{\mathbf{S}}$ Covariance matrix estimated from stochastic model.

$\hat{\mathbf{B}}$ *Treatment* matrix estimated from stochastic model.

Chapter 1

Introduction

Smartcards, such as those provided to their customers by many banks across the world, use a microcontroller to encrypt or decrypt data, in order to authenticate a person (e.g. verify a PIN) or a transaction (e.g. generate an electronic transaction certificate), based on a secret key stored in the microcontroller. However, the physical implementation of a microcontroller *leaks* information via a *side-channel*, such as the power-supply current or electromagnetic emanations. This leakage may allow an attacker to recover the secret key of a microcontroller, and use that to generate valid certificates for unlawful commercial transactions. To reduce this threat, microcontrollers used in the smartcards provided by banks have several layers of countermeasures to limit the amount of side-channel information available to an attacker. But, to develop efficient countermeasures, and to have a correct assessment of the level of security provided by such smartcards, it is important to have a good understanding of the potential of side-channel attacks.

1.1 Cryptography

Encryption has been used for thousands of years, a classic example being the Caesar cipher, devised by the Roman emperor Julius Caesar [57] to replace each letter in a message by another one at a fixed distance in the alphabet. However, until the past century, encryption was developed and analysed more as an art, rather than a science. Another popular example is the Enigma machine, which was used by the Germans during the Second World War to encrypt messages between battlefield troops. This machine used several connected rotors, that could encrypt one character at a time by displaying a permutation of the input character. The secret "key" of these machines was composed mainly of the initial position of the rotors (this position changed after typing a letter), and the selection of rotors (some machines used three out of five possible rotors). Before the end of the war, by a collaboration between Poland and Britain, scientists at Bletchley Park (England), including Alan Turing, found a mechanism to decrypt messages and even

built machines to automate this process. The break of the Enigma made evident the need for obtaining secure encryption systems, based on strong theoretical foundations.

Today, we may refer to *Cryptography* (from Greek: *secret writing*) as the art of *designing* cryptosystems, to *cryptanalysis* as the art of *breaking* cryptosystems, and to *cryptology* (from Greek: *study of secrets*) as the union of cryptography and cryptanalysis [96]. The input to an encryption algorithm (e.g. a document) is known as the *plaintext*, and the encrypted output as the *ciphertext*. More formally, a cryptosystem (e.g. an algorithm that encrypts some data) is considered secure *"if no adversary can compute any function of the plaintext from the ciphertext"* [59, Section 1.4].

The first rigorous theoretical study of cryptography was probably the paper published by Shannon in 1949 [99], where he also defined the conditions necessary for obtaining *perfect secrecy*: the probability of obtaining a ciphertext, given a plaintext, must be *independent* of the plaintext. Shannon also showed that for obtaining perfect secrecy, the secret *key* used to encrypt a plaintext must be as long as the plaintext. This result means that perfect secrecy is impractical for most scenarios, since it would require parties to exchange keys as long as the messages they want to exchange.

However, even if perfect secrecy is not practical, the wider development of computers after the Second World War enabled scientists to develop algorithms with strong theoretical foundations, secure given some practical assumptions. The first notable example was the development at IBM of the Data Encryption Standard (DES) [1], based on a design by Horst Feistel, which had some strong, unpublished, theoretical design rules (e.g. specially designed S-boxes, criteria for choosing the number of encryption rounds, which were discovered after its publication [15]), and was considered *computationally* secure. The notion of computationally secure relies on the assumption that an attacker is limited in the amount of computation he can perform to break the system. In particular, for DES it was assumed that the only possible attack was to iterate over all possible 2^{56} values of the key, which was considered expensive in 1977, when DES was first published as a standard. However, a DES key of 56 bits has long been considered insecure, and the successor of DES, the Advanced Encryption Standard (AES) [2], uses key sizes of 128, 192, or 256 bits. Another notable example was the proposal by Diffie and Hellman in 1976 [35], and the development by Rivest, Shamir and Adleman in 1978 [94], of a public-key cryptosystem, that allowed one party to publish a *public* key, which another party could use to encrypt messages decryptable only by the party knowing the corresponding *private* key (usually the party publishing the public key). The system published by Rivest, Shamir and Adleman (known as *RSA*), relies on the assumption that factoring the product of two sufficiently large randomly chosen prime numbers is computationally hard, i.e. there are no polynomial-time algorithms known for this task [59, Section 7.2.4]. Today, an RSA key modulus of size larger than 2048 bits is still considered secure.

The development of cryptography continued in the direction of providing *semantic secu-*

rity, i.e. formal definitions and proofs of the security of a cryptosystem, when assuming some computational limitation of the adversary. This line of research started the same year I was born, in 1984, with the seminal work of Goldwasser and Micali [49], in the context of public-key cryptosystems, which lead to two widely employed definitions of security: *chosen-plaintext attack* (CPA) security, which means that an adversary should not be able to break the security of the system even when he is supplied with a large number of encryptions of chosen plaintexts, and *chosen-ciphertext attack* (CCA) security, where the adversary is also given access to the decryption of a large number of chosen ciphertexts. In the context of *block ciphers*, such as DES or AES, the description of *pseudorandom functions* [48] and *pseudorandom permutations* [71] established the grounds for proving the security of their applications. More recently, we saw the development of *authenticated encryption* schemes, which provide CPA-secure encryption and *ciphertext integrity* (i.e. an adversary cannot produce an arbitrary ciphertext that decrypts correctly), and that are therefore CCA-secure.

Transport Layer Security (TLS) [34] is a present example of a system using modern cryptography to secure the communication between a web browser and a web server. Since its development, there were many attacks discovered against it, but its latest version, TLS 1.2, using authenticated encryption, such as *Galois Counter Mode* (GCM) [77], is currently considered secure. In a recent paper, Krawczyk, Paterson and Wee [65] provide a detailed security analysis of TLS.

EMV [38] is another widely deployed application of cryptographic algorithms. EMV aims to protect against counterfeiting banking cards and the misuse of stolen cards, by means of a cryptographic protocol between the card, a terminal (card reader), and the bank that issued the card. In contrast to TLS, where the target systems (e.g. web-servers) are not generally physically accessible to an attacker, EMV cards may be physically analysed by an attacker, and therefore they are a good target for the side-channel attacks presented below. For this reason, these cards often contain many countermeasures against these attacks, and are generally required to be security-certified for high assurance levels. As part of my research, I also investigated some aspects of the security of EMV, and found several problems, such as the possibility to exploit a protocol flaw in order to extract information from a card, that may be used later to perform an unlawful commercial transaction [16]. However, in the following chapters, I shall focus my research on side-channel attacks. In Appendix A, I provide some details about EMV, in order to explain why the EMV cards need resistance to side-channel attacks.

1.2 Side-channel attacks

Along the search for better cryptosystems during the two World Wars, to encrypt messages over a particular communication channel, the military discovered the possibility of "lis-

tening" to the main communication channel by means of another, unintentional, channel, known as the *side-channel*. As Kuhn [67, Section 1.1.1] and Markettos [74, Section 2.11.1] describe in more detail, there were many such cases during the past century. Among the first known cases, during the First World War, the Germans were able to retrieve the communications of enemy troops, by analysing the earth return-current of the single-wire telegraph system used by those troops [14]. Another important case, this time involving a cryptosystem, was the side-channel analysis performed by British intelligence on the French embassy in London, around 1960–1963 [112]. MI5 and GCHQ scientists used a broad-band radio-frequency tap on the communication line used by the French embassy to transmit information, encrypted using a *low-grade* cipher, in the hope of obtaining partial information of the plaintext, that may *leak* into the channel. It turned out that they were indeed able to retrieve the plaintext of the communication encrypted using the low-grade cipher. Furthermore, they were also able to retrieve a secondary signal, corresponding to the plaintext of a *high-grade* encrypted communication, which leaked somehow (e.g. via electro-magnetic cross-talk) into the low-grade channel.

While the previous attacks showed that it was possible to use side-channel leakage, such as the signal recovered by the British intelligence, to recover the plaintext message, the publication of side-channel attacks against the cryptosystem itself, e.g. to recover the secret key, came much later. Probably the first such publication was the paper by Paul Kocher in 1996 [63], describing the use of timing information to determine the private-key used by the RSA cryptosystem. Kocher's *timing attack* exploited the fact that the time needed to perform the modular multiplication and exponentiation operations, used by the RSA cryptosystem, depended on the value of the private key bits.[1]

Two years later, in 1998, Kocher, Jaffe and Jun published another side-channel attack, known as *Differential Power Analysis* (DPA) [62], which exploited the monitored power consumption of a microcontroller executing DES encryptions, to determine the secret key used with DES. This publication marked a very important point in history, since a cryptosystem such as DES, which was considered secure against all known cryptanalytic attacks, and was even designed to resist the *differential cryptanalysis* attacks discovered by Biham and Shamir [15] after its publication, could be easily broken (i.e. we could recover the secret key), when implemented on a physical device accessible to an attacker. This had important consequences for the pay-TV industry, and later for the banking industry as well, who provided their customers with a microcontroller (in the form of a smartcard), in order to authenticate them, by using their smartcard to perform some encryption using a cryptosystem such as DES. After the publication of DPA, this technique has also been used with the electromagnetic emissions of microcontrollers [42, 91], and was also immediately analysed for the case of AES [22].

Other applications of side-channel attacks include the use of electromagnetic emissions

[1] RSA basically decrypts a message y as $y^x \bmod n$, where x is the private key, and n is public.

from displays [111], photon emissions from transistors [39] and displays [66], and acoustic waves to recover keystrokes [10] or RSA keys [44].

1.3 Template Attacks

The *Template Attacks* were first described by Chari, Rao and Rohatgi [24] in 2002, demonstrating that this technique can provide results far superior to DPA, by using both signal and noise information. The attack consists of two parts, a *profiling* step in which the parameters of a multivariate normal distribution of the side-channel leakage are computed, and an *attack* step in which these parameters are used to infer some secret value (e.g. the key used by DES or AES). In addition, these attacks are aimed at situations where an attacker can only acquire a limited number of leakage traces (e.g. power-supply current, or electromagnetic emanations) from the target device. For this reason, the template attacks assume the use of a training device, that the attacker can use at will during the profiling step. Therefore, the aim of these attacks is to accumulate as much information as possible during the profiling step, such that the attack step can be successful in recovering the secret data with a very small number of side-channel leakage traces. Due to the profiling requirement, the template attack and its variants are also known as *profiled* attacks. When profiling is possible, the template attack is known to be the most effective side-channel attack in terms of number of attack traces required for a successful attack.

The first practical application of the template attack was shown by Chari et al. [24] on a smartcard running the encryption algorithm RC4 [95], developed by Ron Rivest in 1987. Later, Rechberger and Oswald [92] presented several options to help in the computation of template attacks, such as the transformation of time samples into the frequency domain. Agrawal et al. [6] then showed how to combine DPA and template attacks in order to target single bits. They showed that DPA could be used as a pre-processing step, in order to select the most vulnerable bits (i.e. easier to target by DPA), and the best time instants from the power traces. After this step, they applied the template attack on secret bits processed by DES, and showed that all of them could be classified correctly with probability equal to or higher than 0.72, even when using a single attack trace.

In 2005, Schindler et al. [97] presented a method, called *stochastic models*, to improve the profiling step of template attacks, by modeling each sample of the leakage traces as a linear combination of the bits (or combinations of bits) of the target value (e.g. a DES or AES key byte). Soon afterwards, Gierlichs et al. [46] published an evaluation of classic template attacks and stochastic models, demonstrating the efficiency of the latter.

Another important contribution, in the development of template attacks, were the application of Principal Component Analysis (PCA) [9], and Fisher's Linear Discriminant Analysis (LDA) [101], in order to transform the leakage traces, which may contain from a few hundreds to several hundred thousands leakage samples, into a new dimensional

space, where we can use only a few variables, but still retain most of the information from the original traces.

Together with the enhancements of template attacks, as well as the developments of other powerful attacks, such as *Correlation Power Analysis* (CPA) [18] and *Mutual Information Analysis* (MIA) [45], there were developments on the methodology for evaluating the success of these attacks. A notable example was the paper by Standaert, Malkin and Yung [104]. Based on the definitions of guessing entropy by Massey [75] and Cachin [19], Standaert et al. presented the use of the *guessing entropy* and *success rate* as measures of success of a side-channel attack, and following on the entropy definitions of Shannon [98], they presented the *conditional entropy* and *mutual information* to quantify the leakage provided by a device. These measures are similar, but they aim to provide different views in the evaluation of side-channel attacks. Using these measures, Standaert et al. performed an extended evaluation of classic template attacks and stochastic models [103], as well as an evaluation of template attacks in relation to DPA, CPA and MIA, targetting two software implementations of AES [102].

On another line of research, some studies have analysed the impact of using different devices or acquisition campaigns for the success of template attacks. Renauld et al. [93] have used 20 different samples of a hardware circuit, concluding that template attacks may not work at all when using different devices for the two steps of the attack, but using several devices during profiling can help. Then, Elaabid et al. [37] showed that even using the same device but different campaigns can result in very ineffective template attacks. On the other hand, Lomné et al. [70] showed that, by using electromagnetic emanations, the template attacks can be effective even when using different devices for the profiling and attack steps.

1.4 Contributions

In this thesis, I use several multivariate statistical analysis techniques in order to improve the efficiency of template attacks in different situations.

In Chapter 3, I provide some background on multivariate statistical analysis, with a focus on those details that are used in subsequent chapters.

Then, in Chapter 4, I provide a detailed description of template attacks, present several numerical problems that can arise in practice, and show efficient techniques to overcome the numerical issues, improve the success, and reduce the computation time of these attacks. I also present an extensive evaluation of template attacks using different compression methods, and show that LDA is generally the most efficient, but in general we should not compress the leakage traces beyond what is necessary to enable the computation of the attacks. Using these efficient techniques, I demonstrate that we can determine

almost perfectly an unknown 8-bit value manipulated by a microcontroller, even when this value is used by a single instruction.

Next, in Chapter 5, I provide an evaluation of template attacks on different devices, by using four different samples of the Atmel XMEGA 256 A3U 8-bit microcontroller, and I present several techniques that can improve substantially the success of template attacks. In particular, I show how to adapt the PCA and LDA compression methods for this scenario, such that even when using a single device for profiling, but a different one during the attack step, we can still determine very well an unknown 8-bit value.

In Chapter 6, I provide what is probably a first study of the impact of correlation between leakage samples on template attacks and the impact of acquisition parameters (bandwidth, power supply) on this correlation. Using factor analysis, I demonstrate that the correlation is mainly caused by a small number of factors, and how knowledge of these factors can be used to improve the template attacks. Furthermore, I also show how to use factor analysis for the simulation of leakage traces having any desired correlation, in order to match the observations from real traces.

Finally, in Chapter 7, I show how to implement PCA and LDA with stochastic models, resulting in the most efficient version of template attacks. Furthermore, I also show how to use these PCA and LDA implementations for attacking a 16-bit target value.

Chapter 2

Obtaining side-channel leakage traces

In this chapter, I describe briefly how microcontrollers leak information through the power supply current, how to exploit this leakage in side-channel attacks, and the evaluation setup that I used to acquire power consumption leakage traces. In Section 2.3, I present several datasets that I prepared and used for the evaluations described in the following chapters.

2.1 Digital circuits and side-channel attacks

The physical implementation of cryptographic algorithms, such as Data Encryption Standard (DES) [1] or Advanced Encryption Standard [2], or the simple manipulation of data by microcontrollers, Field Programmable Gate Arrays (FPGA) and other devices, make side-channel attacks possible.

The term *side-channel attack* was probably first proposed by Kelsey et al. [60], and may refer to different channels, such as the use of electromagnetic emanations to eavesdrop on video displays [111, 67], the use of timing information to extract a secret key [63, 33], the use of the acoustic channel to eavesdrop on key strokes [10] or even to recover a cryptographic key [44], or the use of power analysis to recover the secret key used by a cryptographic algorithm [62]. Although the concepts presented in the following chapters can apply to any of these side channels, my focus will be on the latter form: attacks that use the power-consumption leakage of a microcontroller in order to infer the data that it manipulates. In the following, I may use the terms *side-channel attacks* or *power analysis attacks* interchangeably to refer to this type of attacks.

Registers, buses and combinatorial logic cells, which are an essential part of electronic devices such as microcontrollers or FPGAs [108, Chapter 26], are the main target of side-channel attacks.

Most combinatorial cells (such as an OR gate) and registers (also a kind of cell, known as *flip-flop*) are built using transistors. In order to minimise transition time from one state

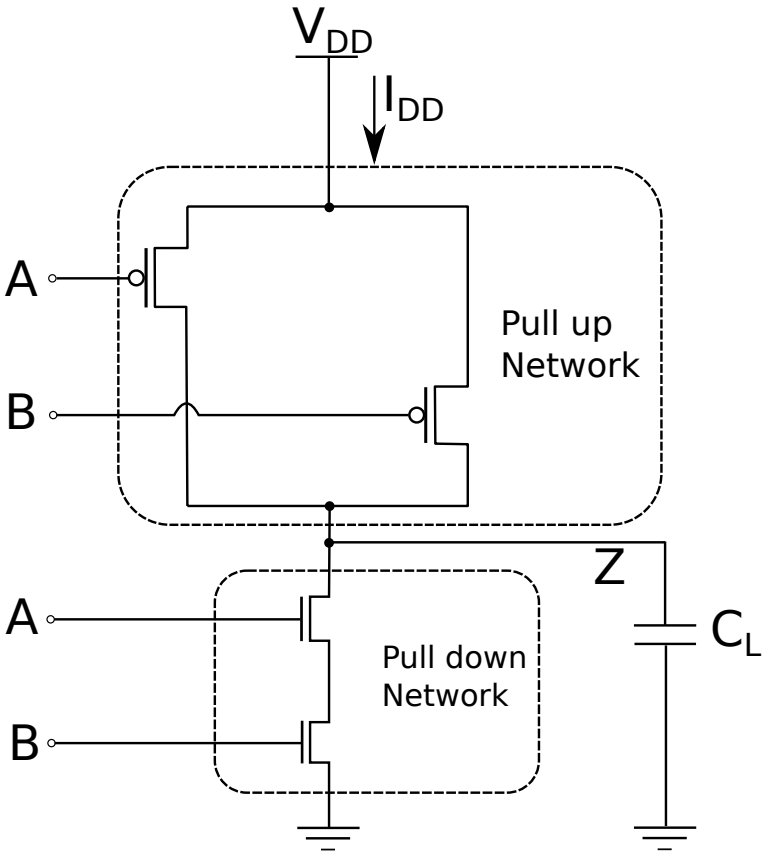

Figure 2.1: A CMOS NAND gate.

to another (e.g. from low to high) and to minimise space (e.g. by avoiding resistors), manufacturers of such cells use complementary metal-oxide semiconductor (CMOS) process technology. Using this technology, a logic cell is formed of a combination of both PMOS (p-channel MOS) and NMOS (n-channel MOS) transistors that create a push-pull network. An example of such a network is shown in the NAND gate from Figure 2.1, where the pull-up network is formed of PMOS transistors that conduct when the inputs (A, B) are low (ground), and the pull-down network is formed of NMOS transistors that conduct when the inputs are high (V_{DD}). When any of the inputs is low (logic 0), the pull-down network is closed and the pull-up network is open such that the current I_{DD} flows through the output of the cell (Z) and this current charges the output (load) capacitor C_L; when this capacitor is charged, the voltage across it is V_{DD}. When both inputs are high (logic 1), the pull-up network is closed but the pull-down network is open. In this case the capacitor is discharged and the output becomes low (ground). In a stable state (i.e. after the load capacitor has been charged or discharged), this kind of CMOS structure consumes very little power, which is known as the *static* power consumption. Giorgetti et al. [47] provide a detailed analysis of this static leakage and they show, based on simulated data as well as from real traces from an ASIC [7] containing several Intel 8051 cores [107, 109] attached to AES modules, that it can be used to differentiate between different keys used with AES. Also, very recently, Amir Moradi [81], using 3 Xilinx FPGAs, has shown that

static power can be used to determine the value stored in registers or look-up-tables, as well as to retrieve the key used with different AES implementations.

Besides the static power, there is a considerable current flow from V_{DD} to ground, when the output switches from one state to another (due to a change in the input), because then there are short periods of time when both pull-up and pull-down networks conduct. This current flow, together with the current caused by charging or discharging the load capacitor, leads to what is known as the *dynamic* power consumption [73, Chapter 3]. Most often, it is this dynamic power that is targeted by side-channel attacks, because it allows us to distinguish between a change in the output vs no change. Furthermore, functional blocks, such as a multiplier, are composed of many interconnected gates like the one in Figure 2.1, which update their output state based on the current input. Therefore, a change in one cell (e.g. at the start of a new clock cycle) propagates through all the connected cells, resulting in a sequence of changes of their outputs until they reach a stable state. This effect is known as *glitching*, and is also an important factor in side-channel attacks, in particular because it makes it difficult to match theoretical assumptions about leakage with real leakages.

A bus (mainly a set of parallel wires) generally connects some component, such as memory, to registers, which are used to store temporary values. Due to the non-negligible length of buses, they form a substantial output capacitance for a register, very much like the output capacitance C_L in Figure 2.1. Therefore, buses are a good target for side-channel attacks, because we may be able to differentiate between a transition of a line in the bus (e.g. from low to high) versus no transition. Furthermore, we may be able to differentiate between a transition from low to high and a transition from high to low and we may be able to differentiate between the transitions of individual lines (wires) in the bus.

2.1.1 Leakage models

As I explained earlier, the dynamic power consumption caused by transitions in the output of CMOS cells or data buses can be used by side-channel attacks. However, in order to mount a particular side-channel attack we might have to assume a certain *leakage model*, i.e. to make an assumption about how the dynamic power consumption leaks information about the values processed by our target device. I now summarize some of the most common leakage models.

Bit difference model

One of the first leakage models was used by Kocher et al. in their Differential Power Analysis (DPA) attack [62]. They simply assumed that the power consumption of a smartcard running a DES encryption is slightly different when a particular bit b of some

value v processed by the encryption algorithm is 1 compared to when it is 0. This assumption is reasonable, since a manipulation of the value v by a physical device (e.g. microcontroller or FPGA) means in practice the use of a register to store the value v, or some combinatorial cells to pass this value to some other processing stages. In either case, the bit b of value v will affect some CMOS cell (see Figure 2.1), and as explained in Section 2.1, this will lead to different power consumption for different values of b.

Hamming weight model

The Hamming weight $\mathrm{HW}(v)$ of a value v (e.g. an 8-bit value) is defined as the number of bits that are 1 in its binary representation. For example, $\mathrm{HW}(5) = 2$ (since the value 5 is "101" in binary). The idea of using the Hamming weight for side-channel attacks was probably first proposed by Kelsey et al. [60]. However, it was Messerges et al. [79] who published some of the first results showing that the power consumption of microcontrollers follows a Hamming weight model. They explain that this model may be suitable when the main cause of the dynamic power consumption is the discharge of the load capacitor (see Figure 2.1), and that this can be observed easily when using a precharged-bus hardware design. In this case, every 0 bit of a value put into the bus will directly influence the power consumption.

Hamming distance model

Messerges et al. [79] also explained that, for some microcontrollers, the power consumption can be modeled by the number of bit transitions (number of bits that change state) on a bus, which in turn determines the number of CMOS cells that will change state (those connected to the bus and those influenced by these), hence reflecting the dynamic power consumption.

If we consider two consecutive values u and v on the data bus, then the number of bit transitions on the bus can be determined by the Hamming weight of $u \oplus v$, where $\oplus$ represents the exclusive-OR (XOR) operator. This leads to what is now known as the *Hamming Distance* model, where the Hamming distance $\mathrm{HD}(u, v) = \mathrm{HW}(u \oplus v)$. Other authors [23, 8] have confirmed this model may work well for many microcontrollers.

As an example, in Figure 2.2, I show average power consumption traces from the Atmel XMEGA A3U microcontroller[1], when loading different 8-bit values k from RAM memory to a register, for the two clock cycles required by the target LOAD instruction. We can see that the leakage during the first clock cycle is well separated into the nine Hamming weight values 0 to 8, while this separation disappears during the second clock cycle. If we look also at the leakage for the values $k = 0$ (HW = 0) and $k = 255$ (HW = 8), shown

[1]These are taken from the *Grizzly Beta* dataset, presented in Section 2.3.1.

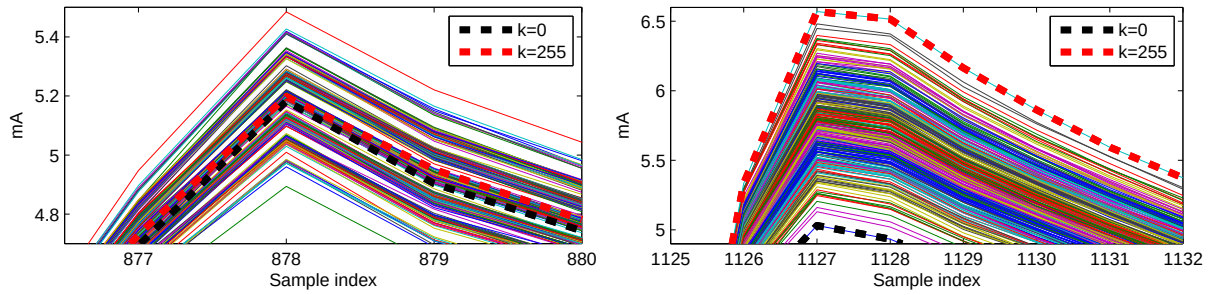

Figure 2.2: Average power consumption traces from the Atmel XMEGA A3U microcontroller, when loading different values from RAM memory to a register. Left: leakage at first clock cycle of LOAD instruction. Right: leakage at second clock cycle.

with bold and dashed lines, we can conclude that the leakage for the first clock cycle follows a Hamming distance model, suggesting that the power consumption is dependent on the XOR between the value k and some value having HW $= 4$, that may be present on the bus before the value k. Also, it seems that the leakage in the second clock cycle is directly influenced by the value k, since the highest leakage corresponds to $k = 255$ (HW $= 8$) and the lowest to $k = 0$ (HW $= 0$).

Some researchers [51, 89] also considered a different model for transitions from 0 to 1 and transitions from 1 to 0 on a data bus, observing improved results in some cases. However, the standard Hamming distance model remains the most common.

Linear model

Akkar et al. [8] mentioned the use of a linear model $P[x] = \sum_{i=0}^{n_{\text{bits}}} x_i \cdot P_i$ to approximate the power consumption $P[x]$ of a value x processed by a device as a linear combination of the bits x_i of x. They mentioned that setting all the P_i to 1 reverts to the Hamming weight model, but they did not provide more details. Later, Schindler et al. [97] refined this model to allow for more general combinations of bits and arbitrary functions of the processed values. In Chapter 7, I will describe this model in more detail and I shall explain how it can be used to obtain very powerful side-channel attacks.

Multivariate normal distribution

The previous models may not accurately represent the leakage of a device, since the actual leakage may depend on more than the Hamming weight of processed values, or the values of the individual bits. For example, past computations might affect the leakage in following clock cycles, or several computations computed in parallel might affect the leakage in a particular clock cycle in the case of hardware implementations of cryptographic algorithms, or perhaps coupling effects between individual bit lines play an important role

for some devices. Therefore, a better model can be obtained if we are able to characterise precisely the leakage caused by processing a particular target value.

In practice, the leakage traces obtained from an electronic device often follow a multivariate normal distribution. As a result, Chari et al. [24] used this distribution to characterise the leakage of a device, obtaining very powerful side-channel attacks, known as *template attacks*. These attacks are the focus of my work, and they will be described in detail throughout the following chapters. As I shall show, such attacks, based on characterising the leakage of each possible target value, worked very well in my experiments, since my target was often the value processed by a LOAD instruction in an 8-bit CPU. However, this approach may not provide the best results when dealing with hardware implementations of cryptographic algorithms, where the target value (often a key byte) is manipulated together with other data in many operations performed in the same clock cycle. In such cases, finding an optimal strategy is not trivial, since we might have to model the leakage of many intermediate values.

2.1.2 The S-box

For the following discussion, it is useful to first introduce briefly the concept of a *substitution box* (known as *S-box*), a fundamental element of many common block ciphers, such as DES or AES, because the S-box is very often the target of side-channel attacks. The S-box is a non-linear mapping between some input value u and some output value $v = \text{Sbox}(u)$. Typically, the input is the XOR between a *known* value p (e.g. part of the plaintext) and an *unknown* value k (e.g. part of the secret key of the block cipher). This is shown in Figure 2.3. For DES, the S-box maps 6 bits input to 4 bits output, while for AES the S-box maps 8 bits to 8 bits. The main property of the S-box is its non-linearity, which means that $\text{Sbox}(a) \oplus \text{Sbox}(b) \neq \text{Sbox}(a \oplus b)$. More details can be found in the book of Paar and Pelzl [84].

A block cipher such as DES or AES iterates a sequence of operations several times to encrypt some plaintext P into a ciphertext C and for each iteration (round) it uses a *round key*, which is derived from the master secret key K. At each round, parts of the round key (the value k in Figure 2.3) are sent to the S-boxes (which are different for DES but identical for AES). In most common side-channel attacks, the target is either the first round of the block cipher for which p is known (is just a part of the plaintext P), or the last round of the block cipher if v is known (e.g. if it can be easily derived from the ciphertext C). In either case, we assume that an attacker can recover the plaintext or the ciphertext (or perhaps both) and is interested in obtaining the full secret key K.

The side-channel attacks against block ciphers are practical because only a part k (e.g. 6 or 8 bits) of the full master key K (e.g. 56 bit for DES or 128 bit or more for AES) is processed by one S-box. Therefore, by targeting each S-box individually, we can break

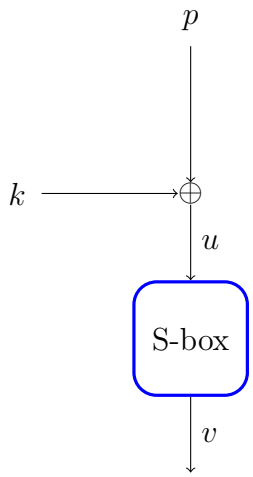

Figure 2.3: A general S-box scenario, where k is a part of the secret key of a block cipher.

the task of finding the full master key into smaller tasks of attacking only small parts k of the entire key.

2.1.3 Side-channel attacks

I now briefly describe the most common side-channel attacks evaluated in the academic community.

Simple Power Analysis (SPA)

Kocher et al. [62] showed that, by simply observing a single power trace of a microcontroller, it is possible to reveal the sequence of instructions being executed. This may be used to extract the secret key of a cryptographic algorithm by targetting the key schedule if this involves conditional branching, by targetting comparison operations, or by targetting the exponentiators needed in public-key cryptographic algorithms such as RSA [94]. Simple and efficient countermeasures for SPA rely on preventing the use of secret data for conditional branching operations. Furthermore, Kocher et al. mentioned that SPA will probably fail on most hardware implementations of block ciphers due to their small power consumption variation. As a result SPA is not considered a major security threat if simple precautions are taken, but the following attacks are.

Differential Power Analysis (DPA)

Kocher et al. [62] also showed a much more powerful attack against DES (which also works very well against AES), known as *Differential Power Analysis* (DPA). It exploits a known[2] relation (such as the input-output relationship of the S-box in Figure 2.3),

[2]This assumes knowledge of the target algorithm.

between a target bit b, a ciphertext C and a part k of the round key, to separate power consumption traces of different encryptions into two groups. Trying each possible value of k (of which there are only $2^6 = 64$ for DES or $2^8 = 256$ for AES), we can split the leakage traces according to the value that b would take for each k. Under Kocher's leakage model, the difference between the mean values of the traces in the two groups should be noticeable when we used the correct k to separate the traces, but this difference will be close to zero when using a wrong candidate k. Given the leakage corresponding to a cryptologically small number (e.g. 1000) of encryptions, Kocher showed that the correct value of k is easily recoverable. This result radically changed the view of the cryptographic community, because an algorithm that is secure in theory (such as AES) can be broken easily (i.e. we can recover the secret key) using side-channel attacks.

Correlation Power Analysis (CPA)

Based on the observation that the leakage of a microcontroller may be directly related to some simple model, such as the Hamming weight or Hamming distance of manipulated values, Brier et al. [18] presented the use of *Pearson's product-moment correlation coefficient* [87], described in Section 3.3.5, to determine the values manipulated by a microcontroller. This main idea behind this attack, known as *Correlation Power Analysis* (CPA), is to obtain many traces, then compute the expected Hamming weight (or other leakage model) of the values corresponding to these traces, and finally compute the correlation coefficient between the power consumption and the leakage model. If our prediction of the processed values and the leakage model are good, then we should obtain a high correlation for those leakage samples that correspond to the processed value. Brier et al. showed how this technique can be used to determine the values processed by a microcontroller, and in particular, how this can be easily used to determine the AES key bytes, by targeting for example the key addition (value $u = p \oplus k$, in Figure 2.3): we simply compute the correlation between the power consumption and $\mathrm{HW}(u)$, for all possible values k, and decide that the correct key byte corresponds to the one providing the highest correlation. However, as I explain in Appendix B, note that it makes a great difference whether we target just the XOR of a key byte and a plaintext byte, or if we target the output of a non-linear function such as the S-box, which leads to better results with CPA.

Template Attacks

As mentioned earlier, Chari et al. [24] used the multivariate normal distribution

$$f(\mathbf{x} \mid k) = \frac{1}{(2\pi)^{m/2}|\mathbf{S}_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \bar{\mathbf{x}}_k)'\mathbf{S}_k^{-1}(\mathbf{x} - \bar{\mathbf{x}}_k)\right) \tag{2.1}$$

to obtain a very powerful side-channel attack, known as the *template attack*. This attack works in two steps. In the first step, we use a *profiling* device, processing *known* data,

to obtain the *template* parameters (a mean vector $\bar{\mathbf{x}}_k \in \mathbb{R}^m$ and a covariance matrix $\mathbf{S}_k \in \mathbb{R}^{m \times m}$, describing the leakage traces $\mathbf{x}_{\text{profile}} \in \mathbb{R}^m$) for each possible value k (e.g. secret key byte). Then, in the second step, we use these parameters, together with (2.1), to classify a trace $\mathbf{x}_{\text{attack}} \in \mathbb{R}^m$, obtained from the *attack* device (of the same model as the profiling device), thus being able to infer the *unknown* value $k\star$ processed by the attacked device, as

$$k\star = \arg\max_k f(\mathbf{x} \mid k). \tag{2.2}$$

Due to the requirement for a profiling step, these attacks are also known as *profiled* attacks.

In Chapter 3, I describe the multivariate normal distribution and how to estimate its parameters. Then, in Chapter 4, I present the template attack in more detail, as well as efficient implementations of this attack. In Chapters 5 and 6, I present further improvements of the template attack for different practical situations, and in Chapter 7, I describe efficient implementations of the template attack using the stochastic model of Schindler et al. [97].

Mutual Information Analysis (MIA)

The Mutual Information Analysis (MIA), presented by Gierlichs et al. [45] in 2008, allows an attacker to determine some secret value k processed by a microcontroller, when this value is processed by some *known* operation, such as the key addition or S-box from Figure 2.3, which combines the secret value k with a known and modifiable value p. The overall process of this attack is very similar to CPA, but instead of using the correlation between power consumption and leakage model, MIA uses the mutual information $\mathbf{I}(L; X)$ as a measure of dependence between the leakage model L of the candidate values (e.g. v, in Figure 2.3) and the leakage values X. As for CPA, the main idea is that, for the correct key, the mutual information will be maximized. Gierlichs et al. mention that one important advantage of MIA is that *"it can exploit arbitrary relationships between [L] and [X]"*. They also mention that Pearson's correlation coefficient requires a linear relation between leakage model and leakage traces, but this is not entirely the case, since Pearson's correlation coefficient may capture other types of relationships as well. For example, two random variables X and $Y = X^2$ can have a very high correlation coefficient when X is positive, so the advantage of MIA over CPA may not be that great in practice.

2.1.4 Countermeasures

Along with the side-channel attacks presented earlier, there have been many countermeasures developed to limit the effects of these attacks. Some of the first countermeasures included the addition of noise, dummy operations and randomized clock [73] to diminish

the ability of using many power consumption traces for statistical attacks such as DPA. However, these countermeasures do not prevent these attacks completely, but only make them more difficult. Therefore, several people proposed and developed a more radical and theoretically sound approach [23, 50, 30, 73], known as *masking*. The main idea of masking is to split the intermediate values targeted by a side-channel attack (e.g. the output of the S-box from Figure 2.3), into a number of secret shares that depend on some random values (the masks), which are refreshed at each iteration of the algorithm. This should remove the dependency of the leakage traces from secret data. However, in practice, masking cannot completely remove this dependency, due to glitches in a CMOS device, pipeline effects and other implementation details that cannot be fully captured by masking. Also, incorrect implementations of masking can lead to even more leakage, therefore resulting in actually better attacks. For example, the current results of an open competition known as *DPA contest V4* [85], show that a standard template attack can easily recover the secret key using a *single* attack trace. Therefore, in practice, a combination of countermeasures is used to design secure devices, such as the smartcards used in the banking industry.

In my work, I did not consider any countermeasures, since my focus has been to explore efficient multivariate statistical techniques for template attacks. As a result, during profiling I could determine the data processed by the microcontroller, leading to a high signal-to-noise ratio, while for the attack the traces were not affected by unknown values (such as masks). However, many of the techniques presented here can be applied also to devices protected by some countermeasure. For example, Lemke-Rust and Paar [69] show how to use stochastic models (which I explore in detail in Chapter 7) against a device protected by masking.

2.1.5 Leakage-resilient cryptography

Besides the physical (e.g. dual rail, noise) and software (e.g. random delays, masking) protections, a new field of study called *Leakage Resilient Cryptography* has emerged in recent years, where the goal is to design algorithms that are by themselves resistant to side-channel attacks. Micali and Reyzin [80] wrote probably the first paper in this field, where they defined the notion of a leakage function, formalised the notion of a computation that *leaks*, and proposed leakage resilient one-way functions and permutations. Later advances include the work of Standaert et al. [105], Balasch et al. [11], and Medwed et al. [78].

Many of the leakage-resilient algorithms assume some bounds on the amount of leakage provided by a device. Therefore, while I have not focused directly on this area of research, my results from using efficient template attacks may help to understand what are some reasonable assumptions about the amount of leakage provided by a device. Nevertheless, there are still no practical results that can prove a bound on the gap between the leakage in the average case and the leakage in a worst-case scenario.

2.1.6 Common Criteria

Together with the academic efforts in developing countermeasures and leakage-resilient algorithms, the industry, together with national security agencies, have developed standards to evaluate and certify the security of IT products used in high-risk applications. One such standard, perhaps the most relevant for secure microcontrollers, is known as *Common Criteria* (CC) [4]. The CC define *protection profiles* for different IT products, which describe the possible threats of these products, the protection that these products must provide, the general components of these products, the overall design and certification process, etc. Manufacturers of these products submit their product to an *evaluation laboratory*, in order to obtain a CC certification at some desired *Evaluation Assurance Level* (EAL). A higher EAL results in higher demands of security, but also requires a larger investment for the development of products that can pass the CC evaluation. The CC is responsible for accrediting and verifying the correct operation of evaluation laboratories.

For the particular case of secure microcontrollers, the CC *Security IC Platform Protection Profile* [28] defines the overall structure of a secure microcontroller and the possible attacks against such device. The relevant definitions for side-channel attacks are *"T.Leak-Inherent"*, which states that an attacker may exploit the side-channel leakage to disclose confidential user data, and *"O.Leak-Inherent"*, which states that the *Target of Evaluation* (TOE) must provide protection against side-channel attacks (e.g. current consumption, time). The Joint Interpretation Library Hardware Attacks Subgroup (JHAS) [55] provides more details regarding the attacks and equipment that should be used during an evaluation of a secure microcontroller, and also provides a method to assess its resistance to an attacker, based on time required for attack, expertise, knowledge of the device, access to the device, equipment, and access to open samples (i.e. available to any attacker).

As a practical example, a currently certified device, the *Infineon M9900 A22* [29] Secure IC has been certified at level *EAL5 Augmented* (EAL5+), which means that it has *resistance to attackers with HIGH attack potential*. The CC definition for EAL5 is that the target device has been *semiformally designed and tested*, meaning that the device has been designed with security features from its early stages, rather than engineering additions. A security evaluation for such Security IC may cost up to hundred thousands EUR, and may require about 3 man months of work.

2.2 Acquisition framework

As explained in the previous section, side-channel attacks work because we are able to observe the power consumption of a microcontroller, and this power consumption in turn depends on the data processed by the microcontroller. Therefore, in order to mount these attacks in practice, the first step is to obtain leakage traces of the power consumption.

For this purpose, I used an evaluation setup consisting of a custom Printed Circuit Board (PCB) that I designed for the Atmel XMEGA A3U microcontroller (the *XMEGA PCB*), some target code (the algorithm or sequence of instructions that we aim to target in a side-channel attack) running on this microcontroller, an oscilloscope used to record the power consumption of the XMEGA, a waveform generator, a power supply, and a PC running some scripts to communicate with both the oscilloscope and the XMEGA PCB. These are described below in more detail.

2.2.1 Atmel XMEGA A3U

When I decided to focus on the field of side channel attacks, I was advised to build my own evaluation device, so that I would know exactly all the components and parameters involved. Therefore, based on my previous experience with AVR microcontrollers, and given their wide use, I decided to build a board for the Atmel XMEGA A3U microcontroller [31].

The XMEGA A3U is an 8-bit microcontroller, i.e. the bus and registers are 8-bit wide, has 32 8-bit registers, can use 16-bit addresses to access memory locations, has many peripherals (e.g. USB, USART, ADC and DAC converters), accepts a wide range of clock frequencies (0–32 MHz), can be powered from 1.6 V to 3.6 V, has 16 KB internal SRAM, 4 KB EEPROM, 256 KB flash, hardware DES and AES crypto engines, and several other features that make it an attractive choice for embedded devices. The block diagram of this device is shown in Figure 2.4.

2.2.2 Design of the XMEGA PCB

I designed the XMEGA PCB with two main goals in mind: (*a*) to allow easy communication with a PC for programming, debugging and data transfer; (*b*) to minimise the negative effects of impedance mismatch and other noise sources derived from a naive design of the PCB.

The first goal is easier to accomplish. For programming and debugging, I used the JTAG interface of the XMEGA A3U microcontroller and I provided headers on the PCB to connect to this interface. For data transfer, I used the USB interface of the XMEGA, linked to a USB MINI connector.

For the second goal, I followed the advice on measurement setup given by Mangard et al. [73, Sections 3.4–3.5] as well as the detailed guidance on signal integrity provided by Paul [86].

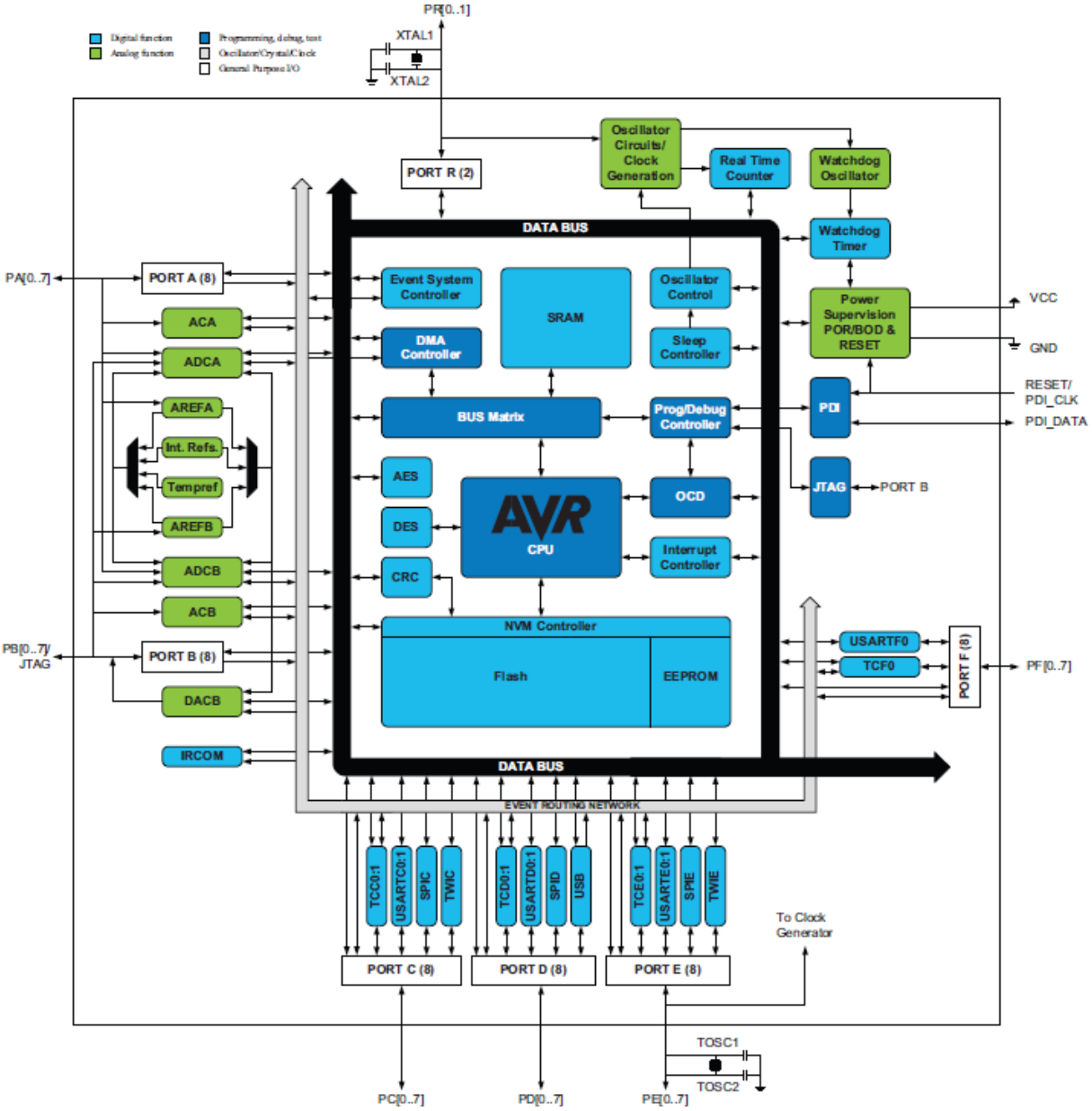

Figure 2.4: XMEGA A3U block diagram. Taken from the XMEGA A3U datasheet [31].

Measurement technique

There are two main ways to obtain power (side-channel) traces from a device. The most popular is to insert a measurement resistor between the VCC or the GND lines of the microcontroller and the corresponding line on the PCB, and then use an oscilloscope to measure the voltage across this resistor, which should be proportional to the power consumption of the microcontroller. The other way is to use an E-field or H-field probe (sometimes referred to as EM probe) that measures the electromagnetic field around the target microcontroller, which is also proportional to its power consumption. The advantage of the EM probes is that they may be able to measure the localized power consumption of only a part of the target device, therefore avoiding switching noise caused

by other components.

For simplicity, I decided to use a measurement resistor of 10 Ohm placed between the joined ground lines of the microcontroller and the ground plane of the XMEGA PCB.

Quality of measurement setup

Mangard et al. [73, Sections 3.5–3.6] describe various causes that lead to electronic noise, which is unrelated to the operation of the target device. They identify five main sources of electronic noise. (1) Noise of the power supply, which can be minimised by using a highly stable power supply such as batteries or a laboratory power supply. (2) Noise of the clock generator. Besides electronic noise, a noisy or unstable clock signal can result in traces that are misaligned, which will require an extra pre-processing step during the side-channel attack. These problems can be minimised by using a very stable clock signal, such as a waveform generator or a crystal oscillator. Furthermore, Mangard et al. recommend using a sinusoidal clock signal. (3) Conducted emissions, i.e. unwanted signals that propagate through the components, wires and material of the PCB. These can be mitigated by good PCB design and by reducing the number of components on the PCB. (4) Radiated emissions, which can be generated by any component carrying current. These emissions can then be picked up by other components on the PCB, therefore producing unwanted noise. Again, a good PCB design can reduce these. (5) Quantization noise, which is introduced by the analog to digital conversion performed by the measurement oscilloscope. This can be reduced by increasing the resolution of the oscilloscope (i.e the number of bits used to store each voltage sample), although 8 bits seem sufficient in practice.

As a result, I designed the XMEGA PCB to allow for different power supply and clock signal, as follows. It has input for unregulated power supply (e.g. input coming from a stable lab power supply), as well as for batteries that are then regulated via a 3.3 V linear voltage regulator. It also allows the use of either an external clock signal (e.g. from a waveform generator), an on-board crystal oscillator, or one of the internal oscillators of the XMEGA microcontroller (although this is not recommended as they are not stable).

A few design aspects also increase the quality of the PCB, such as using only 45 degrees line corners to avoid discontinuities that lead to high fields in the corners, and the use of ground planes that minimise loops, resulting in very low impedance [108, Chapter 21].

Signal integrity

The quality of the leakage signal used by side-channel attacks directly influences the success of these attacks, so it is useful to try increasing the quality of the acquired signal.

In my experiments, the leakage signal comes from the current consumed by the target microcontroller, which is proportional to the voltage across the measurement resistor R_P.

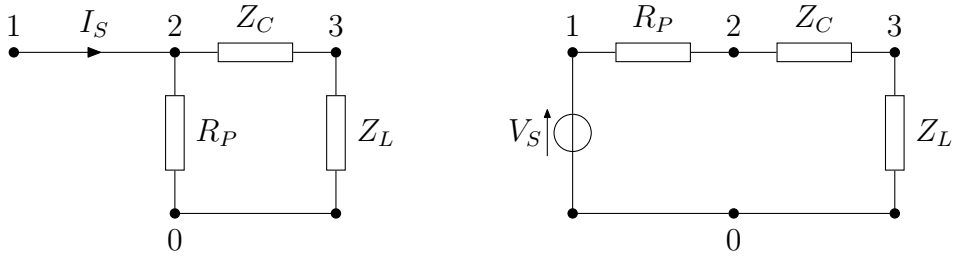

Figure 2.5: Circuit for signal acquisition (left) and Thévenin equivalent circuit (right).

This signal then goes through the PCB, via a microstrip line, then through the probe connector, then through the oscilloscope probe, and finally arrives at the load Z_L in the oscilloscope, which samples this signal. This is illustrated in Figure 2.5 (left), where I_S is a current source modeling the current consumed by the target microcontroller, Z_C is the characteristic impedance of the transmission line between R_P and the oscilloscope (see next paragraphs), and Z_L is the input impedance of the oscilloscope (generally a combination of resistance with very small capacitance). In Figure 2.5 (right) I show the Thévenin equivalent circuit, where V_S represents the open-circuit voltage. This form will be helpful in the following.

The path through which the leakage signal travels is known as a *transmission line* and has several interesting properties [86, Chapter 4]. Firstly, each continuous part of the transmission line can be characterized by a per-length impedance l and a per-length capacitance c, leading to a *characteristic impedance* $Z_C = \sqrt{\frac{l}{c}}$. Then, if some voltage is applied between the line and the ground plane and if some current is flowing through the transmission line, we can derive the (lossless) transmission-line equations

$$V(z,t) = V^+(t - \frac{z}{v}) + V^-(t + \frac{z}{v}) \tag{2.3}$$

$$I(z,t) = \frac{1}{Z_C}V^+(t - \frac{z}{v}) + \frac{1}{Z_C}V^-(t + \frac{z}{v}), \tag{2.4}$$

where z is the distance (location) of the electromagnetic wave flowing through the transmission line, t is the time, v is the velocity of propagation of the wave and Z_C is the characteristic impedance. The important thing from this equation is to notice that both the voltage V and current I at a particular distance z and time t consist of two components: (a) a *forward*-travelling wave V^+ and (b) a *backward*-travelling wave V^-.

In the following discussion I will assume that the signal of interest is the voltage that is generated between the nodes 2 and 0 in Figure 2.5. Then, the forward and backward voltage waves are related at the oscilloscope load, $z = \mathcal{L}$, by the *load reflection coefficient* as

$$\Gamma_L = \frac{V^-(t + \mathcal{L}/v)}{V^+(t - \mathcal{L}/v)} = \frac{Z_L - Z_C}{Z_L + Z_C}. \tag{2.5}$$

An initial wave arrives at the load after a time $T_D = \mathcal{L}/v$, at which point the backward

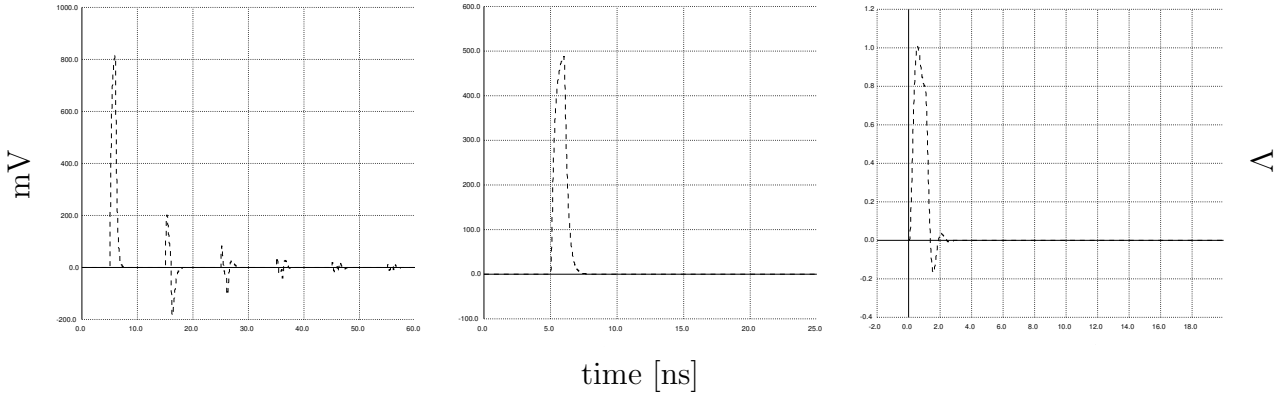

Figure 2.6: Voltage at the oscilloscope load from ngspice simulations. Left: standard circuit from Figure 2.5 (left), when $I_S = 100$ mA. Centre: using a matching resistor at the source, as in Figure 2.7 (left). Right: using a *line does not matter* setup, as in Figure 2.7 (right).

(reflected) wave emerges, with

$$V^-(t + \mathcal{L}/v) = \Gamma_L V^+(t - \mathcal{L}/v). \tag{2.6}$$

After an additional time $T_D = \mathcal{L}/v$, the reflected wave reaches the initial node 2, where the line *sees* another (possibly different) characteristic impedance. Therefore, at the source (node 2) we obtain a voltage reflection coefficient

$$\Gamma_S = \frac{R_P - Z_C}{R_P + Z_C}, \tag{2.7}$$

that causes the reflected wave to be reflected back towards the load. As a result, we get a kind of ping-pong scenario, where an original wave originating at the source (node 2) travels forwards and backwards between the source and the load. This can cause unwanted oscillations (ripples) at the load, which in turn may degrade the signal integrity. To demonstrate this, I have simulated the circuit from Figure 2.5 (left) with ngspice [100], using a source that produces an impulse with rising/falling time of 0.1 ns and width 1 ns, $R_P = 10\ \Omega$, $Z_C = 50\ \Omega$, $T_D = 5$ ns, and a load Z_L composed of a parallel combination of a 50 Ω resistance and a 10 pF capacitance. The voltage at the load is shown in Figure 2.6 (left). We can see that besides the main impulse signal, there are subsequent reflections having a considerable amplitude. For high-frequency signals (e.g. in the order of 100 MHz), these reflections will distort the received signal.

There are two main ways to avoid the ripples at the load, and hence to obtain a cleaner and more useful signal. The first method is to *match* the impedances such that at least one of the reflection coefficients becomes zero. To obtain a zero reflection coefficient at the source (2.7), we should add a *matching* resistor R_M in series with R_P such that the resulting source resistance $R_S = R_P + R_M$ is equal to Z_C, as shown in Figure 2.7 (left). At the load we can either match also the impedance ($Z_C = Z_L$), such that the reflection

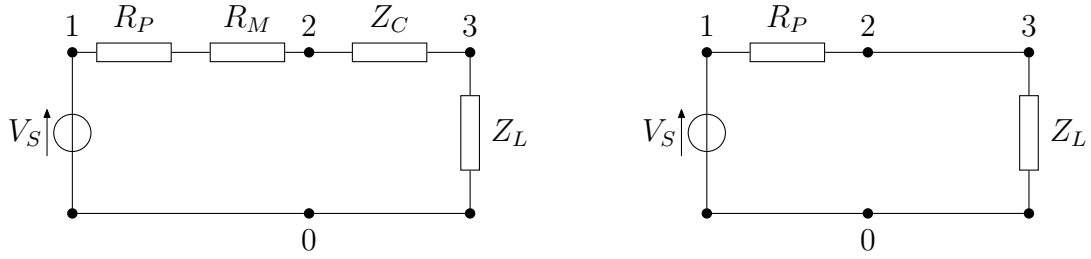

Figure 2.7: Thévenin equivalent circuit for *good* signal acquisition. Left: using matching resistor at the source. Right: ignoring Z_C when the line does not matter.

coefficient at the load (2.5) is zero, or we can use a very large impedance at the load (e.g. $Z_L = 1\text{M}\Omega$), such that the reflection coefficient at the load is one. Note that using a matching resistor at the source implies that the signal provided to the transmission line (at node 2) is effectively $V_S/2$, since at this point we have effectively a voltage divider ($Z_C = R_P + R_M$). Therefore, matching the output ($\Gamma_L = 0$) provides half the signal at the output, while using a very large load resistance ($\Gamma_L = 1$) provides the original signal. This does not affect the signal integrity, although matching the output as well as the input might be better if the amplitude is not important, since it will diminish the risk of residual reflections due to imperfect matching at the source. In Figure 2.6 (middle), I show the results of the ngspice simulation for the circuit in Figure 2.7 (left), using $R_M = 40\ \Omega$ and the same values as before for the other elements. We can see how this setup removes all the unwanted reflections, providing a very clean signal.

The second option to avoid the unwanted ripples at the load is to use a setup such that the transmission line *does not matter*. We can assume this situation if the length $\mathcal{L}$ of the transmission line is electrically short at the highest significant frequency, which translates in practice to the condition

$$\mathcal{L} < \frac{1}{10}\frac{v}{f_{max}}. \tag{2.8}$$

This scenario is shown in Figure 2.7 (right). To simulate a very short line in ngspice I used the previous setup, from Figure 2.7 (left), but with $T_D = 0.05$ ns. The results are shown in Figure 2.6 (right). In this case we can observe that there is a small ripple along the main signal, but nothing else. Therefore, this option can also also provide a good signal.

In the context of power analysis, it is not generally possible to achieve the condition from (2.8), since we must connect the side-channel signal to an oscilloscope somehow and in practice this requires the use of some probe having a cable considerably longer than $\frac{1}{10}\frac{v}{f_{max}}$. However, we can use an *active probe*, such as the Tektronix P6243. This probe basically provides the *line does not matter* condition, by using an operational amplifier with high impedance at the source (i.e. the tip of the probe), and by matching the input of the oscilloscope (i.e. it will match at the load). In addition, the active probe is designed

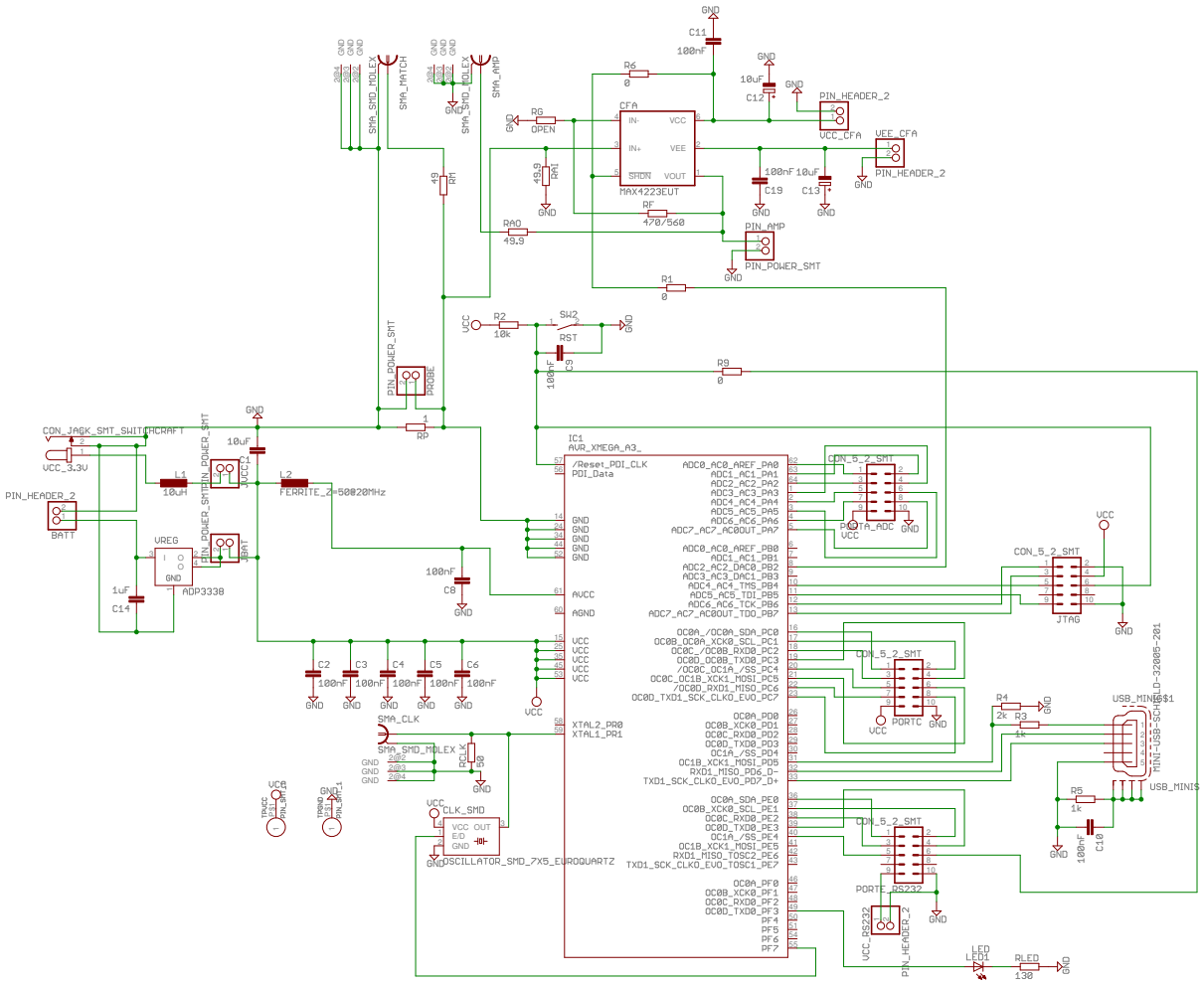

Figure 2.8: Schematic of the XMEGA PCB.

to compensate any loss in the cable.

The XMEGA PCB is designed to allow both options: a matched line leading to an SMA connector that can be connected to a passive probe or coaxial cable having a 50 Ω characteristic impedance, and an unmatched but very short line that can be connected to an active probe. For most of my experiments, I decided to use the P6243 active probe, although at the end of Chapter 4, I discuss briefly also the use of the matched line output.

Schematic and board

I used Eagle [110] to design the XMEGA PCB. The schematic is shown in Figure 2.8, and the board layout is shown in Figure 2.9.

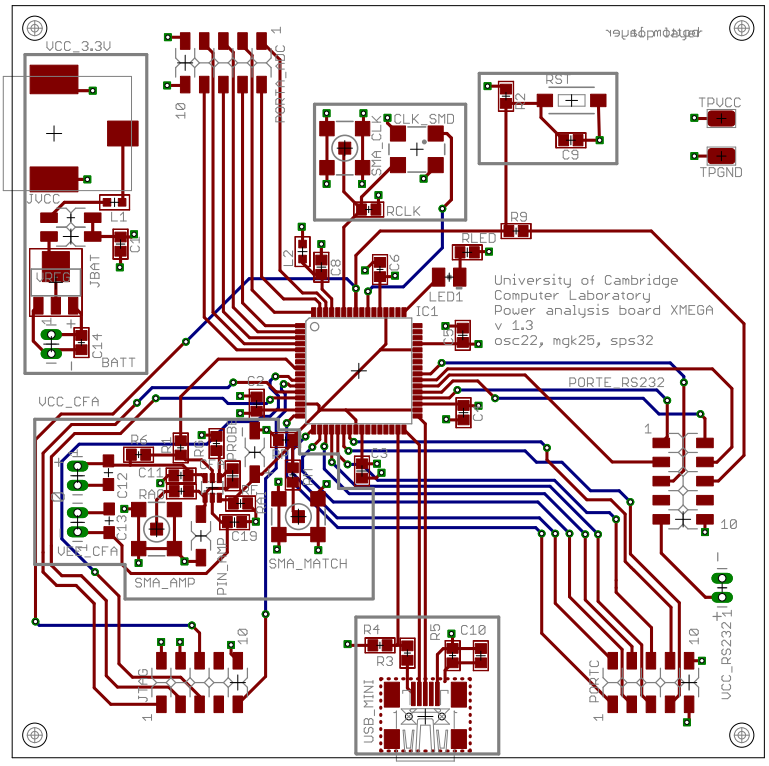

Figure 2.9: Board layout of the XMEGA PCB. Red: top side. Blue: bottom side.

2.2.3 Experimental devices

For the experiments presented in the following chapters, I built 4 instances of the XMEGA PCB, as shown in Figure 2.10. I named these devices *Alpha*, *Beta*, *Gamma* and *Delta*.

2.2.4 Acquisition setup

In order to implement side-channel attacks in practice, I used the combination of hardware and software tools described below.

Hardware

I used a Tektronix TDS 7054 oscilloscope with a P6243 active probe in order to record the voltage across the measurement resistor of the XMEGA PCB. This oscilloscope has a maximum bandwidth of 500 MHz and samples at 5 GS/s, but I used different bandwidth filters for the experiments presented in the following chapters. In order to synchronize measurements, I used one of the pins of the XMEGA, connected to the oscilloscope using a passive probe, as trigger signal.

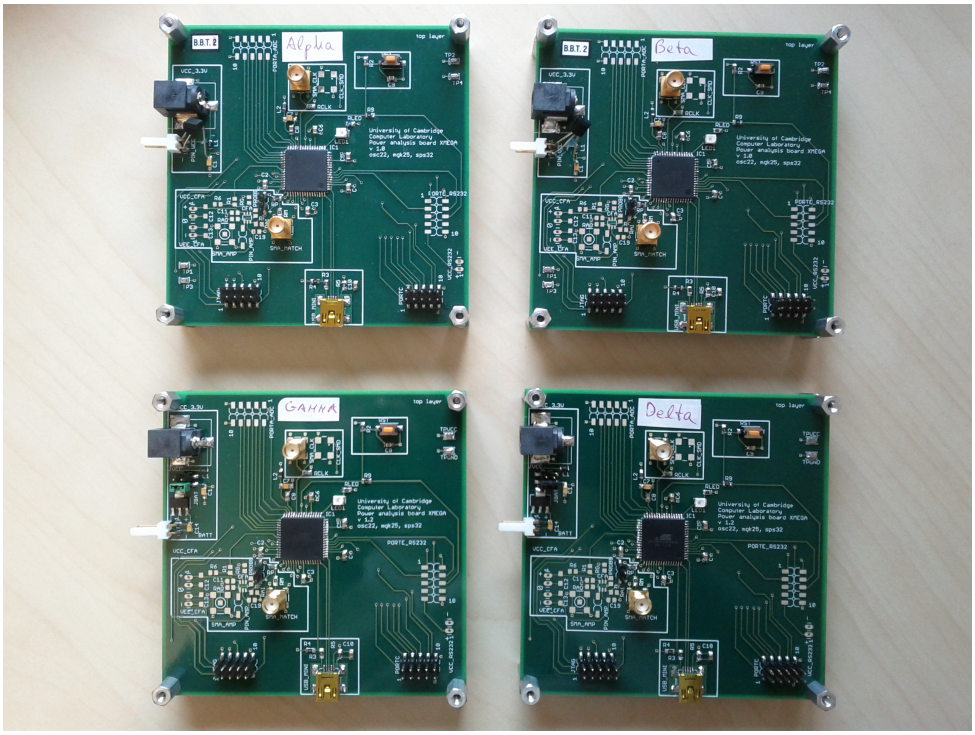

Figure 2.10: The four XMEGA PCB devices that I used in my experiments.

Following the advice of Mangard et al. [73], I used a sine wave clock signal, synthesized by a TTi TGA 1230 waveform generator, to drive the XMEGA CPU. For most experiments, I used a clock signal with 1 MHz frequency.

Depending on the experiment, I used either a set of batteries connected via a linear voltage regulator, or a TTi EL 302 power supply, to power the XMEGA PCB.

In Figure 2.11 (left), I show the XMEGA PCB while it was used for side-channel attacks, and in Figure 2.11 (right), I show the overall acquisition setup.

Software

To automate my acquisition experiments, I used a PC to communicate with both the XMEGA PCB (via the USB virtual serial line) and the oscilloscope (via GPIB [3] or VXI-11 [5]). For this, I developed a series of Python scripts that send acquisition commands to the oscilloscope, send the necessary data to the XMEGA, retrieve the waveforms from the oscilloscope and store the waveforms on a hard-disk. Also, to allow communication between the XMEGA PCB and a PC (to send data and commands), I wrote some code to implement a USB virtual serial line, based on the LUFA library [20].

Then, to run the actual side-channel attacks, I wrote a large set of MATLAB scripts, which together form a library of functions that implement the techniques described throughout the following chapters.

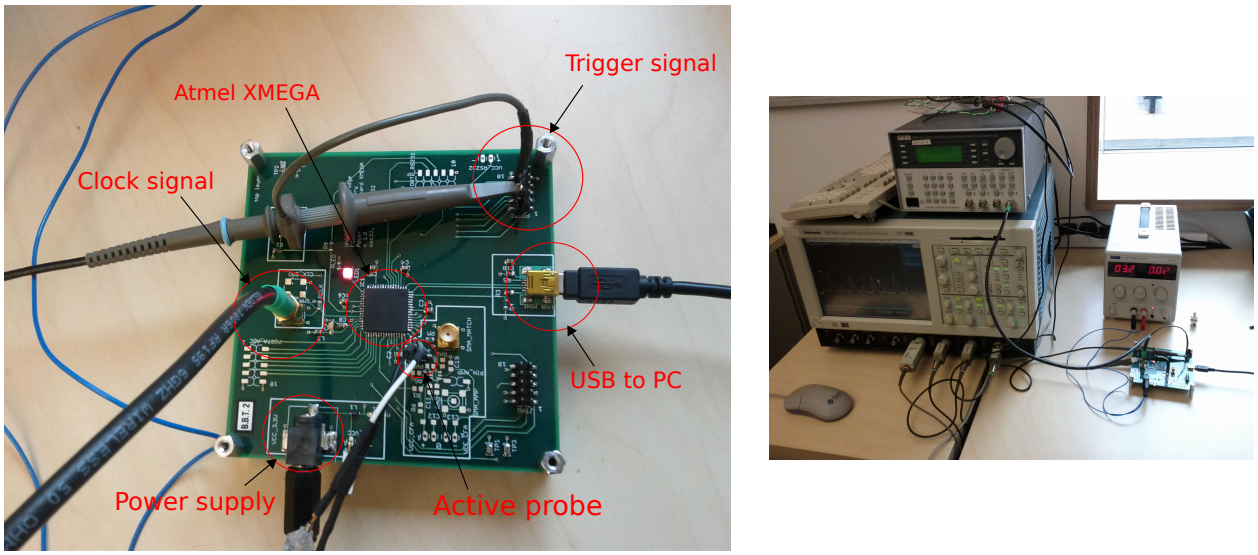

Figure 2.11: Left: XMEGA PCB while it was used for side-channel attacks. Right: overall acquisition setup.

2.3 Data sets

For the experiments presented in the following chapters, I used the different datasets described below. Throughout my experiments, I was not particularly interested in targeting a given cryptographic algorithm (e.g. AES), but rather I was mostly interested in evaluating the success of template attacks using different techniques. For this purpose, most of my datasets focus on a single or at most two LOAD instructions, rather than an entire algorithm (i.e. set of instructions). However, for completeness, one data set (*Polar*) consists of traces related to AES.

2.3.1 *Grizzly* dataset

This dataset consists of power consumption traces of the Atmel XMEGA microcontroller when running the following simple code: a consecutive sequence of LOAD instructions that transfer eight byte values from the SRAM memory to registers r8 to r15. Each LOAD instruction requires 2 clock cycles to execute. In this dataset, only the second byte takes different values, while all the other values remain unchanged. This scenario avoids the noise caused by nearby instructions due to pipelining. However, the pipeline architecture of the Atmel XMEGA AVR core processes the operand of a LOAD instruction during several clock cycles, which spreads the leakage over many samples (see for example Figure 3.8).
The exact sequence of instructions for which I obtained the power consumption is

```
5a5c:   00 00   nop             ; several previous NOPs ommited in this listing
5a5e:   fc 01   movw  r30, r24  ; 1 clock cycle, recorded traces start here
5a60:   81 90   ld  r8, Z+      ; 2 clock cycles per ld instruction
```

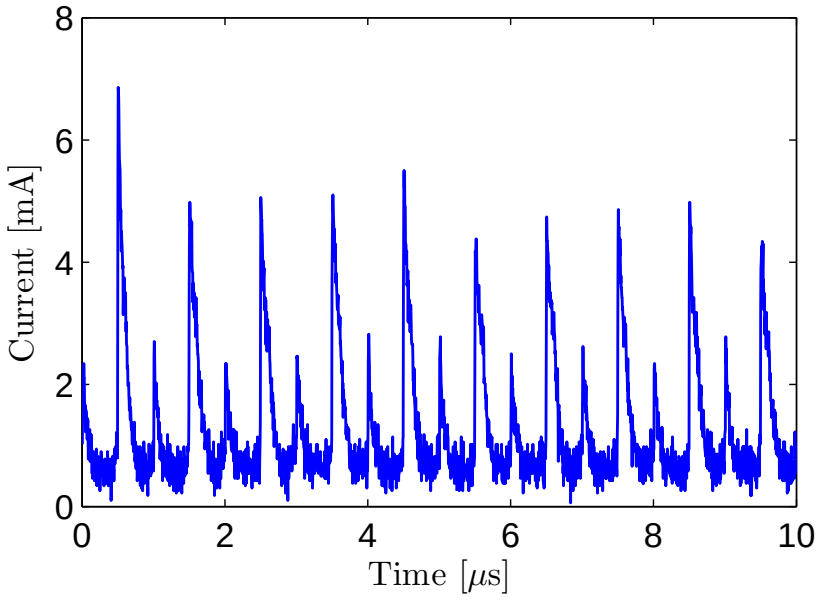

Figure 2.12: Example of leakage trace from the *Grizzly Alpha* data set.

```
5a62:   91 90   ld  r9, Z+      ; this is our target instruction (2 clock cycles)
5a64:   a1 90   ld  r10, Z+     ; we want to infer the data loaded in r9
5a66:   b1 90   ld  r11, Z+
5a68:   c1 90   ld  r12, Z+     ; recorded trace ends after first clock cycle of this ld
```

where the load instructions use the Z pointer (which refers to registers r31:r30) for indirect RAM addressing. The initial value of registers r8–r12 before the load operations is zero. The initial value of Z before the first load instruction is $0x2020$.

In my evaluations of template attacks using this dataset, as in Chapters 4, 5 and 7, the goal of the attacks is to infer the byte value processed by the second LOAD instruction. This represents a bus eavesdropping scenario, which is very general and independent of any algorithmic assumptions. An improvement in this scenario should also lead to an improvement in more particular scenarios such as targeting the S-box output v of AES (see Figure 2.3).

For the acquisition of each leakage trace I used the SAMPLE mode of the oscilloscope, sampling at 250 MS/s and using the full bandwidth (500 MHz). I powered the XMEGA PCB with batteries through a 3.3 V linear voltage regulator.

This dataset contains $n_{\mathrm{p}} = 3072$ traces for each of the 256 possible 8-bit values that can be transferred into the register r9 (a total of 786432 traces). Each trace has $m = 2500$ leakage samples. An example of leakage trace is shown in Figure 2.12.

In order to remove any dependency between the value loaded into r9 and environmental factors such as temperature, I used random permutations of the values $0, 1, \ldots, 255$ during consecutive acquisitions, ensuring that all values are used after every 256 leakage acquisitions.

Table 2.1: Parameters for real experiment

Experiment name	Bandwidth	Power Supply
E1	20 MHz	lab power supply 3.3 V (TTi EL302).
E2	20 MHz	batteries via 3.3 V voltage regulator.
E3	500 MHz	lab power supply 3.3 V (TTi EL302).
E4	500 MHz	batteries via 3.3 V voltage regulator.

To test the side-channel attacks against different devices (see Chapter 5), I performed the same experiments on my four instances of the PCB XMEGA, and I also performed a second acquisition on the device *Beta*, resulting in five actual datasets: *Grizzly Alpha*, *Grizzly Beta*, *Grizzly Gamma*, *Grizzly Delta* and *Grizzly Beta Bis*, respectively.

The *Grizzly* dataset, along with MATLAB scripts to produce some of the results from Chapter 4, is available at this URL:

`http://www.cl.cam.ac.uk/research/security/datasets/grizzly/`

2.3.2 *Koala* dataset

Using the *same* code (a sequence of LOAD instructions) and acquisition setup as for the *Grizzly* datasets (see Section 2.3.1), I varied the bandwidth of the Tektronix TDS 7054 8-bit oscilloscope, and the power supply of the XMEGA PCB *Beta*, to test the influence of these parameters on the correlation between leakage samples. As a result, I obtained the four datasets shown in Table 2.3.2, all using the *Beta* device. These datasets form the basis for the template attacks presented in Chapter 6, which target the value $k\star$ processed by the second LOAD instruction, i.e. the value that is loaded into the register r9, exactly as the evaluations from Chapters 4 and 5.

Similarly to the *Grizzly* datasets, each of the *Koala* datasets contains $n_{\mathrm{p}} = 3072$ traces for each of the 256 possible 8-bit values that can be transferred into the register r9 (a total of 786432 traces). Each trace has $m = 2500$ leakage samples.

Note that the *Koala E4* dataset is in fact the same as the *grizzly Beta* dataset.

2.3.3 *Panda* dataset

In order to test profiled attacks on 16-bit data (see Chapter 7), I varied the 8-bit values processed by two consecutive LOAD instructions, thus obtaining leakage traces that depend on 16-bit values. For this dataset, which I call *Panda*, I acquired $n_{\mathrm{p}} = 200$ traces for each of the $2^{16} = 65536$ values ($N = 13107200$ traces in total). Each trace has $m = 500$ samples, recorded with 125 MS/s using the HIRES mode of the Tektronix TDS 7054 oscilloscope (this provides ≈ 10.5 bits per sample, by averaging 40 consecutive 8-bit samples

acquired internally at 5 GS/s), and contained data over 5 LOAD instructions, of which two contained the target data and the other three processed the constant value 0. The dataset also contains $n_{\mathrm{p}} = 1000$ traces for a random selection of 512 16-bit values (512000 traces in total).

2.3.4 *Polar* dataset

The Atmel XMEGA A3U microcontroller also contains an AES hardware cryptographic module, which can perform one AES encryption with a 128-bit key in 375 CPU clock cycles. Each such AES encryption requires 10 encryption *rounds*. For more details about the AES encryption algorithm, please refer to the book by Paar and Pelzl [84, Chapter 4].

In my last dataset, which I call *Polar*, I used the XMEGA PCB to record power consumption traces targeting this AES implementation. The dataset contains a total of $N = 384000$ traces, taken while running the hardware AES encryption module with the 16-byte fixed key (in hexadecimal notation)

3c53eb11a470e4f7df71b49f2f7e72c6

and uniformly distributed 16-byte plaintexts. Each trace contains 5000 leakage samples, recorded at 500 MS/s using the HIRES mode of the Tektronix TDS 7054 oscilloscope, with 250 MHz bandwidth. The XMEGA PCB was powered from batteries via a 3.3 V linear regulator, and I provided the CPU with a sine wave clock signal at 2 MHz. These traces cover the first 20 CPU clock cycles of the AES encryption, which correspond only to the first encryption round of AES.

Chapter 3

Multivariate statistical analysis

In this chapter, I provide some background on multivariate statistical analysis, which forms the theoretical ground for the following chapters. Much of this material is based on the excellent book by Johnson and Wichern [54], although most of the examples are based on my experimental data. Readers well familiar with this topic may skip this chapter.

3.1 Leakage and random variables

As I explained in Chapter 2, we can obtain leakage traces from a microcontroller by measuring its power consumption using an oscilloscope. Such traces, as shown in Figure 2.12, may contain a large number of *leakage samples* (data points) due to the large oscilloscope sampling rate used to acquire them. For example, the trace in Figure 2.12 has $m = 2500$ samples covering a 10 µs execution interval, while traces from the DPA contest v4 [85] contain $m = 435,002$ samples. I shall refer to a leakage trace as the vector $\mathbf{x} \in \mathbb{R}^m$, which may contain an arbitrary number m of leakage samples $x_1, x_2, \ldots, x_m$.

For side-channel attacks, including the template attacks, which are my focus, we need to consider a large number of leakage traces, which we can refer to as a *statistical sample*. A statistical sample represents a subset of all possible traces that we could obtain. In practice, we will always deal with a statistical sample (subset) of all the possible traces that we may (theoretically) obtain. Note that, throughout the following chapters, I will use the term *sample* to refer both to a *statistical sample* or a *leakage sample* (data point in a trace), but its meaning should be clear from the context or otherwise I will make it clear.

Ignoring synchronisation errors for now, we can consider that each leakage sample x_j corresponds to the power consumption at some time index j, and that all the leakage samples x_j from a statistical sample correspond to the *same* underlying operation of our target microcontroller. Then, we can define X_j to be the random variable representing a leakage sample at time j, and the set of leakage samples $x_{1j}, x_{2j}, \ldots, x_{Nj}$ from a sample

with N leakage traces to be particular instantiations (actual observations) of the random variable X_j. And since we have m leakage samples per trace, we need to deal with m random variables.

The statistical study of such random variables is known as multivariate statistical analysis and will be the focus of the following sections. A good understanding of the theory governing these variables (e.g. the dependence between them, their statistical distribution) can lead to very efficient side-channel attacks, particularly for template attacks, as I will show throughout the following chapters.

3.2 Data representation

As I mentioned earlier, I will use the vector notation $\mathbf{x} \in \mathbb{R}^m$ to represent a leakage trace. Generally I shall be using a sample with a large number N (e.g. $N = 1000$) of traces and I will combine these into a *leakage matrix* $\mathbf{X} \in \mathbb{R}^{N \times m}$. Each row in the leakage matrix $\mathbf{X}$ contains a leakage vector $\mathbf{x}'$:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1{}' \\ \mathbf{x}_2{}' \\ \vdots \\ \mathbf{x}_N{}' \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nm} \end{bmatrix}, \tag{3.1}$$

where $\mathbf{x}'$ represents the transpose of a vector (or matrix) $\mathbf{x}$.

To get an idea of how some data is distributed, we can restrict ourselves to only two variables and show a scatter plot, in which we show bi-dimensional vectors as a cloud of objects in a two-dimensional plane. To illustrate this, I use a sample of $N = 1000$ traces from the *Grizzly Alpha* dataset (see Section 2.3.1), where all traces correspond to the same value loaded into register r9. In Figure 3.1, I show the scatter plot for leakage samples at $j = 878$ and $j = 1128$. It is generally difficult to plot data for more than two dimensions, but we can use several scatter plots to visualize multiple variables, two at a time.

3.3 Descriptive statistics

In order to summarize some of the most important information from large data sets we can use what is known as *descriptive statistics*. In the following I shall describe the most common statistics, which provide a measure of location, variation and linear association.

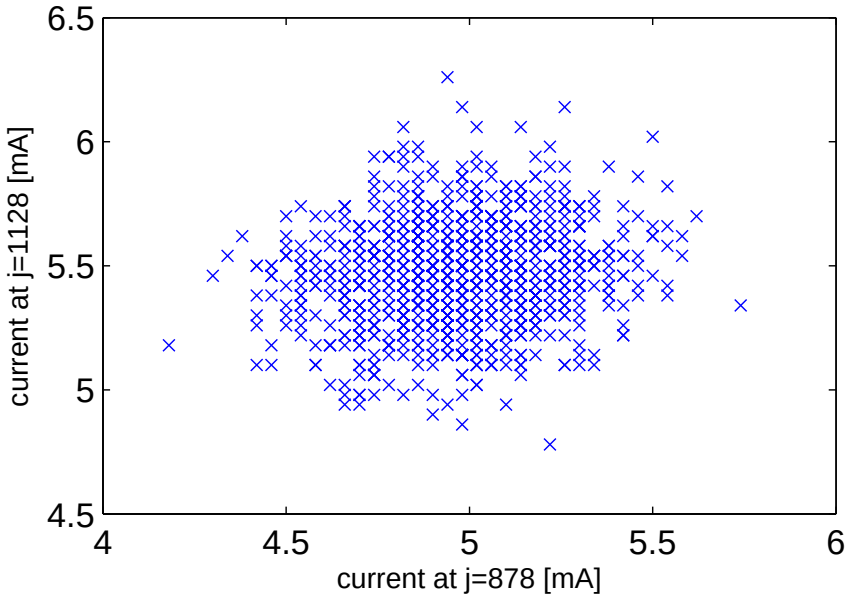

Figure 3.1: Scatter plot of leakage samples at $j = 878$ and $j = 1128$ for $N = 1000$ traces from the *Grizzly Alpha* dataset.

3.3.1 Expectation

In order to help with the definition of descriptive statistics, it is useful to first describe the *expected value*, or *expectation*, of a random variable. Following the notations of Casella and Berger [21], and to keep the definitions general, we have that if X is a random variable, then any function of X, say $g(X)$, is also a random variable.

Given a random variable $g(X)$, we can define its *expected value* or *mean* as

$$\mathbb{E}\, g(X) = \begin{cases} \int_{-\infty}^{\infty} g(x) f_X(x) dx & \text{if } X \text{ is a continuous random variable with probability density function } f_X(x) \\ \sum_{x \in \mathcal{X}} g(x) p_X(x) & \text{if } X \text{ is a discrete random variable with probability mass function } p_X(x), \end{cases} \tag{3.2}$$

where $\mathbb{E}$ denotes the expectation operator, and $\mathcal{X}$ is the discrete set of possible values that X can take.

The expected value of a random variable is merely its average value, weighted according to its probability distribution, and it can be thought of as a measure of centre.

3.3.2 Sample mean

Let us start by looking at the observations $x_{1j}, x_{2j}, \ldots, x_{Nj}$ corresponding to the single random variable X_j. Given these observations, we can compute their arithmetic average

$$\bar{x}_j = \frac{1}{N} \sum_{i=1}^{N} x_{ij}, \tag{3.3}$$

which is known as their *sample mean*. The sample mean provides a measure of location, a central value for a set of numbers. As an example, for the data used in Figure 3.1, we have $\bar{x}_{878} = 4.97$ and $\bar{x}_{1128} = 5.47$, which shows the location of the centre of the data along the x and y axis respectively.

We can also define the *real* mean of the random variable X_j as

$$\mu_j = \mathbb{E}\, X_j. \tag{3.4}$$

In practice, when dealing with observations (samples) from an unknown distribution (as our side-channel leakage traces), we cannot determine the real mean, which is why most of the time I shall use the sample mean. If the distribution is known, then we can use (3.2) to compute its real mean.

We can now compute the sample mean for all the leakage samples (corresponding to the random variables $X_1, X_2, \ldots, X_m$), obtaining the m sample means $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m$. I shall generally combine these into the *sample mean vector* $\bar{\mathbf{x}} \in \mathbb{R}^m$. In practice, given a sample of leakage vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, we can compute the sample mean vector directly as

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i. \tag{3.5}$$

Furthermore, using the leakage matrix $\mathbf{X}$ from (3.1), we can compute the sample mean vector as

$$\bar{\mathbf{x}} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_m \end{bmatrix} = \frac{1}{N} \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1N} \\ x_{21} & x_{22} & \ldots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \ldots & x_{mN} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \frac{1}{N} \mathbf{X}'\mathbf{1}. \tag{3.6}$$

As shown by the previous equations, I shall use Greek letters (e.g. μ_j) to represent the real, but generally unknown, statistics (e.g. the mean), and roman letters (e.g. $\bar{x}_j$) to represent the sample statistic (e.g. the sample mean). Also, I will generally refer to a statistic such as the sample mean simply as the mean. Its symbol should clarify to which statistic I am referring to, or otherwise I shall make it clear.

3.3.3 Sample variance and sample standard deviation

Another useful statistic is the *variance*, which provides a measure of the degree of spread of the data around its mean. Given our set of observations $x_{1j}, x_{2j}, \ldots, x_{Nj}$, corresponding to the random variable X_j, we can compute their *sample variance* as

$$s_{jj} = \frac{1}{N-1} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j)^2. \tag{3.7}$$

The use of the identical double subscript in s_{jj} will become clear in the next section when I shall describe the covariance.

As with the mean, we can also define the *real* variance of the random variable X_j as

$$\mathbf{Var}(X_j) = \sigma_{jj} = \mathbb{E}\,(X_j - \mathbb{E}\,X_j)^2 = \mathbb{E}\,(X_j - \mu_j)^2. \tag{3.8}$$

Small values of the variance mean that the data is very close to its mean (e.g. $\sigma_{jj} = 0$ means that X_j is equal to $\mathbb{E}\,X_j$ with probability 1), while large values mean that the data is very variable.

Notice also the divisor $N - 1$ in the formula of the sample variance instead of N, to obtain an *unbiased* estimator of the sample variance. The use of an unbiased estimator is important when dealing with a small sample size, but for large N the use of either $N - 1$ or N should provide similar results. For the unbiased sample variance estimator, we have

$$\mathbb{E}\,s_{jj} = \sigma_{jj}, \tag{3.9}$$

while the biased estimator

$$s_{jj}^{\text{biased}} = \frac{N-1}{N} s_{jj} \tag{3.10}$$

will contain a bias equal to

$$(\text{bias}) = \mathbb{E}\,(s_{jj}^{\text{biased}}) - \sigma_{jj} = \mathbb{E}\,(\frac{N-1}{N} s_{jj}) - \mathbb{E}\,(s_{jj}) = \mathbb{E}\,(-\frac{1}{N} s_{jj}) = -\frac{1}{N}\sigma_{jj}. \tag{3.11}$$

The positive square root of the sample variance, $\sqrt{s_{jj}}$, is known as the *sample standard deviation* of our sample. The standard deviation has the same qualitative interpretation as the variance but is generally easier to interpret because it has the same measurement unit as the data from which it is computed. For the data used in Figure 3.1, we have $\sqrt{s_{878,878}} = 0.23$ mA and $\sqrt{s_{1128,1128}} = 0.21$ mA.

3.3.4 Sample covariance

A third important statistic is the *covariance*, which measures the linear association between two variables. Given the observations $x_{1j}, x_{2j}, \ldots, x_{Nj}$ and $x_{1k}, x_{2k}, \ldots, x_{Nk}$, corresponding to the two random variables X_j and X_k, we can compute the *sample covariance* as

$$s_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k). \tag{3.12}$$

Note that $s_{jk} = s_{kj}$ and for $j = k$ the sample covariance reduces to the sample variance.

If the distributions of the random variables are known, we can compute the *real* covariance

$$\begin{aligned}\sigma_{jk} &= \mathbb{E}\,(X_j - \mu_j)(X_k - \mu_k) \qquad (3.13)\\ &= \begin{cases} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x_j-\mu_j)(x_k-\mu_k)f_{X_jX_k}(x_j,x_k)dx_jdx_k & \text{if } X_j,\ X_k \text{ are continuous random variables with joint probability density function } f_{X_jX_k}(x_j,x_k) \\ \sum_{x_j}\sum_{x_k}(x_j-\mu_j)(x_k-\mu_k)p_{X_jX_k}(x_j,x_k) & \text{if } X_j,\ X_k \text{ are discrete random variables with joint probability mass function } p_{X_jX_k}(x_j,x_k). \end{cases}\end{aligned}$$

Note that, in the computation of the real covariance, the cross-products $(x_j-\mu_j)(x_k-\mu_k)$ are weighted according to a *joint* probability function $f_{X_jX_k}(x_j,x_k)$. It is this function that describes the association between the random variables. One common such function is the multivariate normal density function, which I describe in Section 3.6.

The random variables $X_1, X_2, \ldots, X_m$ are *mutually statistically independent* if their joint probability function can be factored as

$$f_{X_1X_2\ldots X_m}(x_1,x_2,\ldots,x_m) = f_{X_1}(x_1)f_{X_2}(x_2)\ldots f_{X_m}(x_m). \qquad (3.14)$$

For any two random variables X_j, X_k, statistical independence implies $\sigma_{jk} = 0$. However, $\sigma_{jk} = 0$ does not imply statistical independence, because there might be some *non-linear* dependency between the variables X_j and X_k, which is not captured by the covariance.

I shall use the notations $\mathbf{Cov}(X_j, X_k) = \sigma_{jk}$ to refer to the covariance between two random variables, and $\mathbf{Cov}(\mathbf{x}_j, \mathbf{x}_k) = s_{jk}$ to refer to the sample covariance between the observations $x_{1j}, x_{2j}, \ldots, x_{Nj}$ and $x_{1k}, x_{2k}, \ldots, x_{Nk}$, corresponding to the two random variables X_j and X_k. Furthermore, given a leakage matrix $\mathbf{X} \in \mathbb{R}^{N\times m}$ as in (3.1), I shall define the *sample covariance matrix* $\mathbf{S} \in \mathbb{R}^{m\times m}$ as

$$\begin{aligned}\mathbf{S} = \mathbf{Cov}(\mathbf{X}) &= \begin{bmatrix} \mathbf{Cov}(\mathbf{x}_1,\mathbf{x}_1) & \mathbf{Cov}(\mathbf{x}_1,\mathbf{x}_2) & \ldots & \mathbf{Cov}(\mathbf{x}_1,\mathbf{x}_m) \\ \mathbf{Cov}(\mathbf{x}_2,\mathbf{x}_1) & \mathbf{Cov}(\mathbf{x}_2,\mathbf{x}_2) & \ldots & \mathbf{Cov}(\mathbf{x}_2,\mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Cov}(\mathbf{x}_m,\mathbf{x}_1) & \mathbf{Cov}(\mathbf{x}_m,\mathbf{x}_2) & \ldots & \mathbf{Cov}(\mathbf{x}_m,\mathbf{x}_m) \end{bmatrix} \\ &= \begin{bmatrix} s_{11} & s_{12} & \ldots & s_{1m} \\ s_{21} & s_{22} & \ldots & s_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1} & s_{m2} & \ldots & s_{mm} \end{bmatrix}. \qquad (3.15)\end{aligned}$$

Similarly, we can define the *real* covariance $\mathbf{\Sigma}$ of the random variables $X_1, X_2, \ldots, X_m$ as

$$\begin{aligned}\mathbf{\Sigma} &= \begin{bmatrix} \mathbf{Cov}(X_1, X_1) & \mathbf{Cov}(X_1, X_2) & \ldots & \mathbf{Cov}(X_1, X_m) \\ \mathbf{Cov}(X_2, X_1) & \mathbf{Cov}(X_2, X_2) & \ldots & \mathbf{Cov}(X_2, X_m) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Cov}(X_m, X_1) & \mathbf{Cov}(X_m, X_2) & \ldots & \mathbf{Cov}(X_m, X_m) \end{bmatrix} \\ &= \begin{bmatrix} \sigma_{11} & \sigma_{12} & \ldots & \sigma_{1m} \\ \sigma_{21} & \sigma_{22} & \ldots & \sigma_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{m1} & \sigma_{m2} & \ldots & \sigma_{mm} \end{bmatrix}. \end{aligned} \tag{3.16}$$

Note that the diagonal values of these covariance matrices contain the variances, which is why these matrices are also referred to as the variance-covariance matrices. Also, because $s_{jk} = s_{kj}$ and $\sigma_{jk} = \sigma_{kj}$, both the sample covariance and the real covariance matrices are symmetric, i.e. $\mathbf{S} = \mathbf{S}'$ and $\mathbf{\Sigma} = \mathbf{\Sigma}'$.

Instead of computing the individual sample covariances s_{jk} of the sample covariance matrix $\mathbf{S}$ we can use the following matrix operations, which allow us to take advantage of vectorised linear-algebra routines, resulting in faster computations. Starting from the leakage matrix $\mathbf{X} \in \mathbb{R}^{N \times m}$, we first compute the *zero-mean* leakage matrix

$$\widetilde{\mathbf{X}} = \mathbf{X} - \frac{1}{N}\mathbf{1}\mathbf{1}'\mathbf{X} = \begin{bmatrix} x_{11} - \bar{x}_1 & x_{12} - \bar{x}_2 & \ldots & x_{1m} - \bar{x}_m \\ x_{21} - \bar{x}_1 & x_{22} - \bar{x}_2 & \ldots & x_{2m} - \bar{x}_m \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} - \bar{x}_1 & x_{N2} - \bar{x}_2 & \ldots & x_{Nm} - \bar{x}_m \end{bmatrix}, \tag{3.17}$$

where $\mathbf{1}' = [1, 1, \ldots, 1]$ has N elements. Then, we can compute the sample covariance matrix as

$$\mathbf{S} = \frac{1}{N-1}\widetilde{\mathbf{X}}'\widetilde{\mathbf{X}}. \tag{3.18}$$

3.3.5 Sample correlation

As a last important statistic, the *sample correlation coefficient* (also known as *Pearson's product-moment correlation coefficient*) also provides a measure of linear association. Given the observations $x_{1j}, x_{2j}, \ldots, x_{Nj}$ and $x_{1k}, x_{2k}, \ldots, x_{Nk}$, corresponding to the two random variables X_j and X_k, we can compute the sample correlation coefficient as

$$r_{jk} = \frac{s_{jk}}{\sqrt{s_{jj}}\sqrt{s_{kk}}} = \frac{\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)}{\sqrt{\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)^2}\sqrt{\sum_{i=1}^{N}(x_{ik} - \bar{x}_k)^2}}. \tag{3.19}$$

Note that in this case it does not matter if we use the divisor N or $N - 1$ for the computation of s_{jk}, s_{jj} and s_{kk}.

The sample correlation coefficient is just a standardized version of the sample covariance. That is, if before the computation of the sample covariance from (3.12) we replace the values x_{ij} and x_{ik} by the *standardized* values $(x_{ij} - \bar{x}_j)/\sqrt{s_{jj}}$ and $(x_{ik} - \bar{x}_k)/\sqrt{s_{kk}}$, then we obtain the sample correlation coefficient from (3.19). Note that $r_{jj} = 1$, and if all $s_{jj} = 1$, then all $r_{jj} = s_{jj}$.

Although both the sample covariance and the sample correlation provide a measure of linear association, the sample correlation is generally easier to interpret because it has bounded values: $-1 \leq r_{jk} \leq 1$. $|r_{jk}| = 1$ implies that $X_k = a \cdot X_j + b,\ \forall a, b \in \mathbb{R}$.

Also, $r_{jk} = 0$ implies a lack of linear association, $r_{jk} < 0$ implies that one variable tends to be above its average when the other is below, while $r_{jk} > 0$ implies that both variables tend to be together either above or below their average. The same holds for s_{jk}.

For the data from Figure 3.1, the sample covariance is $s_{878,1128} = 0.0069$ and the sample correlation is $r_{878,1128} = 0.14$, which means that there is a non-negligible dependency between the leakage samples at times $j = 878$ and $j = 1128$.

Similarly to the sample covariance matrix $\mathbf{S}$ from (3.15), given a leakage matrix $\mathbf{X} \in \mathbb{R}^{N \times m}$, we can also define the *sample correlation matrix* $\mathbf{R} \in \mathbb{R}^{m \times m}$ as

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \dots & r_{mm} \end{bmatrix}, \tag{3.20}$$

which is also symmetric.

We can compute $\mathbf{R}$ from $\mathbf{S}$ and vice versa as follows. First, compute the *sample standard deviation matrix*

$$\mathbf{D} = \begin{bmatrix} \sqrt{s_{11}} & 0 & \dots & 0 \\ 0 & \sqrt{s_{22}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sqrt{s_{mm}} \end{bmatrix} \tag{3.21}$$

and its inverse

$$\mathbf{D}^{-1} = \begin{bmatrix} \frac{1}{\sqrt{s_{11}}} & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{s_{22}}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sqrt{s_{mm}}} \end{bmatrix}. \tag{3.22}$$

Then, we have

$$\mathbf{R} = \mathbf{D}^{-1}\mathbf{S}\mathbf{D}^{-1} \tag{3.23}$$

and

$$\mathbf{S} = \mathbf{D}\mathbf{R}\mathbf{D}. \tag{3.24}$$

Note that $\mathbf{R}$ can be obtained from the information in $\mathbf{S}$ alone, but to compute $\mathbf{S}$ we need both $\mathbf{R}$ and $\mathbf{D}$.

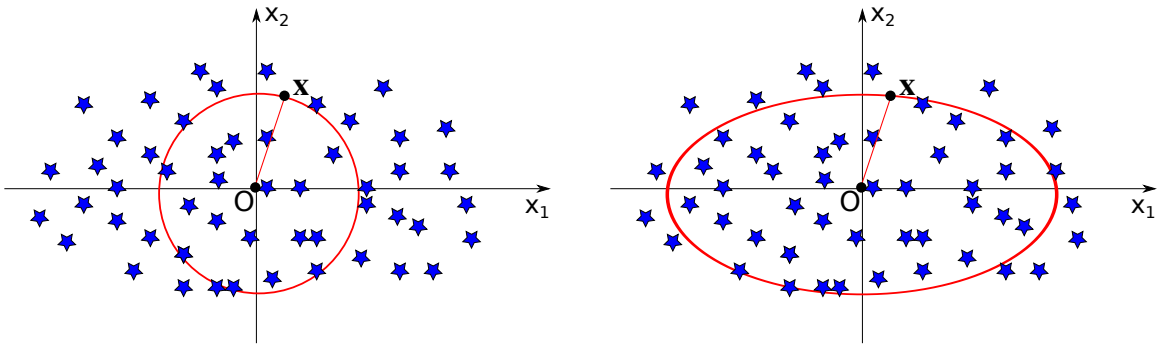

Figure 3.2: Left: points at a constant *Euclidean* distance. Right: points at a constant *statistical* distance.

3.4 Statistical distance

I now present the *statistical distance*, also known as the Mahalanobis distance [72], which is essential for the multivariate statistical analysis techniques used in later chapters.

Given any two leakage vectors $\mathbf{x}_1 \in \mathbb{R}^m$ and $\mathbf{x}_2 \in \mathbb{R}^m$, we shall often need to compute a distance between them, a measure of how far appart they are. This can be directly useful, for example, to find which vector, from a set of vectors, is closest to a given point. For this purpose, in the following, we should consider the leakage samples $x_1, x_2, \ldots, x_m$ of a leakage vector $\mathbf{x}$ as coordinates in an m-dimensional vector space.

Let's start with the case $m = 2$, using samples from two random variables X_1 and X_2 having $\mu_1 = \mu_2 = 0$, $\sigma_{11} > \sigma_{22}$ and $\sigma_{12} = 0$ (i.e. no correlation), as shown in Figure 3.2. Using the classic *Euclidean* distance, we would measure the distance between a vector $\mathbf{x}' = [x_1, x_2]$ and the origin $\mathbf{0}' = [0, 0]$ using the formula

$$d(\mathbf{x}, \mathbf{0}) = \sqrt{x_1^2 + x_2^2} = \mathbf{x}'\mathbf{x}. \tag{3.25}$$

All the points that have the same distance from the origin are represented by the red circle in Figure 3.2 (left).

However, given our data (where $\sigma_{11} > \sigma_{22}$), we could say that it is more likely to find data that deviates more along the X_1 dimension than along X_2. To accommodate for this situation, where the different dimensions of our leakage vectors have different variability, but also to accommodate for correlation between leakage samples, it will be better to use a *statistical distance*. For the data shown in Figure 3.2 we can compensate for the different variances by simply dividing each component by its variance, obtaining the new distance between the vector $\mathbf{x}' = [x_1, x_2]$ and the origin $\mathbf{0}' = [0, 0]$ as

$$d(\mathbf{0}, \mathbf{x}) = \sqrt{\frac{x_1^2}{\sigma_{11}} + \frac{x_2^2}{\sigma_{22}}}. \tag{3.26}$$

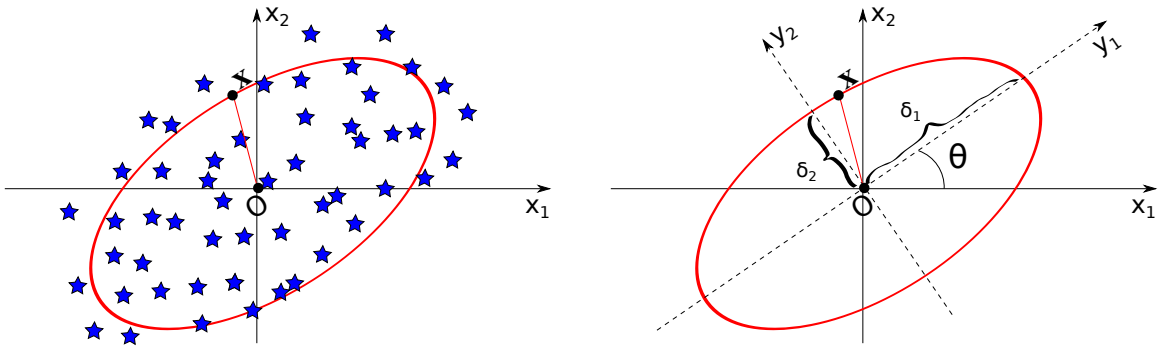

Figure 3.3: Left: correlated points at a constant *statistical* distance. Right: new axes to match rotation of samples due to correlation.

All points having the same *statistical* distance to the origin are represented by the red ellipse in Figure 3.2 (right). This distance can be easily extended to any two m-dimensional vectors $\mathbf{x}' = [x_1, x_2, \ldots, x_m]$ and $\mathbf{q}' = [q_1, q_2, \ldots, q_m]$ as

$$d(\mathbf{x}, \mathbf{q}) = \sqrt{\frac{(x_1 - q_1)^2}{\sigma_{11}} + \frac{(x_2 - q_2)^2}{\sigma_{22}} + \ldots \frac{(x_m - q_m)^2}{\sigma_{mm}}}. \tag{3.27}$$

Now let us examine another situation, where we have a positive correlation $\sigma_{12} > 0$ between the two random variables, as shown in Figure 3.3. We can see that the overall scatter plot is very similar to the previous example, but only rotated by some angle θ. Therefore, we could represent our sample in a rotated coordinate system, represented by the random variables Y_1 and Y_2, as shown in the right side of the figure. Using these new variables, the distance between our leakage vector $\mathbf{x} = [y_1, y_2]$ and the origin $\mathbf{0} = [0, 0]$ has the same form as before, that is

$$d(\mathbf{0}, \mathbf{x}) = \sqrt{\frac{y_1^2}{\tilde{\sigma}_{11}} + \frac{y_2^2}{\tilde{\sigma}_{22}}}, \tag{3.28}$$

where $\tilde{\sigma}_{11}$ and $\tilde{\sigma}_{22}$ are the variances of Y_1 and Y_2, respectively.

However, we are interested in expressing the distance in terms of the original variables X_1 and X_2. The relation between the elements of $\mathbf{x}$ in the two coordinate systems is given by

$$\begin{aligned} y_1 &= x_1 \cos\theta + x_2 \sin\theta \\ y_2 &= -x_1 \sin\theta + x_2 \cos\theta. \end{aligned} \tag{3.29}$$

Then, we can write the statistical distance as

$$d(\mathbf{0}, \mathbf{x}) = \sqrt{a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2}, \tag{3.30}$$

where the coefficients a, which depend on the original variances σ_{11}, σ_{12}, σ_{22} and the rotation angle θ, are such that the distance is *nonnegative* for *all* possible leakage vectors $\mathbf{x} \in \mathbb{R}^2$. What is important to notice in this formula is the apparition of the crossproduct term $2a_{12}x_1x_2$, determined by the nonzero correlation r_{12}. What is also very important is that we do not have to worry about the derivation of the a coefficients. As I will explain shortly, these will be given by the covariance matrix.

We can generalize the above formula to compute the statistical distance between any two m-dimensional vectors $\mathbf{x}' = [x_1, x_2, \ldots, x_m]$ and $\mathbf{q}' = [q_1, q_2, \ldots, q_m]$ as

$$d(\mathbf{x}, \mathbf{q}) = \sqrt{\begin{array}{ll} a_{11}(x_1 - q_1)^2 & + a_{22}(x_2 - q_2)^2 + \ldots + a_{mm}(x_m - q_m)^2 \\ & + 2a_{12}(x_1 - q_1)(x_2 - q_2) + \ldots \\ & + 2a_{m-1,m}(x_{m-1} - q_{m-1})(x_m - q_m) \end{array}} . \tag{3.31}$$

3.4.1 Quadratic forms and positive definite matrices

If we let

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mm} \end{bmatrix}, \tag{3.32}$$

with $a_{jk} = a_{kj}$, and set $\mathbf{q} = \mathbf{0}$ for simplicity, we can rewrite the statistical distance as

$$d(\mathbf{x}, \mathbf{0}) = \sqrt{\mathbf{x}'\mathbf{A}\mathbf{x}} = \sqrt{\begin{array}{ll} a_{11}x_1^2 & + a_{22}x_2^2 + \ldots + a_{mm}x_m^2 + 2a_{12}x_1x_2 + \ldots \\ & + 2a_{m-1,m}x_{m-1}x_m \end{array}} \tag{3.33}$$

From (3.33) we can see that $\mathbf{x}'\mathbf{A}\mathbf{x}$ is a linear combination of the squares x_j^2 and the crossproducts x_jx_k. For this reason, $\mathbf{x}'\mathbf{A}\mathbf{x}$ is called a *quadratic form*. Furthermore, as mentioned earlier, $\mathbf{x}'\mathbf{A}\mathbf{x} \geq 0, \forall \mathbf{x} \in \mathbb{R}^m$. For this reason, both the matrix $\mathbf{A}$ and its quadratic form $\mathbf{x}'\mathbf{A}\mathbf{x}$ are said to be *nonnegative definite* or *semipositive definite*. Any symmetric matrix satisfying this property is semipositive definite. Morover, if $\mathbf{x}'\mathbf{A}\mathbf{x} > 0, \forall \mathbf{x} \neq \mathbf{0}$, then $\mathbf{A}$ and its quadratic form are said to be *positive definite*.

Now it is interesting to find under which condition we can guarantee that a *symmetric* matrix $\mathbf{A}$ is semipositive definite, and hence that it can be used in the computation of the *statistical distance*[1]. This leads us to the concepts of *eigenvectors* and *eigenvalues*.

[1]Any measure of distance $d(\mathbf{x}, \mathbf{q})$ between two vectors $\mathbf{x}$ and $\mathbf{q}$ should satisfy the following conditions: a) $d(\mathbf{x}, \mathbf{q}) = d(\mathbf{q}, \mathbf{x})$, b) $d(\mathbf{x}, \mathbf{q}) > 0$, if $\mathbf{x} \neq \mathbf{q}$, c) $d(\mathbf{x}, \mathbf{q}) = 0$, if $\mathbf{x} = \mathbf{q}$, d) $d(\mathbf{x}, \mathbf{q}) \leq d(\mathbf{x}, \mathbf{r}) + d(\mathbf{r}, \mathbf{q})$ for any other vector $\mathbf{r}$.

Eigenvectors and eigenvalues

A *square* matrix (having equal number of rows and columns) $\mathbf{A} \in \mathbb{R}^{m \times m}$ has an eigenvalue $\lambda \in \mathbb{R}$, with corresponding eigenvector $\mathbf{e} \neq \mathbf{0} \in \mathbb{R}^m$, if

$$\mathbf{Ae} = \lambda \mathbf{e}. \tag{3.34}$$

The matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ has m eigenvectors (and corresponding eigenvalues), all mutually perpendicular, i.e. $\mathbf{e}_j{}'\mathbf{e}_k = 0, \forall j \neq k$. We usually normalize all eigenvectors, such that $\mathbf{e}_1{}'\mathbf{e}_1 = \mathbf{e}_2{}'\mathbf{e}_2 = \ldots = \mathbf{e}_m{}'\mathbf{e}_m = 1$.

The eigenvalues of a square matrix $\mathbf{A}$ can be found by solving the *characteristic equation*

$$|\mathbf{A} - \lambda \mathbf{I}| = 0. \tag{3.35}$$

Then, we can find the corresponding eigenvector from (3.34). However, in practice, we use linear algebra libraries (such as MATLAB) to obtain the *singular value decomposition*

$$\mathbf{A} = \mathbf{UDU}' \tag{3.36}$$

of a square matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$, which gives the matrix of eigenvectors $\mathbf{U} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m]$, and the diagonal matrix $\mathbf{D}$ having the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ along its diagonal. Note that because the normalized eigenvectors are all mutually perpendicular, the matrix $\mathbf{U}$ is *orthonormal*, which means that $\mathbf{UU}' = \mathbf{U}'\mathbf{U} = \mathbf{I}$, and implies that $\mathbf{U}'$ is the inverse of $\mathbf{U}$.

An important property of a symmetric matrix $\mathbf{A}$ is that it has the *spectral decomposition*

$$\mathbf{A} = \lambda_1 \mathbf{e}_1 \mathbf{e}_1{}' + \lambda_2 \mathbf{e}_2 \mathbf{e}_2{}' + \ldots + \lambda_m \mathbf{e}_m \mathbf{e}_m{}'. \tag{3.37}$$

We can now use the spectral decomposition of $\mathbf{A}$ to express the quadratic form $\mathbf{x}'\mathbf{Ax}$ and obtain

$$\begin{aligned} \mathbf{x}'\mathbf{Ax} &= \lambda_1 \mathbf{x}'\mathbf{e}_1\mathbf{e}_1{}'\mathbf{x} + \lambda_2 \mathbf{x}'\mathbf{e}_2\mathbf{e}_2{}'\mathbf{x} + \ldots + \lambda_m \mathbf{x}'\mathbf{e}_m\mathbf{e}_m{}'\mathbf{x} \\ &= \lambda_1 (\mathbf{x}'\mathbf{e}_1)^2 + \lambda_2 (\mathbf{x}'\mathbf{e}_2)^2 + \ldots + \lambda_m (\mathbf{x}'\mathbf{e}_m)^2. \end{aligned} \tag{3.38}$$

From (3.38) we can derive a very important conclusion: a symmetric matrix $\mathbf{A}$ is positive definite ($\mathbf{x}'\mathbf{Ax} > 0, \forall \mathbf{x} \neq \mathbf{0}$) iff all its eigenvalues are positive. Also, $\mathbf{A}$ is semipositive definite ($\mathbf{x}'\mathbf{Ax} \geq 0, \forall \mathbf{x} \neq \mathbf{0}$) iff all its eigenvalues are greater than or equal to zero.

Returning to Figure 3.3, any vector $\mathbf{x} \in \mathbb{R}^2$ on the red ellipse satisfies the relation

$$d^2(\mathbf{0}, \mathbf{x}) = \mathbf{x}'\mathbf{Ax} = \lambda_1 (\mathbf{x}'\mathbf{e}_1)^2 + \lambda_2 (\mathbf{x}'\mathbf{e}_2)^2 = \lambda_1 y_1^2 + \lambda_2 y_1^2 = c^2, \tag{3.39}$$

which represents the ellipse in y_1 and y_2. Now, a very interesting fact is that the vectors $\mathbf{v}_1 = c\lambda_1^{-1/2}\mathbf{e}_1$ and $\mathbf{v}_2 = c\lambda_2^{-1/2}\mathbf{e}_2$ both satisfy (3.39), and represent the intersection with the principal and respectively secondary axes of the ellipse. Therefore, the eigenvectors $\mathbf{e}_1$ and $\mathbf{e}_2$ give precisely the directions of the random variables Y_1 and Y_2. Therefore, the

statistical distance provides the expected transformation, such that in the rotated space (given by the eigenvectors of $\mathbf{A}$) there is no correlation remaining between the variables. This simplifies to using the standard Euclidean distance in the new space, as shown by the right hand side of (3.39).

All the above results hold also when computing the statistical distance between a vector $\mathbf{x}$ and an arbitrary *fixed* vector $\mathbf{q}$. That is, the statistical distance still has the quadratic form

$$d(\mathbf{x}, \mathbf{q}) = \sqrt{(\mathbf{x} - \mathbf{q})' \mathbf{A} (\mathbf{x} - \mathbf{q})}. \tag{3.40}$$

A final question is how to obtain the matrix $\mathbf{A}$, such that the ellipse generated by the points at a constant distance $\mathbf{x}'\mathbf{A}\mathbf{x} = c^2$ indeed lies along the directions of the *independent* variables Y_1 and Y_2. That is, how can we find the matrix $\mathbf{A}$ that provides the correct rotation θ and eliminates the correlation between variables? This is explained next.

3.4.2 Using the covariance matrix in the statistical distance

Let

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_m \end{bmatrix} \tag{3.41}$$

be a vector of m random variables with associated covariance $\mathbf{\Sigma}$, as in (3.16), and let $\mathbf{Q} \in \mathbb{R}^{m \times m}$ be some arbitrary matrix of coefficients. Then, one important fact is that the new vector of random variables

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \dots \\ Y_m \end{bmatrix} = \mathbf{Q}\mathbf{X} = \begin{bmatrix} q_{11}X_1 + \ldots + q_{1m}X_m \\ q_{21}X_1 + \ldots + q_{2m}X_m \\ \vdots \\ q_{m1}X_1 + \ldots + q_{mm}X_m \end{bmatrix} \tag{3.42}$$

has covariance $\mathbf{Q}\mathbf{\Sigma}\mathbf{Q}'$. As a result, the projection $\mathbf{y} = \mathbf{Q}\mathbf{x}$ of any leakage vector $\mathbf{x}$ from a distribution with covariance $\mathbf{\Sigma}$, will follow a distribution with covariance $\mathbf{Q}\mathbf{\Sigma}\mathbf{Q}'$.

Since $\mathbf{\Sigma}$ is a symmetric matrix, we can use (3.36) and (3.37) to obtain

$$\mathbf{\Sigma} = \sum_{j=1}^{m} \lambda_j \mathbf{e}_j \mathbf{e}_j{}' = \mathbf{U}\mathbf{D}\mathbf{U}', \tag{3.43}$$

where $\mathbf{U} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m]$, and $\text{diag}(\mathbf{D}) = [\lambda_1, \lambda_2, \ldots, \lambda_m]$ are now the eigenvectors and eigenvalues of $\mathbf{\Sigma}$. A very important observation now is that we can easily compute the inverse of $\mathbf{\Sigma}$ as

$$\mathbf{\Sigma}^{-1} = \mathbf{U}\mathbf{D}^{-1}\mathbf{U}' = \sum_{j=1}^{m} \frac{1}{\lambda_j} \mathbf{e}_j \mathbf{e}_j{}', \tag{3.44}$$

because $(\mathbf{U}\mathbf{D}^{-1}\mathbf{U}')\mathbf{U}\mathbf{D}\mathbf{U}' = \mathbf{U}\mathbf{D}\mathbf{U}'(\mathbf{U}\mathbf{D}^{-1}\mathbf{U}') = \mathbf{U}\mathbf{U}' = \mathbf{I}$ due to the orthonormality of $\mathbf{U}$. This implies that $\mathbf{\Sigma}$ and $\mathbf{\Sigma}^{-1}$ have the *same* eigenvectors, while the eigenvalues follow an inverse relationship.

Let us now use $\mathbf{\Sigma}^{-1}$ instead of the matrix $\mathbf{A}$ to compute the quadratic form from (3.38). We have that

$$\mathbf{x}'\mathbf{\Sigma}^{-1}\mathbf{x} = \sum_{j=1}^{m} \frac{1}{\lambda_j}(\mathbf{x}'\mathbf{e}_j)(\mathbf{e}_j{}'\mathbf{x}) = \sum_{j=1}^{m} \frac{1}{\lambda_j}(\mathbf{e}_j{}'\mathbf{x})^2 = \sum_{j=1}^{m} \left(\frac{1}{\sqrt{\lambda_j}}\mathbf{e}_j{}'\mathbf{x}\right)^2 = \sum_{j=1}^{m} z_j^2. \tag{3.45}$$

So we are almost there: we now have the squared statistical distance $d^2 = \mathbf{x}'\mathbf{\Sigma}^{-1}\mathbf{x}$ expressed in Euclidean form. However, to verify that indeed $\mathbf{\Sigma}^{-1}$ is the matrix we were looking for in the computation of the statistical distance, we still need to show that there is no correlation between the variables z_j.

If we let $\mathbf{Q} = \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}}\mathbf{e}_1{}' \\ \frac{1}{\sqrt{\lambda_2}}\mathbf{e}_2{}' \\ \vdots \\ \frac{1}{\sqrt{\lambda_m}}\mathbf{e}_m{}' \end{bmatrix}$, we have that $\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}}\mathbf{e}_1{}' \\ \frac{1}{\sqrt{\lambda_2}}\mathbf{e}_2{}' \\ \vdots \\ \frac{1}{\sqrt{\lambda_m}}\mathbf{e}_m{}' \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \mathbf{Q}\mathbf{x}$.
Therefore, as I mentioned earlier, we know that our variables $z_1, z_2, \ldots, z_m$ are based on a distribution with covariance $\mathbf{Q}\mathbf{\Sigma}\mathbf{Q}'$. However,

$$\begin{aligned} \mathbf{Q}\mathbf{\Sigma}\mathbf{Q}' &= \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}}\mathbf{e}_1{}' \\ \vdots \\ \frac{1}{\sqrt{\lambda_m}}\mathbf{e}_m{}' \end{bmatrix} \mathbf{U}\mathbf{D}\mathbf{U}' \left[\frac{1}{\sqrt{\lambda_1}}\mathbf{e}_1, \ldots, \frac{1}{\sqrt{\lambda_m}}\mathbf{e}_m\right] \\ &= \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}}\mathbf{e}_1{}' \\ \vdots \\ \frac{1}{\sqrt{\lambda_m}}\mathbf{e}_m{}' \end{bmatrix} [\mathbf{e}_1, \ldots, \mathbf{e}_m] \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_m \end{bmatrix} \begin{bmatrix} \mathbf{e}_1{}' \\ \ldots \\ \mathbf{e}_m{}' \end{bmatrix} \left[\frac{1}{\sqrt{\lambda_1}}\mathbf{e}_1, \ldots, \frac{1}{\sqrt{\lambda_m}}\mathbf{e}_m\right] \\ &= \mathbf{I}. \end{aligned} \tag{3.46}$$

And that is what we needed to show: the resulting variables $z_1, z_2, \ldots, z_m$ are independent. Therefore, the matrix $\mathbf{\Sigma}^{-1}$ is what we need for the statistical distance: it transforms our variables into a new space where they are not correlated anymore.

As a final important remark, note that we could have used the matrix $\mathbf{\Sigma}$ as well to remove the correlation between the variables, since both $\mathbf{\Sigma}$ and $\mathbf{\Sigma}^{-1}$ have the same eigenvectors and hence can provide the same directions of projection. However, by using the inverse of the covariance matrix we standardize each variable (i.e. we divide by its variance), while if we used the covariance matrix we would be amplifying their variance.

Looking again at Figure 3.3, we can now say that the eigenvectors $\mathbf{e}_1$ and $\mathbf{e}_2$ of $\mathbf{\Sigma}$ and $\mathbf{\Sigma}^{-1}$ give precisely the directions of the variables Y_1 and Y_2. Also, from the description at the end of Section 3.4.1, we find that, when $\mathbf{A} = \mathbf{\Sigma}^{-1}$, the length δ_1 of the principal

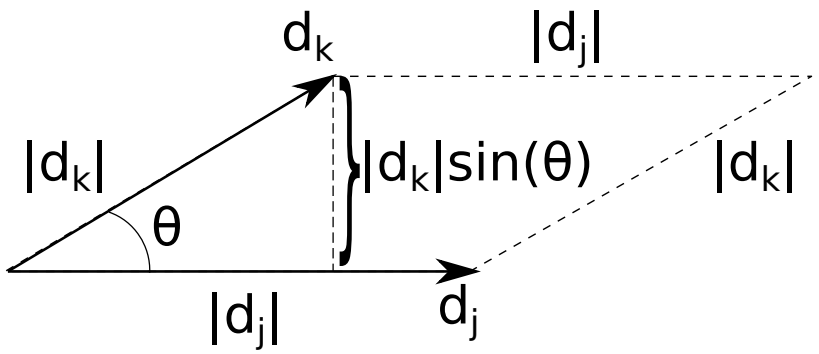

Figure 3.4: Area formed by two deviation vectors $\mathbf{d}_j$ and $\mathbf{d}_k$.

ellipse semi-axis is given by $c\sqrt{\lambda_1}$ and the length δ_2 of the secondary semi-axis is given by $c\sqrt{\lambda_2}$, where $\lambda_1 > \lambda_2$ are the eigenvalues of $\mathbf{\Sigma}$. The use of the inverse $\mathbf{\Sigma}^{-1}$ produces the difference to the values shown in Section 3.4.1.

In conclusion, the use of the inverse of the covariance matrix, $\mathbf{\Sigma}^{-1}$, in the computation of the statistical distance, allows us to eliminate the correlation between the random variables, by rotating the data into a new space where the variables are uncorrelated, and to standardize each variable according to its variance. The new axes are given by the eigenvectors of the matrix $\mathbf{\Sigma}$, while the lengths of these axes are given by the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$ of this matrix. Note that in practice we will use the sample estimate $\mathbf{S}$ of the covariance matrix $\mathbf{\Sigma}$.

3.5 Generalized variance

In Section 3.2, I showed how to represent our leakage traces through the leakage matrix $\mathbf{X}$ and how to visualize these traces in a scatter plot, and in Section 3.4, I presented the importance of the covariance matrix for the statistical distance. Now, I shall extend these concepts and present the relation between the volume generated by the leakage vectors and the covariance matrix.

Looking back at the leakage matrix from (3.1), we can represent each of its columns as a vector $\mathbf{y}_j{}' = [x_{1j}, x_{2j}, \ldots, x_{Nj}]$. If we now subtract the mean $\bar{x}_j$ from each element we obtain the *deviation* vector $\mathbf{d}_j{}' = [d_{1j}, d_{2j}, \ldots, d_{Nj}] = [x_{1j} - \bar{x}_j, x_{2j} - \bar{x}_j, \ldots, x_{Nj} - \bar{x}_j]$. Therefore, we can see our leakage matrix as a set of m vectors in an N-dimensional space.

However, even if each deviation vector is N-dimensional, we can still plot any two such deviation vectors $\mathbf{d}_j$ and $\mathbf{d}_k$ in a two-dimensional space, as shown in Figure 3.4.

Given the definition of the deviation vectors, we have that

$$\mathbf{d}_j{}'\mathbf{d}_k = \sum_{i=1}^{N}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) = (N-1)s_{jk}, \tag{3.47}$$

and from the definition of a scalar product we also know that

$$\mathbf{d}_j{}'\mathbf{d}_k = |\mathbf{d}_j||\mathbf{d}_k|\cos(\theta) = \sqrt{\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)^2}\sqrt{\sum_{i=1}^{N}(x_{ik} - \bar{x}_k)^2}\cos(\theta). \tag{3.48}$$

Then we can derive

$$\cos(\theta) = \frac{s_{jk}}{\sqrt{s_{jj}}\sqrt{s_{kk}}} = r_{jk}. \tag{3.49}$$

This shows that the angle between two deviation vectors $\mathbf{d}_j$ and $\mathbf{d}_k$ corresponds to the sample correlation coefficient between the columns $\mathbf{y}_j$ and $\mathbf{y}_k$ of the leakage matrix $\mathbf{X}$. It is also an estimate for the correlation between the random variables X_j and X_k.

We can also compute the area A spanned by the deviation vectors, shown by the dashed parallelepiped in Figure 3.4, as

$$A = |\mathbf{d}_j||\mathbf{d}_k|\sin(\theta) = (N-1)\sqrt{s_{jj}s_{kk}(1 - r_{jk}^2)} \tag{3.50}$$

Also, we can compute the determinant $|\mathbf{S}|$ of the sample covariance matrix of the two vectors $\mathbf{d}_j$ and $\mathbf{d}_k$, and obtain

$$|\mathbf{S}| = \left|\begin{bmatrix} s_{jj} & s_{jk} \\ s_{jk} & s_{kk} \end{bmatrix}\right| = s_{jj}s_{kk}(1 - r_{jk}^2). \tag{3.51}$$

Comparing (3.50) with (3.51) we get

$$|\mathbf{S}| = \frac{A^2}{(N-1)^2}. \tag{3.52}$$

This can be generalized for the volume V spanned by m deviation vectors, obtaining

$$|\mathbf{S}| = \frac{V^2}{(N-1)^m}, \tag{3.53}$$

where $\mathbf{S} \in \mathbb{R}^{m\times m}$. This shows that the determinant $|\mathbf{S}|$ of the sample covariance matrix, also known as the *generalized sample variance*, is proportional to the square of the volume spanned by the deviation vectors.

These formulas provide the interesting intuition that a smaller correlation may correspond to a larger generalized variance $|\mathbf{S}|$. This can be explained from (3.50) and (3.51), where we see that the generalized variance is proportional to $\sin(\theta)$. Therefore, if *some* deviation vectors are following a similar direction (hence $\sin(\theta)$ will be small and the vectors will be highly correlated) we expect the generalized variance to be small (in the extreme case, if at least two deviation vectors are on the same direction we expect $|\mathbf{S}| = 0$). Conversely, if

the deviation vectors are close to being perpendicular ($\sin(\theta)$ large), then they will span a large volume, resulting in a large value of $|\mathbf{S}|$.

However, the generalized variance $|\mathbf{S}|$ depends also on the lengths of the individual deviation vectors (i.e. the individual variances s_{jj}). Therefore, the generalized variance $|\mathbf{S}|$ is not very well suited to compare the overall correlation between two samples (in my experiments two leakage matrices). A good option for this purpose is to first standardize all the observations from the leakage matrix, i.e. to replace each sample x_{ij} of the leakage matrix $\mathbf{X}$ from (3.1) by $(x_{ij} - \bar{x}_j)/s_{jj}$. This results in deviation vectors having *equal* length, $|\mathbf{d}_1| = |\mathbf{d}_2| = \ldots = |\mathbf{d}_m| = \sqrt{N-1}$.

From Section 3.3.5, we know that the covariance of the standardized variables is the sample correlation matrix $\mathbf{R}$. Then, we can say that that the generalized sample variance $|\mathbf{R}|$ obtained from the standardized variables is proportional to the squared volume of the deviation vectors having *equal* length, since we can replace $\mathbf{S}$ by $\mathbf{R}$ in (3.53) and obtain

$$|\mathbf{R}| = \frac{V^2}{(N-1)^m}. \tag{3.54}$$

As a result, we can use $|\mathbf{R}|$ to compare the overall correlation between different samples.

3.6 Multivariate normal distribution

The *multivariate normal distribution*, which is central to the template attacks from the following chapters, is a useful approximation of the *true* population in many natural phenomena, including side-channel leakage traces. Furthermore, it is mathematically tractable and many nice results can be derived, as I will show throughout this section.

Let us start by looking at the *univariate* normal distribution $\mathcal{N}(\mu, \sigma)$ of a *single* random variable X, with mean μ and variance σ, which can be written as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sqrt{\sigma}}\right)^2\right), \; -\infty < x < \infty. \tag{3.55}$$

This distribution is also known as a *Gaussian* distribution, after Carl Friedrich Gauss, who is believed to be the first to have used this distribution, based on a publication from 1809 [43]. However, it was probably James Clerk Maxwell, in a publication from 1860 [76], who first found the normal distribution to explain physical phenomena. In particular, Maxwell found that this distribution can approximate the number of moving particles of a gas, having some velocity in a given direction.

As explained in Section 3.3, when dealing with a sample of observations (which is usually the case in practice), we replace the real (population) parameters μ and σ by their sample counterparts. Using the data shown in Figure 3.1, we can derive the univariate distribution

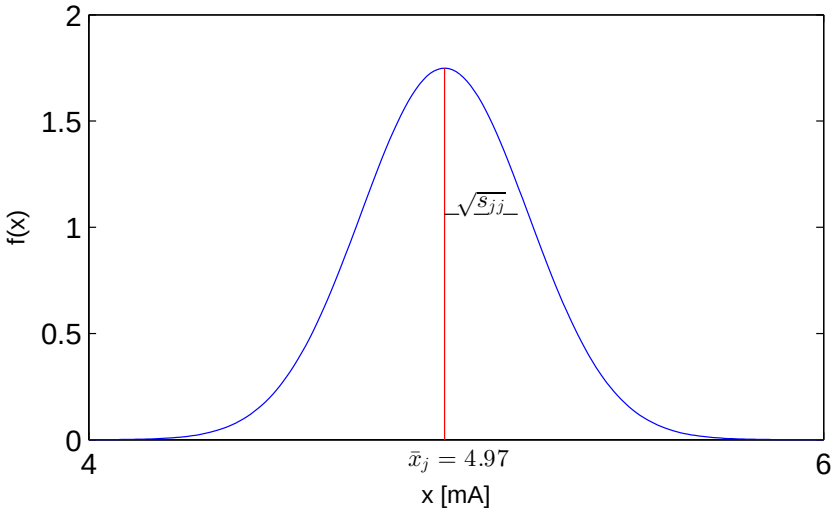

Figure 3.5: Univariate distribution of leakage samples at $j = 878$ for traces from the *Grizzly Alpha* dataset. $\bar{x}_j = 4.97$, $\sqrt{s_{jj}} = 0.23$.

of the leakage samples at $j = 878$, with sample mean $\bar{x}_j = 4.97$ and sample standard deviation $\sqrt{s_{jj}} = 0.23$. This is shown in Figure 3.5.

The univariate normal distribution formula (3.55) is composed of two main terms. The term $\left(\frac{x-\mu}{\sqrt{\sigma}}\right)^2$ is simply the squared Euclidean distance from x to μ, normalized by the standard deviation $\sqrt{\sigma}$, while the term $\frac{1}{\sqrt{2\pi\sigma}}$ is a normalizing constant used to make the area under the distribution (i.e. the probability) equal to 1.

We can extend these terms to the multivariate case, where we deal with several random variables $X_1, X_2, \ldots, X_m$, as follows. First, we replace the Euclidean distance by the statistical distance $(\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$, where $\mathbf{x} \in \mathbb{R}^m$ may be a leakage vector observed from the distribution of the m random variables, $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the covariance of this distribution – see also Section 3.4. Then, we should normalize the resulting exponent by the volume $(|\boldsymbol{\Sigma}|^{-1/2})$ derived from the span of the covariance matrix (see Section 3.5), and the normalizing constant $(2\pi)^{-m/2}$. As a result, we obtain the form of the *multivariate* normal distribution $\mathcal{N}_m(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ of a random vector $\mathbf{X}' = [X_1, X_2, \ldots, X_m]$, with mean $\boldsymbol{\mu} \in \mathbb{R}^m$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times m}$, as

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{m/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right). \tag{3.56}$$

As I mentioned in Section 3.3.4, if the random variables $X_1, X_2, \ldots, X_m$ are independent ($\mathbf{Cov}(X_j, X_k) = 0$), then the multivariate distribution is equal to the product of the univariate distributions, that is

$$f(\mathbf{x}) = f(x_1)f(x_2)\ldots f(x_m). \tag{3.57}$$

Also, note that for $m = 1$ the formula in (3.56) reverts to the univariate form in (3.55).

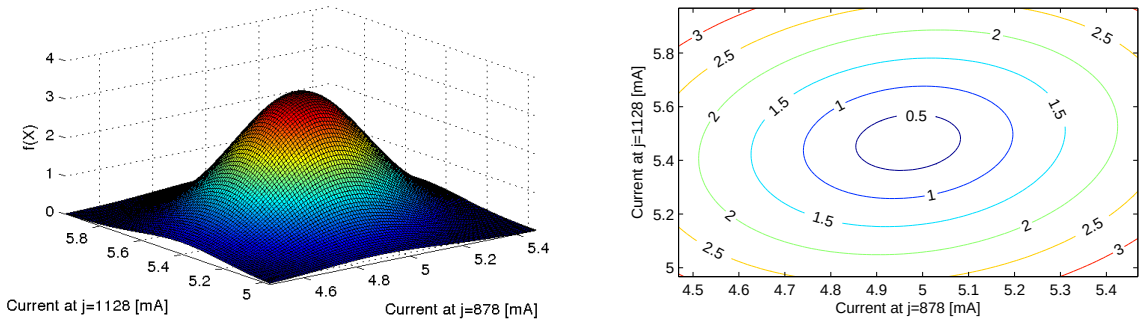

Figure 3.6: Left: multivariate normal distribution of leakage samples at $j = 878$ and $j = 1128$ for $N = 1000$ traces from the *Grizzly Alpha* dataset. Right: contours of points at a constant *statistical* distance.

As I mentioned throughout this chapter, in practice we shall often use the sample mean vector $\bar{\mathbf{x}}$ and the sample covariance matrix $\mathbf{S}$ instead of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Using again the data shown in Figure 3.1, we have $\bar{\mathbf{x}} = [4.97, 5.47]$ and $\mathbf{S} = \begin{bmatrix} 0.052 & 0.007 \\ 0.007 & 0.044 \end{bmatrix}$. The multivariate normal distribution based on this data, $\mathcal{N}_2(\bar{\mathbf{x}}, \mathbf{S})$, is shown in Figure 3.6. The ellipses formed by similar colors on the right hand side of this figure represent the *contours of equal probability*. The points in a given contour are at the same statistical distance $d = \sqrt{(\mathbf{x} - \bar{\mathbf{x}})' \mathbf{S}^{-1} (\mathbf{x} - \bar{\mathbf{x}})}$ from the sample mean $\bar{\mathbf{x}}$.

In Appendix C, I show that the leakage traces from the *Grizzly* dataset follow well the multivariate normal distribution. Hence, this gives a good motivation for using the techniques from this chapter with template attacks on these traces.

3.6.1 Some properties of the multivariate normal distribution

I now present some important facts regarding the multivariate normal distribution. These will be used throughout the following chapters.

Distribution of linear combinations

If the random vector $\mathbf{X}' = [X_1, X_2, \ldots, X_m\ ]$ is distributed as $\mathcal{N}_m(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then any linear combination $\mathbf{a}'\mathbf{X} = a_1 X_1 + a_2 X_2 + \ldots + a_m X_m$ is distributed as $\mathcal{N}(\mathbf{a}'\boldsymbol{\mu}, \mathbf{a}'\boldsymbol{\Sigma}\mathbf{a})$. This property will be essential in the discussion of principal components (see Section 3.9).

Distribution of statistical distance

If the random vector $\mathbf{X}' = [X_1, X_2, \ldots, X_m\ ]$ is distributed as $\mathcal{N}_m(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then the squared statistical distance $d^2 = (\mathbf{x} - \boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$ is distributed as χ^2_m (the *chi-square* distri-

bution with m degrees of freedom), which has the pdf

$$\chi^2_m(d^2) = \frac{d^{(m-\frac{1}{2})} \exp\left(-d^2/2\right)}{2^{m/2}\Gamma(m/2)}. \tag{3.58}$$

For positive integers, the function Γ has the form

$$\Gamma(n+1) = n!, \tag{3.59}$$

for half-integers it has the form

$$\Gamma(n+1/2) = \frac{(2n)!}{4^n n!}\sqrt{\pi}, \tag{3.60}$$

while for arbitrary rational numbers it has no known closed form. The name of the *Chi-square* distribution comes from the Greek letter χ (*chi*), which was used by Karl Pearson in 1900 [88] to refer to the squared statistical distance, $\chi = (\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})$.

Sample distribution of $\bar{\mathbf{x}}$ and $\mathbf{S}$

Let $\mathbf{X}$ be a statistical sample with N observations following a distribution $\mathcal{N}_m(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, such as the leakage matrix from (3.1), having sample mean $\bar{\mathbf{x}}$ and sample covariance $\mathbf{S}$. Then the following hold:

1. $\bar{\mathbf{x}}$ is distributed as $\mathcal{N}_m(\boldsymbol{\mu}, (1/N)\boldsymbol{\Sigma})$.
2. $(N-1)\mathbf{S}$ follows a Wishart distribution.
3. $\bar{\mathbf{x}}$ and $\mathbf{S}$ are independent.

The first statement is very important as it shows that: (a) there is also some variance in the sample mean $\bar{\mathbf{x}}$, around the true mean $\boldsymbol{\mu}$, and (b) the variance of the sample mean is N times smaller than the variance of the data.

The third statement is also important as it allows us to use $\mathbf{S}$ as an estimate of $\boldsymbol{\Sigma}$ in the computation of intervals for the mean vectors (see Section 3.7).

Large sample behaviour

The following result is known as the *central limit theorem*. Given a statistical sample $\mathbf{X} \in \mathbb{R}^{N\times m}$ with a large number N of observations following *any distribution* with some mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, its sample mean $\bar{\mathbf{x}}$ is distributed as $\mathcal{N}_m(\boldsymbol{\mu}, (1/N)\boldsymbol{\Sigma})$, or equivalently $\sqrt{N}(\bar{\mathbf{x}} - \boldsymbol{\mu})$ is distributed as $\mathcal{N}_m(0, \boldsymbol{\Sigma})$. This shows that for N large, the sampling distribution of $\bar{\mathbf{x}}$ is normal, *regardless* of the underlying distribution of the data.

Furthermore, it can be shown that $N(\bar{\mathbf{x}} - \boldsymbol{\mu})'\mathbf{S}^{-1}(\bar{\mathbf{x}} - \boldsymbol{\mu})$ is distributed as $\mathcal{X}^2_m$.

3.7 Confidence regions for mean vectors

Given a random variable X with real mean μ and variance σ, and a set of observations $x_1, x_2, \ldots, x_N$ of this variable, it may be useful to estimate a *confidence region* for the value of the real mean. That is, based on the sample estimates $\bar{x}$ and s, we may wish to determine a probable interval for the real mean μ. This interval is given by

$$\bar{x} - t_{N-1}(\alpha/2)\sqrt{\frac{s}{N}} \leq \mu \leq \bar{x} + t_{N-1}(\alpha/2)\sqrt{\frac{s}{N}}, \tag{3.61}$$

where t_{N-1} is the *t-student* distribution with $N-1$ degrees of freedom and α is a *significance level*. The meaning of this interval and the relation to α is that we expect the real mean to be within the given bounds with probability $(1-\alpha)$. Such interval is also known as a $100(1-\alpha)\%$ confidence interval.

We can extend this concept to multiple variables, as follows. Let $\mathbf{X} \in \mathbb{R}^{N\times m}$ be a statistical sample from a distribution $\mathcal{N}_m(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, such as the leakage matrix from (3.1), having sample mean $\bar{\mathbf{x}}$ and sample covariance $\mathbf{S}$. Then, a $100(1-\alpha)\%$ *confidence region* for the real mean $\boldsymbol{\mu}$ is given by the relation

$$N(\bar{\mathbf{x}} - \boldsymbol{\mu})'\mathbf{S}^{-1}(\bar{\mathbf{x}} - \boldsymbol{\mu}) \leq \frac{m(N-1)}{N-m} F_{m,N-m}(\alpha), \tag{3.62}$$

where $F_{m,N-m}$ represents an F-distribution, with m and $N-m$ degrees of freedom, having the pdf

$$F_{a_1,a_2}(\alpha) = \frac{\sqrt{\frac{(a_1\alpha)^{a_1}a_2^{a_2}}{(a_1\alpha+a_2)^{(a_1+a_2)}}}}{\alpha B(\frac{a_1}{2}, \frac{a_2}{2})}, \quad B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}. \tag{3.63}$$

We can observe that the left hand side of (3.62) defines the sets of points at a particular statistical distance from the sample mean $\bar{\mathbf{x}}$. Therefore, the confidence interval for the real multivariate mean $\boldsymbol{\mu}$ is an ellipsoid centered at $\bar{\mathbf{x}}$, with a boundary given by the points at a squared statistical distance $d^2 = (\bar{\mathbf{x}} - \boldsymbol{\mu})'\mathbf{S}^{-1}(\bar{\mathbf{x}} - \boldsymbol{\mu}) \leq \frac{m(N-1)}{N(N-m)} F_{m,N-m}(\alpha)$.

Furthermore, we can also compute a confidence interval for the observations in our sample $\mathbf{X}$. From Section 3.6.1 we know that $(\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})$ is distributed as $\mathcal{X}^2_m$. As a result, we can compute the $(100-\alpha)\%$ confidence interval for the observations $\mathbf{x}$ in $\mathbf{X}$, using

$$(\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) \leq \mathcal{X}^2_m(\alpha), \tag{3.64}$$

which represents an ellipsoid centered at $\boldsymbol{\mu}$. If we use the sample mean $\bar{\mathbf{x}}$ and sample covariance $\mathbf{S}$, then the $(100-\alpha)\%$ confidence interval for the data is given by

$$(\mathbf{x} - \bar{\mathbf{x}})'\mathbf{S}^{-1}(\mathbf{x} - \bar{\mathbf{x}}) \leq \mathcal{X}^2_m(\alpha), \tag{3.65}$$

which is an ellipsoid centered at $\bar{\mathbf{x}}$.

I shall illustrate these concepts using data from the *Grizzly Alpha* dataset (see Section 2.3.1). Let $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, $\mathbf{X}_4$ and $\mathbf{X}_5$ be 5 different samples (leakage matrices) from

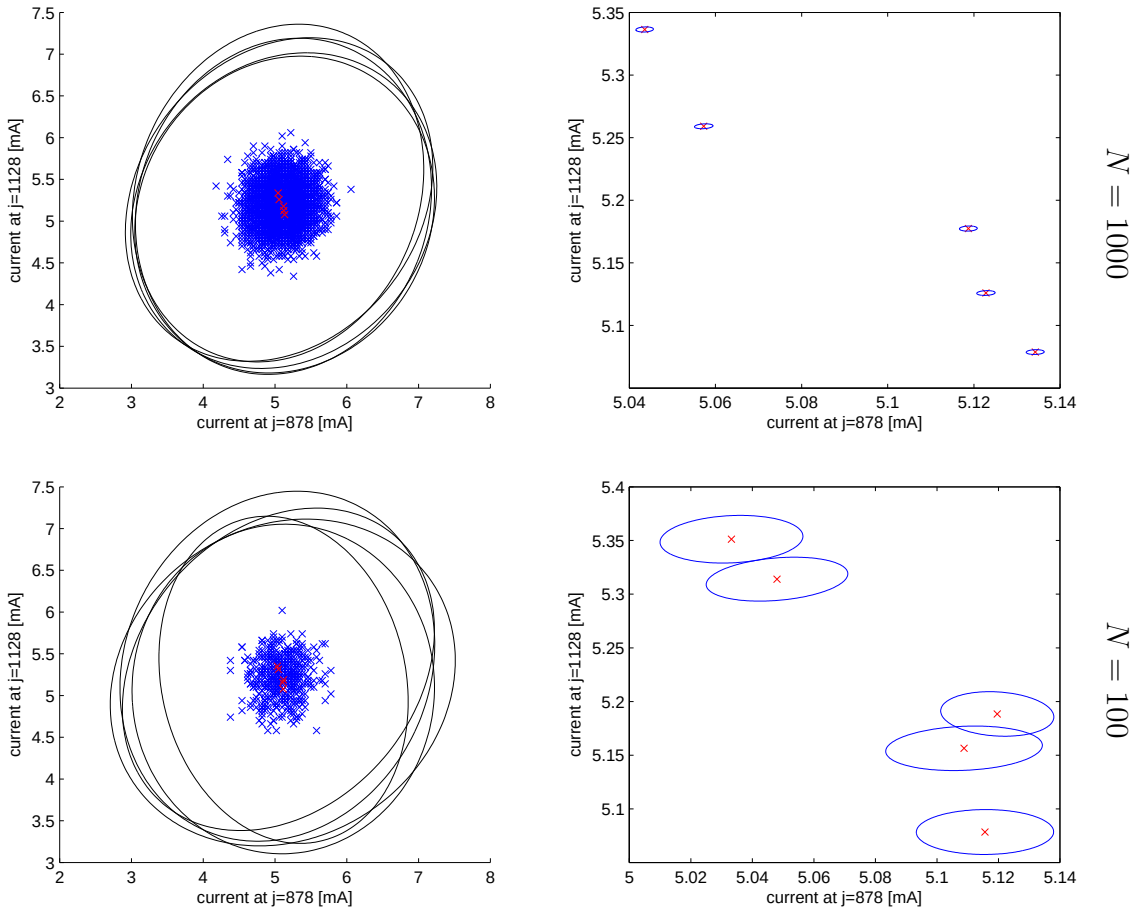

Figure 3.7: Confidence intervals using data from *Grizzly Alpha* dataset for 5 different values of k and $\alpha = 0.01$. Left: intervals for data. Right: intervals for means.

this dataset, corresponding to 5 different register values of the loaded value k. Figure 3.7 shows the confidence intervals for the observations (left), as well as for the real means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_5$ (right), using $\alpha = 0.01$. Note that, for large N (top-right), the confidence intervals of the means are very small and all 5 confidence ellipses are very far apart, while for smaller N (bottom-right) some of the confidence ellipses overlap, meaning that we cannot classify correctly a point from the intersection of the overlapping ellipses. As the following chapters will confirm, the sample size has a strong influence on the effectiveness of template attacks. Generally, the more data we can use the better the attacks.

3.8 Multivariate analysis of variance

Given a set of samples (leakage matrices) $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_G$ for G groups, such as those five shown in Figure 3.7, it can be useful to determine if their mean vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_G$ (or more often their sample estimates) are different. A common technique for this purpose is the (one-way) *analysis of variance* (ANOVA), which I shall describe in the following.

3.8.1 ANOVA

Let us start with the univariate case, where

$$\mathbf{X}_1 = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1N_1} \end{bmatrix}, \mathbf{X}_2 = \begin{bmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2N_2} \end{bmatrix}, \ldots, \mathbf{X}_G = \begin{bmatrix} x_{G1} \\ x_{G2} \\ \vdots \\ x_{GN_G} \end{bmatrix}, \tag{3.66}$$

are *independent* leakage samples corresponding to different values k. We can represent each sample mean $\bar{x}_k$ (where $k \in \{1, \ldots, G\}$) as

$$\bar{x}_k = \bar{x} + \tau_k, \tag{3.67}$$

where $\bar{x} = \dfrac{\sum_{k=1}^{G}\sum_{i=1}^{N_k} x_{ki}}{\sum_{k=1}^{G} N_k}$ is the overall sample mean across all the G samples and $\tau_k = \bar{x}_k - \bar{x}$ is the *treatment* or *effect* of the particular group k. Then, we can represent each observation in our samples as

$$x_{ki} = \bar{x} + (\bar{x}_k - \bar{x}) + (x_{ki} - \bar{x}_k), \tag{3.68}$$

where $e_{ki} = x_{ki} - \bar{x}_k$ is the *residual* associated with x_{ki}, following the distribution $\mathcal{N}(0, \sigma)$. Moving $\bar{x}$ to the left in (3.68) and squaring we obtain

$$(x_{ki} - \bar{x})^2 = (\bar{x}_k - \bar{x})^2 + (x_{ki} - \bar{x}_k)^2 + 2(\bar{x}_k - \bar{x})(x_{ki} - \bar{x}_k), \tag{3.69}$$

and by adding over all observations and groups, and eliminating zero terms, we obtain the *sums of squares* (SS) form

$$\begin{array}{ccccccc} \sum_{k=1}^{G}\sum_{i=1}^{N_k} x_{ki}^2 & = & (\sum_{k=1}^{G} N_k)\bar{x}^2 & + & \sum_{k=1}^{G} N_k(\bar{x}_k - \bar{x})^2 & + & \sum_{k=1}^{G}\sum_{i=1}^{N_k}(x_{ki} - \bar{x}_k)^2 \\ (\mathrm{SS_{obs}}) & = & (\mathrm{SS_{mean}}) & + & (\mathrm{SS_{tr}}) & + & (\mathrm{SS_{res}}), \end{array} \tag{3.70}$$

where $\mathrm{SS_{tr}}$ is known as the *treatment* or *between* (samples) sum of squares and $\mathrm{SS_{res}}$ is known as the *residual* or *within* (samples) sum of squares. In addition, we can define the *total* or *corrected* sum of squares

$$(\mathrm{SS_{cor}}) = (\mathrm{SS_{tr}}) + (\mathrm{SS_{res}}). \tag{3.71}$$

Finally, we can use the F-test, that rejects the *null hypothesis* $H_0 : \tau_1 = \tau_2 = \ldots = \tau_G = 0$ at a *significance level* α if

$$\mathrm{F} = \frac{\mathrm{SS_{tr}}/(G-1)}{\mathrm{SS_{res}}/\big(\sum_{k=1}^{G} N_k - G\big)} > F_{G-1, \sum_k N_k - G}(\alpha), \tag{3.72}$$

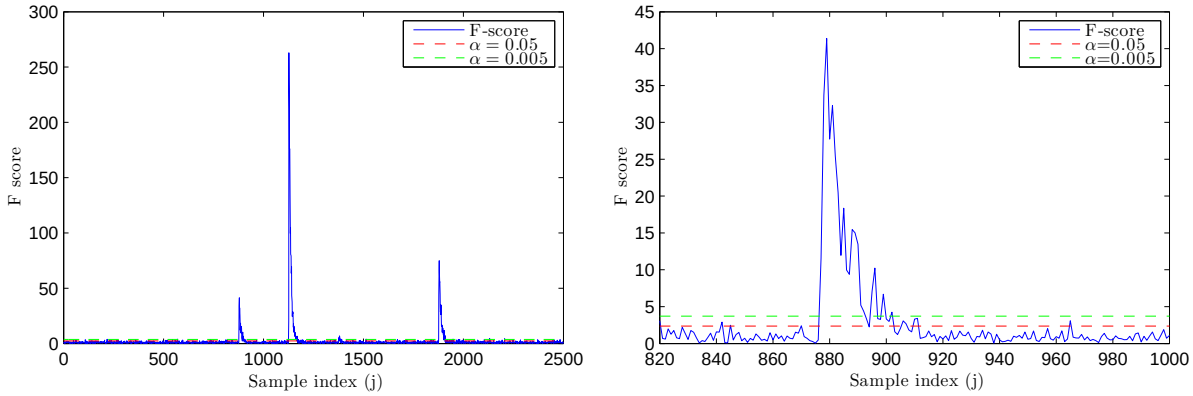

Figure 3.8: F score for the $m = 2500$ variables across $G = 5$ samples (leakage matrices) in the *Grizzly Alpha* dataset, along with threshold lines for $\alpha = 0.05$ (red) and $\alpha = 0.005$ (green). Left: all leakage samples; right: leakage samples around first peak.

where $F_{G-1,\sum_k N_k - G}(\alpha)$ is the upper (100α)th percentile of the F-distribution with $G-1$ and $\sum_k N_k - G$ degrees of freedom. Using the null hypothesis test, we have a probability α of incorrectly rejecting the null hypothesis of equal mean vectors (H_0), when the F value is above the threshold. Therefore, choosing a small α should minimise the chance of incorrectly deciding that there is some difference between the mean vectors.

As I show in Chapter 4, many techniques for finding the best leakage samples to use in a side-channel attack rely on some method that is similar to the F-test. While the null hypothesis testing method may be controversial (e.g. it is not clear how to choose α or the exact meaning of such a value), for side-channel attacks we are not concerned with a particular value of α, but rather, we should compute the F-test for all the leakage samples and select for the attack those providing the highest values. Note that, in general, the values G and N_k are equal across all variables, so to decide which leakage samples to use we merely need to compute the signal-to-noise ratio $\mathrm{SS}_{\mathrm{tr}}/\mathrm{SS}_{\mathrm{res}}$. Here, as throughout the following chapters, the signal is determined by the *treatment* sum of squares $\mathrm{SS}_{\mathrm{tr}}$, which is computed from the deviations of the mean values to the overall mean. This provides the desired signal, since in my experiments I did not use masking (processing random values) or other countermeasures, as explained in Section 2.1.4.

To illustrate the use of the F-test, let's use again the five samples from Section 3.7. I computed the F score from (3.72), for each of the $m = 2500$ leakage samples, and plotted the result in Figure 3.8, along with the rejection threshold for $\alpha = 0.05$ (red) and $\alpha = 0.005$ (green). We can then select only those samples for which the F score is above a desired threshold. The leakage samples with the highest score are around $j = 880$, $j = 1127$ and $j = 1880$, so the mean values $\bar{x}_j$ of the five groups are most different around these instants. We also see that, for $\alpha = 0.05$, some incorrect samples would be selected (those outside the region where we expect the dynamic power consumption to show up), while for $\alpha = 0.005$, the threshold allows us to select samples only around the region of interest.

3.8.2 MANOVA

The technique presented in the previous section can be modified for the multivariate case, in order to take into consideration also the correlation between variables. In this case the technique is called the *multivariate analysis of variance* (MANOVA).

Let's use again a set of independent samples (leakage matrices) $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_G$, but in this case each observation is m-dimensional, i.e. $\mathbf{X}_k \in \mathbb{R}^{N \times m}$. Similarly to the univariate case, we can describe each m-dimensional observation as

$$\mathbf{x}_{ki} = \bar{\mathbf{x}} + (\bar{\mathbf{x}}_k - \bar{\mathbf{x}}) + (\mathbf{x}_{ki} - \bar{\mathbf{x}}_k), \tag{3.73}$$

where $\bar{\mathbf{x}} = \dfrac{\sum_{k=1}^{G}\sum_{i=1}^{N_k}\mathbf{x}_{ki}}{\sum_{k=1}^{G} N_k}$ is the overall mean vector and

$$\boldsymbol{\tau}_k = (\bar{\mathbf{x}}_k - \bar{\mathbf{x}}) \tag{3.74}$$

is the *treatment* or *effect* vector of each group.

We can also define the multivariate versions of the sums of squares, known as the *sums of squares and cross-products* (SSP), as

$$\mathbf{B} = \sum_{k=1}^{G} N_k(\bar{\mathbf{x}}_k - \bar{\mathbf{x}})(\bar{\mathbf{x}}_k - \bar{\mathbf{x}})' \tag{3.75}$$

$$\mathbf{W} = \sum_{k=1}^{G}\sum_{i=1}^{N_k}(\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)(\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)' \tag{3.76}$$

$$\mathbf{B} + \mathbf{W} = \sum_{k=1}^{G}\sum_{i=1}^{N_k}(\mathbf{x}_{ki} - \bar{\mathbf{x}})(\mathbf{x}_{ki} - \bar{\mathbf{x}})', \tag{3.77}$$

where $\mathbf{B}$ is the *treatment* or *between* SSP matrix, $\mathbf{W}$ is the *residual* or *within* SSP matrix, and $\mathbf{B} + \mathbf{W}$ is the *total* or *corrected* SSP matrix. Note that all these matrices can be efficiently computed as shown in (3.17, 3.18) for the sample covariance matrix.

Similarly to the univariate case, $\mathbf{B}$ contains the signal of interest. Then, MANOVA can be used to test if we can reject the null hypothesis $H_0 : \boldsymbol{\tau}_1 = \boldsymbol{\tau}_2 = \ldots = \boldsymbol{\tau}_G = \mathbf{0}$. For this purpose, we can compute the quantity

$$\Lambda^\star = \frac{|\mathbf{W}|}{|\mathbf{B} + \mathbf{W}|}, \tag{3.78}$$

known as *Wilks' Lambda* (named after its proposer), and reject H_0 if $\Lambda^\star$ is too small (there exist tables for checking the expected distribution of $\Lambda^\star$).

I am not aware of this measure having been used in side-channel analysis, although it may be useful in order to compare the leakage of different implementations.

While I have not used $\Lambda^\star$ in my experiments either, I will use the SSP matrices (in particular $\mathbf{W}$ and $\mathbf{B}$) in the following chapters.

3.8.3 Pooled covariance matrix

Note in (3.76) that the *within* SSP matrix $\mathbf{W}$ is in fact a scaled average of the individual sample covariance matrices, that is

$$\mathbf{W} = (N_1 - 1)\mathbf{S}_1 + (N_2 - 1)\mathbf{S}_2 + \ldots + (N_G - 1)\mathbf{S}_G. \tag{3.79}$$

If the underlying real covariances $\mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \ldots, \mathbf{\Sigma}_G$ are the same, we expect their sample counterparts $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_G$ to be similar. In this case, we can define the *pooled covariance* as

$$\mathbf{S}_{\text{pooled}} = \frac{\sum_{k=1}^{G}(N_k - 1)\mathbf{S}_k}{\sum_{k=1}^{G}(N_k - 1)} = \frac{\mathbf{W}}{\sum_{k=1}^{G} N_k - G}. \tag{3.80}$$

The pooled covariance $\mathbf{S}_{\text{pooled}}$ can provide a much better estimate of the real covariance $\mathbf{\Sigma}$ governing all samples. Its use will be very important in the subsequent chapters, as it significantly improves the performance of template attacks.

3.9 Principal Component Analysis

Principal Component Analysis (PCA) is a technique used mainly for two purposes: (a) dimensionality reduction; (b) interpretation of data. In the context of side-channel attacks, in particular for template attacks, we are mostly interested in the first aspect (dimensionality reduction), but generally the application of PCA will also reveal which leakage samples are more important for the attack. Furthermore, when using PCA for factor analysis (see Chapter 6), this technique can provide intuitions about the underlying sources of correlation.

Let $\mathbf{X}' = [X_1, X_2, \ldots, X_m]$ be a vector of random variables representing some data of interest, such as the leakage matrix in (3.1). Then, the *principal components* used by PCA are those linear combinations $Y_j = \mathbf{a}'\mathbf{X}$, with $|\mathbf{a}| = 1$, that maximise the variance of the resulting variables Y_j. PCA aims to find the best $K < m$ linear combinations of the original m variables that maximise the variance of the original data. This allows PCA to capture most of the information from our original data with a smaller (generally much smaller) number of variables, by projecting the data into a space where the variables are uncorrelated.

I already presented some important details about PCA in Section 3.4. If we look back at Figure 3.3, we can recall that for some given data having covariance $\mathbf{\Sigma}$, the directions providing the maximum variance of the data were given by the eigenvectors $\mathbf{e}_j$ of $\mathbf{\Sigma}$.

Therefore, if our random variables have covariance $\mathbf{\Sigma}$, the principal component providing the maximum variance is $Y_1 = \mathbf{e}_1{}'\mathbf{X}$, the component providing the second largest variance is $Y_2 = \mathbf{e}_2{}'\mathbf{X}$, with $\mathbf{e}_2{}'\mathbf{e}_1 = 0$, and so on. All the eigenvectors are perpendicular to each other, i.e. $\mathbf{e}_j{}'\mathbf{e}_k = 0, \forall j \neq k$, and should be normalized, i.e. $|\mathbf{e}_j| = 1, \forall j$. The eigenvectors are selected such that their corresponding eigenvalues are in the order $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$.

As I mentioned in Section 3.4.1, we can easily obtain the eigenvectors and eigenvalues of a symmetric matrix, such as $\mathbf{\Sigma}$ or $\mathbf{S}$, using the singular value decomposition (SVD)

$$\mathbf{\Sigma} = \mathbf{U}\mathbf{D}\mathbf{U}', \tag{3.81}$$

where the orthonormal matrix $\mathbf{U}$ contains the eigenvectors on its columns and the diagonal matrix $\mathbf{D}$ contains the corresponding eigenvalues on its diagonal elements.

Two important properties of the principal components, that can be easily derived from (3.46), are that

$$\mathbf{Var}(Y_j) = \mathbf{e}_j{}'\mathbf{\Sigma}\mathbf{e}_j = \lambda_j \tag{3.82}$$

$$\mathbf{Cov}(Y_j, Y_k) = \mathbf{e}_j{}'\mathbf{\Sigma}\mathbf{e}_k = 0. \quad (j \neq k) \tag{3.83}$$

Another important property is that

$$\sum_{j=1}^{m} \mathbf{Var}(X_j) = \sum_{j=1}^{m} \mathbf{Var}(Y_j) = \sum_{j=1}^{m} \lambda_j, \tag{3.84}$$

which shows that the total variance across all the random variables in the original data does not change with the PCA transform, and can be computed from the eigenvalues of the covariance matrix.

3.9.1 Choosing the number of principal components

Based on (3.84), we can define the contribution of a principal component Y_j to the total variance as

$$(\text{contribution of } Y_j) = \frac{\lambda_j}{\sum_{j=1}^{m} \lambda_j}. \tag{3.85}$$

Furthermore, we can compute the contribution of the first K principal components to the total variance as

$$\phi(K) = \frac{\sum_{j=1}^{K} \lambda_j}{\sum_{j=1}^{m} \lambda_j}. \tag{3.86}$$

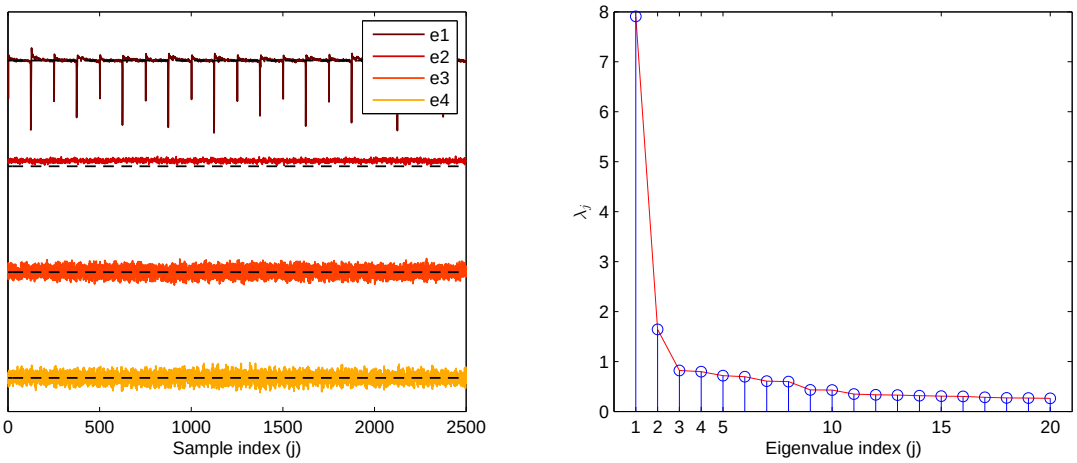

Figure 3.9: PCA eigenvectors (left) and eigenvalues (right) from leakage matrix $\mathbf{X}_1$ ($k = 1$) of *Grizzly Alpha* dataset.

This leads to a rule, known as the *cumulative percentage of total variation* [56], by which we should select the smallest K for which $\phi(K)$ is greater than some chosen threshold (e.g. 0.9).

Another option is to use the *elbow rule* [56], which advises to plot the eigenvalues and then select K as the point after which there is no considerable difference between consecutive eigenvalues.

3.9.2 PCA on the Grizzly dataset

As an example for using PCA, let's use again the leakage matrix $\mathbf{X}_1$, corresponding to $k = 1$, from the *Grizzly Alpha* dataset. Figure 3.9 shows the first four eigenvectors (left) out of the total of $m = 2500$ eigenvectors, and first 20 eigenvalues (right) of the sample covariance matrix $\mathbf{S} = \mathbf{Cov}(\mathbf{X}_1)$. Looking at the eigenvectors, it seems that only the first two either contain large peaks or are constantly different than zero. Looking now at the eigenvalues, we can see an elbow at $j = 3$, which confirms that most of the information can be extracted with only $K = 2$ PCA components.

Visualising the directions of the eigenvectors in the space spanned by the covariance matrix $\mathbf{X}_1$ is not possible in this case ($m = 2500$). However, just for exemplification, we can focus only on the leakage samples at times $j = 878$ and $j = 1128$, as I did for the example in Figure 3.7, and compute the eigenvectors $\mathbf{e}_1$ and $\mathbf{e}_2$ of the covariance matrix $\mathbf{S} = \begin{bmatrix} s_{878,878} & s_{878,1128} \\ s_{1128,878} & s_{1128,1128} \end{bmatrix}$. The directions of these eigenvectors are shown in Figure 3.10 (left), along with the ellipse formed by points at a constant statistical distance from the sample mean vector $\bar{\mathbf{x}} = [\bar{x}_{878}, \bar{x}_{1128}]$. Note that this ellipse is one of those shown already in the left side of Figure 3.7.

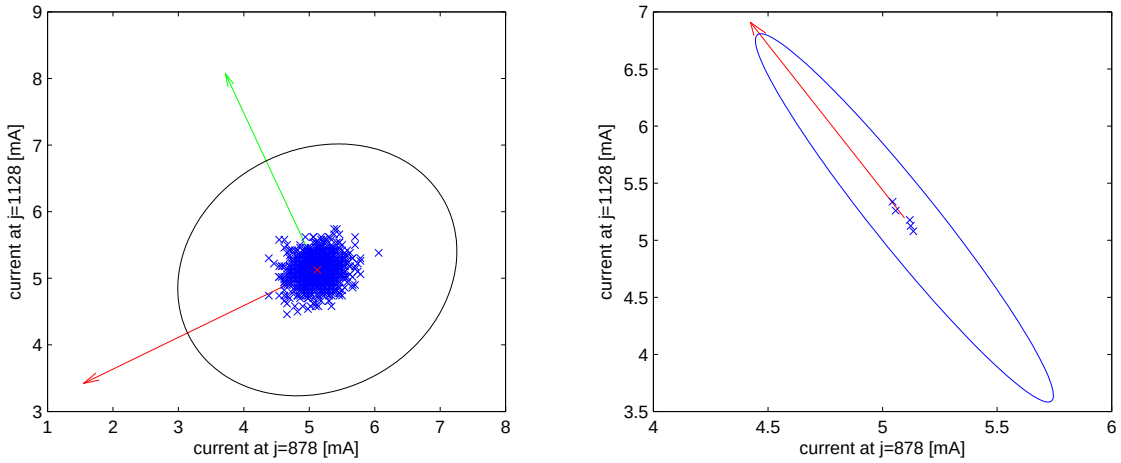

Figure 3.10: Directions of PCA eigenvectors from sample covariance matrix $\mathbf{S}_1$ (left) and from the treatment SSP matrix $\mathbf{B}$ (right), using leakage matrices $\mathbf{X}_1, \ldots, \mathbf{X}_5$ of the *Grizzly Alpha* dataset.

3.9.3 PCA on the treatment vectors

Another good application of PCA, which will be most useful in the template attacks described throughout the following chapters, can be used when dealing with samples having different mean vectors. Let's use again the leakage matrices $\mathbf{X}_1, \ldots, \mathbf{X}_5$ of the *Grizzly Alpha* dataset. The idea then, is to use the matrix of treatment vectors

$$\mathbf{T} = \begin{bmatrix} \boldsymbol{\tau}_1{}' \\ \boldsymbol{\tau}_2{}' \\ \boldsymbol{\tau}_3{}' \\ \boldsymbol{\tau}_4{}' \\ \boldsymbol{\tau}_5{}' \end{bmatrix} = \begin{bmatrix} (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}})' \\ (\bar{\mathbf{x}}_2 - \bar{\mathbf{x}})' \\ (\bar{\mathbf{x}}_3 - \bar{\mathbf{x}})' \\ (\bar{\mathbf{x}}_4 - \bar{\mathbf{x}})' \\ (\bar{\mathbf{x}}_5 - \bar{\mathbf{x}})' \end{bmatrix} \tag{3.87}$$

in order to find the principal components that maximise the variance between the mean vectors. We simply apply PCA to the covariance matrix $\mathbf{T}'\mathbf{T}$, which, except for a constant factor, is precisely the treatment matrix $\mathbf{B}$ from (3.75). Focusing again only on the samples at $j = 878$ and $j = 1128$, I show in Figure 3.10 (right) the direction of the first eigenvector $\mathbf{e}_1$, along with the treatment vectors $\boldsymbol{\tau}_k$. Comparing this figure with the right side of Figure 3.7, we can see that the treatment vectors have the same relative position as the mean vectors $\bar{\mathbf{x}}_k$, and that the first eigenvector of $\mathbf{B}$ indeed provides the direction where the variance between the treatment vectors (and the mean vectors) is maximised.

3.10 Fisher's Linear Discriminant Analysis

Fisher [40] proposed another technique, known as *Fisher's Linear Discriminant Analysis* (LDA), which can also be used for dimensionality reduction. LDA is aimed at maximising

the classification performance between different groups, which is the main goal of the template attacks presented in the following chapters.

Let $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_G$ be G random vectors in $\mathbb{R}^m$ (representing for example the leakage matrices used in Figure 3.10), with the corresponding mean vectors $\boldsymbol{\mu}_1^X, \boldsymbol{\mu}_2^X, \ldots, \boldsymbol{\mu}_G^X$ and having equal covariances $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \ldots = \boldsymbol{\Sigma}$. Let also $Y_{kj} = \mathbf{a}_j{}'\mathbf{X}_k$ represent a linear combination of the random vector $\mathbf{X}_k$, having mean $\mu_{kj}^Y = \mathbf{a}_j{}'\boldsymbol{\mu}_k^X$ and variance $\mathbf{Var}(Y_{kj}) = \mathbf{Var}(\mathbf{a}_j{}'\mathbf{X}_k) = \mathbf{a}_j{}'\boldsymbol{\Sigma}_k\mathbf{a}_j = \mathbf{a}_j{}'\boldsymbol{\Sigma}\mathbf{a}_j$. Due to the equality of covariances (*homoscedasticity*), we can observe that the variance $\mathbf{Var}(Y_{kj})$ of the linear combinations is independent of k, so I shall use the notation $\mathbf{Var}(Y_j) = \mathbf{Var}(Y_{kj})$ for all groups. Furthermore, let $\boldsymbol{\mu}^X = \frac{1}{G}\sum_{k=1}^{G}\boldsymbol{\mu}_k^X$ and $\mu_j^Y = \frac{1}{G}\sum_{k=1}^{G}\mu_{kj}^Y$ represent the means across the G random vectors.

The goal of LDA is to find those linear combinations $Y_{kj} = \mathbf{a}_j{}'\mathbf{X}_k$ that maximise the ratio

$$\begin{aligned}\frac{\sum_{k=1}^{G}(\mu_{kj}^Y - \bar{\mu}_j^Y)^2}{\mathbf{Var}(Y_j)} &= \frac{\sum_{k=1}^{G}(\mathbf{a}_j{}'(\boldsymbol{\mu}_k^X - \boldsymbol{\mu}^X))^2}{\mathbf{Var}(\mathbf{a}_j{}'\mathbf{X})} \\ &= \frac{\mathbf{a}_j{}'\big(\sum_{k=1}^{G}(\boldsymbol{\mu}_k^X - \boldsymbol{\mu}^X)(\boldsymbol{\mu}_k^X - \boldsymbol{\mu}^X)'\big)\mathbf{a}_j}{\mathbf{a}_j{}'\boldsymbol{\Sigma}\mathbf{a}_j} = \frac{\mathbf{a}_j{}'\mathbf{B}\mathbf{a}_j}{\mathbf{a}_j{}'\boldsymbol{\Sigma}\mathbf{a}_j},\end{aligned} \tag{3.88}$$

where $\mathbf{B}$ is the treatment matrix presented in Section 3.8.2. We can observe that this is precisely a multivariate signal-to-noise ratio, since $\mathbf{B}$ contains our signal of interest, and $\boldsymbol{\Sigma}$ contains the noise (variance and correlation) common to all traces.

The linear combinations $Y_{kj} = \mathbf{a}_j{}'\mathbf{X}_k$ that maximise (3.88) are known as *sample discriminants*, and the coefficients $\mathbf{a}_j$ are given by the eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_s$ corresponding to the largest eigenvalues of $\boldsymbol{\Sigma}^{-1}\mathbf{B}$, where $s = \min(m, G-1)$ is the maximum number of non-zero eigenvectors, as that is the maximum number of independent linear combinations available in $\mathbf{B}$. In practice, we shall often use the sample estimates of $\boldsymbol{\Sigma}$ and $\mathbf{B}$, such that the coefficients of LDA will be given by the eigenvectors of $\mathbf{S}_{\text{pooled}}^{-1}\mathbf{B}$. Also, we should use some rule, such as those in Section 3.9.1, to select the minimum number K of discriminants that are needed for a good discrimination.

There is an interesting similarity between the left hand side of (3.88) and the F-score in (3.72), in particular when used to discriminate between leakage samples of different classes (e.g. different k). While with (3.72), we compute the *individual* F-score for each sample and then select the samples with the highest score, Fisher's LDA finds the linear combinations of the samples that maximise the *overall* signal-to-noise ratio from (3.88), taking into account the overall variation and correlation between samples. In the following chapters, I will show that this has a great impact on the performance of template attacks.

Chapter 4

Efficient template attacks

In this chapter, I provide a detailed introduction to template attacks (Section 4.1), and then explain some efficient techniques to implement these attacks in practice (Section 4.4), complemented with results from the *Grizzly* dataset (Section 4.6). These results have been published in the following conference paper:

> Omar Choudary and Markus G. Kuhn. *Efficient Template Attacks*, CARDIS 2013, Berlin, 27–29 November 2013, LNCS 8419, pp. 253–270 [26].

As I will detail in the following sections, these results dismiss some previous misconceptions about the implementation of template attacks and show that with enough data and good algorithms we can extract much more information from side-channel leakage traces than was previously thought. In particular, I shall show that we can determine almost perfectly an unknown 8-bit value processed by a single load instruction in a microcontroller. These are probably among the best published results on eavesdropping a single data value (not in the context of any specific cryptographic algorithm) on an 8-bit microcontroller. Furthermore, in Section 4.7, I demonstrate how the efficient techniques presented in this chapter can be used to attack a hardware implementation of AES.

4.1 Template attacks

I now provide a detailed description of the template attack. In the rest of this chapter, and in the following chapters, I shall use the notations introduced in Chapter 3.

As I mentioned in the previous section, to implement a template attack we need physical access to a pair of identical devices, which I refer to as the *profiling* and the *attacked* device. We wish to infer some secret value $k\star \in \mathcal{S}$, processed by the attacked device at some point. For an 8-bit microcontroller, $\mathcal{S} = \{0, \ldots, 255\}$ might be the set of possible byte values manipulated by a particular machine instruction. However, note that this attack is not

restricted to 8-bit microcontrollers. As I show in Chapter 7, we can combine the efficient attacks described here with Stochastic Models [97] to obtain practical attacks on 16-bit targets. Furthermore, it is generally possible to target 8 bits (e.g. the bits processed by an AES S-box) even in architectures with 64-bit bus, by considering the other bits as electronic noise (although in this case we shall need more traces).

We assume that we determined the approximate moments of time when the secret value $k\star$ is manipulated and we are able to record signal traces (e.g. supply current or electromagnetic waveforms) around these moments. Let $\mathbf{x} \in \mathbb{R}^{m^{\mathrm{r}}}$ be such a leakage trace. I shall use the symbol m^{r} to refer to the *total* number of leakage samples that we obtain from the oscilloscope, i.e. the number of leakage samples in a *raw* trace, and the symbol m to refer to an arbitrary selection or projection of these leakage samples, such that $m \leq m^{\mathrm{r}}$. More generally, I shall use the superscript $^{\mathrm{r}}$ whenever a symbol applies to the raw traces.

During the *profiling* phase we record n_{p} leakage vectors $\mathbf{x}^{\mathrm{r}}_{ki} \in \mathbb{R}^{m^{\mathrm{r}}}$ from the profiling device for each possible value $k \in \mathcal{S}$, and combine these as row vectors $\mathbf{x}^{\mathrm{r}}_{ki}{}'$ in the leakage matrix $\mathbf{X}^{\mathrm{r}}_{k} \in \mathbb{R}^{n_{\mathrm{p}} \times m^{\mathrm{r}}}$.

Typically, the *raw* leakage vectors $\mathbf{x}^{\mathrm{r}}_{ki}$ provided by the data acquisition device contain a large number m^{r} of leakage samples, due to high sampling rates used. Therefore, we might *compress* them before further processing, either by selecting only a subset of $m \ll m^{\mathrm{r}}$ of those samples, or by applying some other data-dimensionality reduction method (see Section 4.3). I refer to such compressed leakage vectors as $\mathbf{x}_{ki} \in \mathbb{R}^{m}$ and combine all of these as rows into the compressed leakage matrix $\mathbf{X}_k \in \mathbb{R}^{n_{\mathrm{p}} \times m}$. (Without any such compression step, we would have $\mathbf{X}_k = \mathbf{X}^{\mathrm{r}}_k$ and $m = m^{\mathrm{r}}$.)

Then, as presented in Section 3.3, we can compute the sample mean $\bar{\mathbf{x}}_k \in \mathbb{R}^{m}$ and sample covariance $\mathbf{S}_k \in \mathbb{R}^{m \times m}$ for each possible value $k \in \mathcal{S}$ as

$$\bar{\mathbf{x}}_k = \tfrac{1}{n_{\mathrm{p}}} \sum_{i=1}^{n_{\mathrm{p}}} \mathbf{x}_{ki}, \qquad \mathbf{S}_k = \tfrac{1}{n_{\mathrm{p}}-1} \sum_{i=1}^{n_{\mathrm{p}}} (\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)(\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)'. \tag{4.1}$$

These are known as the *template parameters*. Note that others [9, 101, 36] have used $1/n_{\mathrm{p}}$ rather than $1/(n_{\mathrm{p}} - 1)$ in the computation of $\mathbf{S}_k$, thereby computing the *maximum likelihood estimator (MLE)* of $\mathbf{\Sigma}_k$. In theory, the correct estimator for $\mathbf{\Sigma}_k$ is the unbiased estimator with $1/(n_{\mathrm{p}} - 1)$; the MLE merely maximises the joint likelihood from the multivariate normal distribution. In practice, I found this choice makes no significant performance difference (even down to $n_{\mathrm{p}} = 10, m = 6$).

As I show in Appendix C, side-channel leakage traces can be modeled well by the multivariate normal distribution, which I presented in detail in Chapter 3. Then, the probability density function (pdf) of a leakage vector $\mathbf{x}$, given $\bar{\mathbf{x}}_k$ and $\mathbf{S}_k$, is

$$f(\mathbf{x} \mid \bar{\mathbf{x}}_k, \mathbf{S}_k) = \frac{1}{\sqrt{(2\pi)^m |\mathbf{S}_k|}} \exp\left(-\frac{1}{2} (\mathbf{x} - \bar{\mathbf{x}}_k)' \mathbf{S}_k^{-1} (\mathbf{x} - \bar{\mathbf{x}}_k) \right). \tag{4.2}$$

In the *attack* phase, we try to infer the secret value $k\star \in \mathcal{S}$ processed by the attacked device. We obtain n_a leakage vectors $\mathbf{x}_i \in \mathbb{R}^m$ from the attacked device, using the same recording technique and compression method as in the profiling phase, resulting in the leakage matrix $\mathbf{X}_{k\star} \in \mathbb{R}^{n_\mathrm{a} \times m}$. Then, for each $k \in \mathcal{S}$, we compute a *discriminant score* $\mathrm{d}(k \mid \mathbf{X}_{k\star})$. Finally, we try all $k \in \mathcal{S}$ on the attacked device, in order of decreasing score (optimized brute-force search, e.g. for a password or cryptographic key), until we find the correct $k\star$. Given a trace $\mathbf{x}_i$ from $\mathbf{X}_{k\star}$, a commonly used discriminant [9, 101, 36], derived from Bayes' rule, is

$$\mathrm{d}(k \mid \mathbf{x}_i) = f(\mathbf{x}_i \mid \bar{\mathbf{x}}_k, \mathbf{S}_k) P(k), \tag{4.3}$$

where the denominator from Bayes' rule is omitted, as it is the same for each k. Assuming a uniform a-priori probability $P(k) = |\mathcal{S}|^{-1}$, applying Bayes' rule becomes equivalent to computing the likelihood

$$\mathrm{l}(k \mid \mathbf{x}_i) = \mathrm{d}(k \mid \mathbf{x}_i) = \mathrm{l}(\bar{\mathbf{x}}_k, \mathbf{S}_k \mid \mathbf{x}_i) = f(\mathbf{x}_i \mid \bar{\mathbf{x}}_k, \mathbf{S}_k), \tag{4.4}$$

where the latter can be computed from (4.2). However, we do not need to compute a proper a-posteriori probability for each candidate k given a trace $\mathbf{x}_i$, but only a discriminant function that allows us to sort scores and identify the most likely candidates. In Section 4.4, I show how the latter can be much more efficient.

4.2 Implementation caveats

Here I present several problems that can appear when implementing the template attack, especially when using a large number of samples m.

4.2.1 Inverse of covariance matrix

Several authors [73, 36, 12] noted that inverting the covariance matrix $\mathbf{S}_k$ from (4.1), as needed in (4.2), can cause numerical problems for large m. However, it is important to understand why $\mathbf{S}_k$ can become singular ($|\mathbf{S}_k| \approx 0$), causing these problems.

Since $\mathbf{S}_k$ is essentially the matrix product $\widetilde{\mathbf{X}}'_k \widetilde{\mathbf{X}}_k$ (see (3.18)), both $\mathbf{S}_k$ and $\widetilde{\mathbf{X}}_k$ have the same rank. Therefore $\mathbf{S}_k$ is singular iff $\widetilde{\mathbf{X}}_k$ has dependent columns, which is guaranteed if $n_\mathrm{p} < m$. The constraint on $\widetilde{\mathbf{X}}_k$ to have zero-mean rows implies that it has dependent columns even for $n_\mathrm{p} = m$. Therefore, $n_\mathrm{p} > m$ is a *necessary* condition for $\mathbf{S}_k$ to be non-singular. See [54, Result 3.3] for a more detailed proof.

The restriction $m < n_\mathrm{p}$ is one main reason for reducing m through compression (see Section 4.3). However, it is not mandatory to compress m further than what is needed to keep the columns of $\widetilde{\mathbf{X}}_k$ independent. In practice, some of the leakage samples can be highly correlated, in which case n_p needs to be somewhat larger than m. For example,

using all the $N = 3072$ raw traces from the leakage matrix $\mathbf{X}_1^{\mathrm{r}}$ of the *Grizzly Alpha* dataset, the resulting covariance matrix $\mathbf{S}_1 = \mathbf{Cov}(\mathbf{X}_1^{\mathrm{r}})$ has full rank ($\mathrm{rank}(\mathbf{S}_1) = m = 2500$), and hence poses no problems for inversion.

If we cannot obtain $n_{\mathrm{p}} > m$, then we may try to correct the covariance matrix [68, 41, 113]. With the covariance estimator of Ledoit and Wolf [68] I obtained a non-singular $\mathbf{S}_k$ even for $n_{\mathrm{p}} < m$. However, when possible, a much better option is to use the pooled covariance matrix $\mathbf{S}_{\mathrm{pooled}}$, presented in Section 3.8.3.

4.2.2 Floating-point limitations

One practical problem with (4.2) is that, for large m, the statistical distance

$$(\mathbf{x} - \bar{\mathbf{x}}_k)' \mathbf{S}_k^{-1} (\mathbf{x} - \bar{\mathbf{x}}_k)$$

can reach values that cause the subsequent exponentiation operation to overflow. For example, in IEEE double precision, $\exp(x)$ is only safe with $|x| < 710$, easily exceeded for large m. For example, taking again the leakage matrix $\mathbf{X}_1^{\mathrm{r}}$ of the *Grizzly Alpha* dataset, I obtain $(\mathbf{x}_{1,1} - \bar{\mathbf{x}}_1)' \mathbf{S}_1^{-1} (\mathbf{x}_{1,1} - \bar{\mathbf{x}}_1) = 2496.2$.

Another problem is that, for large m, the determinant $|\mathbf{S}_k|$ can overflow or underflow. This can be explained easily, observing that $|\mathbf{S}_k| = \lambda_1 \lambda_2 \ldots \lambda_m$, which can be easily verified from the SVD in (3.81) and noting that $|\mathbf{U}| = \pm 1$. For the covariance matrix $\mathbf{S}_1$ of the *Grizzly Alpha* dataset, $\lambda_{50} \approx 5.28 \times 10^6$ (all the previous eigenvalues are larger). Multiplying merely 50 such values again overflows the IEEE double precision format.

4.3 Compression methods

A compression method can be used to reduce the length (dimensionality) of leakage vectors from m^{r} to m. As detailed in Section 4.2, this may be needed if we do not have enough traces for a full rank covariance matrix or to cope with computational or memory restrictions. There are two main common approaches used in the literature: (a) selecting some of the samples based on some criteria; (b) using some linear combinations of the leakage vectors, based on the principal components or Fisher's linear discriminant. All of these approaches, as I will show in the following, rely on the treatment vectors

$$\tau_k = (\bar{\mathbf{x}}_k - \bar{\mathbf{x}}), \tag{4.5}$$

that define the signal of interest (see Section 3.8.2), where

$$\bar{\mathbf{x}} = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \bar{\mathbf{x}}_k. \tag{4.6}$$

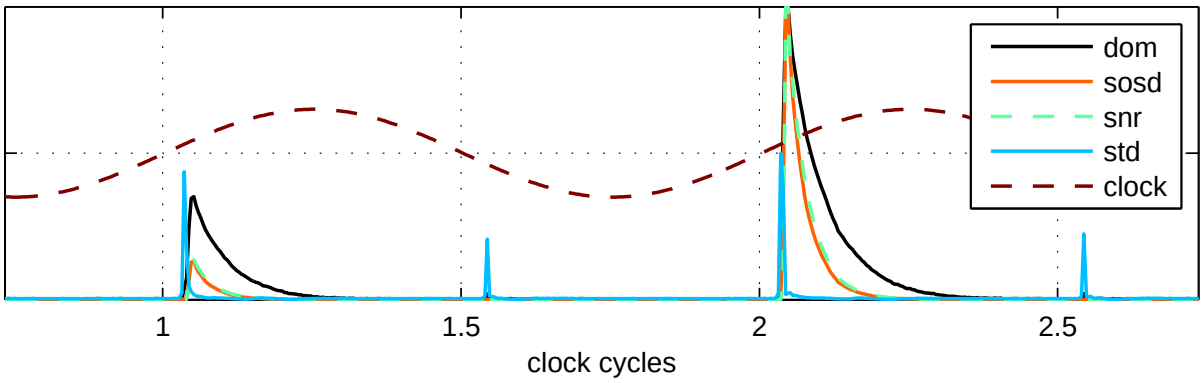

Figure 4.1: Signal-strength estimates from DOM, SOSD and SNR (identical to SOST) on the *Grizzly Beta* dataset, along with the average standard deviation (STD) of the traces and clock signal. $n_{\mathrm{p}} = 2000$. All estimates are rescaled to fit into the plot, so the vertical axis (linear) has no scale.

4.3.1 Selection of samples

In this method we first compute a signal-strength estimate $\mathbf{s}(t), t \in \{t_1, \ldots, t_{m^{\mathrm{r}}}\}$, and then we select a subset of m points based on this estimate.

There are several proposals for producing $\mathbf{s}(t)$, such as *difference of means (DOM)* [24, Section 2.1], the *sum of squared differences (SOSD)* [46], the *Signal to Noise Ratio (SNR)* [73] and *SOST* [46]. All these are similar, with the notable difference that the first two do not take the variance of the traces into consideration, while the latter two do. The SNR and SOST methods are variants of the F-test, presented in Section 3.8.1.

The DOM method was first proposed by Chari et al. [24, Section 2.1], where they proposed to select samples at which large pairwise differences between the means show up. Later, Rechberger and Oswald [92, Section 3.2] explicitly suggested to sum these pairwise differences and then select the samples from the traces with largest peaks. Gierlichs et al. [46, Section 2.1] observed that using the sum is not appropriate, proposing the sum of squared differences (SOSD) instead.

I found that the sum of the absolute value of pairwise differences

$$\mathbf{s}_{\mathrm{DOM}} = \sum_{1 \le i < j < |\mathcal{S}|} |\bar{\mathbf{x}}^{\mathrm{r}}_i - \bar{\mathbf{x}}^{\mathrm{r}}_j| \tag{4.7}$$

gives very good results, which is what I refer to as DOM from now on. Here, $\bar{\mathbf{x}}^{\mathrm{r}}_i$, $\bar{\mathbf{x}}^{\mathrm{r}}_j$ are the mean vectors, as in (4.1), but calculated from the raw leakage vectors $\mathbf{x}^{\mathrm{r}}_i$.

In Figure 4.1, I show these estimates for the *Grizzly Beta* dataset. The methods SNR and SOST are in fact the same if we consider the variance at each sample point to be independent of the candidate k, which is expected in our setting. Under this condition SNR and SOST reduce to computing the F-score from Section 3.8.1.

In the second step of this compression method we need to choose m samples based on the signal-strength estimate $\mathbf{s}$. The goal is to select the smallest set of samples that contains most of the information about our target. A guideline, proposed by Rechberger and Oswald [92, Section 3.2], is to select at most one sample per clock cycle among the samples with highest $\mathbf{s}$. In Section 4.6, I evaluate several other options, and show that selecting several leakage samples per clock cycle can provide better results.

4.3.2 Principal Component Analysis (PCA)

In Section 3.9, I explained in detail how PCA can be used to reduce the dimensionality of leakage traces, providing examples for the *Grizzly Alpha* dataset. However, it was Archambeau et al. [9], who proposed the use of PCA on the treatment matrix

$$\mathbf{B} = n_\mathrm{p} \sum_{k \in \mathcal{S}} (\bar{\mathbf{x}}_k^\mathrm{r} - \bar{\mathbf{x}}^\mathrm{r})(\bar{\mathbf{x}}_k^\mathrm{r} - \bar{\mathbf{x}}^\mathrm{r})' \tag{4.8}$$

for template attacks (see also Section 3.9.3).

Using PCA on the matrix $\mathbf{B} \in \mathbb{R}^{m^\mathrm{r} \times m^\mathrm{r}}$, we obtain the SVD $\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{U}'$, where the eigenvectors $\mathbf{e}_j \in \mathbb{R}^{m^\mathrm{r}}$ (the columns of $\mathbf{U}$) provide the directions such that the variance between the treatment vectors τ_k is maximised (see also Figure 3.10 right). As a result, only the first m eigenvectors $[\mathbf{e}_1 \dots \mathbf{e}_m] = \mathbf{U}^m$ are needed in order to preserve most of the information from the mean vectors $\bar{\mathbf{x}}_k^\mathrm{r}$. Therefore, we can project the raw mean vectors $\bar{\mathbf{x}}_k^\mathrm{r}$ and covariance matrices $\mathbf{S}_k^\mathrm{r}$ into the new coordinate system defined by $\mathbf{U}^m$ to obtain the PCA template parameters $\bar{\mathbf{x}}_k \in \mathbb{R}^m$ and $\mathbf{S}_k \in \mathbb{R}^{m \times m}$, where

$$\bar{\mathbf{x}}_k = \mathbf{U}^{m\prime} \bar{\mathbf{x}}_k^\mathrm{r}, \qquad \mathbf{S}_k = \mathbf{U}^{m\prime} \mathbf{S}_k^\mathrm{r} \mathbf{U}^m. \tag{4.9}$$

For situations where $m^\mathrm{r} \gg |\mathcal{S}|$ (which is typical for 8-bit targets, but not so for 16-bit or larger targets), Archambeau et al. [9] show a better method for computing $\mathbf{U}$. However, in my experiments with $m^\mathrm{r} = 2500$, the direct computation of $\mathbf{U}$ from the SVD of $\mathbf{B}$ worked well, requiring only 14 s to complete on a normal PC (Intel i3 CPU 3.07 GHz).

Alternative computation of PCA templates

Even though in [101, Section 4.1] the authors mention that PCA can help where computing the full covariance matrix $\mathbf{S}_k^\mathrm{r}$ is prohibitive (due to large m^r), the original PCA approach [9] still requires the computation of $\mathbf{S}_k^\mathrm{r}$ (see (4.9)). Also, numerical artifacts during the double matrix multiplication in (4.9) can make $\mathbf{S}_k$ non-symmetric. One way to avoid the latter is to use the Cholesky decomposition $\mathbf{S}_k^\mathrm{r} = \mathbf{C}'\mathbf{C}$ and compute

$$\mathbf{S}_k = \mathbf{U}^{m\prime} \mathbf{S}_k^\mathrm{r} \mathbf{U}^m = \mathbf{U}^{m\prime} \mathbf{C}' \mathbf{C} \mathbf{U}^m = (\mathbf{C}\mathbf{U}^m)'(\mathbf{C}\mathbf{U}^m) = \mathbf{V}'\mathbf{V}. \tag{4.10}$$

However, based on the facts shown in (3.6.1), we know that

$$\mathbf{S}_k = \mathrm{Cov}(\mathbf{X}_k^{\mathrm{r}}\mathbf{U}^m) = \mathbf{U}^{m\prime}\mathrm{Cov}(\mathbf{X}_k^{\mathrm{r}})\mathbf{U}^m = \mathbf{U}^{m\prime}\mathbf{S}_k^{\mathrm{r}}\mathbf{U}^m. \tag{4.11}$$

Therefore, to avoid both the numerical artifacts and the computation of large covariance matrices, we can first compute the projected leakage matrix

$$\mathbf{X}_k = \mathbf{X}_k^{\mathrm{r}}\mathbf{U}^m \tag{4.12}$$

and then compute the PCA-based template parameters using (4.1) on $\mathbf{X}_k$. I use this method throughout my experiments.

4.3.3 Fisher's Linear Discriminant Analysis (LDA)

As explained in Section 3.10, we can also use Fisher's LDA to compress the leakage traces. LDA takes into account both the variability of the means (from $\mathbf{B}$) and the variability within each leakage matrix (from $\mathbf{S}_{\mathrm{pooled}}$). This allows LDA to perform better than PCA, although, if the pooled covariance matrix $\mathbf{S}_{\mathrm{pooled}}$ (see Section 4.4.2) cannot be well estimated, then PCA might be a better choice.

LDA provides the eigenvectors $[\mathbf{e}_1 \dots \mathbf{e}_{m^{\mathrm{r}}}] = \mathbf{U}$ corresponding to the largest eigenvalues of $\mathbf{S}_{\mathrm{pooled}}^{-1}\mathbf{B}$. As with PCA, we only need to use the first m eigenvectors $[\mathbf{e}_1 \dots \mathbf{e}_m] = \mathbf{U}^m$ to preserve most of the information from the original traces. Then, we can project each leakage matrix as

$$\mathbf{X}_k = \mathbf{X}_k^{\mathrm{r}}\mathbf{U}^m \tag{4.13}$$

and compute the LDA-based template parameters using (4.1).

Several authors [101, 36] have used Fisher's LDA for template attacks, but without mentioning two important aspects. Firstly, the condition of equal covariances (known as *homoscedasticity*) may be important for the success of Fisher's LDA. Therefore, the PCA method (Section 4.3.2), which does not depend on this condition, might be a better choice in some settings. Secondly, the coefficients that maximise (3.88) can be obtained using scaled versions of $\mathbf{S}_{\mathrm{pooled}}$[1] or different approaches [101, 36], which will result in a different scale of the coefficients $\mathbf{a}_j$. This difference has a major impact on the template attack: *only* if we scale the coefficients $\mathbf{a}_j$, such that $\mathbf{a}_j{}'\mathbf{S}_{\mathrm{pooled}}\mathbf{a}_j = 1$, will the covariance of the compressed traces become the identity matrix [54, Exercise 11.21], i.e. $\mathbf{S}_k = \mathbf{I}$. That means that we only need to use the sample means during the attack, which greatly reduces computation and storage requirements. Therefore, we can compute the diagonal matrix $\mathbf{Q} \in \mathbb{R}^{m\times m}$, having the values $q_{jj} = (\frac{1}{\mathbf{a}_j{}'\mathbf{S}_{\mathrm{pooled}}\mathbf{a}_j})^{\frac{1}{2}} = (\frac{1}{\mathbf{e}_j{}'\mathbf{S}_{\mathrm{pooled}}\mathbf{e}_j})^{\frac{1}{2}}$ on its diagonal, to obtain the scaled coefficients $\mathbf{AQ} = \mathbf{U}^m\mathbf{Q}$, and replace (4.13) by

$$\mathbf{X}_k = \mathbf{X}_k^{\mathrm{r}}\mathbf{AQ} = \mathbf{X}_k^{\mathrm{r}}\mathbf{U}^m\mathbf{Q}. \tag{4.14}$$

[1] Instead of $\mathbf{S}_{\mathrm{pooled}}$ we could use $\mathbf{W} = |\mathcal{S}|(n_{\mathrm{p}} - 1)\mathbf{S}_{\mathrm{pooled}}$, the *sample within groups* matrix.

An alternative approach is to compute the eigenvectors $\mathbf{e}_j$ of $\mathbf{S}_{\text{pooled}}^{-\frac{1}{2}}\mathbf{B}\mathbf{S}_{\text{pooled}}^{-\frac{1}{2}}$ and then obtain the coefficients $\mathbf{a}_j = \mathbf{S}_{\text{pooled}}^{-\frac{1}{2}}\mathbf{e}_j$, which leads directly to coefficients that satisfy $\mathbf{a}_j{}'\mathbf{S}_{\text{pooled}}\mathbf{a}_j = 1$. Throughout my experiments I used the first approach.

4.4 Efficient implementation of template attacks

In this section I present methods that avoid the problems identified in Section 4.2 and implement template attacks very efficiently.

4.4.1 Using the logarithm of the multivariate normal distribution

Mangard et al. [73, p. 108] suggested calculating the logarithm of (4.2), as in

$$\log f(\mathbf{x} \mid \bar{\mathbf{x}}_k, \mathbf{S}_k) = -\frac{1}{2}\left(\log\left[(2\pi)^m|\mathbf{S}_k|\right] + (\mathbf{x}-\bar{\mathbf{x}}_k)'\mathbf{S}_k^{-1}(\mathbf{x}-\bar{\mathbf{x}}_k)\right). \tag{4.15}$$

They then claim that *"the template that leads to the smallest absolute value [of (4.15)] indicates the correct [candidate]"*.

The first problem with this approach is that (4.15) does not avoid the computation of $|\mathbf{S}_k|$, which I have shown to be problematic. Therefore, a better method is to compute the logarithm of the multivariate normal pdf as

$$\log \mathrm{f}(\mathbf{x} \mid \bar{\mathbf{x}}_k, \mathbf{S}_k) = -\frac{m}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{S}_k| - \frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}}_k)'\mathbf{S}_k^{-1}(\mathbf{x}-\bar{\mathbf{x}}_k), \tag{4.16}$$

where we can compute the logarithm of the determinant as

$$\log|\mathbf{S}_k| = 2\sum_{c_{ii}\in\mathrm{diag}(\mathbf{C})}\log c_{ii}, \tag{4.17}$$

using the Cholesky decomposition $\mathbf{S}_k = \mathbf{C}'\mathbf{C}$ of the symmetric matrix $\mathbf{S}_k$. (Since $\mathbf{C}$ is triangular, its determinant is the product of its diagonal elements.)

Secondly, it is incorrect to choose the candidate k that leads to the *"smallest absolute value"* of (4.15,4.16), since the logarithm is a monotonic function and preserves the property that the *largest value* corresponds to the correct k.[2]

Using the log-likelihood from (4.16), and dropping the first term which is constant across all k, we can compute the following discriminant score:

$$\begin{aligned} \mathrm{d}_{\mathrm{LOG}}(k \mid \mathbf{x}_i) &= -\frac{1}{2}\log|\mathbf{S}_k| - \frac{1}{2}(\mathbf{x}_i-\bar{\mathbf{x}}_k)'\mathbf{S}_k^{-1}(\mathbf{x}_i-\bar{\mathbf{x}}_k) \\ &= \log \mathrm{f}(\mathbf{x}_i \mid \bar{\mathbf{x}}_k, \mathbf{S}_k) + \frac{m}{2}\log 2\pi = \log \mathrm{l}(k \mid \mathbf{x}_i) + \text{const}, \end{aligned} \tag{4.18}$$

which avoids the numerical issues that can appear with (4.3).

[2]Note that a probability *density* function (continuous), such as f from (4.2), unlike a probability *mass* function (discrete), can be both larger or smaller than 1 and therefore its logarithm can be both positive or negative.

4.4.2 Using a pooled covariance matrix

As I explained in Section 3.8.3, if the leakages from different candidates k have different means but the same *real* covariance $\mathbf{\Sigma} = \mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 = \cdots = \mathbf{\Sigma}_k$, it is possible to use a pooled covariance

$$\mathbf{S}_{\text{pooled}} = \frac{1}{|\mathcal{S}|(n_{\text{p}} - 1)} \sum_{k \in \mathcal{S}} \sum_{i=1}^{n_{\text{p}}} (\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)(\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)', \tag{4.19}$$

which represents a much better estimate of the real covariance $\mathbf{\Sigma}$, since $\mathbf{S}_{\text{pooled}}$ estimates the covariance using $n_{\text{p}}|\mathcal{S}|$ traces, while the individual covariance matrices $\mathbf{S}_k$ use only n_{p}. This in turn means that the condition for a non-singular matrix (see Section 4.2.1) relaxes to $n_{\text{p}}|\mathcal{S}| > m$ or $n_{\text{p}} > \frac{m}{|\mathcal{S}|}$. Therefore, the number of traces that we must obtain for each candidate k is reduced by a factor of $|\mathcal{S}|$, a great advantage in practice. Nevertheless, the quality of the mean estimates $\bar{\mathbf{x}}_k$ still depends directly on n_{p}. Also note that for Fisher's LDA (see Sections 3.10 and 4.3.3) we need to compute the inverse of $\mathbf{S}_{\text{pooled}} \in \mathbb{R}^{m^{\text{r}} \times m^{\text{r}}}$, which requires $n_{\text{p}}|\mathcal{S}| > m^{\text{r}}$.

Several authors used $\mathbf{S}_{\text{pooled}}$ with template attacks [13, 83], but did not provide a clear motivation for its use. We expect the assumption of equal covariances to hold for many side-channel applications, because the covariance matrices $\mathbf{\Sigma}_k$ capture primarily information about how *noise*, that is variation in the recorded traces unrelated to k, is correlated across trace samples. After all, the data-dependent signal $\bar{\mathbf{x}}_k$ was already subtracted. As a result, we should not expect substantial differences between the sample covariances $\mathbf{S}_k$ for different candidate values k, unless the target device contains some mechanism by which k can modify the correlation between samples. Note that in this assumption I do not consider masking implementations [106, 90], and therefore I do not consider scenarios in which leakage samples depend on k due to missing information during profiling.

Box's test [17] can be used to reject the hypothesis of equal covariances, although it can be misleading for large $|\mathcal{S}|$ or large m. In my experiments, with $|\mathcal{S}| = 2^8$, $m = 6$ and $n_{\text{p}} = 2000$, Box's variable $C \sim F_{f1,f2}(\alpha)$ had the value 2.03, which was above the rejection threshold for any realistic significance level (e.g. $F_{f1,f2}(0.99) = 1.045$). Nevertheless, the different $\mathbf{S}_k$ are visually similar (viewed as bitmaps with linear colour mapping), and we can consider that the hypothesis of equal covariances (homoscedasticity) is confirmed by the superior results obtained with the pooled covariance (see Section 4.6).

When using $\mathbf{S}_{\text{pooled}}$, the first term in (4.18) becomes constant. The remaining term is based on the statistical distance (or Mahalanobis distance)

$$d_{\text{M}}^2(\mathbf{x} \mid \bar{\mathbf{x}}_k, \mathbf{S}_{\text{pooled}}) = (\mathbf{x} - \bar{\mathbf{x}}_k)' \mathbf{S}_{\text{pooled}}^{-1} (\mathbf{x} - \bar{\mathbf{x}}_k) \geq 0, \tag{4.20}$$

which I described in detail in Section 3.4. As a result, we can use the discriminant score

$$\text{d}_{\text{MD}}(k \mid \mathbf{x}_i) = -\frac{1}{2} d_{\text{M}}^2(\mathbf{x}_i \mid \bar{\mathbf{x}}_k, \mathbf{S}_{\text{pooled}}) = \text{d}_{\text{LOG}}(k \mid \mathbf{x}_i) + \text{const.} \tag{4.21}$$

to compare the candidates k.

4.4.3 Linear discriminant score

When using the pooled covariance matrix $\mathbf{S}_{\text{pooled}}$ we can rewrite the distance from (4.20) as

$$d_{\text{M}}^2(\mathbf{x} \mid \bar{\mathbf{x}}_k, \mathbf{S}_{\text{pooled}}) = \mathbf{x}'\mathbf{S}_{\text{pooled}}^{-1}\mathbf{x} - 2\bar{\mathbf{x}}_k'\mathbf{S}_{\text{pooled}}^{-1}\mathbf{x} + \bar{\mathbf{x}}_k'\mathbf{S}_{\text{pooled}}^{-1}\bar{\mathbf{x}}_k, \tag{4.22}$$

because

$$\bar{\mathbf{x}}_k'\mathbf{S}_{\text{pooled}}^{-1}\mathbf{x} = (\bar{\mathbf{x}}_k'\mathbf{S}_{\text{pooled}}^{-1}\mathbf{x})' = \mathbf{x}'\mathbf{S}_{\text{pooled}}^{-1}{}'\bar{\mathbf{x}}_k = \mathbf{x}'\mathbf{S}_{\text{pooled}}^{-1}\bar{\mathbf{x}}_k. \tag{4.23}$$

The first term in (4.22) is constant for all groups k so we can discard it. That means, that we can now use the following *linear* discriminant score:

$$\text{d}_{\text{LINEAR}}(k \mid \mathbf{x}_i) = \bar{\mathbf{x}}_k'\mathbf{S}_{\text{pooled}}^{-1}\mathbf{x}_i - \frac{1}{2}\bar{\mathbf{x}}_k'\mathbf{S}_{\text{pooled}}^{-1}\bar{\mathbf{x}}_k = \text{d}_{\text{MD}}(k \mid \mathbf{x}_i) + \text{const.}, \tag{4.24}$$

which depends *linearly* on $\mathbf{x}_i$ (where const. does not depend on k). Although equivalent, the linear discriminant d_{LINEAR} can be far more efficient to compute than the quadratic d_{MD}.

4.4.4 Combining multiple attack traces

In the previous sections I presented a couple of discriminants used with a single leakage trace. However, an attacker might be able to use many attack traces. Therefore, I now present two sound options for combining the n_{a} individual leakage traces $\mathbf{x}_i$ from $\mathbf{X}_{k\star}$ into the final discriminant score $\text{d}(k \mid \mathbf{X}_{k\star})$.

Option 1:

Average all the traces in $\mathbf{X}_{k\star}$ (similar to the mean computation in (4.1)) in order to remove as much noise as possible and then use this single mean trace $\bar{\mathbf{x}}_{k\star}$ to compute

$$\text{d}^{\text{avg}}(k \mid \mathbf{X}_{k\star}) = \text{d}(k \mid \bar{\mathbf{x}}_{k\star}). \tag{4.25}$$

This option is computationally fast, requiring $O(n_{\text{a}}m + m^2)$ time for any presented discriminant, but it does not use all the information from the available attack traces (in particular the noise).

Option 2:

Compute the joint likelihood $\mathrm{l}(k \mid \mathbf{X}_{k\star}) = \prod_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathrm{l}(k \mid \mathbf{x}_i)$. By applying the logarithm to both sides we have $\log \mathrm{l}(k \mid \mathbf{X}_{k\star}) = \sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \log \mathrm{l}(k \mid \mathbf{x}_i)$ and we obtain the derived scores:

$$\mathrm{d}^{\text{joint}}_{\text{LOG}}(k \mid \mathbf{X}_{k\star}) = -\frac{n_\mathrm{a}}{2} \log |\mathbf{S}_k| - \frac{1}{2} \sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} (\mathbf{x}_i - \bar{\mathbf{x}}_k)' \mathbf{S}_k^{-1} (\mathbf{x}_i - \bar{\mathbf{x}}_k), \tag{4.26}$$

$$\mathrm{d}^{\text{joint}}_{\text{MD}}(k \mid \mathbf{X}_{k\star}) = -\frac{1}{2} \sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} (\mathbf{x}_i - \bar{\mathbf{x}}_k)' \mathbf{S}_k^{-1} (\mathbf{x}_i - \bar{\mathbf{x}}_k), \tag{4.27}$$

$$\mathrm{d}^{\text{joint}}_{\text{LINEAR}}(k \mid \mathbf{X}_{k\star}) = \bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\text{pooled}} \left(\sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) - \frac{n_\mathrm{a}}{2} \bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\text{pooled}} \bar{\mathbf{x}}_k. \tag{4.28}$$

Given the n_a leakage traces $\mathbf{x}_i \in \mathbf{X}_{k\star}$, d_{LOG} and d_{MD} require time $O(n_\mathrm{a} m^2)$ while $\mathrm{d}_{\text{LINEAR}}$ only requires $O(n_\mathrm{a} m + m^2)$, since the multiplications $\bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\text{pooled}}$ and $\bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\text{pooled}} \bar{\mathbf{x}}_k$ only need to be done once, which is a great advantage in practice. As a practical example, my evaluations of the guessing entropy (see Section 4.6) for $m = 125$ and $n_\mathrm{a} \in \{1, 2, \ldots, 10, 20, \ldots, 100, 200, \ldots, 1000\}$ took about 3.5 days with d_{LOG}, but only 30 minutes with $\mathrm{d}_{\text{LINEAR}}$.[3]

Notice that for $\mathrm{d}_{\text{LINEAR}}$ the computation time is the same regardless of which option we use to combine the traces, and both give the same results for the template attack. This is because if we let $c_k = -\frac{1}{2} \bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\text{pooled}} \bar{\mathbf{x}}_k$ for any k, then we have

$$\mathrm{d}^{\text{joint}}_{\text{LINEAR}}(k \mid \mathbf{X}_{k\star}) = \bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\text{pooled}} \left(\sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) + n_\mathrm{a} c_k, \tag{4.29}$$

$$\mathrm{d}^{\text{avg}}_{\text{LINEAR}}(k \mid \mathbf{X}_{k\star}) = \bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\text{pooled}} \left(\frac{1}{n_\mathrm{a}} \sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) + c_k, \tag{4.30}$$

and therefore for any $u, v \in \mathcal{S}$ it is true that

$$\begin{aligned}
\mathrm{d}^{\text{avg}}_{\text{LINEAR}}(u \mid \mathbf{X}_{k\star}) &> \mathrm{d}^{\text{avg}}_{\text{LINEAR}}(v \mid \mathbf{X}_{k\star}) \Leftrightarrow \\
\bar{\mathbf{x}}_u' \mathbf{S}^{-1}_{\text{pooled}} \left(\frac{1}{n_\mathrm{a}} \sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) + c_u &> \bar{\mathbf{x}}_v' \mathbf{S}^{-1}_{\text{pooled}} \left(\frac{1}{n_\mathrm{a}} \sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) + c_v \Leftrightarrow \\
\bar{\mathbf{x}}_u' \mathbf{S}^{-1}_{\text{pooled}} \left(\sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) + n_\mathrm{a} c_u &> \bar{\mathbf{x}}_v' \mathbf{S}^{-1}_{\text{pooled}} \left(\sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) + n_\mathrm{a} c_v \Leftrightarrow \\
\mathrm{d}^{\text{joint}}_{\text{LINEAR}}(u \mid \mathbf{X}_{k\star}) &> \mathrm{d}^{\text{joint}}_{\text{LINEAR}}(v \mid \mathbf{X}_{k\star}).
\end{aligned}$$

Karsmakers et al. [58] have mentioned the use of the linear discriminant with template attacks in a technical report in 2007, before the publication of my paper at CARIDS in 2013 [26]. However, they evaluated only the linear discriminant, without commenting on

[3] MATLAB, single core CPU with 3794 MIPS.

the great advantage over the other discriminants, and did not consider Fisher's LDA as a compression method. In the following section I present an extensive evaluation of all the discriminants presented earlier, using the compression methods from Section 4.3.

4.5 Metrics for the evaluation of template attacks

There are three common metrics used to evaluate side-channel attacks: the *guessing entropy*, the *success rate*, and the *mutual information* [104]. The guessing entropy was analysed by Massey [75] and Cachin [19], and later used by Köph et al. [64] and Standaert et al. [104] in the context of side-channel attacks. In my evaluations, I used mainly the guessing entropy, as it is defined below.

4.5.1 Guessing entropy

The guessing entropy estimates the remaining cost of an optimised brute-force search for the correct $k\star$, i.e. the average number of trials needed to find $k\star$ after performing a template attack, when searching over the candidate values k in decreasing order of their discriminant score. Basically, the guessing entropy provides the average depth of the correct value $k\star$, in the sorted vector of discriminant scores. Its logarithm gives the expected number of bits of uncertainty remaining about the target value $k\star$. The lower the guessing entropy, the more successful the attack has been and the less effort remains to search for the correct $k\star$.

To compute the guessing entropy, we first need to obtain the scores $\mathrm{d}(k \mid \mathbf{X}_{k\star})$ (see Section 4.4) for each combination of candidate value k and target value $k\star$, resulting in a score matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ with $\mathbf{M}(k\star, k) = \mathrm{d}(k \mid \mathbf{X}_{k\star})$. Each row in $\mathbf{M}$ contains the score of each candidate value k given the traces $\mathbf{X}_{k\star}$ corresponding to a given target value $k\star$. Next we sort each row of $\mathbf{M}$, in decreasing order, to obtain a depth matrix $\mathbf{D} \in \mathbb{N}^{|\mathcal{S}|\times|\mathcal{S}|}$ with

$$\mathbf{D}(k\star, k) = \text{position of } \mathrm{d}(k \mid \mathbf{X}_{k\star}) \text{ in the sorted row of } \mathbf{M}(k\star, \cdot). \tag{4.31}$$

Finally, using the matrix $\mathbf{D}$ we can compute the guessing entropy[4] as

$$g = \log_2 \frac{1}{|\mathcal{S}|} \sum_{k\in\mathcal{S}} \mathbf{D}(k, k). \tag{4.32}$$

Note that it is possible to compute the guessing entropy even if we do not run the attack for all possible target values $k\star$, by computing the score matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{S}_s|\times|\mathcal{S}|}$ and corresponding depth matrix $\mathbf{D} \in \mathbb{N}^{|\mathcal{S}_s|\times|\mathcal{S}|}$ for only a subset $\mathcal{S}_s \in \mathcal{S}$ of target values. In this

[4]Standaert et al. [104] presented this measure without the logarithm.

case we obtain the *partial* guessing entropy

$$g = \log_2 \frac{1}{|\mathcal{S}_s|} \sum_{k \in \mathcal{S}_s} \mathbf{D}(k, k). \tag{4.33}$$

However, for the *Grizzly* dataset (see Section 2.3.1), using a partial guessing entropy may be misleading because some values are much easier to attack than others. For example, the value $k = 0$ is the only one with a Hamming weight 0. Therefore, in scenarios where we attack a single *fixed* value (as in the *Grizzly* dataset), we should compute the full guessing entropy. On the other hand, when our target is for example the output v of the AES S-box (see Figure 2.3), computing the guessing entropy even for a single secret key byte value $k\star$ might be fine, if the attack traces correspond to a uniformly distributed choice of values v, which are dependent on the secret key byte hypothesis k and the plaintext bytes p.

4.5.2 Guessing entropy of Hamming weight leakage

As I explained in Section 2.1.1, a common assumption is that the leakage of a device can be modeled using the Hamming weight of the target value $k\star$. Therefore, in order to compare my results on *Grizzly* (see next section), with a Hamming weight leakage, I provide below a computation of the guessing entropy for the Hamming weight leakage. For this, I assume a leakage function that leaks *exactly* the Hamming weight of $k\star$, and I use the *theoretical* definition of the guessing entropy used by Cachin in his thesis [19].

Given a random variable X having n possible outcomes, with probabilities $p_1, p_2, \ldots, p_n$ such that $p_1 \geq p_2 \geq \ldots \geq p_n$, the guessing entropy is defined as[5]:

$$\mathbb{E}[G(X)] = \sum_{i=1}^{n} i \cdot p_i, \tag{4.34}$$

which is simply the expectation (mean) of the position (i) according to the probability of each value. $G(X)$ represents the number of guesses needed to determine the value of X.

In order to use this definition with a Hamming weight leakage, we need to use a conditional probability, defining the values $p_{k|h}$ as the probability of the candidate value $k \in \mathcal{S}$ given a Hamming weight leakage with value $h \in \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$. Then, after relabeling and reordering the conditional probabilities such that $p_{1|h} \geq p_{2|h} \geq \ldots \geq p_{n|h}$, where $p_{1|h} = \arg\max_k p_{k|h}$, $p_{2|h}$ is the second largest, etc., the definition of the guessing entropy for a Hamming weight leakage h becomes:

$$\mathbb{E}[G(X \mid h)] = \sum_{i=1}^{n} i \cdot p_{i|h}. \tag{4.35}$$

[5]This definition can lead to different results than using the definition from Section 4.5.1. In the remaining of this thesis I used the definition from Section 4.5.1.

For example, given $h = 1$ we have that $p_{1|h} = p_{2|h} = \ldots = p_{8|h} = 1/8$ (corresponding to the values $k \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ having the Hamming weight $h = 1$), and $p_{9|h} = \ldots = p_{256|h} = 0$. As a result, we obtain

$$\mathbb{E}[G(X \mid h = 1)] = \sum_{i=1}^{8} i \cdot p_{i|h=1} = 4.5. \tag{4.36}$$

More generally, for a particular b-bit candidate k there are a total of

$$t_k = \left(\begin{array}{c} b \\ h_k \end{array} \right) = \frac{b!}{(b - h_k)!h_k!} \tag{4.37}$$

values with the same Hamming weight value $h_k = \mathrm{HW}(k)$. Therefore, we have that $p_{1|h_k} = p_{2|h_k} = \ldots = p_{t_k|h_k} = 1/t_k$ and $p_{t_k+1|h_k} = \ldots = p_{2^b|h_k} = 0$, so that the guessing entropy is

$$\mathbb{E}[G(X \mid h_k)] = \sum_{i=1}^{t_k} i \cdot p_{i|h_k} = \frac{t_k + 1}{2}. \tag{4.38}$$

As a result, we can compute the overall guessing entropy, as an average over all possible values k as

$$\mathbb{E}[G(X)] = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \mathbb{E}[G(X \mid h_k = \mathrm{HW}(k))] = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \frac{t_k + 1}{2}. \tag{4.39}$$

For the 8-bit value target used in my experiments, we have that $\mathbb{E}[G(X)] = 25.63$, or $\log_2 \mathbb{E}[G(X)] = 4.68$.

4.6 Attacks on a single LOAD instruction

In this section, I present the results of evaluating template attacks on the *Grizzly Beta* dataset from Section 2.3.1, where the target $k\star$ is the value processed by the second LOAD instruction, i.e. the value that gets loaded into r9.

I compare the compression methods from Table 4.1[6] and the discriminants from Section 4.4. The sample selection choices are used to compare the previously proposed guideline [92] of 1 sample per clock at most (*1ppc*), against multiple points (leakage samples) per clock: the *3ppc*, *20ppc* and *allap* selections.[7] I show these selections in Figure 4.2.

I performed each attack 10 times for each combination of n_a, k and $k\star$, using a different random selection of $\mathbf{X}_{k\star}$ for each n_a and $k\star$. In the results presented below, I plot the averaged guessing entropy, resulting in highly reproducible graphs. The standard deviation across all experiments is around 0.1 bits.

[6] Using SNR instead of DOM as the signal strength estimate $\mathbf{s}(t)$ provided very similar results.

[7] The selections *1ppc*, *3ppc* and *20ppc* provide a variable number of samples because of the additional restriction that the selected samples must be above the highest 95th percentile of $\mathbf{s}(t)$, which varies with n_p for each clock edge.

Table 4.1: List of compression methods evaluated in this chapter.

Name	**Description**	m
DOM 1ppc	DOM, 1 sample per clock at most	6–10
DOM 3ppc	DOM, 3 samples per clock at most	18–30
DOM 20ppc	DOM, 20 samples per clock at most	75–79
DOM allap	DOM, all samples above 95th percentile of F(t)	125
PCA	Fixed selection of number of principal components	4
LDA	Fixed selection of number of coefficients	4

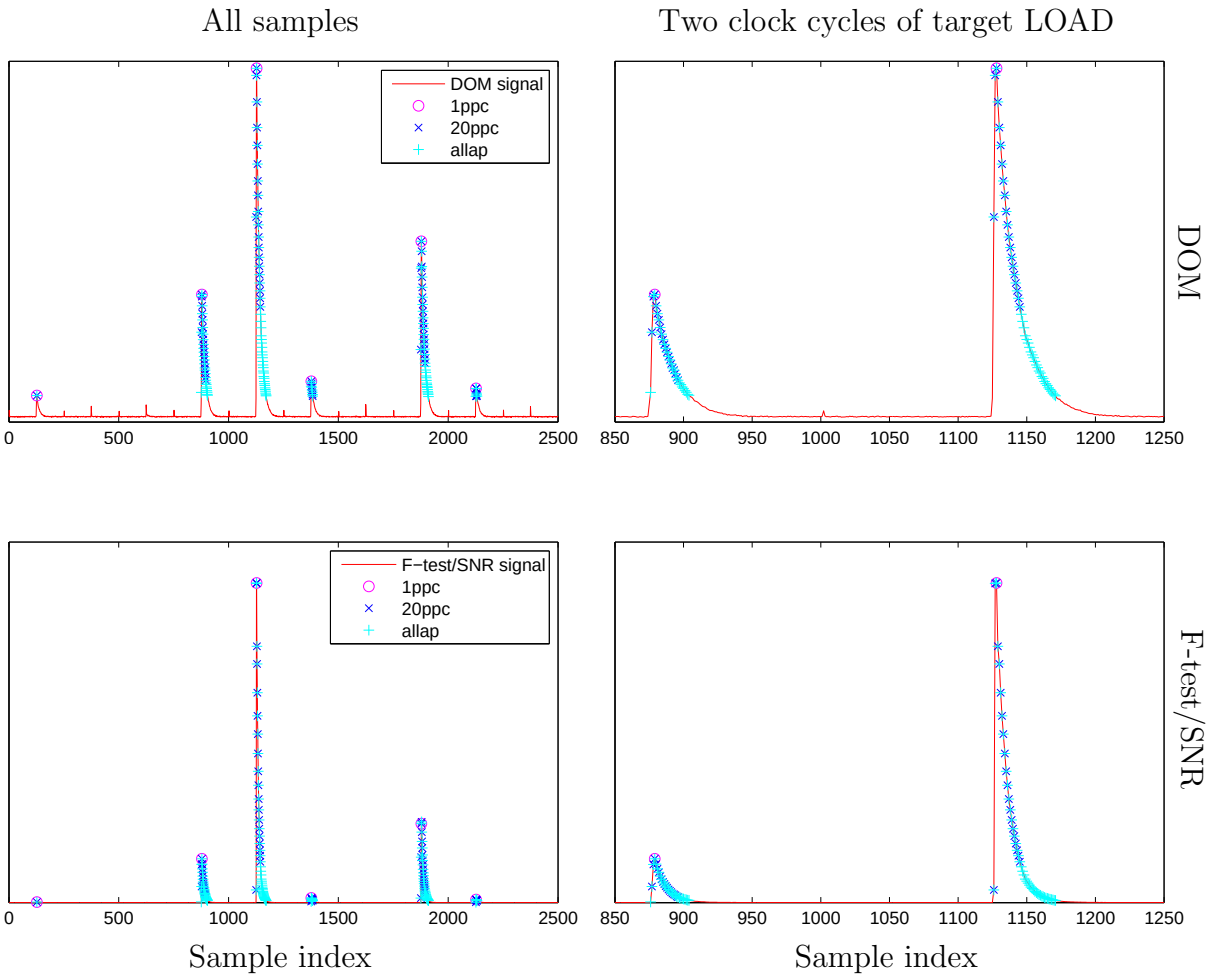

Figure 4.2: Signal strength estimates for DOM (top) and F-test (bottom), showing the sample selections *1ppc*, *20ppc*, and *allap*, for all samples (left), and for only the two clock cycles of the target LOAD instruction (right).

4.6.1 Results using individual covariances

In Figure 4.3, I show the results for $\mathrm{d}_{\mathrm{LOG}}$. We can see that using $\mathrm{d}_{\mathrm{LOG}}^{\mathrm{avg}}$ with $n_{\mathrm{p}} = 200$ and large n_{a}, the selection methods with more points perform better than the methods with fewer points, while for $\mathrm{d}_{\mathrm{LOG}}^{\mathrm{joint}}$ the opposite is true. For small n_{p}, the covariance matrix

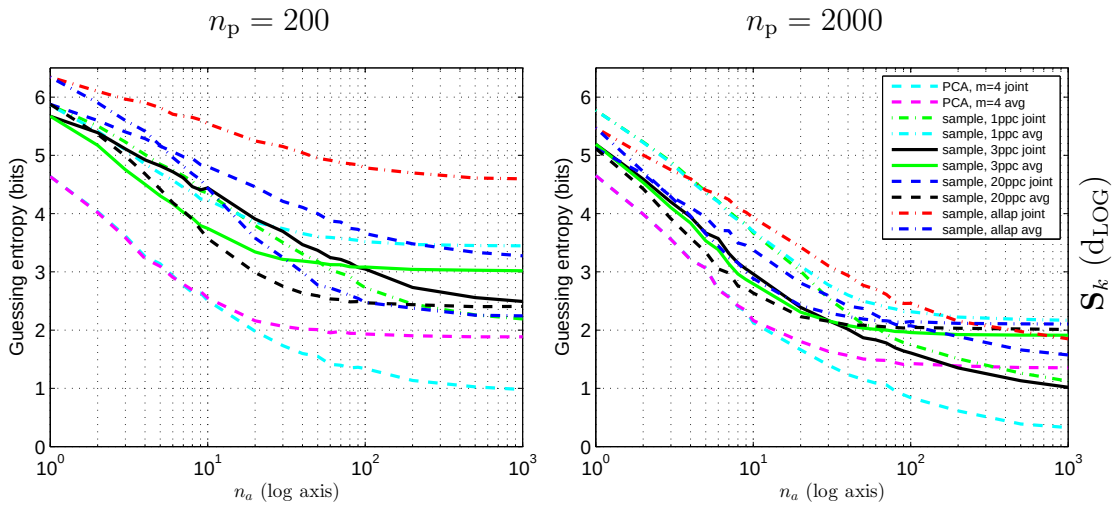

Figure 4.3: Guessing entropy remaining after template attacks, with different compressions, for $n_\mathrm{p} = 200$ (left) and $n_\mathrm{p} = 2000$ (right) profiling traces, using individual covariances $\mathbf{S}_k$ with $\mathrm{d}_{\mathrm{LOG}}$.

is not well estimated, and therefore the joint likelihood with large m will provide weak results, whereas with the average trace $\bar{\mathbf{x}}_{k\star}$ we rely more on the estimation of the mean vector and in this case having more points helps.

As expected, on the average attack trace $\bar{\mathbf{x}}_{k\star}$, PCA performs best: it discriminates along the directions of maximum variance of the means $\bar{\mathbf{x}}_k$. However, PCA also performs very well using individual attack traces, and we see that PCA with $\mathrm{d}^{\mathrm{joint}}$ provides the best results. This shows that PCA is a very efficient compression method, as it accumulates most of the information from the traces in a very small number of dimensions, which also helps to obtain a good estimate of the covariance matrix.

For $n_\mathrm{p} = 2000$, as n_a increases, the average-trace results not using PCA converge: good $\bar{\mathbf{x}}_k$ estimates lead to good discrimination, even with few samples. In this case, the methods using more samples are also better for small n_a.

On the other hand, using $\mathrm{d}^{\mathrm{joint}}_{\mathrm{LOG}}$, we see that the methods using more samples lead to similar results as the methods with fewer samples. However, the method *3ppc* has outperformed *1ppc* and for small n_a all the methods using more samples are better than *1ppc*. This happens because as n_p increases, the estimate of the covariance matrix gets better and methods with larger m start to perform better. We also observe that $\mathrm{d}^{\mathrm{joint}}_{\mathrm{LOG}}$ outperforms $\mathrm{d}^{\mathrm{avg}}_{\mathrm{LOG}}$ for large n_a. As n_p increases and $\mathbf{S}_k$ becomes very close to $\mathbf{\Sigma}_k$, we should expect this trend to become more clear.

Note also that for these figures, using individual covariances, I did not plot the results of LDA, because LDA requires the use of a common covariance, and in that case we should better use $\mathrm{d}_{\mathrm{LINEAR}}$, as shown next.

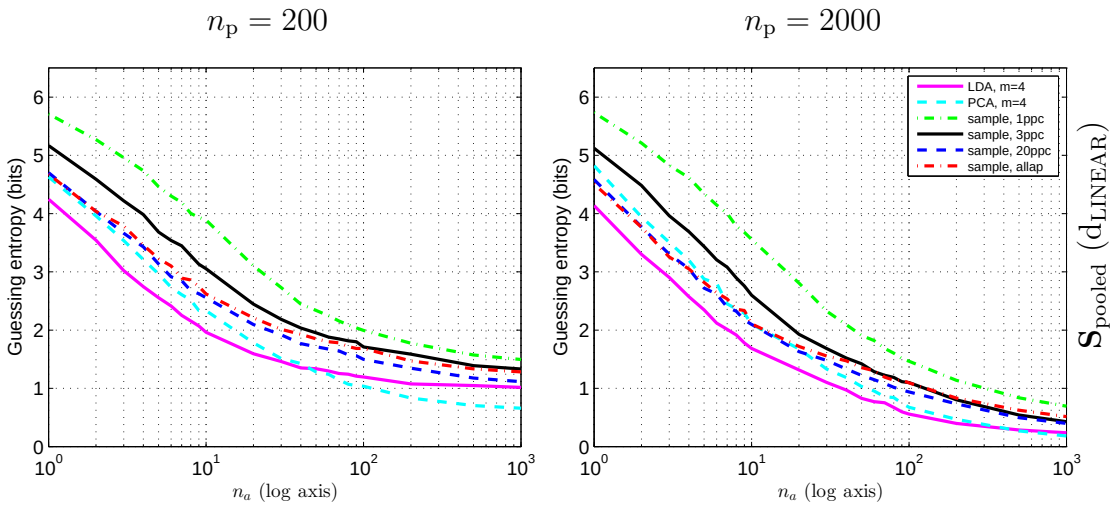

Figure 4.4: Guessing entropy remaining after template attacks, with different compressions, for $n_\text{p} = 200$ (left) and $n_\text{p} = 2000$ (right) profiling traces, using a pooled covariance $\mathbf{S}_\text{pooled}$ with d_LINEAR.

4.6.2 Results using the pooled covariance

As explained in Section 4.4.2, using a pooled covariance we can expect to obtain improved results, since in my case the covariances $\mathbf{S}_k$ are very similar and therefore we have $n_\text{p}|\mathcal{S}|$ traces for the estimation of $\mathbf{S}_\text{pooled}$. In Figure 4.4, I show the results of the template attacks for different n_p using $\mathbf{S}_\text{pooled}$. In this case, there is no difference between using the average trace and the joint likelihood, as I proved in Section 4.4.4. We can see that, even for small n_p (e.g. $n_\text{p} = 200$), the selection methods using more samples are now always better than *1ppc*. For $n_\text{a} < 10$, there is about 1 bit of entropy difference between *1ppc* and *20ppc* or *allap*. The results are very similar for $n_\text{p} = 2000$, with a few differences: (a) there is almost no difference between *20ppc* and *allap* now, as the covariance estimate for *allap* is getting close to the real value (although there seems to be room for improvement); (b) for small n_a, the methods *20ppc* and *allap* are now even better than PCA; (c) for very large n_a, PCA is still better but the difference to *20ppc* and *allap* is now very small. LDA seems to be the best method overall, although we observe that, for $n_\text{p} = 200$ and large n_a, PCA is better; this is because LDA depends also on the estimate of the common covariance matrix, while PCA does not. This might be an important difference to consider when comparing PCA and LDA.

Note that $n_\text{a} = 1$ is a very important case, as in a real scenario an attacker might be able to record only a single trace. In this case, the methods using many samples not only provide a clear advantage over *1ppc*, but they can even provide better results than PCA. The case $n_\text{a} = 1000$ is also interesting, as it represents the power of the template attack for an attacker with unlimited access to the target device. I show the success of the template attacks on these two cases for different values of n_p in Figure 4.5. In this

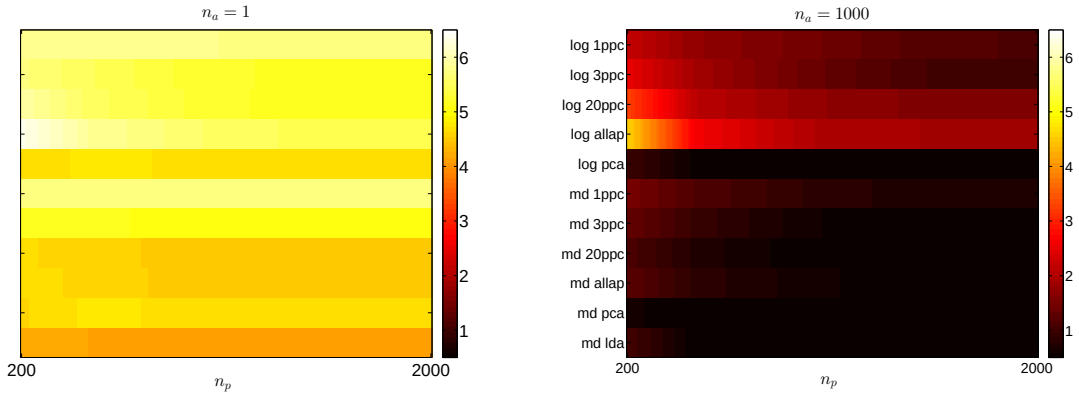

Figure 4.5: Guessing entropy from the methods discussed, for $n_\mathrm{a} = 1$ (left) and $n_\mathrm{a} = 1000$ (right), using $\mathrm{d}^{\mathrm{joint}}$ (at $n_\mathrm{p} \in \{200, 500, 1000, 1500, 2000\}$, linearly interpolated).

figure, I show only the results for using the joint likelihood, since for $n_\mathrm{a} = 1$ there is no difference to using the average trace (since there is just one trace), while for $n_\mathrm{a} = 1000$ the joint likelihood provides the best results, and as said earlier, $\mathrm{d}^{\mathrm{avg}}_{\mathrm{LINEAR}} = \mathrm{d}^{\mathrm{joint}}_{\mathrm{LINEAR}}$.

4.6.3 Practical guidance

The results and considerations presented earlier, lead to the following practical guidance regarding the choice of algorithm:

1. *Use Option 2 ($\mathrm{d}^{\mathrm{joint}}$) in preference to Option 1 ($\mathrm{d}^{\mathrm{avg}}$) to combine the discriminant scores for $n_\mathrm{a} > 1$ attack traces.* For $n_\mathrm{a} = 1$ or when using $\mathbf{S}_{\mathrm{pooled}}$, these options are equivalent. Otherwise, as the number n_a of attack traces increases and the covariance matrix is better estimated (e.g. due to a large number n_p of profiling traces or small number m of variables) $\mathrm{d}^{\mathrm{joint}}$ outperforms $\mathrm{d}^{\mathrm{avg}}$.

2. *Try using a common covariance matrix $\mathbf{S}_{\mathrm{pooled}}$ with $\mathrm{d}_{\mathrm{LINEAR}}$* (unless differences between individual estimates $\mathbf{S}_k$ are very evident, e.g. from visual inspection, or expected). Failing a statistical test for homoscedasticity (e.g. Box's test [17]) alone does not seem to imply that using individual estimates $\mathbf{S}_k$ will improve the template attack. Using individual estimates $\mathbf{S}_k$ prevents use of the significantly faster and more robust discriminant $\mathrm{d}_{\mathrm{LINEAR}}$. Then:

   (a) *If your target allows you to acquire a large number of traces ($n_\mathrm{a} > 100$):* try the compression methods PCA, LDA and sample-selection with *large* m since they may perform differently based on the level of noise from the profiling traces $\mathbf{X}_k$.

   (b) *If your target allows only acquisition of a limited number of attack traces ($n_\mathrm{a} < 10$):* use LDA. Note that in this case, as the covariance estimate improves due

to large $|\mathcal{S}|n_{\mathrm{p}}$, performance increases with larger m (cf. *3ppc*, *20ppc*, *allap*). In particular, for $n_{\mathrm{a}} < 10$, we see in Figure 4.4 that we got more than 1 bit of data from *20ppc* and *allap* compared to *1ppc*, which contradicts the claim [92, Section 3.2] that "additional [samples] in the same clock cycle do not provide additional information". In this setting, *20ppc* and *allap* can outperform PCA.

3. *If you cannot use the pooled covariance* $\mathbf{S}_{\mathrm{pooled}}$*, then use the individual covariances* $\mathbf{S}_k$ *with* $\mathrm{d}_{\mathrm{LOG}}$ *and use PCA as the compression method.*

This guidance should work well in situations similar to the *Grizzly* dataset, with the mention that the success of LDA depends very much on the ratio $n_{\mathrm{p}}/m^{\mathrm{r}}$, since for LDA we need to estimate the covariance matrix corresponding to the raw (full) traces. Although in my results I was able to use LDA successfully, providing the best results among all compression methods for low n_{a}, this technique might not be appropriate when dealing with much larger leakage traces. In that case, using PCA or sample selection might be the better choice for any n_{a}.

4.6.4 Results with a matched line

In Figure 4.6, I compare the results of template attacks when using either the active probe or a coaxial cable with 50 Ω characteristic impedance, connected to the matched line of my XMEGA PCB (see Section 2.2.2). Due to lack of amplification when using the coaxial cable, I had to reduce the vertical scale of the Tektronix oscilloscope from 10mV/div to 5mV/div. From these plots, we see that using the coaxial cable with the matched line provides better results. This is mainly because the active probe uses an operational amplifier, which introduces a considerable amount of noise. For all the experiments presented in the following I used the active probe, even if it does not provide the best results, mainly because it was not possible for the author to redo all the experiments.

Note that the methods and conclusions presented in this thesis apply whether we use an active probe or a matched line, the main difference being the noise level in the acquired traces. In practical attack scenarios we might not be able to have a matched line, due to lack of control over the target circuit, so in this respect the results from using the active probe can be regarded more general. However, if the circuit allows it, using a matched line may be a better choice, as shown in Figure 4.6.

4.6.5 Implications of my results

The results presented earlier have some important implications. Firstly, they show that, for my target microcontroller, we can almost completely determine an unknown 8-bit value manipulated by a single LOAD instruction, not just its Hamming weight. Comparing

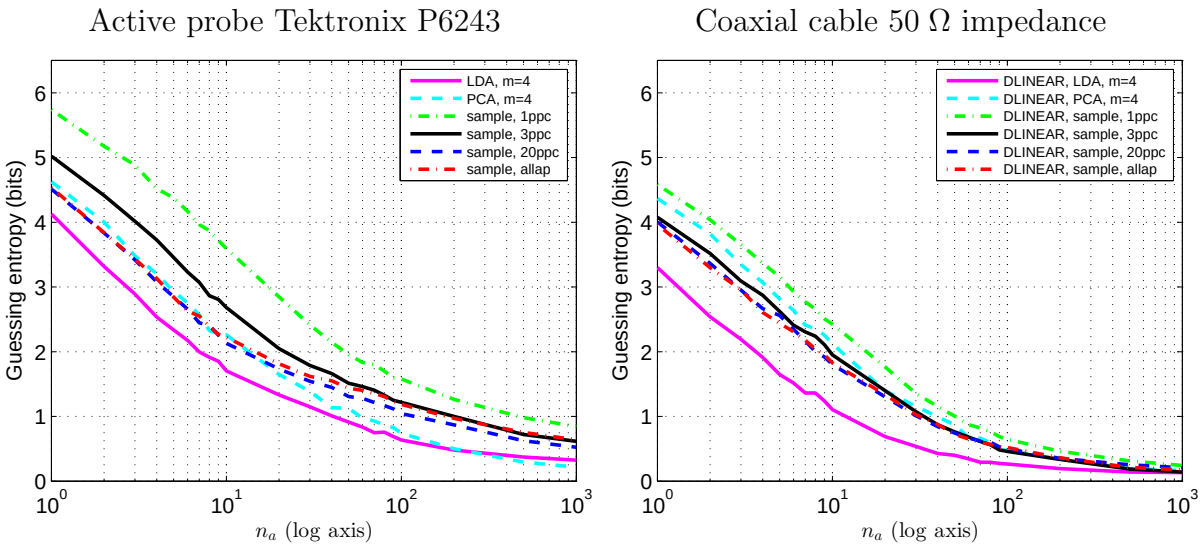

Figure 4.6: Guessing entropy after template attacks on the *Grizzly Beta* dataset, using the linear discriminant, and $n_\mathrm{p} = 1000$ profiling traces. Left: results using active probe Tektronix P6243; right: results from using a coaxial cable with 50 Ω characteristic impedance, connected to the matched line of the XMEGA PCB.

the value $\log_2 \mathbb{E}[G(X)] = 4.68$, obtained from a perfect Hamming weight leakage (see Section 4.5.2), with my results from Figures 4.4 and 4.6, we can see that using LDA with $\mathrm{d}_{\mathrm{LINEAR}}$ we can already obtain more than Hamming weight leakage even with a single attack trace. When using more than $n_\mathrm{a} = 100$ attack traces, the logarithmic guessing entropy decreases close to zero, showing a great advantage compared to the Hamming weight leakage.

A second important implication is that these results may provide one of the best published results on attacking a single data value (bus eavesdropping). A similar experiment was carried out by D. Oswald and C. Paar [83], where they applied the template attacks on some key bytes that were transferred over the bus of the Mifare DESFire microcontroller before executing an authentication protocol. In his thesis [82, Table 7.1], D. Oswald shows that the average key rank he obtains with $n_\mathrm{p} = 4000$ profiling traces and $n_\mathrm{a} = 4000$ attack traces is 3.66, which corresponds to a guessing entropy of $g = 1.87$.

A third implication is that evaluation laboratories can benefit from the computational advantage of the linear discriminant, meaning they can use this discriminant to perform the evaluations in a shorter time. This is important because evaluation laboratories typically have about 3 months to perform the entire Common Criteria evaluation of a security microcontroller, leaving only a few weeks for the evaluation of side-channel attacks [70].

4.7 Attacks on a hardware AES implementation

I now present the success of template attacks when attacking the hardware AES implementation of the XMEGA A3U 256 microcontroller, using the *Polar* dataset from Section 2.3.4. These traces cover the first 20 CPU clock cycles of an AES encryption, corresponding mainly to the first AES round. The dataset contains $N = 384000$ traces, taken while running the hardware AES encryption module with a fixed key and uniformly distributed plaintexts. For the template attacks described below I split this dataset into two disjoint sets of traces, one for profiling and one for the attack. The target of the attacks is the byte $v = \text{Sbox}(p \oplus k)$ from the first AES encryption round (see Section 2.1.2), where p and k represent the first byte of the plaintext and key, respectively.

I implemented the template attack using the linear discriminant, in a very similar manner to the attacks on the *Grizzly* dataset (see Section 4.6). However, there are a few important differences. Firstly, during the profiling step, we first need to compute the value $v = \text{Sbox}(p \oplus k)$ corresponding to each plaintext byte p, and compute the template mean vectors $\bar{\mathbf{x}}_v$ for each value of v instead of k. Similarly, the pooled covariance matrix from (4.19) must be computed using these vectors. Secondly, during the attack, for each attack trace corresponding to a *known* plaintext byte p, and each candidate key byte value k, we need to compute the S-box output value $v = \text{Sbox}(p \oplus k)$, and then obtain the linear discriminant of each k as

$$\mathrm{d}_{\text{LINEAR}}(k \mid \mathbf{x}_i) = \bar{\mathbf{x}}_v' \mathbf{S}_{\text{pooled}}^{-1} \mathbf{x}_i - \frac{1}{2} \bar{\mathbf{x}}_v' \mathbf{S}_{\text{pooled}}^{-1} \bar{\mathbf{x}}_v. \tag{4.40}$$

Thirdly, when combining multiple attack traces, we cannot first sum the attack traces, as shown in (4.28), but now we have to add the linear discriminant of *each* attack trace, obtaining

$$\mathrm{d}_{\text{LINEAR}}^{\text{joint}}(k \mid \mathbf{X}_{k\star}) = \sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \left(\bar{\mathbf{x}}_v' \mathbf{S}_{\text{pooled}}^{-1} \mathbf{x}_i - \frac{1}{2} \bar{\mathbf{x}}_v' \mathbf{S}_{\text{pooled}}^{-1} \bar{\mathbf{x}}_v \right). \tag{4.41}$$

While this computation now takes $O(n_{\text{a}} \cdot m^2)$, we can precompute the fixed factors corresponding to the value v, as shown in Section 7.4.3, and obtain a computation in $O(n_{\text{a}} \cdot m)$. Thirdly, for the computation of the guessing entropy, I only used a fixed key value k, since the target value v depends also on the plaintext bytes p.

In Figure 4.7, I plot the guessing entropy remaining after using different amounts n_{p} of profiling traces per target value v with the compression methods LDA, PCA, *1ppc*, *3ppc* and *20ppc*. Each line represents the average over 1000 experiments. Comparing these results to those from attacking a LOAD instruction (see Figure 4.4), we can make several observations. Firstly, as n_{p} increases, LDA becomes the most efficient method, followed by PCA and *3ppc*, similarly to the results on the LOAD instruction. But, even for $n_{\text{p}} = 3000$, *20ppc* is worse than *1ppc*, due to the higher level of noise from the AES engine (where many operations are done in parallel), which suggests that more profiling traces are needed for this compression option to be useful. Secondly, when attacking

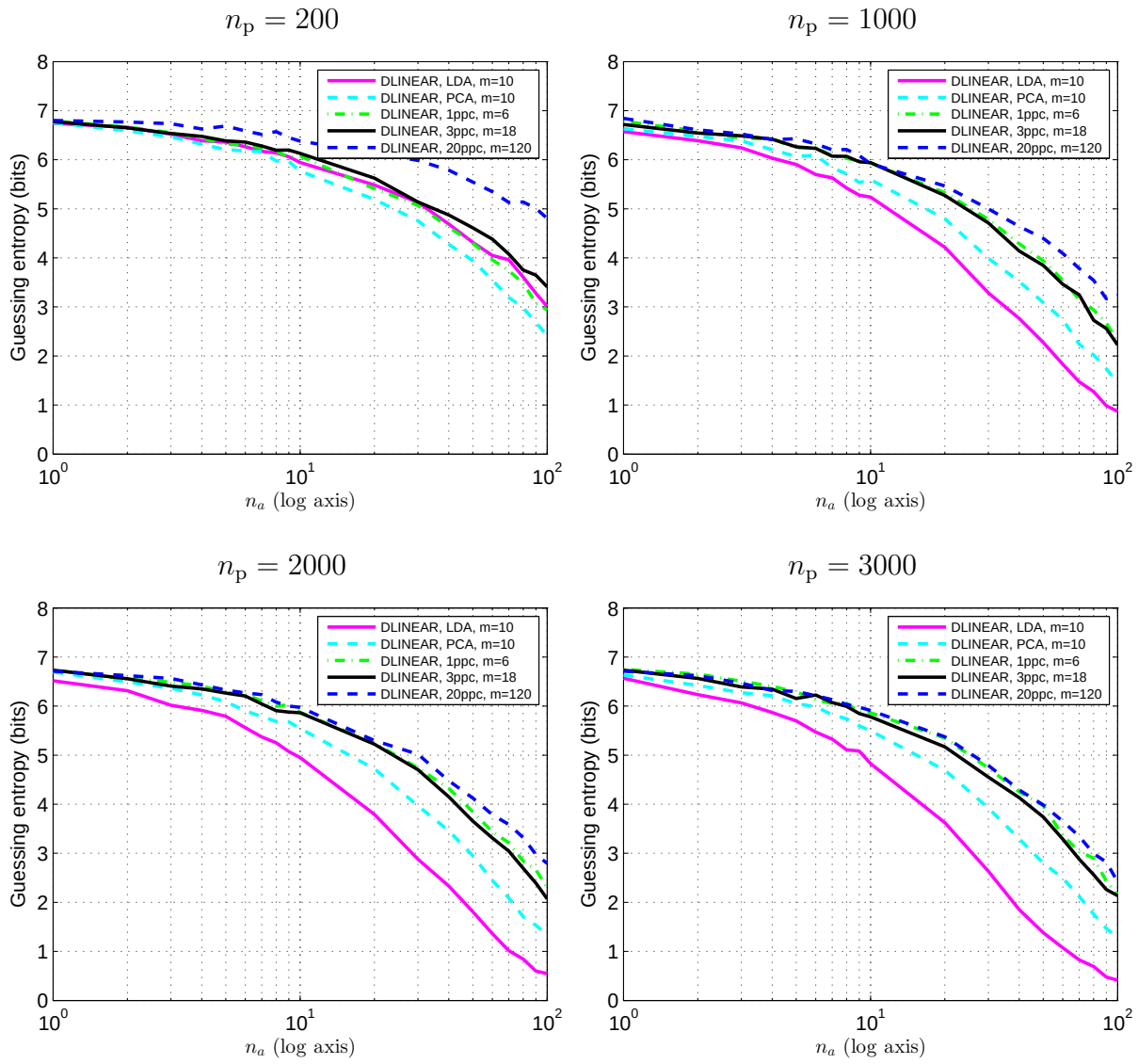

Figure 4.7: Guessing entropy after template attack on *Polar* dataset (AES engine), using different values of n_p and different compression methods.

a LOAD instruction the guessing entropy decreases abruptly whithin the first $n_a = 10$ traces and then starts to level off, but here the guessing entropy decreases steady within the attacked $n_a = 100$ traces[8]. This happens because now we target different values v, which reduces the set of possible candidates with each new attack trace (e.g. there is a single value with Hamming weight 0) and helps the attack to converge faster.

Finally, I also mention that a successful DPA attack on the same microcontroller requires about 3000 attack traces [61], while my results show that with template attacks we can recover almost perfectly a key byte with about 100 traces. This shows the important feature of template attacks: while they may require a large number of training traces (e.g. 256000 in this case), they can help to reduce the number of attack traces.

[8]Notice the logarithm on the horizontal axis in Figure 4.7.

Chapter 5

Template attacks on different devices

In this chapter, I present an extensive evaluation of the template attack, when using different devices and campaigns for the profiling and attack steps (please refer to Chapter 4, for a detailed description of template attacks).

Most publications [24, 46, 101, 104], and also my evaluations from Chapter 4, have used a single device (and possibly acquisition campaign) for the evaluation of template attacks, in order to demonstrate some particular idea. However, using the same device for profiling and attack does not represent well a real scenario, where an attacker uses a device during profiling for training, but needs to use another (the target) device during the attack.

Only recently, Renauld et al. [93] performed an extensive study on 20 physical realizations of the same electronic circuit, showing that the template attack may not work at all when the profiling and attack steps are performed on different devices; Elaabid et al. [37] showed that acquisition campaigns on the same device, but conducted at different times, also lead to worse template-attack results; and Lomné et al. [70] evaluated this scenario using electromagnetic leakage.

Throughout this chapter, I explore further the causes that make template attacks perform worse across different devices. I show that, for my experiments, a main difference across devices and acquisition campaigns is a DC offset, and this difference decreases very much the performance of template attacks (Section 5.1). To compensate for differences between devices or campaigns, I evaluate several variants of the template attack (Section 5.2). One of them needs multiple profiling devices, but provides good results when using the sample selection compression method (Section 5.2.3). However, based on investigations of Fisher's Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA), I explain how to use these two compression techniques to maximise the performance of template attacks on different devices, even when profiling on a single device (Section 5.2.4).

Overall, my results show that a good choice of compression method and parameters can dramatically improve template attacks across different devices or acquisition campaigns. Previous studies [93, 37] may have missed this by evaluating only one compression method.

Most of the content in this chapter has been published in the following conference paper:

> Omar Choudary and Markus G. Kuhn. *Template Attacks on Different Devices*, COSADE 2014, Paris, 14–15 April 2014, LNCS 8622, pp. 179–198 [27].

5.1 Ideal vs real scenario

Throughout this chapter, in order to evaluate the template attacks in different conditions, I use all the *Grizzly* datasets described in Section 2.3.1, and the compression methods from Table 4.1 (see also Section 4.3), targeting the value $k\star$ processed by the second LOAD instruction. For PCA and LDA, I often used $m = 4$, based on observation of the eigenvalues (see Section 3.9.1), but in Section 5.2.4 I shall provide a better method for selecting the components of PCA and LDA. All the evaluations are based on the guessing entropy, defined in Section 4.5.1, computed using the linear discriminant score from Section 4.4.3. Please refer to the previous chapter, Section 4.1, for a detailed description of the template attack. The standard implementation of this attack, which I shall refer to as the *standard* method in this chapter, is summarised in Algorithm 1.

Algorithm 1 (Standard)

1: Obtain the n_p leakage traces in $\mathbf{X}_k$ from the profiling device, for each k.
2: Compute the template parameters $(\bar{\mathbf{x}}_k, \mathbf{S}_\mathrm{pooled})$ using (4.1,4.19).
3: Obtain the leakage traces $\mathbf{X}_{k\star}$ from the attacked device.
4: Compute the guessing entropy as described in Section 4.5.1.

In an ideal scenario, as used in many publications [24, 46, 101, 104, 26], the attacker can use the the same device and campaign for the profiling and attack steps of the template attack. The results of Algorithm 1 in this ideal case are shown in Figure 5.1 (top-left). We can see that most compression methods perform very well for large n_a, while for smaller n_a, LDA is generally the best. This is what I have shown previously in Section 4.6.

However, in a more realistic scenario, an attacker who wants to infer some secret data from a target device may be forced to use a different device for profiling. Indeed, there are situations where we could use non-profiled attacks, such as DPA [62], CPA [18], or MIA [45], to infer secret data using a single device (e.g. by targeting values that represent a known relationship between key and plaintext). But these methods cannot be used in more general situations, where we want to infer a single secret data value that does not depend on any other values, which is the setting of our experiments. In such cases, the template attacks or the stochastic approach (see Chapter 7) might be the only viable side-channel attack.[1] Moreover, the template attacks are expected to perform better than

[1] In our setting we cannot use the non-profiled stochastic method (termed *on-the-fly* attacks by Renauld et al. [93]) either, because our attacker only has data dependent on the target secret value.

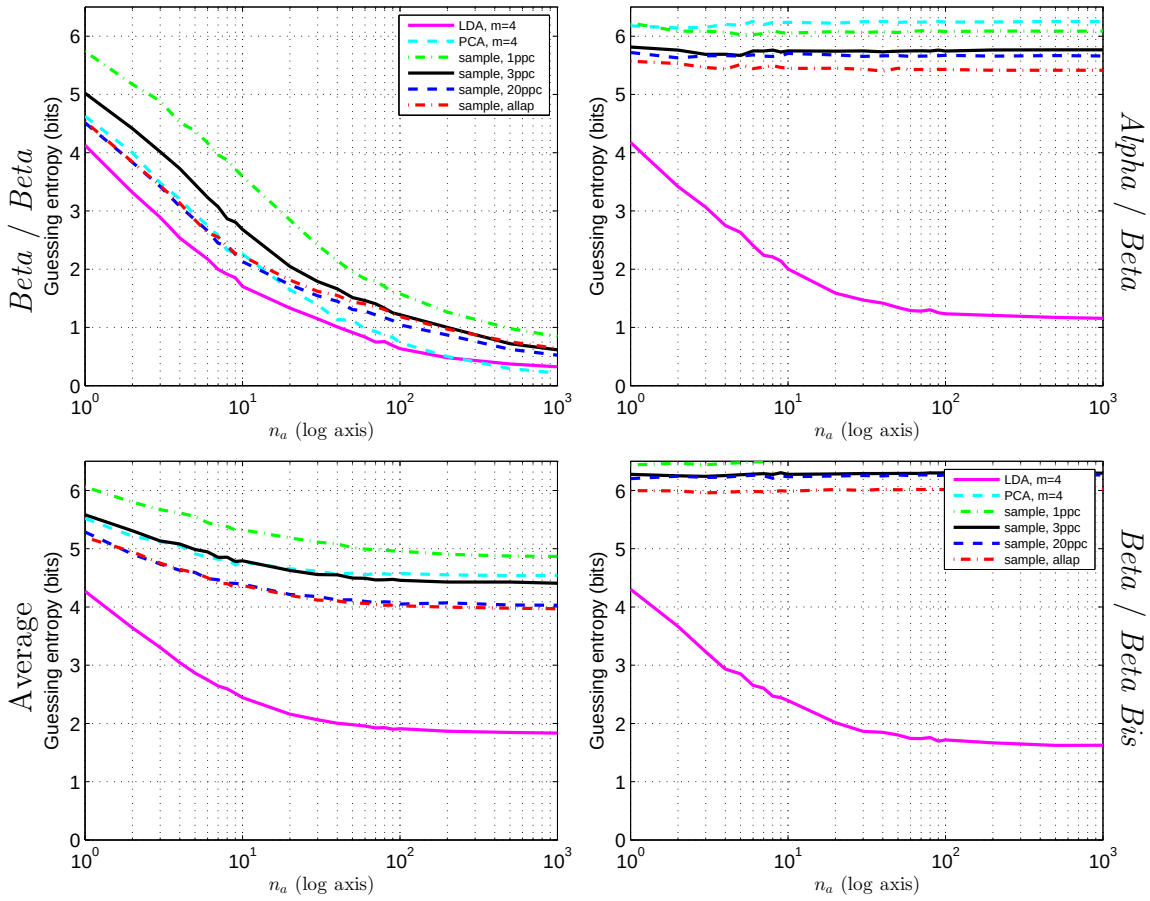

Figure 5.1: Template attacks using Algorithm 1 in different scenarios. Top-left (ideal): using same device and acquisition campaign (*Beta*) for profiling and attack. Top-right: using *Alpha* for profiling and *Beta* for attack. Bottom-left: arithmetic average of guessing entropy over all combinations of different pairs of devices for profile and attack. Bottom-right: using same device (Beta) but different acquisition campaigns for profile (*Beta*) and attack (*Beta Bis*).

the other attacks, when provided with enough profiling data [103]. Therefore, we would like to use template attacks also with different devices for profiling and attack.

As I show in Figure 5.1 (top-right), the efficacy of template attacks using the standard Algorithm 1 drops dramatically when using different devices for the profiling and attack steps. This was also observed by Renauld et al. [93], by testing the success of template attacks on 20 different devices with 65 nm CMOS transistor technology. Moreover, Elaabid et al. [37] mentioned that even if the profiling and attack steps are performed on the same device but on different acquisition campaigns we will also observe weak success of the template attacks. In Figure 5.1 (bottom-right), I confirm that indeed, even when using the same device but different acquisition campaigns (same acquisition settings), we can get results as bad or even worse as when using different devices. In Section 5.2, I offer an explanation for why LDA can perform well across different devices.

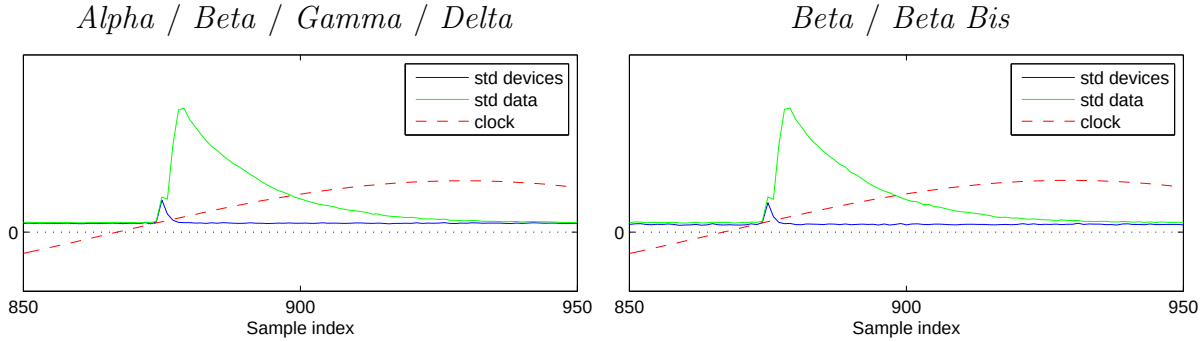

Figure 5.2: *std devices*(j) and *std data*(j), along with clock signal for a selection of samples around the first clock cycle of our target LOAD instruction. Left: using the four campaigns on different devices. Right: using Beta and Beta Bis.

5.1.1 Causes of trouble

In order to explore the causes that lead to worse attack performance on different acquisition campaigns, let's start by looking at two measures of standard deviation (std), that I call *std devices* (average deviation of same data across devices) and *std data* (average deviation of data per device).

Let $\bar{x}_{kj}^{(i)}$ be the mean value of sample $j \in \{1, \ldots, m\}$ for the candidate $k \in \mathcal{S}$ on the campaign $i \in \{1, \ldots, n_c\}$, $\bar{x}_j^{(i)} = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \bar{x}_{kj}^{(i)}$, $\tau_k^{(i)}(j) = (\bar{x}_{kj}^{(i)} - \bar{x}_j^{(i)})$ is the *treatment* of k for campaign i at sample j (see also Section 3.8.1), and $\bar{\tau}_k(j) = \frac{1}{n_c} \sum_{i=1}^{n_c} \tau_k^{(i)}(j)$. Then,

$$\text{std devices}(j) = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \sqrt{\frac{1}{n_c - 1} \sum_{i=1}^{n_c} \left(\tau_k^{(i)}(j) - \bar{\tau}_k(j)\right)^2}, \tag{5.1}$$

and

$$\text{std data}(j) = \frac{1}{n_c} \sum_{i=1}^{n_c} \sqrt{\frac{1}{|\mathcal{S}| - 1} \sum_{k \in \mathcal{S}} \left(\bar{x}_{kj}^{(i)} - \bar{x}_j^{(i)}\right)^2}. \tag{5.2}$$

I show these values in Figure 5.2. The results on the left plot are from the $n_c = 4$ campaigns on different devices (*Grizzly Alpha*, *Grizzly Beta*, *Grizzly Gamma*, and *Grizzly Delta*), while the results on the right plot are from the $n_c = 2$ campaigns on the device Beta (*Grizzly Beta* and *Grizzly Beta Bis*). We can observe that both plots are very similar, which suggests that the differences between campaigns are not entirely due to different devices being used, but largely due to different sources of noise (e.g. temperature, interference, etc.) that may affect in a particular manner each acquisition campaign. Using a similar type of plots, Renauld et al. [93, Fig. 1] observed a much stronger difference, attributed to physical variability. Their observed differences are not evident in my experiments, possibly because my devices use a larger transistor size (around 0.12 µm)[2].

[2]See `http://www.avrfreaks.net/?name=PNphpBB2&file=viewtopic&p=976590`.

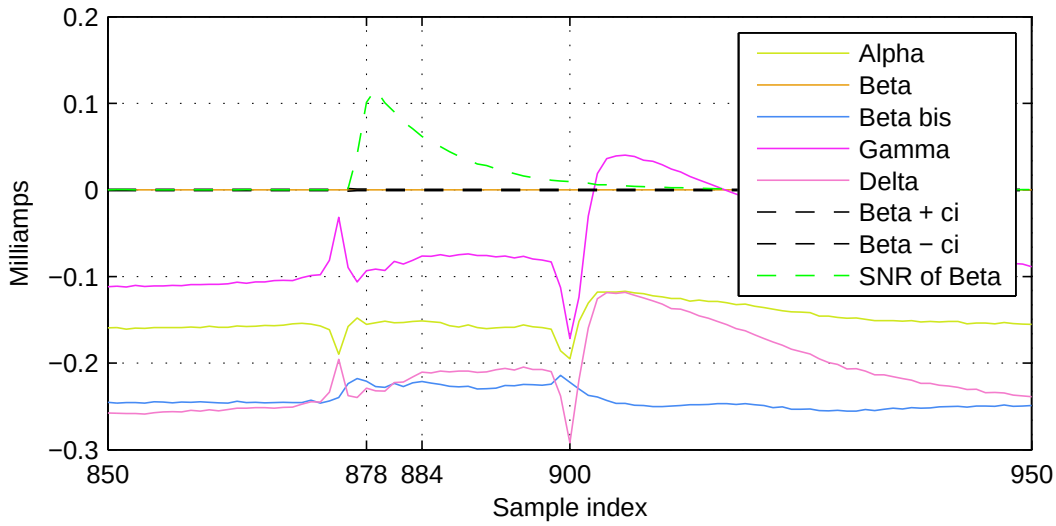

Figure 5.3: Overall mean vectors $\bar{\mathbf{x}}$ for all campaigns, from which the overall mean vector of Beta was subtracted. *Beta+ci* and *Beta−ci* represent the confidence region ($\alpha = 0.05$) for the overall mean vector of Beta. *SNR of Beta* is the Signal-to-Noise signal strength estimate of Beta (rescaled). Samples at first clock cycle of target LOAD instruction.

5.1.2 How it differs

Next, let's look at how the overall power consumption differs between acquisition campaigns. In Figure 5.3, I show the overall mean vectors $\bar{\mathbf{x}} = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \bar{\mathbf{x}}_k$ for each campaign, from which I removed the overall mean vector of Beta (hence the vector for Beta is 0). From this figure, we see that all overall mean vectors $\bar{\mathbf{x}}$ (except the one for Beta) are far outside the confidence region of Beta (for a significance level $\alpha = 0.05$). Moreover, we see that the overall mean vector $\bar{\mathbf{x}}$ for Beta Bis is the most distant from the overall mean vector of Beta. This confirms my previous assumption, that the main difference between acquisition campaigns is caused by campaign-dependent factors, such as temperature drift, environmental noise, etc. and not necessarily by the use of different devices. A similar observation was made by Elaabid et al. [37], however they used different setups for the different campaigns on the same devices. In my experiments, I used the same setup for the acquisition of data, while replacing only the tested device (evaluation board).

It is clear from Figure 5.3, that a main difference between the different campaigns is an overall offset. We see that this is particularly the case over the samples corresponding to the highest SNR. If we now look at the distributions of the data, as shown in Figure 5.4 for Alpha and Beta, we observe that the distributions are very similar (in particular the ordering of the different candidates k), but differ mainly by an overall offset. This suggests that, for my experiments, this offset is a main reason why template attacks perform badly when using different campaigns for the profiling and attack steps.

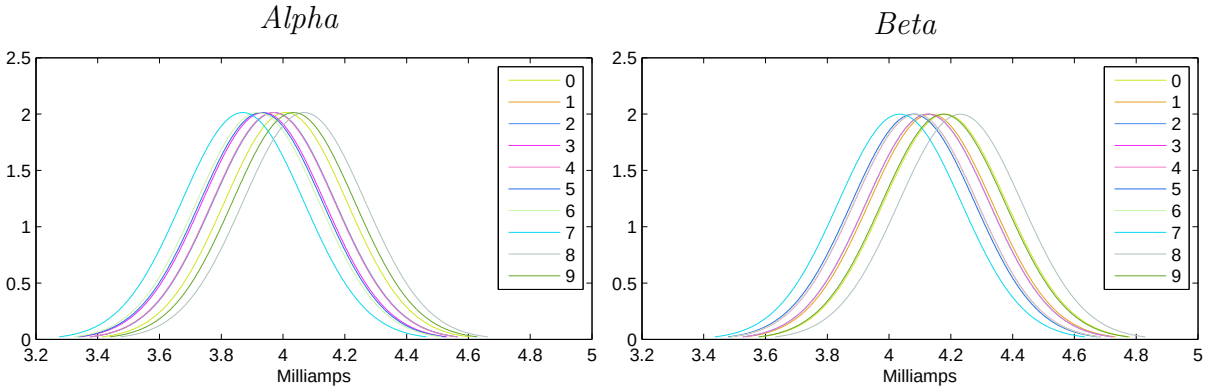

Figure 5.4: Normal distribution at sample index $j = 884$ based on the template parameters $(\bar{\mathbf{x}}_k, \mathbf{S}_{\text{pooled}})$ for $k \in \{0, 1, \ldots, 9\}$. Left: on Alpha. Right: on Beta.

5.1.3 Misalignment

I also mention that, in some circumstances, the recorded traces might be misaligned, e.g. due to lack of a good trigger signal, or random delays introduced by some countermeasure. In such cases, we should first apply a resynchronisation method, such as those proposed by Homma et al. [53]. In my experiments, I used a very stable trigger, as shown by the exact alignments of sharp peaks in Figure 5.3, so I did not preprocess the traces.

5.2 Improved attacks on different devices

In this section, I present several methods to improve the success of template attacks, when using different devices or campaigns for the profiling and attack steps. I assume that the attacker can profile well a particular device or set of devices, i.e. he can get a large number n_{p} of traces for each candidate k, but needs to attack a different device for which he only has access to a set of n_{a} traces for a particular unknown target value $k\star$. Unless otherwise mentioned, in the following evaluations I considered all possible combinations of the campaigns *Grizzly Alpha*, *Grizzly Beta*, *Grizzly Gamma* and *Grizzly Delta*, always ensuring that the campaign of one device is only used in either the profiling or attack phases, but not in both.

5.2.1 Profiling on multiple devices

Renauld et al. [93] proposed to accumulate the sample means $\bar{\mathbf{x}}_k$ and variances s_{jj} (where $\mathbf{S}$ can be either $\mathbf{S}_k$ or $\mathbf{S}_{\text{pooled}}$) of each sample x_j across multiple devices in order to make the templates more robust against differences between different devices. That is, for each candidate k and sample j, and given the sample means $\bar{\mathbf{x}}_k$ and covariances $\mathbf{S}$ from n_{c}

training devices, they compute the robust sample means as

$$\bar{x}_{kj}^{(\text{robust})} = \frac{1}{n_\text{c}}(\bar{x}_{kj}^{(1)} + \cdots + \bar{x}_{kj}^{(n_\text{c})}) \tag{5.3}$$

(i.e. an average over the sample means of each device), and the robust variance as

$$s_{jj}^{(\text{robust})} = s_{jj}^{(1)} + \frac{1}{n_\text{c} - 1} \sum_{i=1}^{n_\text{c}} \left(\bar{x}_{kj}^{(i)} - \bar{x}_{kj}^{(\text{robust})}\right)^2 \tag{5.4}$$

(i.e. they add the variance of one device to the variance of the sample means across devices, using simulated univariate noise for each device). However, this approach does not consider the correlation between samples or the differences between the covariances of different devices. Therefore, I instead use Algorithm 2, where I use the traces from all available campaigns.

Algorithm 2 (Robust Templates from Multiple Devices)

1: Obtain the leakage traces $\mathbf{X}_k^{(i)}$ from each profiling device $i \in \{1, \ldots, n_\text{c}\}$, for each k.
2: Pull together the leakage traces of each candidate k from all n_c devices into an overall leakage matrix $\mathbf{X}_k^{(\text{robust})} \in \mathbb{R}^{n_\text{p} n_\text{c} \times m}$ composed as

$$\mathbf{X}_k^{(\text{robust})\prime} = [\mathbf{X}_k^{(1)\prime}, \ldots, \mathbf{X}_k^{(n_\text{c})\prime}]. \tag{5.5}$$

3: Compute the template parameters ($\bar{\mathbf{x}}_k$,$\mathbf{S}_\text{pooled}$) using (4.1,4.19) on $\mathbf{X}_k^{(\text{robust})}$.
4: Obtain the leakage traces $\mathbf{X}_{k^\star}$ from the attacked device.
5: Compute the guessing entropy as described in Section 4.5.1.

In my evaluation of Algorithm 2, I used the data from the campaigns on the four devices (Alpha, Beta, Gamma, Delta), by profiling on three devices and attacking the fourth. The results are shown in Figure 5.5. We can see that, on average, all the compression methods perform better than using Algorithm 1 (compare Figures 5.1 and 5.5, bottom-left). This is because, with Algorithm 2, the pooled covariance $\mathbf{S}_\text{pooled}$ captures noise from many different devices, allowing more variability in the attack traces. However, the additional noise from different devices also has the negative effect of increasing the variability of each leakage sample [93, Fig. 4]. As a result, we can see that for the attacks on Beta, LDA performs better when profiling on a single device (Alpha) than when using three devices (compare Figures 5.1 and 5.5, top-right).

5.2.2 Compensating for the offset

In Section 5.1.2, I showed that a main difference between acquisition campaigns and devices is a constant offset between the overall mean vector $\bar{\mathbf{x}}$. Therefore, we expect that a template attack that removes this offset should provide better results. Elaabid

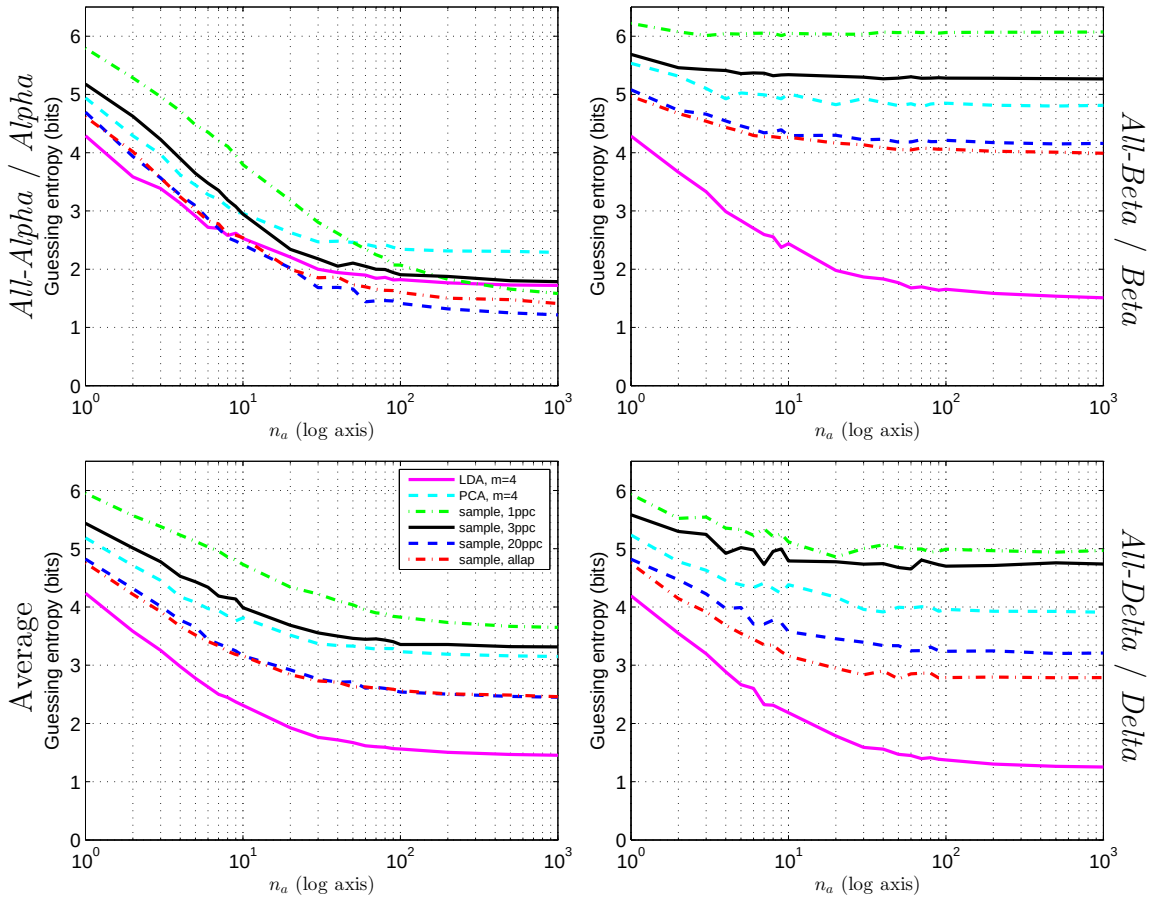

Figure 5.5: Results of Algorithm 2, profiling on three devices and attacking the fourth one. Top-left: attack on Alpha; top-right: attack on Beta; bottom-left: arithmetic average of guessing entropy over all four combinations; bottom-right: attack on Delta.

et al. [37] have shown that, indeed, if we replace each trace from each campaign by the difference between itself and the overall mean $\bar{\mathbf{x}}$ of that campaign (they refer to this process as *normalisation*, and this process may also include division by the overall standard deviation), we can obtain results very similar to those in the ideal case (profiling and attack on the same campaign). However, this approach does not work straight away in a more realistic scenario, in which the attacker only has access to a limited number n_a of traces from the target device, for a particular target value $k\star$, and hence he cannot compute the overall mean $\bar{\mathbf{x}}$ of the campaign. Nevertheless, if the difference between campaigns is mainly a constant overall offset (as I showed in Section 5.1.2), then an attacker may still use the subset of available attack traces $\mathbf{X}_{k\star}$ to improve the template attack. I describe such method in Algorithm 3.[3]

Note that, instead of Algorithm 3, we could also compensate for the offset $(\mathrm{c}^{(\mathrm{attack})} - \mathrm{c}^{(\mathrm{profile})})$ in the template parameters $(\bar{\mathbf{x}}_k, \mathbf{S}_{\mathrm{pooled}})$, but that would require more compu-

[3]For the offset, I used the median of $\bar{\mathbf{x}}^{\mathrm{r(profile)}}$, since it provides a very good approximation with my data. However, for higher clock frequencies the median can become noisy, so this might not work.

Algorithm 3 (Adapt for the Offset)

1: Obtain the *raw* leakage traces $\mathbf{X}_k^{\mathrm{r}}$ from the profiling device, for each k.
2: Compress the *raw* leakage traces $\mathbf{X}_k^{\mathrm{r}}$ to obtain $\mathbf{X}_k$, for each k.
3: Compute the template parameters ($\bar{\mathbf{x}}_k$,$\mathbf{S}_{\mathrm{pooled}}$) using (4.1,4.19) on $\mathbf{X}_k$.
4: Compute the overall mean vector $\bar{\mathbf{x}}^{\mathrm{r(profile)}} = \frac{1}{|\mathcal{S}|}\sum_k \bar{\mathbf{x}}_k^{\mathrm{r}}$ from $\mathbf{X}_k^{\mathrm{r}}$.
5: Compute the constant offset $\mathrm{c}^{(\mathrm{profile})} = \mathrm{offset}(\bar{\mathbf{x}}^{\mathrm{r(profile)}}) \in \mathbb{R}$.
6: Obtain the leakage traces $\mathbf{X}_{k\star}$ from the attacked device.
7: Compute the offset $\mathrm{c}^{(\mathrm{attack})} = \mathrm{offset}(\mathbf{x}^{\mathrm{r}}) \in \mathbb{R}$ from each *raw* attack trace $\mathbf{x}^{\mathrm{r}}$ (row of $\mathbf{X}_{k\star}^{\mathrm{r}}$). As in step 5, for my data, I used the median of $\mathbf{x}^{\mathrm{r}}$.
8: Replace each trace $\mathbf{x}^{\mathrm{r}}$ (row of $\mathbf{X}_{k\star}^{\mathrm{r}}$) by $\mathbf{x}^{\mathrm{r(robust)}} = \mathbf{x}^{\mathrm{r}} - \mathbf{1}^{\mathrm{r}} \cdot (\mathrm{c}^{(\mathrm{attack})} - \mathrm{c}^{(\mathrm{profile})})$, where $\mathbf{1}^{\mathrm{r}\prime} = [1, 1, \ldots, 1] \in \mathbb{R}^{m^{\mathrm{r}}}$.
9: Apply the compression method to each of the modified attack traces $\mathbf{x}^{\mathrm{r(robust)}}$, obtaining the robust attack leakage matrix $\mathbf{X}_{k\star}^{(\mathrm{robust})}$.
10: Compute the guessing entropy as described in Section 4.5.1 using $\mathbf{X}_{k\star}^{(\mathrm{robust})}$.

tation, especially if we want to evaluate the expected success of an attacker using this method with an arbitrary number of attack traces, as I do in my evaluations. Note also that, in my evaluation, each additional attack trace improves the offset difference estimation of the attacker: the use of the linear discriminant from (4.28) implies that, as we get more attack traces, we are basically averaging the differences $(\mathrm{c}^{(\mathrm{attack})} - \mathrm{c}^{(\mathrm{profile})})$, thus getting a better estimate of this difference.

In Figure 5.6, I show the results of Algorithm 3. We can see that, on average, the results are similar to those obtained with Algorithm 2, but slightly worse. For the best case (top-right), LDA is now achieving less than 1 bit of entropy at $n_{\mathrm{a}} = 1000$, thus approaching the results on the ideal scenario (Figure 5.1, top-left). On the other hand, we also see that, for the worst case (top-left), the results are very bad, since even using LDA with $n_{\mathrm{a}} = 1000$ doesn't provide any real improvement. This large difference between the best and worst cases can be explained by looking at Figure 5.3. There, we see that the difference between the overall means $\bar{\mathbf{x}}$ of Alpha and Beta is constant across the regions of high SNR (e.g. around samples 878 and 884), while the difference between Beta and Delta varies around these samples. This suggests that, in general, there is more than a simple DC offset involved between different campaigns, and therefore this offset compensation method alone is not likely to be helpful.

We could also try to use a high-pass filter to deal with the low-frequency offset variation, but note that a simple DC block has a non-local effect, i.e. a far-away bump in the trace not related to k can affect the leakage samples that matter most. Another possibility is to use the first derivative of the signal, which ignores DC offsets. Using EM leakage does essentially that, as the commonly used H-field probes measure the change of current flowing, so it may provide better results, as reported by Lomné et al. [70].

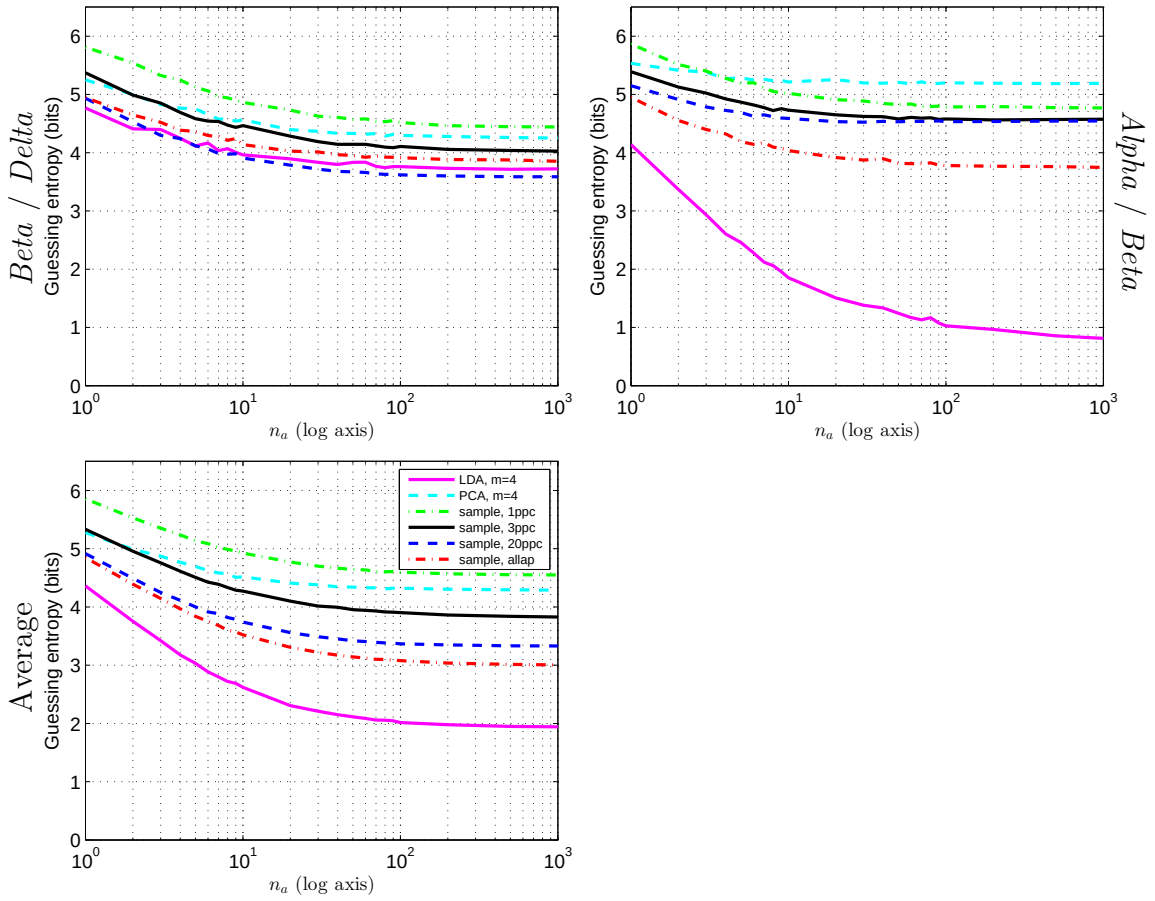

Figure 5.6: Results of Algorithm 3, profiling on one device and attacking a different device. Top-left: worst case (profiling on Beta, attack on Delta); top-right: best case (profiling on Alpha, attack on Beta); bottom-left: average over all possible 12 combinations of campaigns *Grizzly Alpha*, *Grizzly Beta*, *Grizzly Gamma*, and *Grizzly Delta*.

5.2.3 Multiple devices and offset compensation

If an attacker can use multiple devices during profiling, and since compensating for the offset may help where this offset is a main difference between campaigns, a possible option is to combine the previous two methods. This leads to Algorithm 4.

Algorithm 4 (Robust Templates and Adapt for the Offset)

1: Obtain the overall *raw* leakage matrix $\mathbf{X}_k^{r(\text{robust})}$ using Steps (1,2) of Algorithm 2.
2: Use Algorithm 3 with $\mathbf{X}_k^{r(\text{robust})}$ instead of $\mathbf{X}_k^{\text{r}}$.

The results from Algorithm 4 are shown in Figure 5.7. We can see that on average, the sample selections (in particular *20ppc*, *allap*) perform much better, when using Algorithm 4, than when using Algorithms 1–3. Furthermore, in this case the sample selection *allap* can even outperform LDA (Figure 5.7, bottom-left). This can be explained as follows: the profiling on multiple devices allows the estimation of a more robust covariance

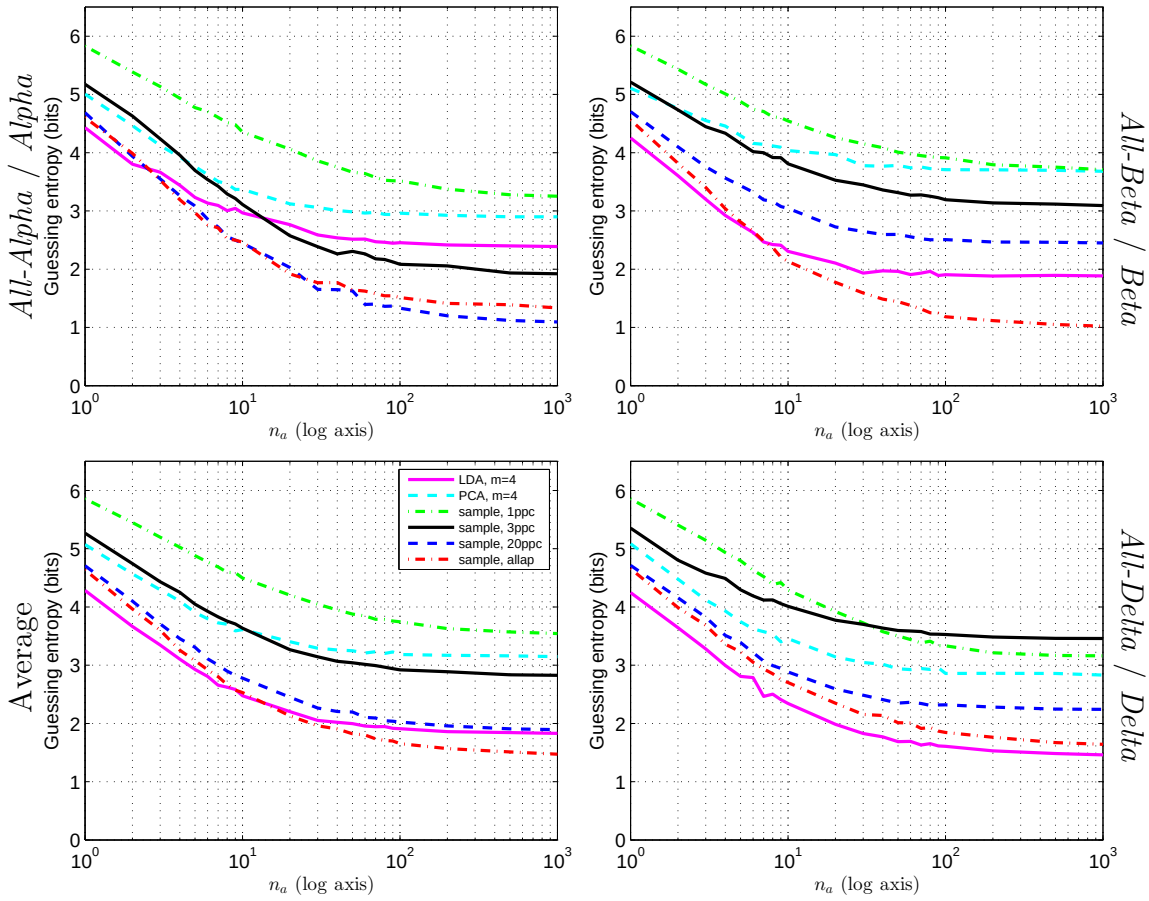

Figure 5.7: Results of Algorithm 4, profiling three devices and attacking the fourth one. Top-left: attack on Alpha; top-right: attack on Beta; bottom-left: average over all 4 combinations; bottom-right: attack on Delta.

matrix (which helps both the sample selection methods and LDA), while the offset compensation helps more the sample selection methods than LDA. We also notice that PCA still performs poorly, which was somewhat expected since the standard PCA compression method does not take advantage of the robust covariance matrix. In the following sections, I explain how to improve the template attacks, when using LDA or PCA.

5.2.4 Efficient use of LDA and PCA

In the previous sections, I showed that LDA did not benefit much from profiling on different devices or adapting the attack traces for a DC offset. In fact, using the standard Algorithm 1, LDA was already able to provide good results across different devices (see Figure 5.1). To understand why this happens, we need to look at the details of Fisher's LDA, presented in Sections 3.10 and 4.3.3. There, we can see that LDA takes into consideration the *raw* pooled covariance $\mathbf{S}_{\text{pooled}}$. Also, as I explained in Section 2.3.1, I acquired traces in batches, using random permutations of all values k per batch, and my acquisition campaigns took a few hours to complete. Therefore, the pooled covariance

$\mathbf{S}_{\text{pooled}}$ of a given campaign contains information about the different noise sources that have influenced the current consumption of my microcontrollers over the acquisition period. But one of the major sources of low-frequency noise is temperature variation (which can affect the CPU, the voltage regulator of my boards, the voltage reference of the oscilloscope, the measurement resistor; see also the study by Heuser et al. [52]), and we can expect this temperature variation to be similar within a campaign as it is across campaigns, if each acquisition campaign takes several hours. As a result, the temperature variation captured by the covariance matrix $\mathbf{S}_{\text{pooled}}$ of one campaign should be similar across different campaigns. However, the mean vectors $\bar{\mathbf{x}}_k$ across different campaigns can be different due to different DC offsets (even if the overall temperature variation is similar), and this is why the sample selection methods (e.g. *20ppc*, *allap*) perform poorly across different campaigns. Nevertheless, the LDA algorithm is able to remove the DC component and use only the rest of the trace for the attack. This, combined with the fact that with LDA we no longer need a covariance matrix after compression, allows LDA to filter out temperature variations and other noise sources that are similar across campaigns, thus providing good results even across different devices.

To see how LDA and PCA deal with the DC offset, I plot in Figure 5.8 (top) the DC components (mean) of the LDA and PCA eigenvectors. For LDA, there is a peak at the fifth DC component, which shows that my choice of $m = 4$ avoided the component with largest DC offset by chance. For PCA, there is a similar peak, also for the fifth component, and again my choice $m = 4$ avoided this component. However, for PCA this turned out to be bad, because PCA does use a covariance matrix after projection, and therefore it would benefit from knowledge of the temperature variation. This temperature variation will be given by the eigenvector with a high DC offset, and therefore we can expect that adding this eigenvector may provide better results. I also show in Figure 5.8, the first six eigenvectors of LDA ($\mathbf{S}_{\text{pooled}}^{-1}\mathbf{B}$), PCA ($\mathbf{B}$) and $\mathbf{S}_{\text{pooled}}$, along with the first ten eigenvalues of LDA and PCA. The fifth eigenvector of PCA clearly contains a DC offset (most of the eigenvector is above the dotted black line, which represents the zero value), while this is not obvious for LDA. Also, we see that the division by $\mathbf{S}_{\text{pooled}}$ in LDA has removed much of the noise found in the PCA eigenvectors, and it appears that LDA has reduced the number of components extracting most information from four (in PCA) down to three.

In Figure 5.9 (left), I show the results of template attacks when using PCA and LDA with different values of m. We see that, for LDA, there is a great gap between using $m = 4$ and $m = 5$, no gap between $m = 3$ and $m = 4$, while the gap between $m = 5$ and $m = 40$ is very small. This confirms that, with LDA, we should ignore the eigenvector containing a strong DC coefficient. Also, for PCA there is a huge gap between using $m = 4$ and $m = 5$ (in the opposite sense as with LDA), but the gap between $m = 5$ and $m = 40$ is negligible. Therefore, PCA can work well across devices, if we include the eigenvectors containing the DC offset information. These results provide an important lesson for implementing template attacks across different devices or campaigns: the choice of components should

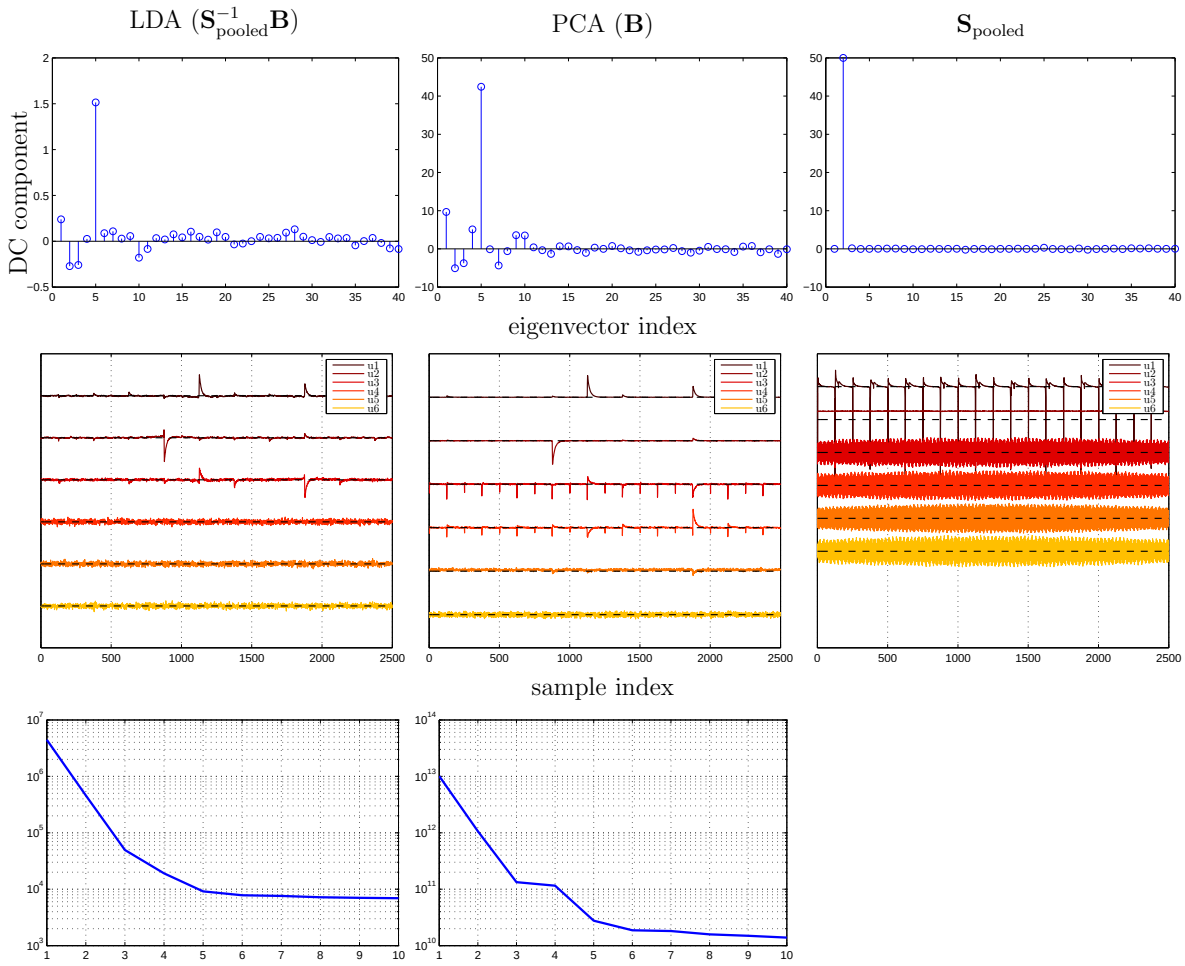

Figure 5.8: Top: DC components of eigenvectors of LDA ($\mathbf{S}_{\text{pooled}}^{-1}\mathbf{B}$), PCA ($\mathbf{B}$) and $\mathbf{S}_{\text{pooled}}$. Middle: First six eigenvectors of LDA ($\mathbf{S}_{\text{pooled}}^{-1}\mathbf{B}$), PCA ($\mathbf{B}$) and $\mathbf{S}_{\text{pooled}}$. Bottom: eigenvalues (log y axis) of LDA and PCA.

consider the DC offset contribution of each eigenvector. This suggests that previous studies may have missed important information, by using only sample selections with one to three samples [93] or only the first PCA component [37]. However, even using PCA or LDA, as proposed in these sections, may only help where the main difference between campaigns or devices is a low-frequency offset. What is the optimal method for dealing with arbitrary leakage differences between devices remains an open question.

5.2.5 Add DC offset variation to PCA

Renauld et al. [93] mentioned that *"physical variability makes the application of PCA irrelevant, as it cannot distinguish between inter-plaintext and inter-chip variances"*. While it is true that the standard PCA approach [9] is not aimed at distinguishing between the two types of variance (see Figure 5.2 for a plot of the square root of these variances), I showed in Section 5.2.4 that PCA can actually provide good results if we select the

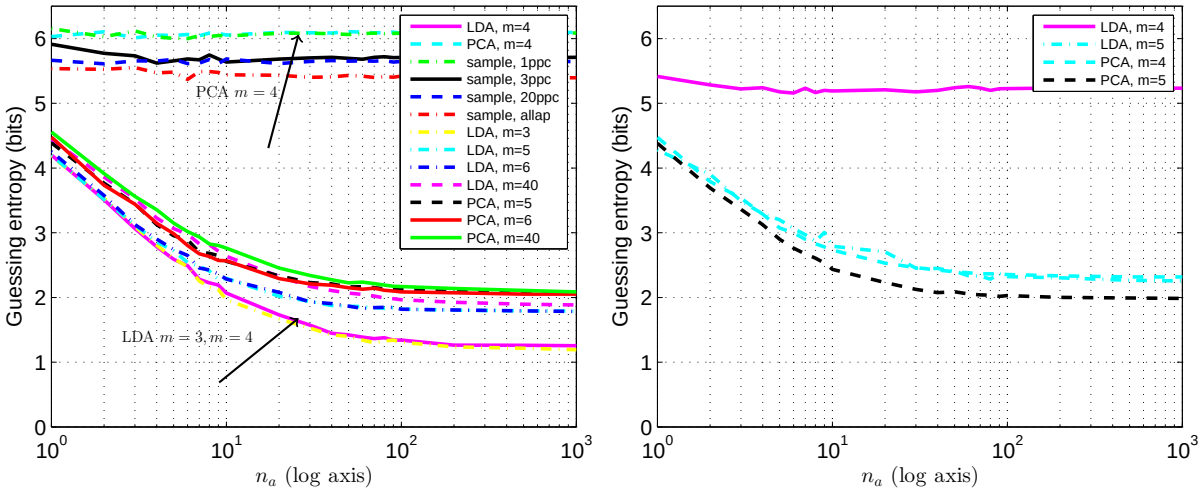

Figure 5.9: Template attack on different campaigns (profiling on Alpha, attack on Beta). Left: using various compressions with Algorithm 1. Right: using PCA and LDA with Algorithm 5.

eigenvectors carefully. Starting from this observation, we can try to enhance the PCA algorithm by deliberately adding DC noise, in the hope of concentrating the DC sensitivity in one of the first eigenvectors, thereby making the other eigenvectors less DC sensitive (as all eigenvectors are orthogonal). I present this approach in Algorithm 5.[4]

Algorithm 5 (Add Random Offsets to Matrix $\mathbf{B}$ – PCA and LDA only)

1: Obtain the *raw* leakage traces $\mathbf{X}_k^{\mathrm{r}}$ from the profiling device, for each k.
2: Obtain the pooled covariance matrix $\mathbf{S}_{\text{pooled}} \in \mathbb{R}^{m^{\mathrm{r}} \times m^{\mathrm{r}}}$.
3: Pick a random offset c_k for each *raw* mean vector $\bar{\mathbf{x}}_k^{\mathrm{r}}$.
4: Compute the between-groups matrix as
$$\mathbf{B} = \sum_{k \in \mathcal{S}} (\bar{\mathbf{x}}_k^{\mathrm{r}} - \bar{\mathbf{x}}^{\mathrm{r}} + \mathbf{1}^{\mathrm{r}} \cdot c_k)(\bar{\mathbf{x}}_k^{\mathrm{r}} - \bar{\mathbf{x}}^{\mathrm{r}} + \mathbf{1}^{\mathrm{r}} \cdot c_k)'.$$
5: Use PCA (uses $\mathbf{B}$ only) or LDA (uses both $\mathbf{B}$ and $\mathbf{S}_{\text{pooled}}$) to compress the *raw* leakage traces and obtain $\mathbf{X}_k$ for each k.
6: Compute the template parameters ($\bar{\mathbf{x}}_k$,$\mathbf{S}_{\text{pooled}}$) using (4.1,4.19).
7: Obtain the compressed leakage traces $\mathbf{X}_{k\star}$ from the attacked device.
8: Compute the guessing entropy as described in Section 4.5.1.

The results of this method are shown in Figure 5.9 (right). Now, PCA provides good results even with $m = 4$, while, for the same m, LDA gives bad results. In Figure 5.10, I show the eigenvectors from LDA and PCA, along with their DC component. We see that Algorithm 5 pushed the eigenvector having the strongest DC component first (i.e. corresponding to the largest eigenvalue), which was useful for PCA, but not for LDA. Therefore, I recommend this method only for PCA.

[4]For my implementation of Algorithm 5, I have chosen c_k uniformly from the interval $[-u, u]$, where u is the absolute average offset between the overall mean vectors shown in Figure 5.3.

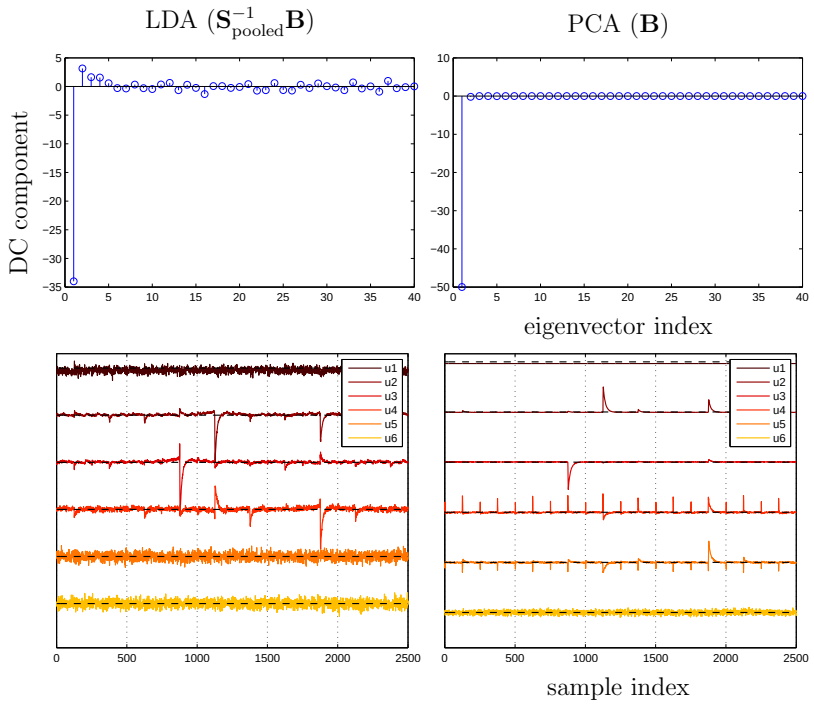

Figure 5.10: Top: DC components of eigenvectors of LDA ($\mathbf{S}_{\text{pooled}}^{-1}\mathbf{B}$) and PCA ($\mathbf{B}$) after using Algorithm 5. Bottom: First six eigenvectors of LDA ($\mathbf{S}_{\text{pooled}}^{-1}\mathbf{B}$) and PCA ($\mathbf{B}$).

Chapter 6

Correlation and factor analysis

In this chapter, I provide a first detailed analysis of the structure of the correlation between samples in side channel traces (Section 6.1.1) and of the effects of this correlation on template attacks (Section 6.1.2). I vary several acquisition parameters in order to understand their effect on the inter-sample correlation, and then I examine the effects of this correlation on template attacks.

Based on analysis of the correlation, I can observe some initial structure. Then, using factor analysis (Section 6.2), I extract the main factors that make up the correlation structure in my data, confirming my initial hypothesis on the structure of the correlation (Section 6.2.1). Furthermore, in Section 6.2.2, I show that factor analysis can be an effective method to obtain better covariance matrix estimates, which can lead to better template attacks, particularly when dealing with a large number of samples.

Some publications (e.g. [93]) have analysed the effect of individual variability (i.e. the standard deviation of each trace sample), but not the effect of the entire inter-sample correlation. Based on the factor analysis model, I am able to develop an algorithm to synthesize covariance matrices with arbitrary noise and correlation values (Section 6.3.1). Using this algorithm, I synthesize covariances based on many combinations of noise and correlation, providing an overview on the effects of these parameters on template attacks (Section 6.3.2).

Please refer to Chapter 4, for a detailed description of template attacks.

6.1 Analysis on real data

In order to explore the influence of bandwidth and power supply on the correlation between leakage samples, and on the template attacks, I used the *Koala* datasets *E1*, *E2*, *E3* and *E4*, described in Section 2.3.2. The different parameters of these datasets are shown in Table 6.1.

Table 6.1: Parameters for real experiment

Experiment name	Bandwidth	Power Supply
E1	20 MHz	lab power supply 3.3 V (TTi EL302).
E2	20 MHz	batteries via 3.3 V voltage regulator.
E3	500 MHz	lab power supply 3.3 V (TTi EL302).
E4	500 MHz	batteries via 3.3 V voltage regulator.

6.1.1 Analysis of correlation

As explained in Section 3.3.5, given a sample covariance matrix $\mathbf{S}$ (either $\mathbf{S}_k$ from (4.1) or $\mathbf{S}_{\text{pooled}}$ from (4.19)), we can obtain the corresponding *sample correlation matrix*

$$\mathbf{R} = \mathbf{D}\mathbf{S}\mathbf{D}^{-1}, \tag{6.1}$$

having the same size as $\mathbf{S}$, where $\mathbf{D}$ is the diagonal *sample standard deviation matrix*, having the standard deviation values $\sqrt{s_{jj}}$ of each leakage sample x_j on its diagonal. $\mathbf{R}$ allows us to visualise the correlation between trace samples using the standardised variables $(x_j - \bar{x}_j)/\sqrt{s_{jj}}$, which do not depend on the standard deviation (i.e. noise) of the samples. Therefore, we can obtain a meaningful comparison between the different correlation effects.

Figure 6.1 shows the sample correlation matrix $\mathbf{R}$ for each experiment. From this visualisation, we can observe that the bandwidth and power supply choices have an important effect on the correlation: for *E1* the median correlation is higher than 0.7, while this is lower for *E2*, even lower for *E3* and finally the median correlation is almost zero for *E4*. The correlation is influenced as follows. Firstly, the lab power supply introduces low-frequency noise into the side-channel, which causes significant correlation. We can see this effect by comparing experiments *E1* vs *E2* and *E3* vs *E4*. Secondly, a low acquisition bandwidth applies a low pass filter that eliminates high-frequency signals (e.g. noise), but also increases the correlation between neighboring samples significantly. We can observe this effect by comparing *E1* vs *E3* and *E2* vs *E4*.

Besides a visual examination of the sample correlation matrices $\mathbf{R}$, we can also compute the determinant $|\mathbf{R}|$, also known as the *generalized sample variance from standardized variables*. As I explain in more detail in Section 3.5, this generalized variance can be interpreted geometrically as the volume created by the standardized deviation vectors

$$\mathbf{d}_j = \begin{bmatrix} \frac{x_{1j}-\bar{x}_j}{s_{jj}} \\ \frac{x_{2j}-\bar{x}_j}{s_{jj}} \\ \vdots \\ \frac{x_{Nj}-\bar{x}_j}{s_{jj}} \end{bmatrix}. \tag{6.2}$$

A larger $|\mathbf{R}|$ corresponds to a smaller correlation: the deviation vectors $\mathbf{d}_j$ are nearly perpendicular, creating a larger volume. Therefore, we can use $|\mathbf{R}|$ as a single value

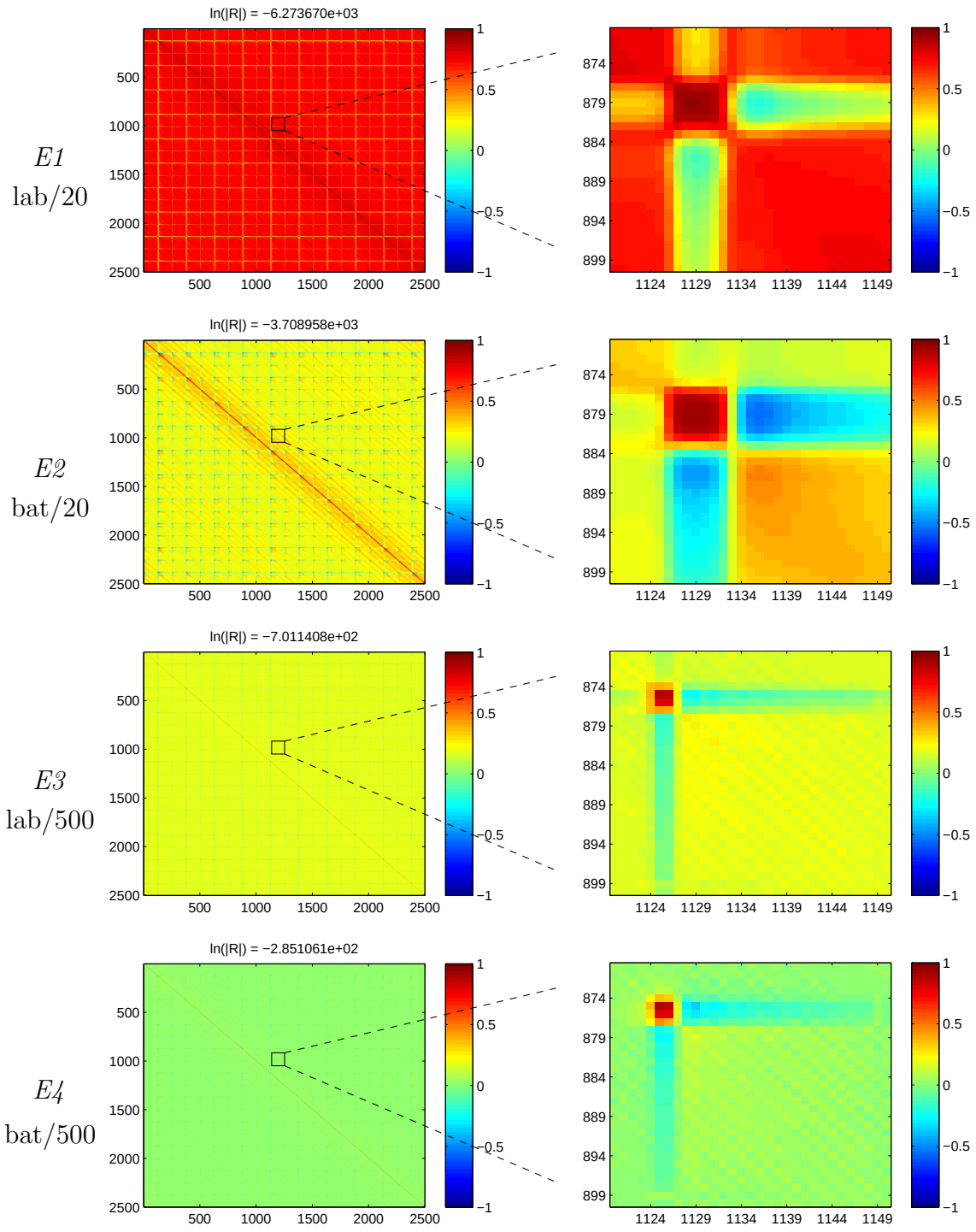

Figure 6.1: Sample correlation matrix $\mathbf{R}$ for experiments *E1* to *E4*. Left: correlation between all sample points; right: correlation between samples around first and second peaks of signal strength estimate $\mathbf{s}(t)$.

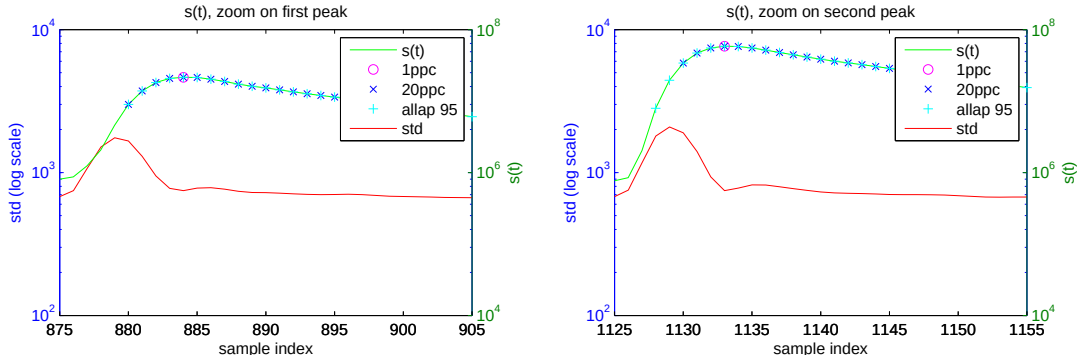

Figure 6.2: Signal strength estimate $\mathbf{s}_{\mathrm{DOM}}$, along with standard deviation for each sample points. Left: leakage samples around first clock cycle of target LOAD instruction; right: leakage samples around second clock cycle of the target LOAD instruction. Markers indicate the samples chosen using the sample-selection methods from Section 4.3.1.

to compare the correlation between all samples in our traces across different settings. Note that in practice, we should use $\log|\mathbf{R}|$ instead, since $|\mathbf{R}|$ can overflow floating point arithmetic for large m (see Section 4.2.2).

From Figure 6.1 (bottom-right), we can observe that even after removing the correlation due to bandwidth and power supply, there is still a strong correlation between the samples around the clock edges, possibly caused by trigger jitter. In Figure 6.2, I show the signal strength estimate $\mathbf{s}_{\mathrm{DOM}}(t)$ from (4.7), along with the standard deviation (*std*) for each leakage sample, for the first two clock edges. The strong correlation happens during the periods of glitching (high *std*), which is when the CMOS cells fluctuate before reaching a stable state. Comparing Figures 6.1 and 6.2, we can see that the correlation returns to a low level for the samples with high signal strength.

6.1.2 Results from template attacks

In order to understand the effects of the acquisition parameters, and indirectly of the correlation, on the template attack, I computed the guessing entropy from Section 4.5.1, using the linear discriminant from Section 4.4.3, on the *Koala* datasets *E1* to *E4* from Section 2.3.2, always targeting the value $k\star$ processed by the second LOAD instruction. In order to provide a more comprehensive view of the results, I evaluated the attacks using all the compression methods from Table 4.1, described in detail in Section 4.3.

The results of these evaluations are shown in Figure 6.3. We can make several observations. Firstly, the results using the analog low-pass filter (*E1*, *E2*) are considerably better than the results using the full bandwidth (*E3*, *E4*). This somewhat surprising result shows that, for my particular CPU (designed for a maximum clock frequency of 36 MHz, but running at 1 MHz), the high SNR obtained by using the analog low-pass filter provides much better results, even with a single sample per clock (*1ppc*), than when

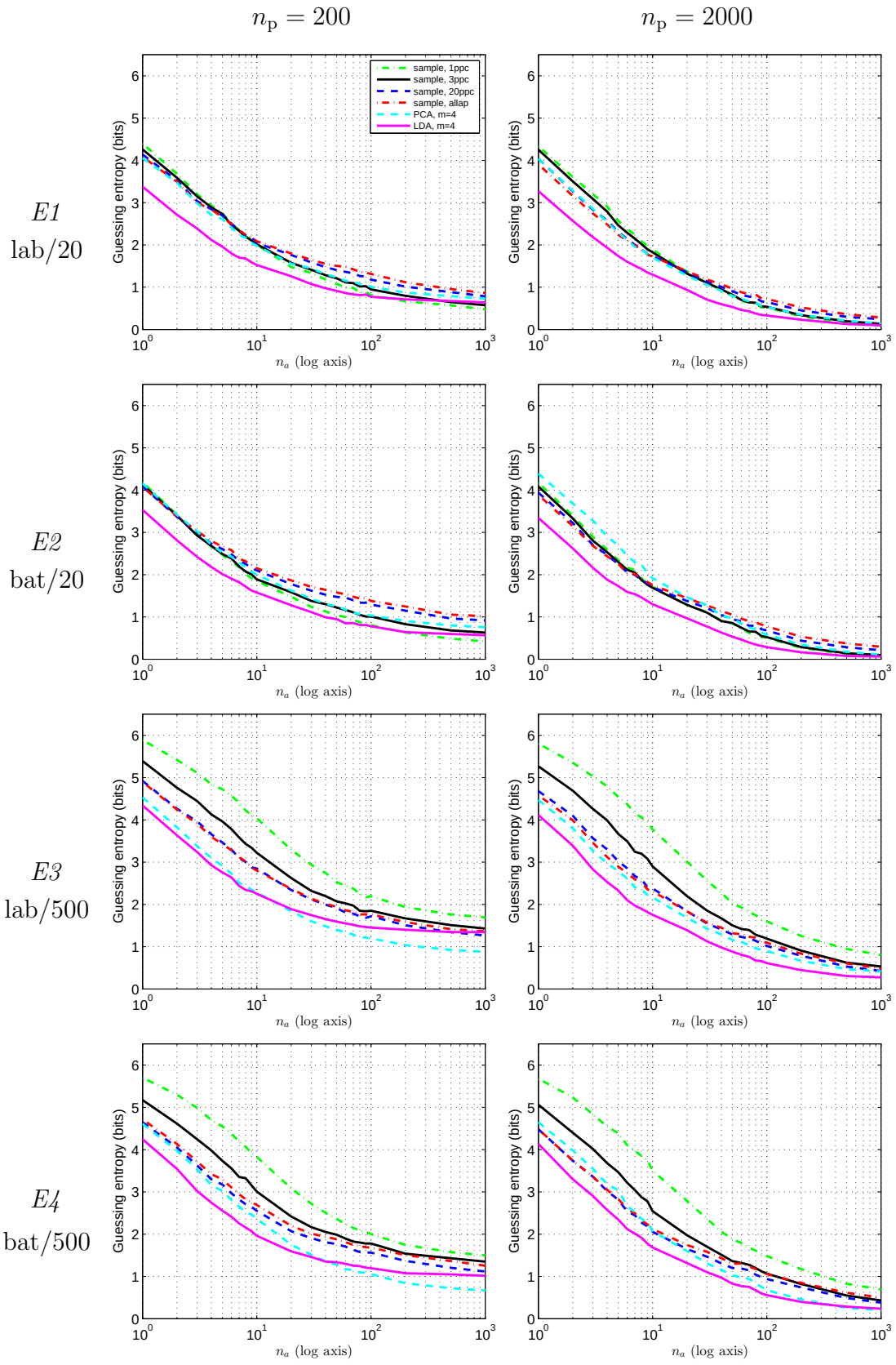

Figure 6.3: Results of template attacks on experiments *E1* to *E4*. Left: using $n_p = 200$; right: using $n_p = 2000$.

using many samples or LDA with the full bandwidth. Nevertheless, this will likely not hold in situations where there is signal at much higher frequencies. Secondly, for small n_a, using more samples (*20ppc*, *allap*) is always better than using a single sample per clock, which confirms that using a single sample per clock is not optimal in general. Thirdly, we see that for the experiments with low correlation (*E3*, *E4*) selecting more samples is vastly better than using fewer samples (in some cases as good or even better than PCA). This is expected, since the low correlation from (*E3*, *E4*) results in more information per additional sample, while for (*E1*, *E2*) there is little information that can be obtained from additional samples per clock. In general, LDA provides the best results, but if we do not have enough samples for a very good estimation of $\mathbf{S}_{\text{pooled}}$ (computed on the *raw* traces, see Section 4.3), we observe that for $n_p = 200$ and as the number n_a of attack traces increases, LDA is outperformed by *1ppc* (in the high correlation experiments *E1*, *E2*) and PCA (in the low correlation experiments *E3*, *E4*). All these results are consistent with my previous observations from Chapter 4.

Looking at the results from the left hand side of Figure 6.3, we can see another interesting aspect: LDA cannot provide better results once $n_a > n_p$. This happens because LDA does not use the covariance matrix, and therefore all the estimation is based on mean vectors only. For $n_a > n_p$, we may only observe errors due to training. That is, we cannot get a lower estimation error than what we have during profiling. The confidence region for the means is based on n_p, not n_a.

Note also, that the correlations shown in Figure 6.1 are directly linked to these results, since the covariance matrices $\mathbf{S}_k$ and $\mathbf{S}_{\text{pooled}}$, from which the correlations are computed using (6.1), are essential in template attacks (see Section 4.1). For completeness, in Figure 6.4, I plot the first five LDA eigenvectors (top) for *E1* (left) and *E4* (right), along with the first ten eigenvalues. We see that for *E1*, where there is a very strong correlation between leakage samples, the first three eigenvectors are more *noisy* than for *E4* over the entire trace, meaning that in this case LDA will use information from most of the trace. This is expected due to the high correlation. Furthermore, for *E1*, the fourth eigenvalue is almost as large as the third, meaning that the fourth eigenvector (which does not contain any peak, as the first three do) is as important as the third, confirming that in this case LDA tries to extract information from the entire trace. On the other hand, for *E4*, the third eigenvalue is much larger than the fourth, meaning that in this case the contents of the overall trace are not that useful (due to low correlation).

6.2 Factor analysis

Looking at the correlation matrices from Figure 6.1, I suspected that the correlation matrices can be decomposed into two factors: a measurement-dependent factor (bandwidth, current supply) and an intrinsic correlation factor (between the samples corresponding

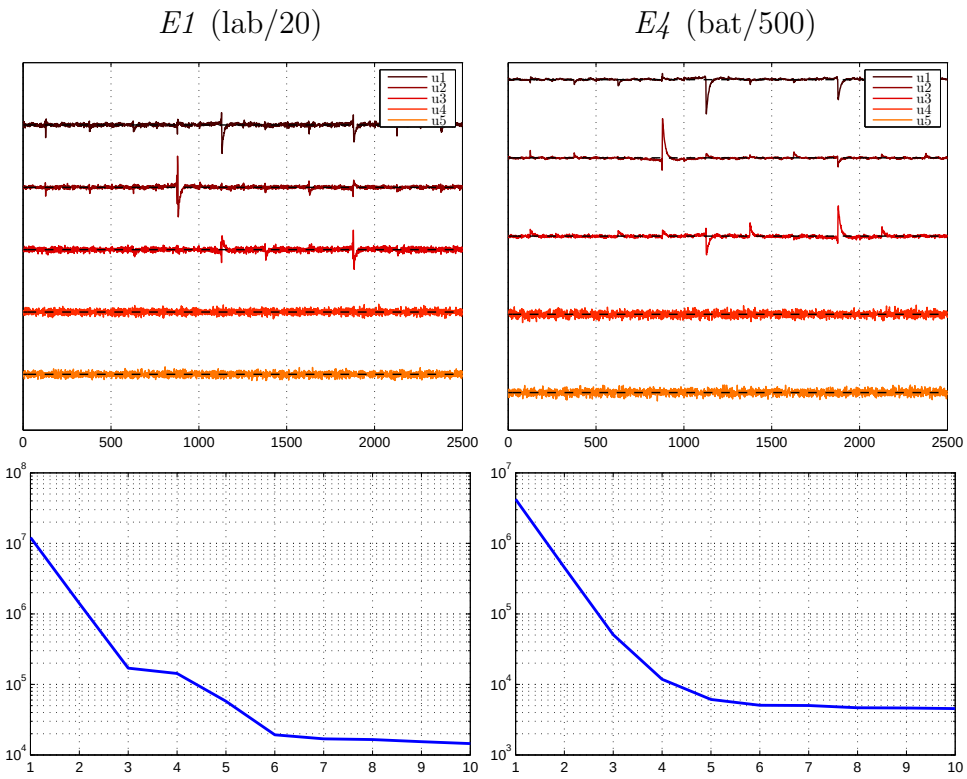

Figure 6.4: LDA eigenvectors (top) and eigenvalues (bottom) for experiments *E1* (left) and *E4* (right), computed using $n_{\mathrm{p}} = 1000$.

to high levels of glitching). If this is the case, then factor analysis [54, Chapter 9] might retrieve these factors, and reconstruct the correlation matrix only from these factors.

In factor analysis, we assume that the structure of the covariance or correlation matrix is determined by a small number of factors, even if the size of the correlation matrix is large, e.g. $K = 2$ factors may determine a correlation matrix of $m = 2500$ variables. This technique works as follows.

Let X_j be the random variable from which leakage samples at time j are drawn, and $\mathbf{X} = [X_1, X_2, \ldots, X_m]$ be the random vector representing the leakage traces, as in Section 3.1. Then, we can write each zero-mean value of a variable X_j as a linear combination of *common factors* $\mathbf{F} \in \mathbb{R}^K$ and a *specific factor* $E_j \in \mathbb{R}$, which is unique to each variable:

$$X_j - \mu_j = l_{j1}F_1 + l_{j2}F_2 + \cdots + l_{jK}F_K + E_j, \quad j \in \{1, \ldots, m\}, \tag{6.3}$$

where we call $\{l_{j1}, \ldots, l_{jK}\}$ the *factor loadings* (constant for all traces but specific to each variable). In matrix notation, we can write

$$\mathbf{X} - \boldsymbol{\mu} = \mathbf{LF} + \mathbf{E}, \tag{6.4}$$

where $\mathbf{L} \in \mathbb{R}^{m \times K}$ is the matrix of factor loadings, and $\mathbf{E} \in \mathbb{R}^m$ is the vector of specific factors.

Without any additional constraints, it is difficult to obtain useful values for this model. Therefore, we make the following assumptions, which form the *orthogonal factor model*. In this model, the random vectors $\mathbf{F}$ and $\mathbf{E}$ must satisfy the following conditions [54, 9.2]:

$$\begin{aligned} &\mathbf{F} \text{ and } \mathbf{E} \text{ are independent, so } \mathbf{Cov}(\mathbf{E}, \mathbf{F}) = \mathbb{E}(\mathbf{E}\mathbf{F}') = \mathbf{0}, \\ &\mathbb{E}(\mathbf{F}) = \mathbf{0}, \mathbf{Cov}(\mathbf{F}) = \mathbb{E}(\mathbf{F}\mathbf{F}') = \mathbf{I}, \\ &\mathbb{E}(\mathbf{E}) = \mathbf{0}, \mathbf{Cov}(\mathbf{E}) = \mathbb{E}(\mathbf{E}\mathbf{E}') = \mathbf{\Psi}, \text{ where } \mathbf{\Psi} \text{ is a diagonal matrix.} \end{aligned} \tag{6.5}$$

The orthogonal factor model implies a particular covariance structure of the random vector $\mathbf{X}$. Starting from (6.4), we obtain

$$(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})' = \mathbf{L}\mathbf{F}(\mathbf{L}\mathbf{F})' + \mathbf{E}(\mathbf{L}\mathbf{F})' + \mathbf{L}\mathbf{F}\mathbf{E}' + \mathbf{E}\mathbf{E}', \tag{6.6}$$

which leads to the real (population) covariance

$$\begin{aligned} \mathbf{\Sigma} &= \mathbf{Cov}(\mathbf{X}) = \mathbb{E}(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})' \\ &= \mathbf{L}\,\mathbb{E}(\mathbf{F}\mathbf{F}')\mathbf{L}' + \mathbb{E}(\mathbf{E}\mathbf{F}')\mathbf{L}' + \mathbf{L}\,\mathbb{E}(\mathbf{F}\mathbf{E}') + \mathbb{E}(\mathbf{E}\mathbf{E}') \\ &= \mathbf{L}\mathbf{L}' + \mathbf{\Psi}. \end{aligned} \tag{6.7}$$

Therefore, we may be able to use a small number K of components, to determine the entire structure of the covariance matrix. For example, with $K = 2$ and $m = 2500$, we may be able to represent a covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{m \times m}$, having $m(m-1)/2 = 3123750$ parameters, with only $m(K+1) = 7500$ parameters. In addition, factor analysis allows us to understand the main structure of the covariance, as I will show in the next section.

There are several methods to obtain the matrix of factor loadings $\mathbf{L}$, but I will focus only on the *Principal Component* method, which is relatively simple to implement and works well in my experimental context. This method relies on the spectral decomposition (see also Section 3.37) of $\mathbf{\Sigma}$,

$$\mathbf{\Sigma} = \lambda_1 \mathbf{e}_1 \mathbf{e}_1{}' + \lambda_2 \mathbf{e}_2 \mathbf{e}_2{}' + \cdots + \lambda_m \mathbf{e}_m \mathbf{e}_m{}', \tag{6.8}$$

which can also be written as the matrix product:

$$\mathbf{\Sigma} = [\sqrt{\lambda_1}\mathbf{e}_1, \sqrt{\lambda_2}\mathbf{e}_2, \ldots, \sqrt{\lambda_m}\mathbf{e}_m] \begin{bmatrix} \sqrt{\lambda_1}\mathbf{e}_1{}' \\ \sqrt{\lambda_2}\mathbf{e}_2{}' \\ \vdots \\ \sqrt{\lambda_m}\mathbf{e}_m{}' \end{bmatrix}, \tag{6.9}$$

where $(\lambda_j, \mathbf{e}_j)$ are the (eigenvalue, eigenvector) pairs of $\mathbf{\Sigma}$, with $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$. As before, since in practice we do not know the true matrix $\mathbf{\Sigma}$, we should estimate it using the sample covariance matrix $\mathbf{S}$, computed using (4.1) or (4.19). Comparing (6.7) and (6.9), and ignoring $\mathbf{\Psi}$ for now, we can identify $\mathbf{L} = [\mathbf{l}_1, \ldots, \mathbf{l}_m]$ as the matrix having the vector $\mathbf{l}_j = \sqrt{\lambda_j}\mathbf{e}_j \in \mathbb{R}^m$ as it's j-th column, obtaining $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}'$. The main idea then, is to select only those K columns of $\mathbf{L}$ that correspond to the minimum number of factors

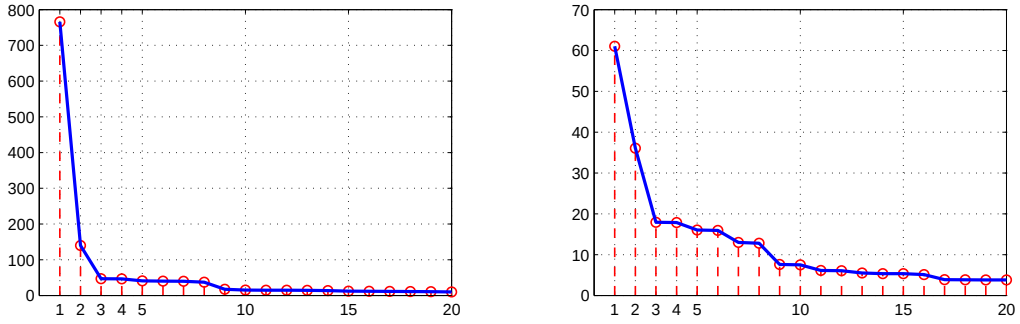

Figure 6.5: First 20 eigenvalues from the sample correlation matrix $\mathbf{R}$ obtained from experiments *E2* (left) and *E4* (right).

describing well enough our covariance matrix. The number of retained factors can be chosen based on prior knowledge of the data, or using the rules presented in Section 3.9.1. As a result, we can use the restricted matrix of factor loadings $\widetilde{\mathbf{L}} = [\mathbf{l}_1, \ldots, \mathbf{l}_K]$, having only K columns, and obtain the estimated sample covariance matrix

$$\widetilde{\mathbf{S}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{L}}' + \mathbf{\Psi}, \tag{6.10}$$

where the matrix $\mathbf{\Psi}$ of specific variances can be obtained from the diagonal elements of $\mathbf{S} - \widetilde{\mathbf{L}}\widetilde{\mathbf{L}}'$:

$$\mathbf{\Psi} = \begin{bmatrix} \psi_1 & 0 & \ldots & 0 \\ 0 & \psi_2 & \ldots & 0 \\ \vdots & \ldots & \ldots & \vdots \\ 0 & 0 & \ldots & \psi_m \end{bmatrix}, \text{ with } \psi_j = s_{jj} - \sum_{i=1}^{K} l_{ji}^2. \tag{6.11}$$

The Principal Component method can be used with both, the sample correlation matrix $\mathbf{R}$, and the sample covariance matrix $\mathbf{S}$, as I will show in the following sections.

6.2.1 Factor analysis on Koala

In Figure 6.5, I show the first 20 eigenvalues λ_j of the sample correlation matrix $\mathbf{R}$ obtained using (6.1) on the pooled covariance matrix $\mathbf{S}_{\text{pooled}}$ of experiments *E2* (left) and *E4* (right). We can see that, in both cases, there is a clear elbow at $K = 3$, which is in line with our hypothesis that the first two factors (corresponding to the largest two eigenvalues) make up for most of the correlation. We can also see that in *E2* both eigenvalues λ_1 and λ_2 are at least a factor three larger than the rest, while for *E4* only the first eigenvalue is a factor three larger than the others. This suggests that, for *E4*, there is merely one factor that influences the correlation, as observed also in Figure 6.1.

Based on these results, and the observations from the beginning of Section 6.2 that only 2 factors seem to influence the correlation between samples, I have set $K = 2$ and obtained

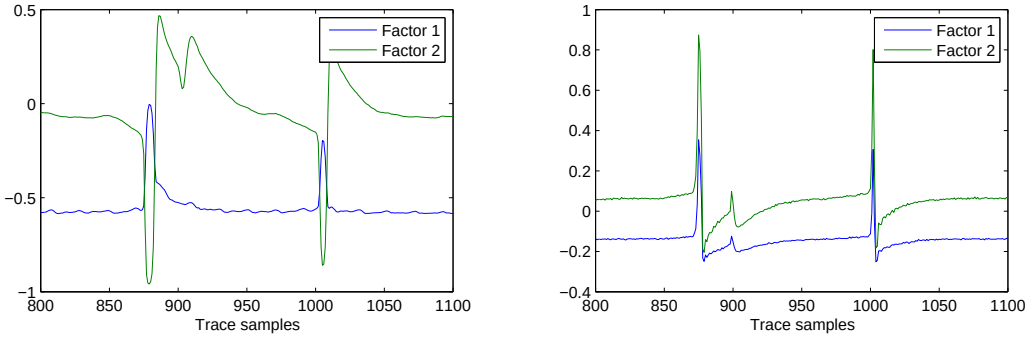

Figure 6.6: Factor loadings l_{j1} and l_{j2} corresponding to the factors F_1 and respectively F_2 for the samples $j \in \{800, \ldots, 1100\}$. Left: for *E2*; right: for *E4*.

the matrix of factor loadings $\widetilde{\mathbf{L}} = [\mathbf{l}_1, \mathbf{l}_2]$ from the matrix $\mathbf{R}$ of experiments *E2* and *E4*. The vectors of factor loadings $(\mathbf{l}_1, \mathbf{l}_2)$ for these experiments are shown in Figure 6.6. From the factors on *E2* (left), we can infer that F_1 represents the measurement-dependent factor (bandwidth, current supply), since it has a large non-zero value at most samples, except around the samples corresponding to high levels of glitching, and F_2 represents the intrinsic correlation factor, with large non-zero values around the samples corresponding to high levels of glitching. From *E4* (right), we can see that only the intrinsic correlation factor appears in both $\mathbf{l}_1$ and $\mathbf{l}_2$ (there are large non-zero values only around the samples corresponding to high levels of glitching), as we expect in this experiment, since the correlation due to measurement-dependent factor is almost zero (see Figure 6.1, bottom).

In Figure 6.7, I show the correlation matrices estimated using (6.10, 6.1), where $\widetilde{\mathbf{L}} = \begin{bmatrix} l_{11} & l_{12} \\ \vdots & \\ l_{m1} & l_{m2} \end{bmatrix} = [\mathbf{l}_1, \mathbf{l}_2]$, $m = m^{\mathrm{r}} = 2500$. Comparing Figures 6.1 and 6.7, we see that using only the factors loadings $\mathbf{l}_1$ and $\mathbf{l}_2$, along with the specific variations $\mathbf{\Psi}$, we can reproduce very well the entire correlation matrix.

6.2.2 Using factor analysis to improve template attacks

In the previous section, I showed that, using factor analysis, we can estimate a large correlation matrix $\mathbf{R} \in \mathbb{R}^{2500 \times 2500}$ using a small number of factors, corresponding to the main components of the correlation matrix. If indeed, the matrix $\widetilde{\mathbf{R}}$ computed from only $K = 2$ factors estimates well the real correlation matrix, then we may be able to use factor analysis to remove noise from the sample correlation matrix. Therefore, although in the previous section I used factor analysis on the correlation matrix $\mathbf{R}$, in order to focus on the structure of the correlation between samples, we can also use factor analysis on the individual covariances $\mathbf{S}_k$ or the pooled covariance $\mathbf{S}_{\text{pooled}}$, in order to reduce noise from these matrices and obtain cleaner estimates ($\widetilde{\mathbf{S}}_k$ or $\widetilde{\mathbf{S}}_{\text{pooled}}$) of the real covariance $\mathbf{\Sigma}$. This

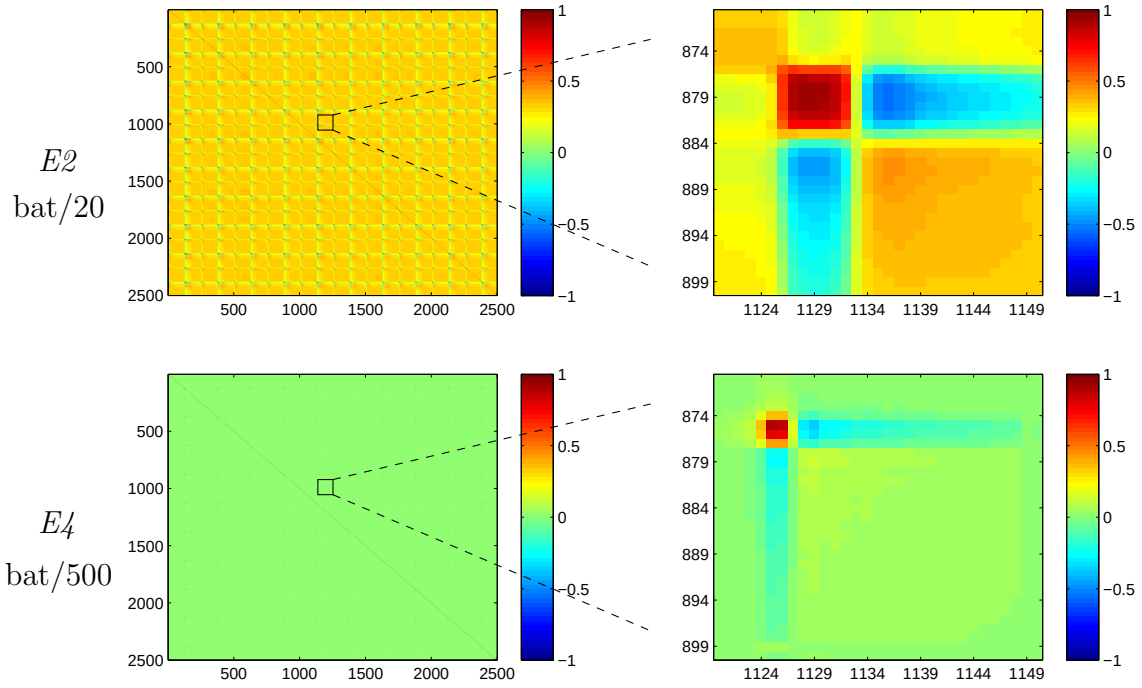

Figure 6.7: Correlation matrices estimated from the factor loadings on factors F_1 and F_2 and from the specific factors $\mathbf{E}$ only, for experiments *E2* (top) and *E4* (bottom); full matrices (left) and zoom around subsamples (right).

in turn may provide similar or better template attacks than using the covariances $\mathbf{S}_k$ or $\mathbf{S}_{\text{pooled}}$, since in Chapter 4, I showed that the estimation of the covariance matrix plays a very important role in the success of these attacks. In Algorithm 7, I describe how to use factor analysis on the *raw* covariance matrix $\mathbf{S}^{\text{r}}$ ($\mathbf{S}^{\text{r}}_k$ or $\mathbf{S}^{\text{r}}_{\text{pooled}}$), in order to obtain the estimate $\widetilde{\mathbf{S}}^{\text{r}}$, which can then be used for the template attack.

Algorithm 6 (Estimating $\widetilde{\mathbf{S}}^{\text{r}}$ using factor analysis)

Require: $\mathbf{S}^{\text{r}} \in \mathbb{R}^{m^{\text{r}} \times m^{\text{r}}}$ ($\mathbf{S}^{\text{r}}_k$ or $\mathbf{S}^{\text{r}}_{\text{pooled}}$)

1: Obtain the restricted matrix of factor loadings $\widetilde{\mathbf{L}} = [\sqrt{\lambda_1}\mathbf{e}_1, \sqrt{\lambda_2}\mathbf{e}_2, \ldots, \sqrt{\lambda_K}\mathbf{e}_K]$ (where $(\mathbf{e}_j, \lambda_j)$ are the eigenvectors and eigenvalues of $\mathbf{S}^{\text{r}}$)

2: Obtain the matrix of specific factors $\boldsymbol{\Psi} \in \mathbb{R}^{m^{\text{r}} \times m^{\text{r}}}$ ▷ See (6.11)

3: Obtain the covariance estimate $\widetilde{\mathbf{S}}^{\text{r}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{L}}' + \boldsymbol{\Psi}$,

In order to verify if factor analysis can improve template attacks, I used Algorithm 6, with $K = 2$ factors (based on the observations and results from Section 6.2), to estimate both the individual covariances $\mathbf{S}^{\text{r}}_k$ and the pooled covariances $\mathbf{S}^{\text{r}}_{\text{pooled}}$ from experiment *E4*. Then, I evaluated the template attacks, using the estimated covariances $\widetilde{\mathbf{S}}^{\text{r}}_k$ and $\widetilde{\mathbf{S}}^{\text{r}}_{\text{pooled}}$, with the discriminants $\text{d}^{\text{joint}}_{\text{LOG}}$ (4.26) and $\text{d}^{\text{joint}}_{\text{LINEAR}}$ (4.28), respectively. The results are shown in Figure 6.8.[1] We can see that, when using $\mathbf{S}^{\text{r}}_{\text{pooled}}$ (top), the estimates obtained

[1]For the results using the individual covariances $\mathbf{S}^{\text{r}}_k$ (see bottom of Figure 6.8), I did not use LDA

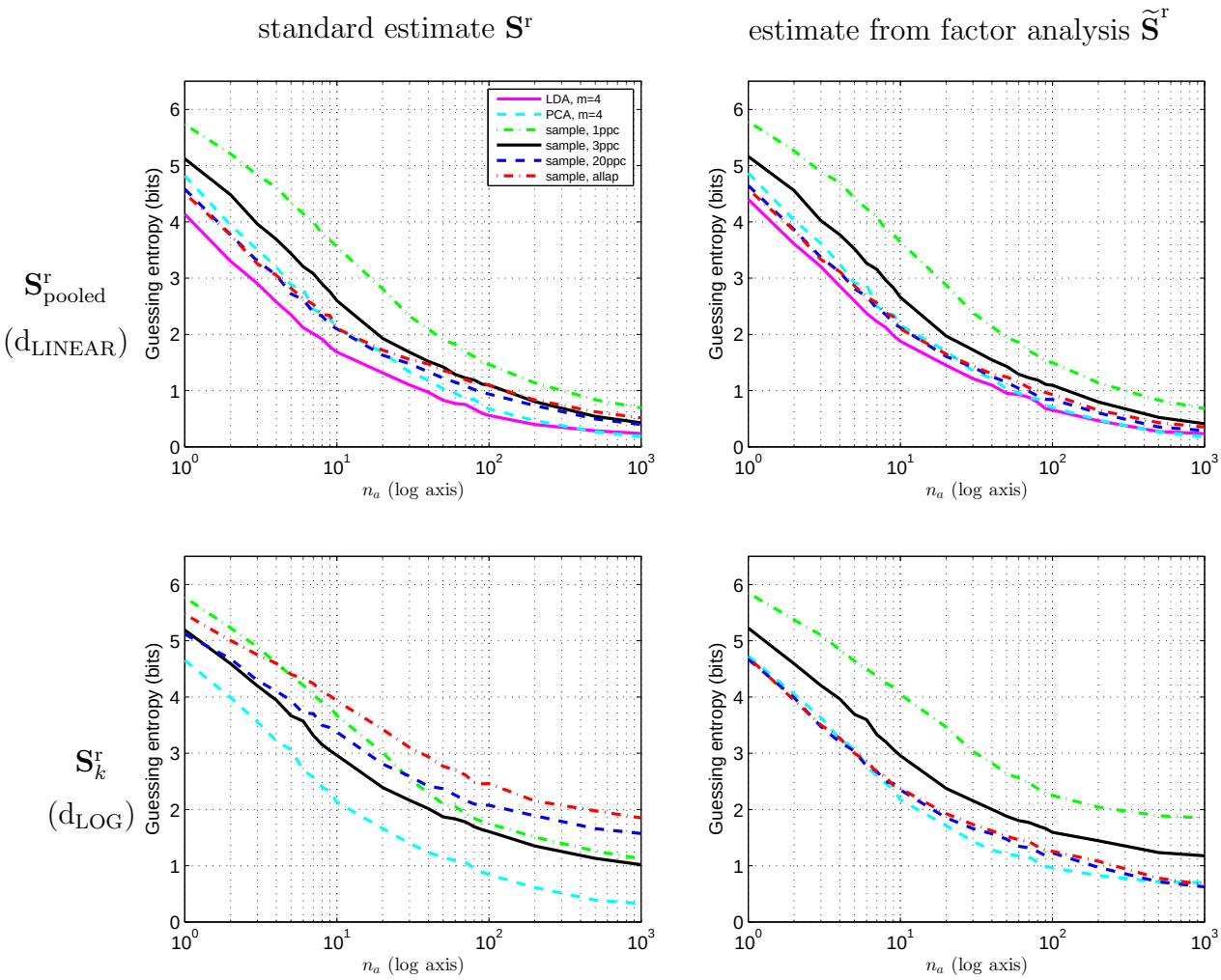

Figure 6.8: Results of template attacks on real data using the individual covariances with $\mathrm{d}_{\mathrm{LOG}}$ (top) and the pooled covariances with $\mathrm{d}_{\mathrm{LINEAR}}$ (bottom). Left: using the standard estimators for $\mathbf{S}_k$ and $\mathbf{S}_{\mathrm{pooled}}$ from (4.1, 4.19). Right: using factor analysis to estimate $\widetilde{\mathbf{S}}_k$ and $\widetilde{\mathbf{S}}_{\mathrm{pooled}}$. In all cases I used $n_{\mathrm{p}} = 2000$ traces for each candidate k during profile.

from factor analysis provide very similar results. This means that, (a) the $K = 2$ factors are indeed able to capture most of the correlation between samples, and (b) the pooled covariance $\mathbf{S}^{\mathrm{r}}_{\mathrm{pooled}}$ already does a good job at estimating the real covariance $\mathbf{\Sigma}^{\mathrm{r}}$, such that factor analysis brings little or no improvement. However, we can observe the differences better, when looking at the attacks using the individual covariances $\mathbf{S}^{\mathrm{r}}_k$ (bottom), since in this case only $n_{\mathrm{p}} = 200$ traces were used for the estimation of each $\mathbf{S}^{\mathrm{r}}_k$, while for $\mathbf{S}^{\mathrm{r}}_{\mathrm{pooled}}$, I used $n_{\mathrm{p}}|\mathcal{S}| = 51200$ traces. Firstly, we see that, when using the standard covariances $\mathbf{S}^{\mathrm{r}}_k$ (bottom, left), PCA and *1ppc* perform better (as n_{a} increases) than in the case of using the covariances $\widetilde{\mathbf{S}}^{\mathrm{r}}_k$. This means that there are still some minor factors that, in my choice of $K = 2$ factors, I did not consider when computing $\widetilde{\mathbf{S}}^{\mathrm{r}}_k$, but which PCA and *1ppc* were able to use, since due to the small number of variables ($m = 4$ for PCA, $m = 8$ for *1ppc*),

since it requires the use of the common covariance $\mathbf{S}^{\mathrm{r}}_{\mathrm{pooled}}$.

even the $n_{\mathrm{p}} = 200$ traces were enough to obtain a good covariance estimate $\mathbf{S}_k^{\mathrm{r}}$. Secondly, the bottom of Figure 6.8 shows that factor analysis can provide a cleaner estimate of $\boldsymbol{\Sigma}^{\mathrm{r}}$, since the methods *20ppc* and *allap* (which use a large number of samples) perform vastly better when the estimates $\widetilde{\mathbf{S}}_k^{\mathrm{r}}$ (bottom, right) are used instead of the standard covariances $\mathbf{S}_k^{\mathrm{r}}$ (bottom, left). In particular, we can see that, for low n_{a}, the *20ppc* and *allap* methods using $\widetilde{\mathbf{S}}_k^{\mathrm{r}}$, provide results as good as PCA with the standard estimates $\mathbf{S}_k^{\mathrm{r}}$.

In summary, my results show that factor analysis can be useful for template attacks, in situations where we have a low number n_{p} of profiling traces and a large number of samples (see *20ppc* and *allap* in the bottom of Figure 6.8). However, this technique does not appear to be very useful otherwise (e.g. when using PCA or LDA with a small m).

6.3 Analysis on synthesized data

In order to observe the influence of the correlation on the template attack, I synthesized data that resembles what I obtained from the experiments on real data. This synthesized data permits me to isolate the effects of correlation (matrix $\mathbf{R}$) and noise (matrix $\mathbf{D}$) on the results of the template attack, thus providing a good setting for an experimental statistical analysis.

6.3.1 Synthesized data

To generate the synthesized data, I used a simple model of power consumption for three load instructions during five clock cycles, which depend all on a single byte, in a similar manner as my real target CPU, the Atmel XMEGA 256 A3U. More precise models can be built, e.g. using stochastic models [97], as Renzo did in his master thesis [32]. However, for the purposes of analysing the effects of correlation and noise, and demonstrating the usefulness of factor analysis for synthesizing data, my simpler model should suffice.

My synthesized noise-free samples $\bar{x}_{kj}$ of a particular candidate k are modeled as

$$\bar{x}_{kj} = c_j + a \left(\sum_{i=1}^{8} r_i \cdot b_{ki} - 4 \right), \tag{6.12}$$

where a is a constant value for each clock cycle, the -4 is used to remove the average value $\bar{x}_j$ over all candidates k, and b_{ki} is the value of bit i of candidate k. During the clock cycles 1,3,4 and 5, I set all the values $r_i = 1$, which reduce my model to the Hamming weight model, while for the second clock cycle I used random values $r_i \in [0.8, 1.2]$, to obtain different influences of each bit in the synthesized sample $\bar{x}_j$. This reflects my observations from the real traces, where the second clock cycle of my target LOAD instruction leaks clearly more than the Hamming weight of the target value k. The values c_j follow a simple

exponential curve, particular to each clock cycle, that increases or decreases from some constant value c_{static} (this represents the static power consumption of each clock cycle), to a maximum (respectively minimum) determined by the leakage model described above, and then decreases (respectively increases) back to the constant value c_{static}.

After obtaining the noise-free vectors $\bar{\mathbf{x}}_k$, I also need a method to synthesize the correlation between samples. Earlier, I showed how to compute the correlation matrix $\mathbf{R}$ from a covariance matrix $\mathbf{S}$, using (6.1). However, as described in Section 3.3.5, we can also perform the opposite, and compute the covariance matrix $\mathbf{S}$ from $\mathbf{R}$ and $\mathbf{D}$ as

$$\mathbf{S} = \mathbf{DRD}. \tag{6.13}$$

Then, using factor analysis, and equations (6.1, 6.13), I designed Algorithm 7 to synthesize arbitrary covariance matrices.

Algorithm 7 (Synthesize covariance matrix using factor analysis)

1: Compute $\mathbf{S}$ (either $\mathbf{S}_k$ or $\mathbf{S}_{\text{pooled}}$) using (4.1, 4.19) on the real data.
2: Compute $\mathbf{D}$ and $\mathbf{R}$ using (6.1) on the covariance estimate $\mathbf{S}$.
3: Based on the spectral decomposition (6.8) of $\mathbf{R}$, obtain $\mathbf{l}_1 = \sqrt{\lambda_1}\mathbf{e}_1$ corresponding to the measurement-dependent *overall* correlation (F_1) and $\mathbf{l}_2 = \sqrt{\lambda_2}\mathbf{e}_2$ corresponding to the measurement-independent *intrinsic* correlation (F_2).
4: Scale $\mathbf{l}_1$, obtaining $\mathbf{l}_1^{\text{new}} = \rho_{\text{corr}}\mathbf{l}_1$, in order to increase or decrease the overall correlation (F_1).
5: Use $\mathbf{R}$ and $\widetilde{\mathbf{L}} = [\mathbf{l}_1^{\text{new}}, \mathbf{l}_2]$ with (6.10,6.11) to compute $\widetilde{\mathbf{R}}$.
6: Scale the diagonal elements of $\mathbf{D}$, obtaining $\text{diag}(\mathbf{D}^{(\text{new})}) = \rho_{\text{std}}\text{diag}(\mathbf{D})$, in order to increase or decrease the desired level of noise.
7: Compute $\widetilde{\mathbf{S}}$ from $\widetilde{\mathbf{R}}$ and $\mathbf{D}^{(\text{new})}$ using (6.13).

I make the following remarks about Algorithm 7: (a) instead of steps (1–4,6) we could use arbitrary values for the noise (diagonal of $\mathbf{D}$) and factor loadings ($\mathbf{l}_1, \mathbf{l}_2$), or even use more factors. I used these steps in order to change only the level of noise ($\mathbf{D}$) and overall correlation ($\mathbf{l}_1$); (b) I used only two factors to match my observations on real data. However, for other scenarios, the choice of factors might be different; (c) I select from $\widetilde{\mathbf{S}}$, only the rows and columns that correspond to the samples j, for which I synthesized the noise-free vectors $\bar{\mathbf{x}}_k$.

Note also, that it is not possible to use arbitrary covariance matrices with template attacks, since the covariance matrices need to be positive-semidefinite [26]. For this reason, Algorithm 7 provides a helpful and reliable method to synthesize different, correct (positive-semidefinite), covariance matrices, allowing to test independently the effects of inter-sample correlation ($\mathbf{R}$) and noise ($\mathbf{D}$) on template attacks. This approach may also be useful also for simulations where the noise-free leakage of a CPU is available but the correlation is not, as was the case in the work of Renauld et al. [93].

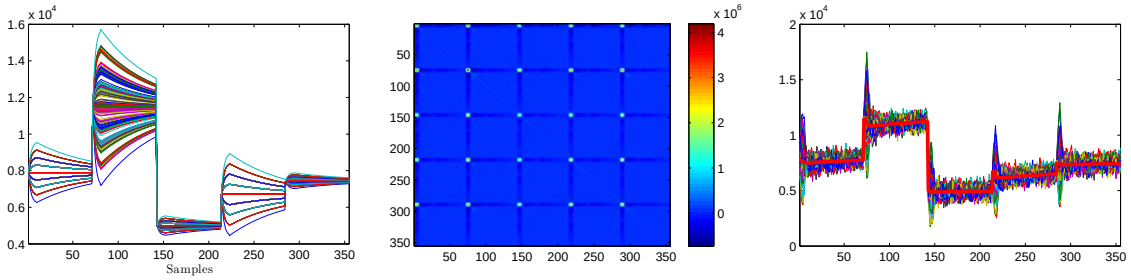

Figure 6.9: Simulated data of 71 samples per clock for 5 clocks. Left: noise-free mean vectors; center: synthesized covariance matrix; right: 50 random vectors generated using MATLAB's *mvnrnd* on one of the mean vectors (bold) and the covariance matrix.

6.3.2 Analysis of synthesized correlation

Using the leakage models from (6.12) and Algorithm 7, I synthesized data with 71 samples per clock over five clock cycles (355 samples in total), having different values of noise (matrix $\mathbf{D}$) and correlation (matrix $\mathbf{R}$), obtaining traces such as those shown in Figure 6.9. I used $\rho_{\text{std}}, \rho_{\text{corr}} \in \{0.2, 0.4, 0.6, \ldots, 2\}$, resulting in 100 different covariance estimates $\widetilde{\mathbf{S}}$. In Figure 6.10, I show the guessing entropy[2] obtained by running the template attacks on this synthesized data for $n_{\text{p}} = 100$, $n_{\text{a}} = 1$, and in Figure 6.11 for $n_{\text{p}} = n_{\text{a}} = 100$. All attacks are performed five times with different data, and the resulting guessing entropy is obtained as an average over the five iterations. We see that the overall correlation (ρ_{corr}) has a much weaker influence on the template attack than the noise (ρ_{std}). The correlation affects the results of the sample selection methods (*1ppc*, *3ppc*, *20ppc*, *allap*) when the noise is higher, and this is is more visible for $n_{\text{a}} = 1$. LDA and PCA are generally not affected by the strength of the correlation, even at higher levels of noise: this can be explained by the fact that both LDA and PCA project the samples into a new space, where these samples are not correlated anymore. We observe as well that *1ppc* and *3ppc* are always worse than all the other methods (including *20ppc* and *allap*), which is in line with my observations from Chapter 4. In conclusion, factor analysis can be used to simulate traces having a similar structure as the real traces, therefore providing a useful tool for the evaluation of template attacks.

[2]The color bar on the right of the plots provides the value of the guessing entropy corresponding to each result.

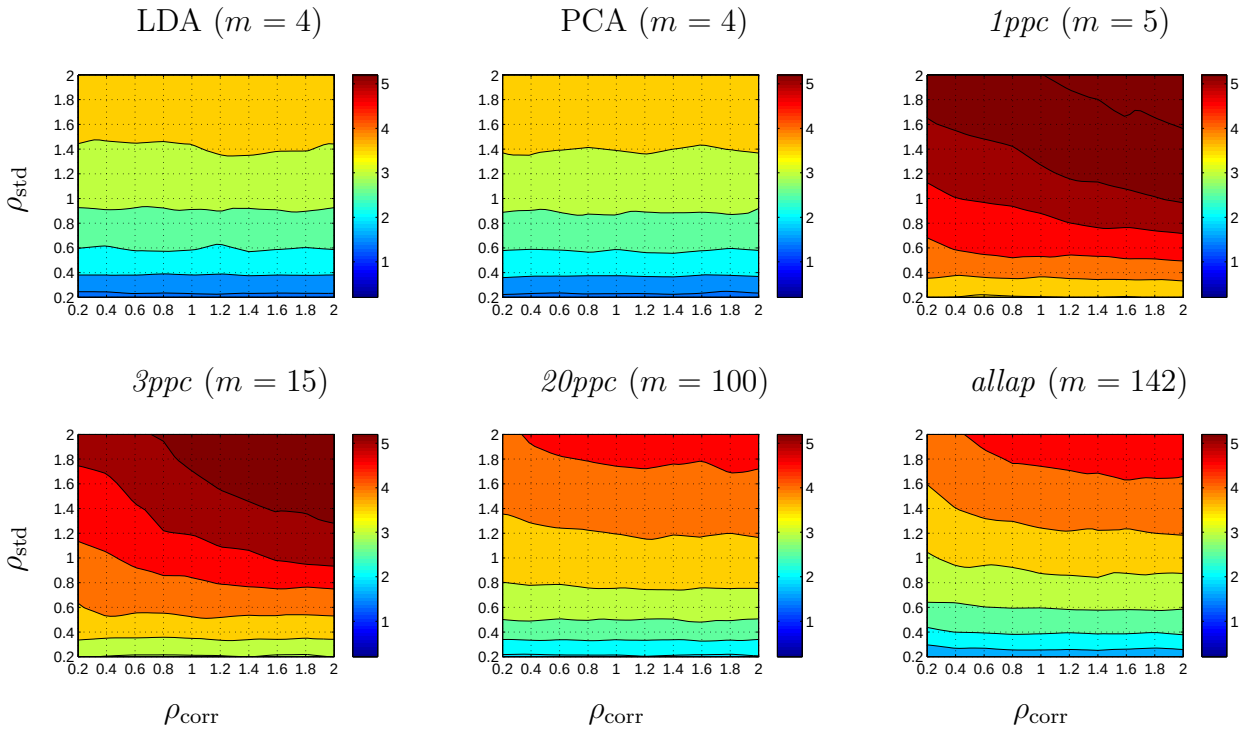

Figure 6.10: Contour plots of guessing entropy over 100 combinations of noise ($\mathbf{D}^{(\text{new})}$) and correlation ($\mathbf{l}_1^{\text{new}}$) for all compression methods from Table 4.1. $n_{\text{p}} = 100$, $n_{\text{a}} = 1$.

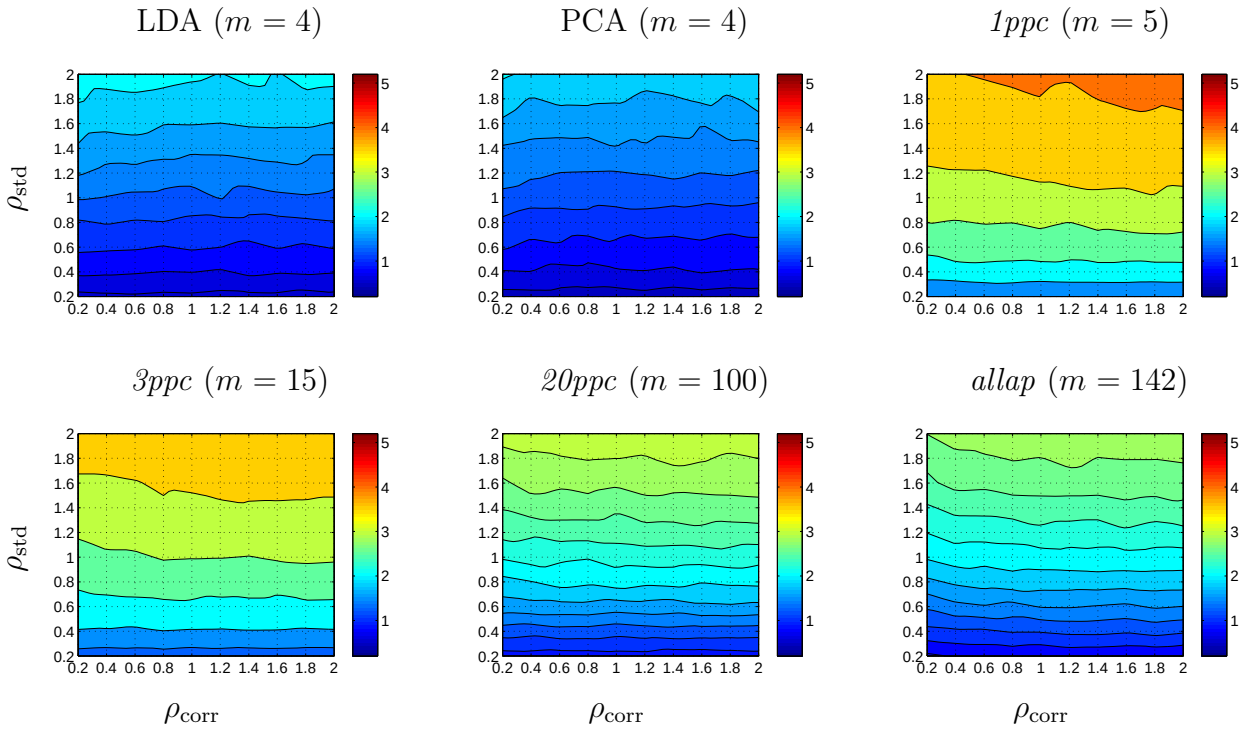

Figure 6.11: Contour plots of guessing entropy over 100 combinations of noise ($\mathbf{D}^{(\text{new})}$) and correlation ($\mathbf{l}_1^{\text{new}}$) for all compression methods from Table 4.1. $n_{\text{p}} = n_{\text{a}} = 100$.

Chapter 7

Efficient stochastic methods: profiled attacks beyond 8 bits

Throughout the previous chapters, I presented several methods to improve the efficiency of template attacks in different situations, such as when dealing with a large number of leakage samples (Chapter 4), with different devices (Chapter 5), or with different acquisition parameters (Chapter 6).

In this chapter, I describe the use of the "stochastic model" [97] to obtain very efficient *profiled* attacks, i.e. attacks in which we use a profiling phase, as we do with template attacks. While the template attacks presented in previous chapters are very general, in that they only assume a multivariate normal distribution of the underlying leakage traces, they also require a large number of profiling traces in order to obtain good mean and covariance estimates, resulting in one independent mean vector per target value. In contrast, the stochastic method models the leakage through a small number of functions of a data word, such as the value of each bit, obtaining the mean vectors of all possible target values mainly as a linear combination of the leakage vectors corresponding to each bit. This results in much fewer parameters to be estimated, thereby trading generality of the model for efficiency of profiling. As I show in Section 7.1, the stochastic model merely provides an efficient method to estimate the mean and covariance parameters, while the rest of the attack can proceed exactly as with template attacks. In particular, we can use all the efficient methods presented in Chapter 4. The results presented in this chapter have been published in the following conference paper [25]:

> Marios O. Choudary and Markus G. Kuhn, *Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits*, CARDIS 2014, Springer LNCS 8968, pp. 85–103.

As I have shown throughout the previous chapters, the PCA and LDA compression methods can improve significantly the success of template attacks. The question arose, whether similar benefits could be achieved with the stochastic model [101]. In Section 7.2, I present

four efficient ways of implementing PCA and LDA with stochastic models, which allow us to combine the profiling efficiency of the stochastic method with the compression efficiency of PCA and LDA, resulting in very efficient profiled attacks.

In Section 7.3, I evaluate my four implementations of PCA and LDA using the *Grizzly Beta* dataset from Section 2.3.1, in order to provide a clear comparison between the template attack and the stochastic model. The results show that these PCA and LDA implementations can indeed increase the success of profiled attacks, while preserving the profiling efficiency of the stochastic model.

Then, in Section 7.4, I demonstrate how to profile and evaluate stochastic models simultaneously for more than eight bits, using the *Panda* dataset from Section 2.3.3, and I show that my applications of LDA and PCA are particularly helpful in this context.

In these evaluations, I use the linear discriminant from Section 4.4.3 for the attack step, and the logarithmic guessing entropy from Section 4.5.1 to measure the success of the attack. Also, I compare the PCA and LDA methods against the sample selections *1ppc*, and *20ppc* from Section 4.3.1. Please refer to Chapter 4 for more information, and for a detailed description of the template attack.

7.1 Stochastic models

Stochastic models were introduced by Schindler et al. [97] as another kind of profiled attack, where the profiling phase can be more efficient than for template attacks.

In the original description, as well as in following publications on stochastic model [46, 103, 52], each leakage sample x_j of a trace $\mathbf{x}_i$ is modelled as a combination of a deterministic part $\delta_j(d_i, k)$, which depends on a part (e.g. byte) of plaintext d_i and a part (e.g. byte) of key k, and a random part ρ_j which models the noise:

$$x_j = \delta_j(d_i, k) + \rho_j. \tag{7.1}$$

The formulation of the stochastic model in this form was guided by the assumption that a side channel attack will use intermediate results such as the output of an S-box of a block cipher (e.g. AES), which depend on both a known value (the plaintext) and an unknown value (the key).

Here, I simply assume that a leakage sample x_j depends on a deterministic part $\delta_j(k)$, which takes as input a single value k, and the random part ρ_j, which models the noise:

$$x_j = \delta_j(k) + \rho_j. \tag{7.2}$$

This model can be used to attack any data, not just intermediate results that depend on combinations of a plaintext and a key, just like the template attacks presented in

Section 4.1.[1]

The deterministic function $\delta_j(k)$ is modelled as a linear combination of base functions $g_{jb} : \mathcal{S} \to \mathbb{R}$, with

$$\delta_j(k) = \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k), \tag{7.3}$$

where the coefficients $\beta_{jb} \in \mathbb{R}$ model the contribution of each base function g_{jb}. The essential idea behind stochastic models is to find a good set of base functions that matches well the leakage of the values k. A common and generally good option for 8-bit architectures is to use the set of $u = 9$ base functions, known as $\mathcal{F}_9$, for which $g_{j0}(k) = 1$ and $g_{jb}(k) = \text{bit}_b(k)$. In this case, the coefficients β_{jb} model the contribution of each bit. As an example, in Figure 7.1, I show the bit-coefficients β_{jb} for the *Grizzly Beta* dataset. For my experiments, $\mathcal{F}_9$ produced good results, because the leakage caused by the bus lines in my target CPU can be modelled well with $\mathcal{F}_9$, but this model may not work well when dealing, for example, with hardware implementations of cryptographic algorithms. In such cases, we might need to use combinations of bits [97, 46], or model the leakage produced by combinations of several data values. To obtain the most efficient model, we may require precise knowledge of the internal architecture of the device. However, when targeting 8-bit values (e.g. the S-box output of AES) in unknown hardware implementations of cryptographic algorithms, we may still try to use $\mathcal{F}_9$ and compensate the lack of knowledge of a very good model with an increase of the number of profiling traces.

During profiling, instead of acquiring n_p leakage traces $\mathbf{x}^\text{r}_{ki}$ for each candidate k and then use (4.1, 4.19) to compute the mean vectors $\bar{\mathbf{x}}_k$ and covariance $\mathbf{S}_\text{pooled}$ needed for template attacks, we only use a *total* of N leakage traces $\mathbf{x}^\text{r}_i \in \mathbb{R}^{m^\text{r}}$ from a uniform distribution of the values $k \in \mathcal{S}$. As with template attacks, we generally compress these leakage traces to obtain the compressed traces $\mathbf{x}_i \in \mathbb{R}^m$ ($m \ll m^\text{r}$, see Section 7.2). Then, we combine all these leakage traces into the leakage matrix $\mathbf{X} \in \mathbb{R}^{N \times m}$. Next, for each sample index $j \in \{1, \ldots, m\}$ we build the matrix

$$\mathbf{F}_j = \begin{bmatrix} g_{j0}(k^1) & g_{j1}(k^1) & \ldots & g_{ju-1}(k^1) \\ g_{j0}(k^2) & g_{j1}(k^2) & \ldots & g_{ju-1}(k^2) \\ \vdots & \vdots & \ddots & \vdots \\ g_{j0}(k^N) & g_{j1}(k^N) & \ldots & g_{ju-1}(k^N) \end{bmatrix}, \tag{7.4}$$

[1] In the original approach from (7.1) the deterministic function $\delta_j(d_i, k)$ was intended to capture any combination of plaintext and key and then use a mapping function that reduced this combination into a value to be modelled by the set of base functions g_{jb}. However, the most common mapping is to use the XOR between d_i and k [97, 103] or the XOR between these and a mask value [69]. Therefore, in most cases, there is a single value (e.g. the XOR result) that is modelled by the base functions. For the cases where we want to target several values (e.g. for masking [97, 69], the proposed solution is to use a set of base functions that depend on both a mask y and the XOR between this mask, a plaintext and a key), we can simply form the candidate k to be the concatenation of the bits of these values (e.g. k = [bits mask|bits XOR]).

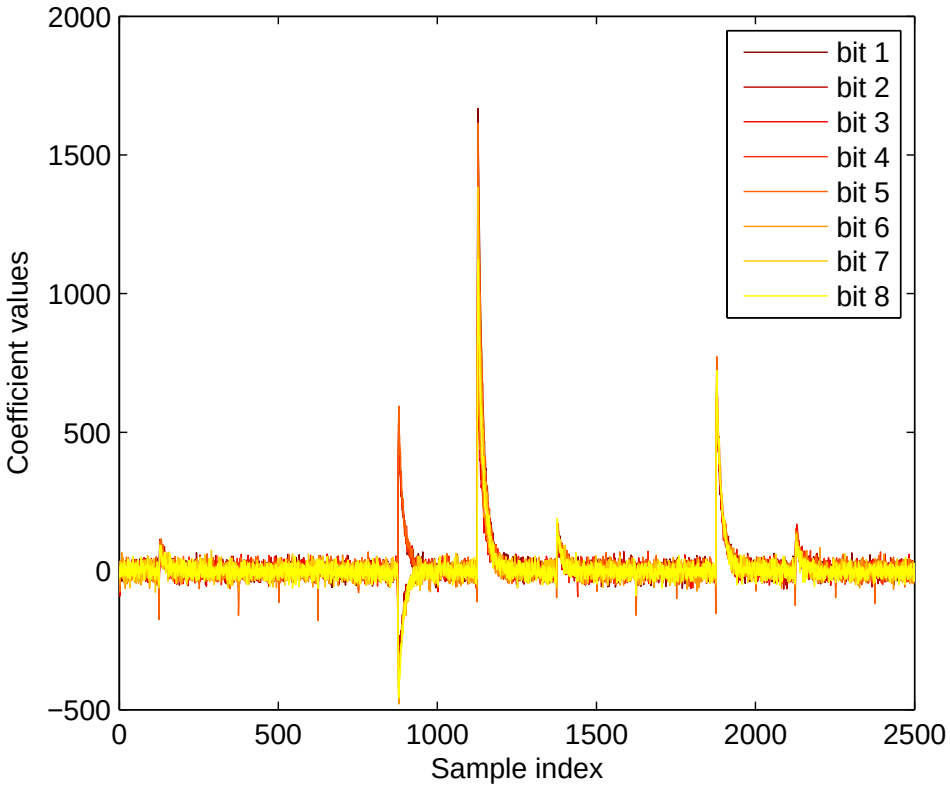

Figure 7.1: Coefficients β_{jb} of $\mathcal{F}_9$, for the *Grizzly Beta* dataset from Section 2.3.1.

and use the stochastic model

$$\mathrm{col}_j(\mathbf{X}) = \mathbf{d}_j + \mathbf{r}_j = \mathbf{F}_j\mathbf{v}_j + \mathbf{r}_j, \tag{7.5}$$

where $\mathrm{col}_j(\mathbf{X})$ contains the leakage samples x_j of all traces $\mathbf{x}_i$ in $\mathbf{X}$,

$$\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}] \tag{7.6}$$

represents the vector of coefficients modelling the base functions,

$$\mathbf{r}_j{}' = [\rho_j^1, \ldots, \rho_j^N] \tag{7.7}$$

is the vector of noise terms,

$$\mathbf{d}_j{}' = [\delta_j(k^1), \ldots, \delta_j(k^N)] \tag{7.8}$$

is the vector containing the deterministic part of each value, and k^i represents the value of k corresponding to the trace $\mathbf{x}_i$. To find the vector of coefficients $\mathbf{v}_j$, we try to minimise the distance $\|\mathrm{col}_j(\mathbf{X}) - \mathbf{F}_j\mathbf{v}_j\|^2$, leading to the least-squares solution

$$\mathbf{v}_j = (\mathbf{F}_j{}'\mathbf{F}_j)^{-1}\mathbf{F}_j{}'\mathrm{col}_j(\mathbf{X}). \tag{7.9}$$

Note that the matrix inversion in (7.9) requires $\mathrm{rank}(\mathbf{F}_j) = u$ (see Section 4.2), which in turn requires $\mathbf{F}_j$ to have u independent rows.

In practice, we may use the same set of base functions (e.g. $\mathcal{F}_9$) for all samples j (or at least for a subset of all samples). In this case, we can drop the subscript j from (7.4) and use the same $\mathbf{F}$ for all samples j. This allows us to compute all the coefficients at once as

$$\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m] = (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}, \tag{7.10}$$

which is computationally more efficient. The coefficient vectors $\mathbf{v}_j$, computed either with (7.9) or (7.10), can be used with (7.3) to compute the deterministic part $\delta_j(k)$ of a sample x_j for any value k. Note that this deterministic part is assumed to be noise-free, since the noise is captured by the term ρ_j. Therefore, as mentioned also by Gierlichs et al. [46], we can use the values $\delta_j(k)$ to compute the stochastic mean vectors $\hat{\mathbf{x}}_\mathbf{k} \in \mathbb{R}^m$ as

$$\hat{\mathbf{x}}'_\mathbf{k} = [\delta_1(k), \ldots, \delta_m(k)]. \tag{7.11}$$

While these correspond to the template mean vectors $\bar{\mathbf{x}}_k$ from (4.1), their approximation of the real trace means $\boldsymbol{\mu}_k$ is limited by how well the choice of base functions models the actual leakage.

In order to use also the noise information, we need to compute a covariance matrix $\hat{\mathbf{S}} \in \mathbb{R}^{m \times m}$, similar to the pooled covariance $\mathbf{S}_{\text{pooled}}$ from (4.19). For this, we can use the same N traces that we used to estimate the coefficients $\mathbf{v}_j$, and compute the noise vector $\mathbf{z}_i \in \mathbb{R}^m$, specific to each trace $\mathbf{x}_i$, as

$$\mathbf{z}_i{}' = [\rho_1^i, \ldots, \rho_m^i],\ \rho_j^i = x_j^i - \delta_j(k^i). \tag{7.12}$$

These vectors can then be used to compute the noise matrix

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1{}' \\ \vdots \\ \mathbf{z}_N{}' \end{bmatrix} = \begin{bmatrix} \rho_1^1 & \ldots & \rho_m^1 \\ \vdots & \ddots & \vdots \\ \rho_1^N & \ldots & \rho_m^N \end{bmatrix}, \tag{7.13}$$

and finally, we can compute the covariance matrix as

$$\hat{\mathbf{S}} = \frac{1}{N-1} \sum_{i=1}^{N} \mathbf{z}_i \mathbf{z}_i{}' = \frac{1}{N-1} \mathbf{Z}'\mathbf{Z}. \tag{7.14}$$

In the attack step, we proceed similarly to the template attack, using the linear discriminant from Section 4.4.3, but replacing the template mean vectors $\bar{\mathbf{x}}_k$ by the mean vectors $\hat{\mathbf{x}}_\mathbf{k}$ from (7.11), and the pooled covariance $\mathbf{S}_{\text{pooled}}$ by the covariance $\hat{\mathbf{S}}$ from (7.14).

7.1.1 Note on the estimation of the covariance

In the original publication of the stochastic model, Schindler et al. [97], and then also following publications [69, 103], suggested to use one set of N_1 traces for the estimation of

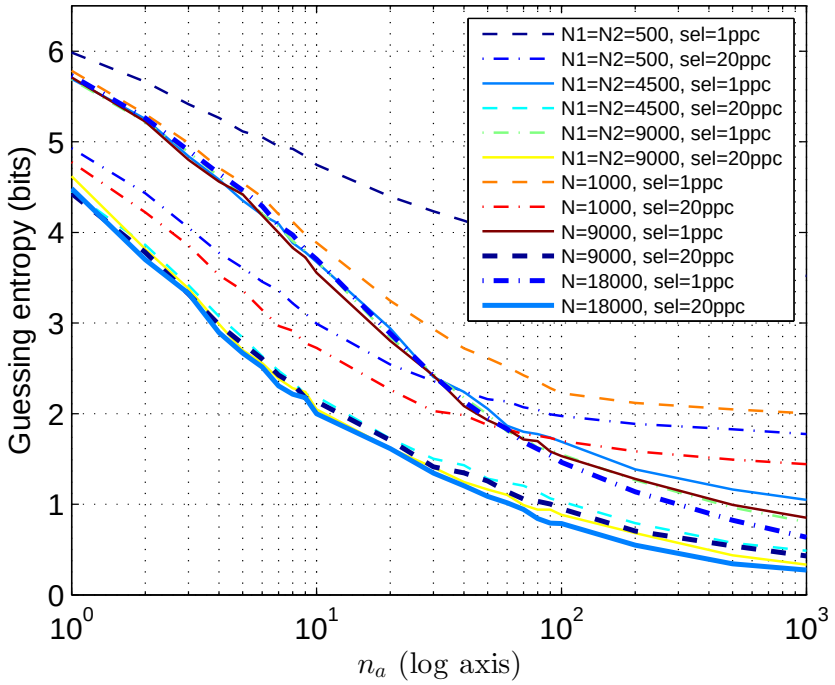

Figure 7.2: Results of stochastic model attacks using either (a) two different sets, having $N_1 = N_2 = N/2$ traces, for the estimation of the stocastic mean vectors $\hat{\mathbf{x}}_\mathbf{k}$ and covariance $\hat{\mathbf{S}}$, or (b) a single set of N traces for the estimation of both $\hat{\mathbf{x}}_\mathbf{k}$ and $\hat{\mathbf{S}}$.

the coefficients β_{jb}, and an *additional* training set of $N_2 = N - N_1$ traces to compute the covariance matrix $\hat{\mathbf{S}}$. However, it was never clearly motivated why we need the additional set of N_2 training traces. The already available $N = N_1$ traces that were used to estimate the coefficients β_{jb} seem good for this purpose, since in (7.2) the deterministic part $\delta_j(k)$ approximates the noise-free part, *common* to all the N traces, while $\mathbf{z} \in \mathbb{R}^m$ is the noise vector *specific* to each trace.

The approach I proposed and used, i.e. using the same N traces to compute both the mean vectors $\hat{\mathbf{x}}_\mathbf{k}$ and covariance matrix $\hat{\mathbf{S}}$ is more efficient than the previously proposed methods, because it allows us to use a larger number N of training traces compared to the original method, resulting in better estimates of both the mean vectors $\hat{\mathbf{x}}_\mathbf{k}$ and covariance $\hat{\mathbf{S}}$. The advantage of using all the N traces to estimate both the means $\hat{\mathbf{x}}_\mathbf{k}$ and covariance $\hat{\mathbf{S}}$ is shown in Figure 7.2 (compare for example $N_1 = N_2 = 500, \mathit{1ppc}$ with $N = 1000, \mathit{1ppc}$).

7.2 Compression methods for stochastic models

7.2.1 Sample selection

All the sample selection methods from Section 4.3 can be adapted for stochastic models by using (7.11) and (7.14) to compute the stochastic mean vectors $\hat{\mathbf{x}}_\mathbf{k}$ and covariance matrix

$\hat{\mathbf{S}}$, and then using these to obtain the desired signal-strength estimate $\mathbf{s}(t)$. In addition, Schindler et al. [97] proposed to use $\mathbf{s}(j) = \sum_{b=1}^{u-1} \beta_{jb}^2$, i.e. the norm of the data-dependent coefficients, which I shall refer to as *bnorm*.

7.2.2 PCA and LDA

PCA and LDA provided a significant break-through in the application of template attacks, and Standaert et al. [101] mentioned that *"Combining data dimensionality reduction techniques with stochastic models is a scope for further research"*. However, until now, the sole published attempt to apply PCA to stochastic models, by Heuser et al. [52], is inefficient. As I explained in Sections 3.9.3 and 4.3.2, the goal of PCA is to find the smallest set of eigenvectors $[\mathbf{e}_1, \ldots, \mathbf{e}_m] = \mathbf{U}^m$, such that the projection $\mathbf{X}_k = \mathbf{X}_k^{\mathrm{r}}\mathbf{U}^m$ from (4.12) maximises the distance between the compressed traces corresponding to different values k. The main idea is that the *treatment* vectors $\boldsymbol{\tau}_k = (\bar{\mathbf{x}}_k - \bar{\mathbf{x}})$, from Section 3.8.2, define the signal of interest, i.e. they provide the location and magnitude of the leakage caused by processing our target value. Therefore, to obtain the PCA eigenvectors that maximise this signal, we must apply PCA on the *treatment* matrix $\mathbf{B}$ from (4.8), which is derived from the *treatment* vectors. Instead, Heuser et al. [52] used the eigenvalues of the *raw* covariance matrix $\hat{\mathbf{S}}^{\mathrm{r}}$, computed as in (7.14), to project the leakage traces. This removes the correlation between leakage samples, but does not maximise the discrimination between means, since the matrix $\hat{\mathbf{S}}^{\mathrm{r}}$ contains no information about the different *raw* mean vectors $\hat{\mathbf{x}}_{\mathbf{k}}^{\mathrm{r}}$, obtained from (7.11). I suspect that the lack of 'mean' information in $\hat{\mathbf{S}}^{\mathrm{r}}$ is also the reason why only the first eigenvalue was significant in the results of Heuser et al., which lead them to use a univariate attack. I verified that for the *Grizzly* dataset this method provides no useful attack (i.e. the guessing entropy does not decrease).

I now describe four efficient methods of implementing PCA and LDA with stochastic models. All these methods work in three main steps. In the first step, for which I offer two methods (labelled "S" and "T" below), we compute the matrix $\hat{\mathbf{B}}$, as an approximation of the *treatment* matrix $\mathbf{B}$ from (4.8), and the *raw* covariance matrix $\hat{\mathbf{S}}^{\mathrm{r}}$ (only needed for LDA). Next, we use either PCA or LDA to obtain the matrix of eigenvectors $\mathbf{U}^m$, and use that to compress the *raw* leakage matrix $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$ into $\mathbf{X} \in \mathbb{R}^{N \times m}$. Finally, for the third step, we use the stochastic model, on the compressed (projected) traces, to model each sample x_j of a *compressed* trace $\mathbf{x}_i = [x_1, \ldots, x_m]$ in $\mathbf{X}$. The general method is shown in Algorithm 8.

S-PCA

My first PCA method for stochastic models, which I call *S-PCA*, relies on the stochastic model from Section 7.1, to build the mean vectors $\hat{\mathbf{x}}_{\mathbf{k}}^{\mathrm{r}}$ of the *raw* traces. In the first step,

Algorithm 8 (Generic PCA/LDA algorithm for stochastic models)

Require: $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$

Step 1:

Obtain the matrix $\hat{\mathbf{B}}$ (Algorithm 9 or 10),
and the matrix $\hat{\mathbf{S}^{\mathrm{r}}}$ (Algorithm 11 or 12; LDA only)

Step 2:

Obtain the matrix $\mathbf{U}^m$ from $\hat{\mathbf{B}}$ (PCA) or $\hat{\mathbf{S}^{\mathrm{r}}}^{-1}\mathbf{B}$ (LDA)

$\mathbf{X} \leftarrow \mathbf{X}^{\mathrm{r}}\mathbf{U}^m,\ \mathbf{X} \in \mathbb{R}^{N \times m}$

Step 3:

Compute $\mathbf{F}$ (same for all samples) ▷ See (7.4)

$\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m] \leftarrow (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}$
where $\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}]$

for all $k \in \mathcal{S}$ **do**

$\hat{\mathbf{x}}'_{\mathbf{k}} = [\delta_1(k), \ldots, \delta_m(k)],\ \delta_j(k) \leftarrow \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k)$

end for

for $i \leftarrow 1, N$ **do**

$\mathbf{z}_i{}' = [\rho_1^i, \ldots, \rho_m^i],\ \rho_j^i \leftarrow x_j^i - \delta_j(k^i)$

end for

$\mathbf{Z}' = [\mathbf{z}_1, \ldots, \mathbf{z}_N]$

$\hat{\mathbf{S}} \leftarrow \frac{1}{N-1}\mathbf{Z}'\mathbf{Z}$

Use $(\hat{\mathbf{x}}_{\mathbf{k}}, \hat{\mathbf{S}})$ in the attack step

Algorithm 9 (Compute $\hat{\mathbf{B}}$ for S-PCA/S-LDA)

Require: $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$

Compute $\mathbf{F}$ (same for all samples) ▷ See (7.4)

$\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_{m^{\mathrm{r}}}] \leftarrow (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}^{\mathrm{r}}$
where $\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}]$

for all $k \in \mathcal{S}$ **do**

$\hat{\mathbf{x}}^{\mathrm{r}'}_{\mathbf{k}} = [\delta_1(k), \ldots, \delta_{m^{\mathrm{r}}}(k)],\ \delta_j(k) \leftarrow \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k)$

end for

$\hat{\mathbf{x}}^{\mathrm{r}} \leftarrow \frac{1}{|\mathcal{S}|}\sum_{k \in \mathcal{S}} \hat{\mathbf{x}}^{\mathrm{r}}_{\mathbf{k}}$

$\hat{\mathbf{B}} \leftarrow \sum_{k \in \mathcal{S}} (\hat{\mathbf{x}}^{\mathrm{r}}_{\mathbf{k}} - \hat{\mathbf{x}}^{\mathrm{r}})(\hat{\mathbf{x}}^{\mathrm{r}}_{\mathbf{k}} - \hat{\mathbf{x}}^{\mathrm{r}})'$

we use these vectors to compute $\hat{\mathbf{B}}$ (see Algorithm 9), and in the second step, we obtain $\mathbf{U}^m$ as the eigenvectors of $\hat{\mathbf{B}}$ (see Section 4.3.2).

Algorithm 10 (Compute $\hat{\mathbf{B}}$ for T-PCA/T-LDA)

Require: $\mathbf{X}_k^{\mathrm{r}} \in \mathbb{R}^{n_{\mathrm{p}} \times m^{\mathrm{r}}}$, $\forall k \in \mathcal{S}_s$

1: **for all** $k \in \mathcal{S}_s$ **do**

2: $\quad \bar{\mathbf{x}}_k^{\mathrm{r}} \leftarrow \frac{1}{n_{\mathrm{p}}} \sum_{i=1}^{n_{\mathrm{p}}} \mathbf{x}_{ki}^{\mathrm{r}}$

3: **end for**

4: $\bar{\mathbf{x}}^{\mathrm{r}} \leftarrow \frac{1}{|\mathcal{S}_s|} \sum_{k \in \mathcal{S}_s} \bar{\mathbf{x}}_k^{\mathrm{r}}$

5: $\hat{\mathbf{B}} \leftarrow \sum_{k \in \mathcal{S}_s} (\bar{\mathbf{x}}_k^{\mathrm{r}} - \bar{\mathbf{x}}^{\mathrm{r}})(\bar{\mathbf{x}}_k^{\mathrm{r}} - \bar{\mathbf{x}}^{\mathrm{r}})'$

Algorithm 11 (Compute $\hat{\mathbf{S}}^{\mathrm{r}}$ for S-LDA)

Require: $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$

1: Compute $\mathbf{F}$ (same for all samples) ▷ See (7.4)

2: $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_{m^{\mathrm{r}}}] \leftarrow (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}^{\mathrm{r}}$
where $\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}]$

3: **for** $i \leftarrow 1, N$ **do**

4: $\quad \mathbf{z}_i{}' = [\rho_1^i, \ldots, \rho_{m^{\mathrm{r}}}^i],\ \rho_j^i \leftarrow x_j^i - \delta_j(k^i),\ \delta_j(k^i) \leftarrow \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k^i)$

5: **end for**

6: $\mathbf{Z}' = [\mathbf{z}_1, \ldots, \mathbf{z}_N]$

7: $\hat{\mathbf{S}}^{\mathrm{r}} \leftarrow \frac{1}{N-1}\mathbf{Z}'\mathbf{Z}$

T-PCA

My second PCA method for stochastic models, which I call *T-PCA*, is based on the observation that the matrix $\mathbf{B}$ in (4.8) may be approximated from only a subset $\mathcal{S}_s \in \mathcal{S}$ of values k. Therefore, in the first step, we obtain *raw* traces for the subset $\mathcal{S}_s$, and we use the resulting leakage matrices $\mathbf{X}_k^{\mathrm{r}}$ to compute the matrix $\hat{\mathbf{B}}$ (see Algorithm 10). In the second step, we obtain $\mathbf{U}^m$ as the eigenvectors of $\hat{\mathbf{B}}$. Note that for this method (as well as for *T-LDA*, described next), we need two sets of *raw* traces: (a) the traces in $\mathbf{X}^{\mathrm{r}}$ (used in step 2 and then, compressed, in step 3), and (b) the traces for the matrices $\mathbf{X}_k^{\mathrm{r}}$.

Algorithm 12 (Compute $\hat{\mathbf{S}}^{\mathrm{r}}$ for T-LDA)

Require: $\mathbf{X}_k^{\mathrm{r}} \in \mathbb{R}^{n_{\mathrm{p}} \times m^{\mathrm{r}}}$, $\forall k \in \mathcal{S}_s$

1: **for all** $k \in \mathcal{S}_s$ **do**

2: $\quad \bar{\mathbf{x}}_k^{\mathrm{r}} \leftarrow \frac{1}{n_{\mathrm{p}}} \sum_{i=1}^{n_{\mathrm{p}}} \mathbf{x}_{ki}^{\mathrm{r}}$

3: **end for**

4: $\hat{\mathbf{S}}^{\mathrm{r}} \leftarrow \frac{1}{(n_{\mathrm{p}}-1)|\mathcal{S}_s|} \sum_{k \in \mathcal{S}_s} \sum_{i=1}^{n_{\mathrm{p}}} (\mathbf{x}_{ki}^{\mathrm{r}} - \bar{\mathbf{x}}_k^{\mathrm{r}})(\mathbf{x}_{ki}^{\mathrm{r}} - \bar{\mathbf{x}}_k^{\mathrm{r}})'$

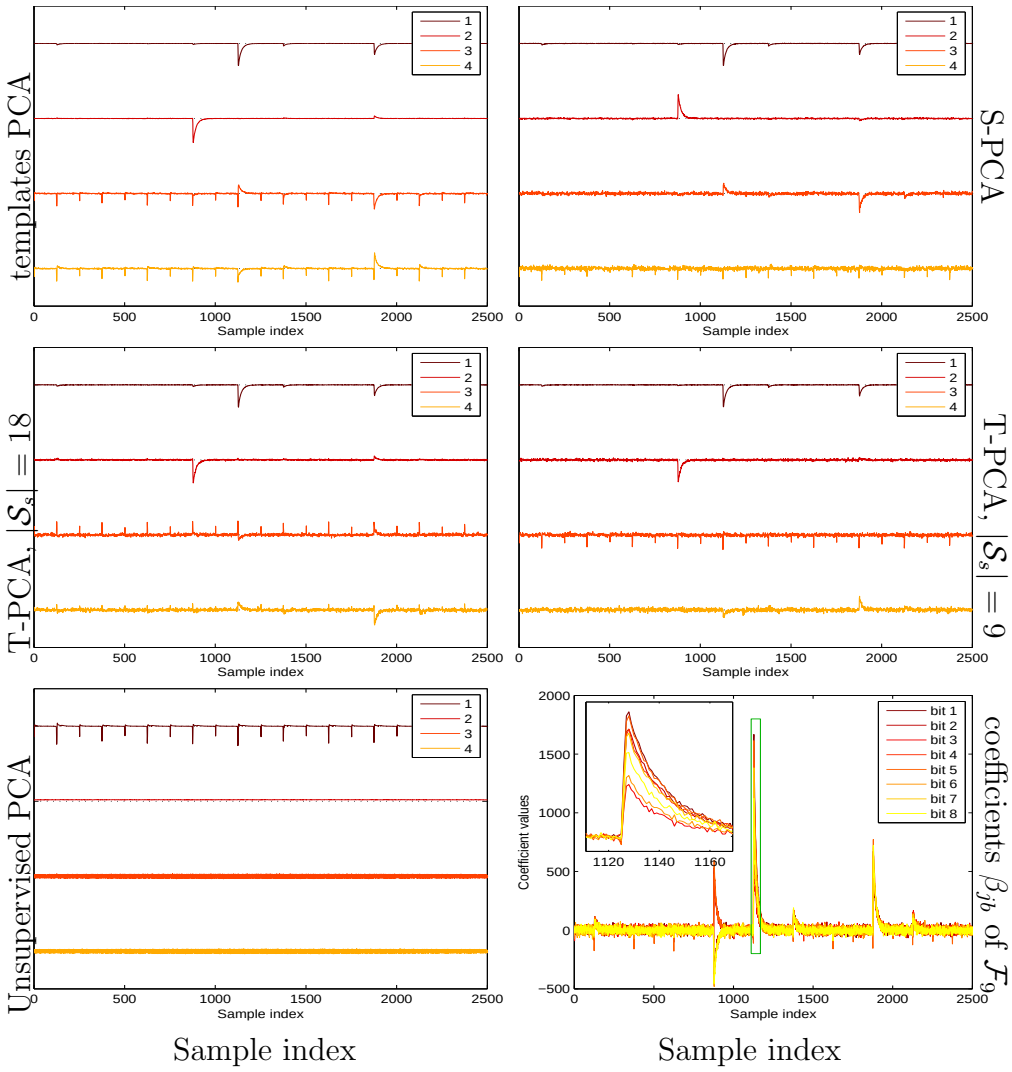

Figure 7.3: Normalized eigenvectors for PCA methods, and coefficients β_{jb} of $\mathcal{F}_9$.

S-LDA and T-LDA

I propose two methods for using LDA with stochastic models, which I name *S-LDA* and *T-LDA*. These are very similar to their PCA counterparts, with S-LDA using Algorithm 9, and T-LDA using Algorithm 10, to compute $\hat{\mathbf{B}}$. The main difference is that, besides the matrix $\hat{\mathbf{B}}$, we also need to compute the covariance matrix $\hat{\mathbf{S}}^{\mathrm{r}} \in \mathbb{R}^{m^{\mathrm{r}} \times m^{\mathrm{r}}}$ of the *raw* traces. Then, we can obtain $\mathbf{U}^m$ from the eigenvectors of $\hat{\mathbf{S}}^{\mathrm{r}\,-1}\hat{\mathbf{B}}$, as explained in Section 4.3.3. Algorithms 11 and 12 show how to obtain $\hat{\mathbf{S}}^{\mathrm{r}}$ for S-LDA and T-LDA, respectively.

In Figure 7.3, I show the first four PCA eigenvectors of the *Grizzly* dataset for template PCA, S-PCA, T-PCA with different subsets $\mathcal{S}_s$, and the unsupervised PCA of Heuser et al. [52], along with the coefficients β_{jb}. For the unsupervised PCA, the eigenvectors fail to provide useful information. For the other methods, the first two eigenvectors are very similar. This suggests that S-PCA and T-PCA can produce eigenvectors similar to those from template attacks.

7.3 Evaluation on 8-bit data

As I mentioned at the beginning of this chapter, I use the *Grizzly Beta* dataset, from Section 2.3.1, to to compare the template attacks (TA) with stochastic models (SM). The target of these attacks is again the value k, processed by a single LOAD instruction. Due to the architecture of the Atmel XMEGA 256 microcontroller, the value k affects the traces over several clock cycles. In these evaluations, as well as in the evaluations on the *Panda* dataset from the next section, I use the linear discriminant from Section 4.4.3 for the attack step, and the logarithmic guessing entropy from Section 4.5.1 to measure the success of the attack.

In Figure 7.4, I show the results of SM using my PCA/LDA methods, along with TA using PCA/LDA ($m = 4$ for all the evaluations using PCA or LDA). For TA, I used $n_\mathrm{p} = 1000$ traces per value k during profiling, while for SM, I used different N and subsets $\mathcal{S}_s$. I also show the results for SM and TA using *1ppc* ($m = 10$) and *20ppc* ($m = 80$), computed using the absolute difference of means (see Section 4.3.1).

From these figures, we can observe several things. Firstly, it is clear that all the SM methods provide a guessing entropy equal or better than their TA equivalent, even when supplied with a much smaller amount of training data. Therefore, these results confirm the observations of Standaert et al. [103], that SM can be at least one order of magnitude more efficient than TA. Theoretically, given enough training data, SM cannot perform better than TA. However, with a limited number of profiling traces, SM may outperform TA when the leakage is modelled well by the chosen base functions. With the 256×1000 profiling traces of the *Grizzly* dataset, SM reaches nearly 0-bit guessing entropy with 1000 attack traces, whereas TA does not (Fig. 7.4, bottom right). Furthermore, if we want to use profiled attacks against data having more than 8 bits, as I show in the next section, the SM may be the only practical choice.

Secondly, we can observe that both S-PCA and T-PCA reach the TA boundary quicker than S-LDA and T-LDA. This is because the PCA methods only depend on $\hat{\mathbf{B}}$ (the approximation of $\mathbf{B}$), while the LDA methods depend on both $\hat{\mathbf{B}}$ and $\hat{\mathbf{S}}^\mathrm{r}$.

Thirdly, we observe that, for large n_a, the *T-PCA*, *T-LDA*, *S-PCA*, *S-LDA*, and *20ppc* methods provide similar results, but for small n_a, the best results are obtained by LDA. In particular, note that, using *T-LDA* and *S-LDA*, we can reach 4.1 bits of entropy when $n_\mathrm{a} = 1$, while this limit is unreachable for *1ppc* (5.7 bits), *20ppc* (4.5 bits) or PCA (4.7 bits).

From Chapter 4, we knew that PCA and LDA are the most efficient compression methods. Now, we have seen that my PCA/LDA implementations for SM can achieve the same performance. On the other hand, the SM provide more efficient profiling than TA and, moreover, the SM may be the only viable solution to implement profiled attacks against more than 8-bit targets. Therefore, the *S-PCA*, *S-LDA*, *T-PCA* and *T-LDA* methods

really combine the best compression methods (PCA, LDA) with the most efficient profiled attack (SM).

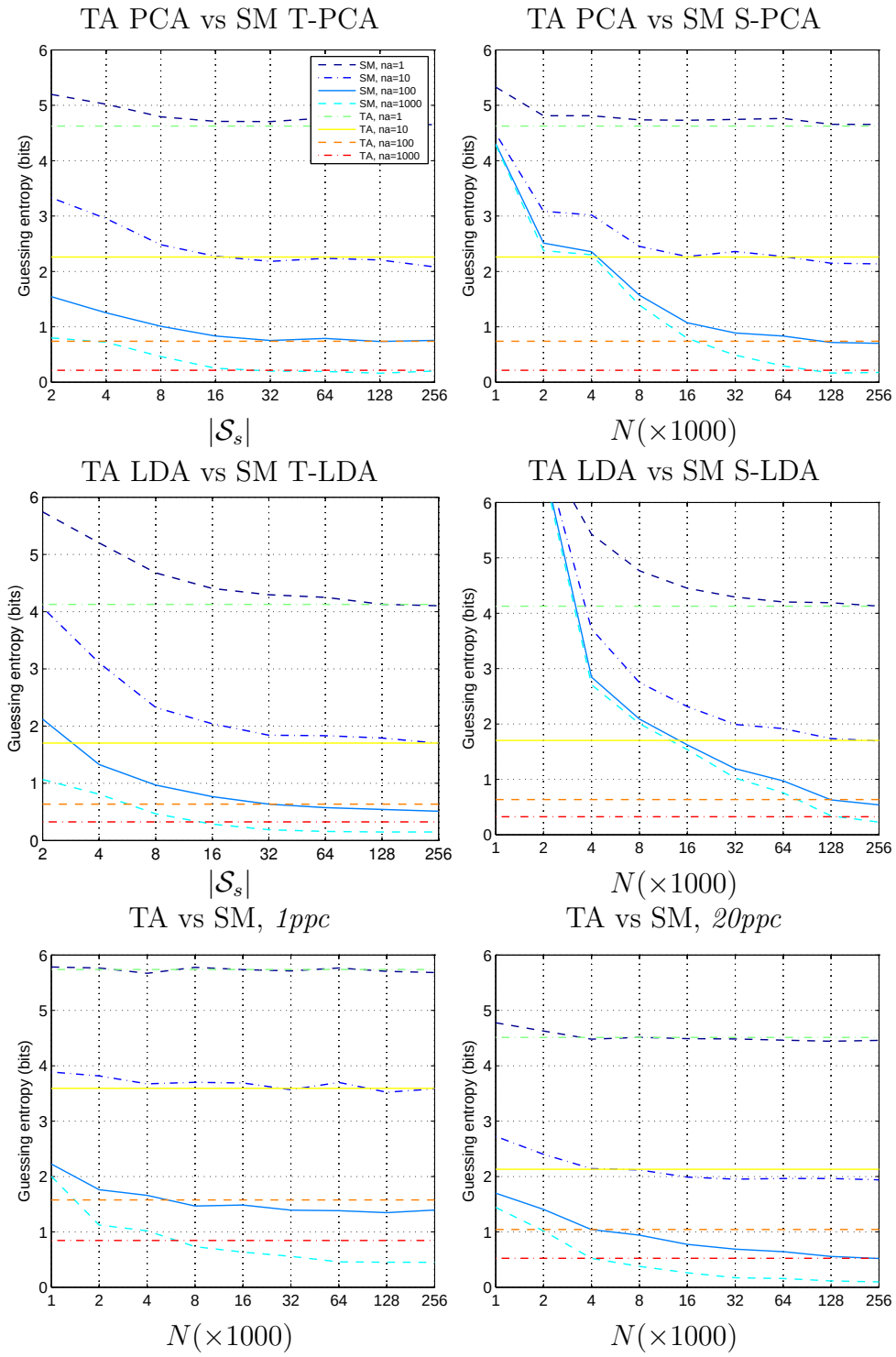

Figure 7.4: Comparing TA with SM using PCA, LDA, *1ppc* and *20ppc* with different N and n_a. For TA, I used $n_\mathrm{p} = 1000$. For *T-PCA* and *T-LDA*, I used $N = 16000$.

7.4 Profiled attacks on 16-bit data and more

So far, most publications on profiled attacks have focused on 8-bit attacks. The possibility of attacking 16-bits was mentioned in passing [52], but I am not aware of any public description of the challenges involved in attacking 16-bit data. Therefore, in this section, I consider and demonstrate a profiled 16-bit attack.

7.4.1 Considerations for the attacker

It is not feasible to mount a template attack on much more than 8 bits, as we need to obtain leakage traces for *each* value k to compute and store the mean vectors $\bar{\mathbf{x}}_k$. However, for the stochastic model, all we need is a selection of traces from a random subset of values k, to estimate the coefficient vectors $\mathbf{v}_j$, from which we can derive any desired stochastic mean vector $\hat{\mathbf{x}}_\mathbf{k}$. The remaining limitation is that, in the attack phase, we still need to compute the discriminant $\mathrm{d}_{\mathrm{LINEAR}}$ from (4.24) over all possible values k. While doing so for 2^{32} candidate values is no problem with normal PCs, attempting to do this for 2^{64} candidates would certainly require special hardware.

7.4.2 Considerations for evaluation laboratories

Even if SM are practical given a *single* attack trace $\mathbf{x}_i$, a problem that remains, in particular for evaluation labs, is computing the guessing entropy (see Section 4.5.1), which requires to store n_a traces for *each* value $k\star \in \mathcal{S}$ and run the attack on each of these. This is not practical for values having 16 bits or more. However, one practical solution is to run the attack merely over a subset $\mathcal{S}_s$ of the target values $k\star$, to compute the *partial* guessing entropy defined in Section 4.5.1.

7.4.3 Efficient attacks and evaluations on more than 8-bit

As explained in Section 4.4.4, the complexity of $\mathrm{d}_{\mathrm{LINEAR}}$ is $O(m^2+n_\mathrm{a}\cdot m)$. However, that implies the use of a covariance in (4.24). But with LDA, we no longer use a covariance matrix (see Section 4.3), so the complexity of $\mathrm{d}_{\mathrm{LINEAR}}$ reduces to $O(m+n_\mathrm{a}\cdot m) = O(n_\mathrm{a}\cdot m)$. Then, an attacker who simply wants to find the most likely k, requires a computation of complexity $O(|\mathcal{S}|\cdot n_\mathrm{a}\cdot m)$ when using LDA (since we need to compute $\mathrm{d}_{\mathrm{LINEAR}}$ for each $k\in\mathcal{S}$), and $O(|\mathcal{S}|(m^2+n_\mathrm{a}\cdot m))$ when using PCA or sample selection. If n_a is of a lower order than m, then the use of LDA will provide a computational advantage to an attacker. Also, both PCA and LDA will typically select an m much smaller than for sample selection. In my experiments, on the *Grizzly* dataset, $m=4$ for PCA and LDA, while for *20ppc*, $m=80$ (note that with sample selection we should use many samples

per clock – see Chapter 4). In the extreme case, for $n_\mathrm{a} = 1$, an attack using LDA will be 1600 times faster than using *20ppc*, and PCA will be be 400 times faster than *20ppc*. For larger traces, covering many clock cycles (e.g. for a cryptographic algorithm), I expect this difference to increase. Therefore, my PCA and LDA implementations for SM can offer great computational advantage.[2]

An evaluator who wants to compute the partial guessing entropy will run the attack for each $k\star \in \mathcal{S}_s$. Therefore, the complexity of the evaluation is $O(|\mathcal{S}_s| \cdot |\mathcal{S}| \cdot n_\mathrm{a} \cdot m)$ for LDA and $O(|\mathcal{S}_s| \cdot |\mathcal{S}| \cdot (m^2 + n_\mathrm{a} \cdot m))$ for PCA or sample selection. However, we can optimise the computation of the partial guessing entropy by precomputing $\mathbf{y}_k = \hat{\mathbf{x}}'_\mathbf{k} \hat{\mathbf{S}}^{-1}$, and $z_k = \mathbf{y}_k{}' \hat{\mathbf{x}}_\mathbf{k}$, which require a computation of $O(|\mathcal{S}|m^2)$. With these values, the discriminant $\mathrm{d}_\mathrm{LINEAR}$ can be computed as

$$\mathrm{d}^{\mathrm{fast}}_\mathrm{LINEAR}(k \mid \mathbf{X}_{k\star}) = \mathbf{y}_k{}' \left(\sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) - \frac{n_\mathrm{a}}{2} z_k, \tag{7.15}$$

which has complexity $O(n_\mathrm{a} \cdot m)$. Therefore, the evaluation of the partial guessing entropy can be done with complexity $O(|\mathcal{S}|m^2 + |\mathcal{S}_s| \cdot |\mathcal{S}| \cdot n_\mathrm{a} \cdot m)$. For PCA, the value m may be comparable to or smaller than $|\mathcal{S}_s|$ and therefore an evaluation using $\mathrm{d}^{\mathrm{fast}}_\mathrm{LINEAR}$ will run as fast as an evaluation using LDA. However, if we need to use a sample selection method with very large m, then the evaluation will be considerably slower. Remember also that, while *1ppc* with low m may be as fast as LDA in this case, we confirmed in Figure 7.4 that both PCA and LDA provide better results than *1ppc*.

These considerations show that the choice of compression method depends also on who will need to use it: an attacker who only wants the correct $k\star$, or an evaluator who wants to know the average attack cost. In both cases, my LDA and PCA methods will help.

7.4.4 Results on 16-bit data

In order to verify that an attack on 16-bit data is feasible, and to obtain an estimate on the actual run time, I used the *Panda* dataset from Section 2.3.3, where the target is composed of two 8-bit values, processed by two consecutive LOAD instructions, thus providing a 16-bit target $k\star$. Using this dataset, I cannot evaluate the limit of SM on a 16-bit parallel bus, but I can evaluate the feasibility of profiled attacks on more than 8 bits of data. For the evaluation, I split the dataset into two disjoint sets, for profiling and attack. Also, I use the $n_\mathrm{p} = 1000$ traces, obtained for each of the 16-bit values $k \in \mathcal{S}_s$, where the subset has $|\mathcal{S}_s| = 512$ random values, for the computation of $\hat{\mathbf{B}}$ and $\hat{\mathbf{S}}^\mathrm{r}$ in *T-PCA* and *T-LDA*. For the implementation of the SM, I simply extended the set of base functions to include the individual contributions of all the 16 bits of the values k,

[2] I also note that, for SM with sample selection, we should use *bnorm* (see Section 7.2.1), as it is more computationally efficient than the other methods for estimating the signal-strength estimate $\mathbf{s}(t)$.

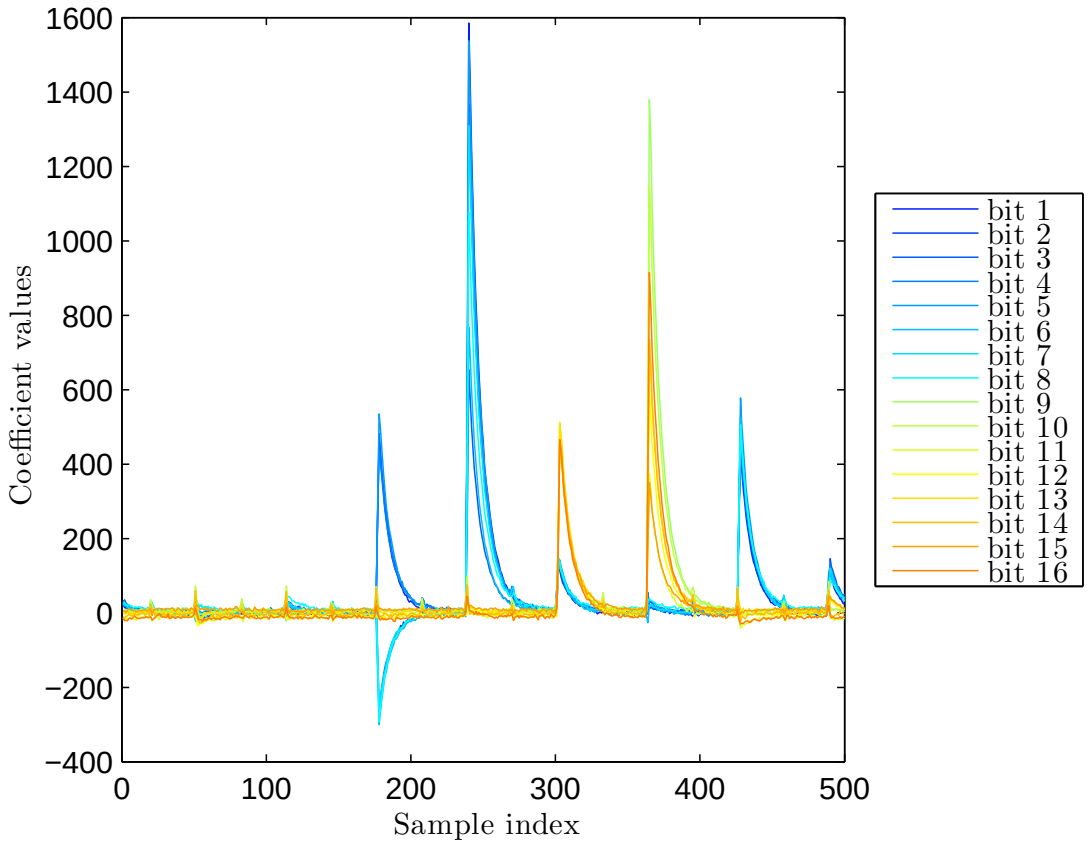

Figure 7.5: Contribution of each coefficients β_{jb} of $\mathcal{F}_{17}$ for the *Panda* dataset from Section 7.4.4.

resulting in the set $\mathcal{F}_{17}$. The contribution of each base function is shown in Figure 7.7. This figure shows the effect of pipelining in my traces, since the values processed by the two target LOAD instructions can influence the same portion of the traces (e.g. around sample 300).

In Figure 7.6, I show the results of *S-PCA*, *S-LDA*, *T-PCA*, *T-LDA*, and *20ppc* for the full 16-bit attack. We see that, with most methods, the guessing entropy converges after about $N = 1000 \times 2^4 = 16000$ traces, which confirms the efficiency of stochastic models. Then, we see that *S-LDA* reduces the guessing entropy below 4 bits when using $n_a = 100$ traces, which means that, in this case, we can find the correct $k\star$ in a brute-force search attack with 16 trials, on average. We also see that *S-PCA*, *S-LDA* and *T-PCA* are better than *20ppc* but *T-LDA* is not. Also, both *S-PCA* and *S-LDA* are better than *T-PCA* and *T-LDA*. This suggests that the subset of $|\mathcal{S}_s| = 512$ values I used for the estimation of the *T-PCA*/*T-LDA* parameters $\hat{\mathbf{B}}$ and $\hat{\mathbf{S}}^{\mathrm{r}}$ were not enough to reach the full potential of PCA and LDA. Therefore, for attacks on more than 8 bits, the methods *S-PCA* and *S-LDA* may be the best option, as they can use all the available N traces with the stochastic model for both the modeling of the compressed mean vectors $\hat{\mathbf{x}}_{\mathbf{k}}$ (step 3 in Algorithm 8), as well as for the modeling of *all* the raw mean vectors $\hat{\mathbf{x}}^{\mathrm{r}}_{\mathbf{k}}$ (lines 3–5 in Algorithm 9). This in turn can result in a better estimation of the matrix $\hat{\mathbf{B}}$ (step 1 in Algorithm 8), than

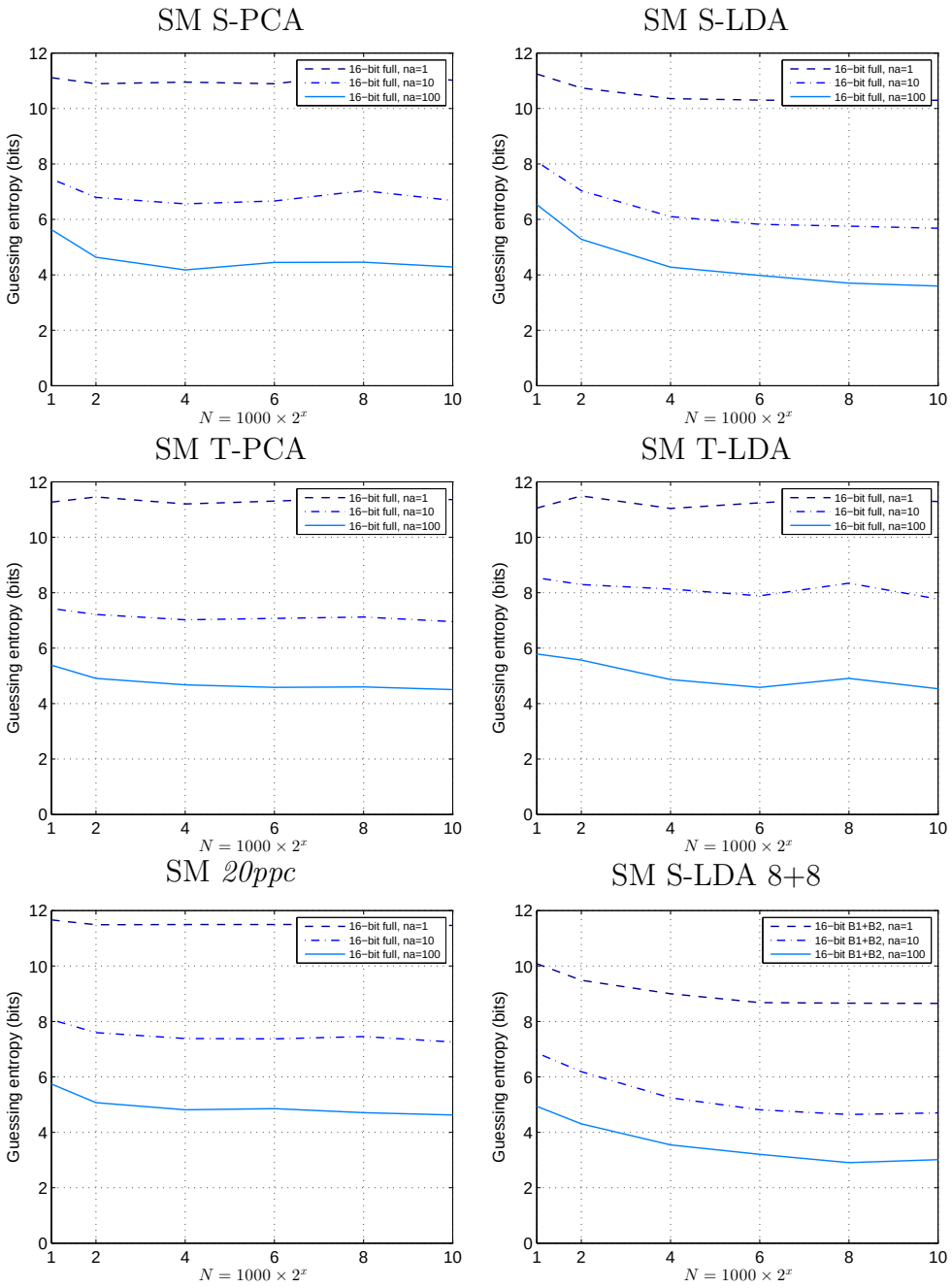

Figure 7.6: Results on full 16-bit attack for pipelined data, with *S-PCA*, *S-LDA*, *20ppc*, *T-PCA*, *T-LDA*, and results from *S-PCA* on 8 bits at a time, using different N and n_{a}. I tried $N = 1000 \cdot 2^x$, where x is the value shown in the logarithmic x-axis. For *T-PCA* and *T-LDA*, I used $|\mathcal{S}_s| = 512$.

what can be achieved with a small subset of real vectors $\bar{\mathbf{x}}_k^{\mathrm{r}}$ for the *T-PCA* and *T-LDA* methods.

In the bottom-right of Figure 7.6, I also show the results when performing the SM attack separately, for each of the two bytes of my target value (i.e. during profiling, I only consider one byte known, while the varying value of the other represents noise). I computed the

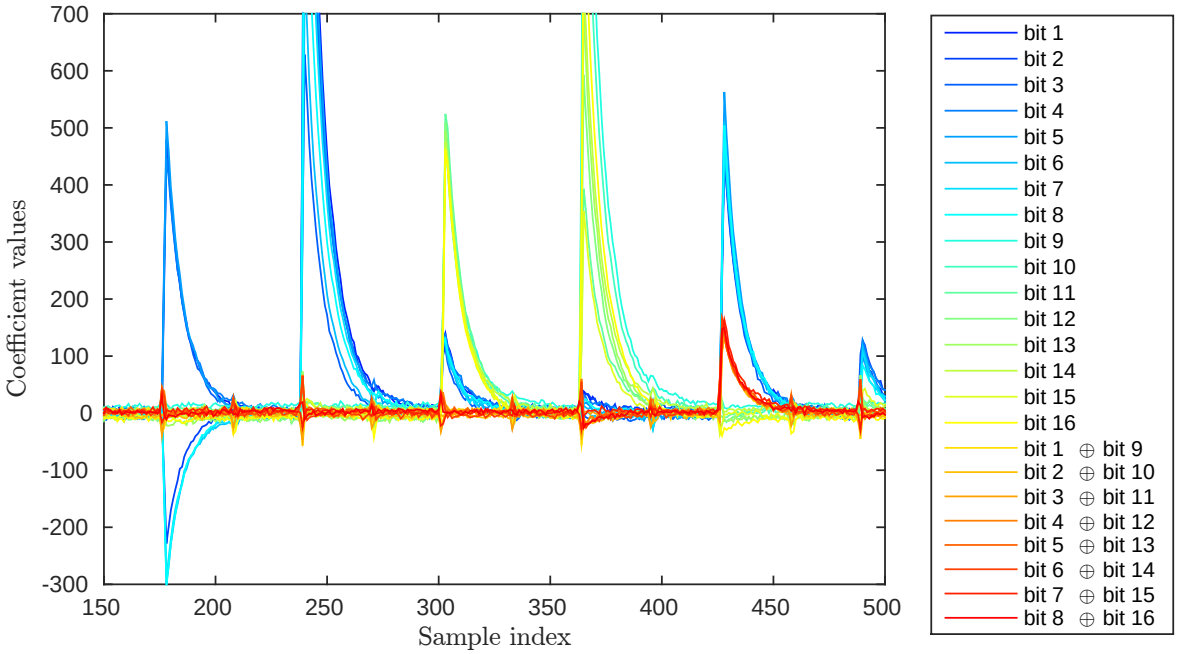

Figure 7.7: Contribution of coefficients β_{jb} in $\mathcal{F}_{17}$ (bits 1 to 16) and $\mathcal{F}_{17x}$ ($\mathcal{F}_{17}$ enhanced with XOR between bits of 8-bit halves) for the *Panda* dataset. Pipelining causes leakage of the two 8-bit halves to overlap (e.g. around sample 300). Their consecutive processing also leaks their XOR value (e.g. around sample 430).

results by adding the guessing entropy from each 8-bit attack. This figure shows that, in my particular scenario, performing two 8-bit attacks (each with $\mathcal{F}_9$) provided better results than any of the 16-bit attacks with $\mathcal{F}_{17}$. This could potentially be due to several factors. Firstly, by attacking only 8 bits, there are fewer parameters to be estimated during the attack (e.g. the SM coefficients). Secondly, the signal-to-noise ratio in the acquisition setup might have been too low to provide sufficient separation between the $|\mathcal{S}| = 2^{16}$ classes to be distinguished by our classifier. Finally, the base function set $\mathcal{F}_{17}$ may simply not have adequately modelled the leakage.

The latter turned out to be the main factor, which was easily fixed. Our 16-bit target value $k = [k_1|k_2]$ is composed of two 8-bit halves (k_1 and k_2), which are processed consecutively in the XMEGA CPU. If these two values pass through the same parts of the circuit, their XOR difference is likely to affect part of the leakage traces. Therefore, I also evaluated an attack where the stochastic model included the XOR between the bits of k_1 and k_2, resulting in the set $\mathcal{F}_{17x}$ (see Figure 7.7). Figure 7.8 shows the results of the SM attacks using S-PCA (left) and S-LDA (right) with $\mathcal{F}_{17x}$. We see that, using $\mathcal{F}_{17x}$, both S-PCA and S-LDA perform better than with $\mathcal{F}_{17}$. Also, in this case S-LDA reduces the guessing entropy to about one bit, which is far better than any of the other results, including the attack on k_1 and k_2 separately. Therefore, a 16-bit attack can perform better than two 8-bit attacks, if a model is used that also takes into consideration differences between the bits, as done in $\mathcal{F}_{17x}$.

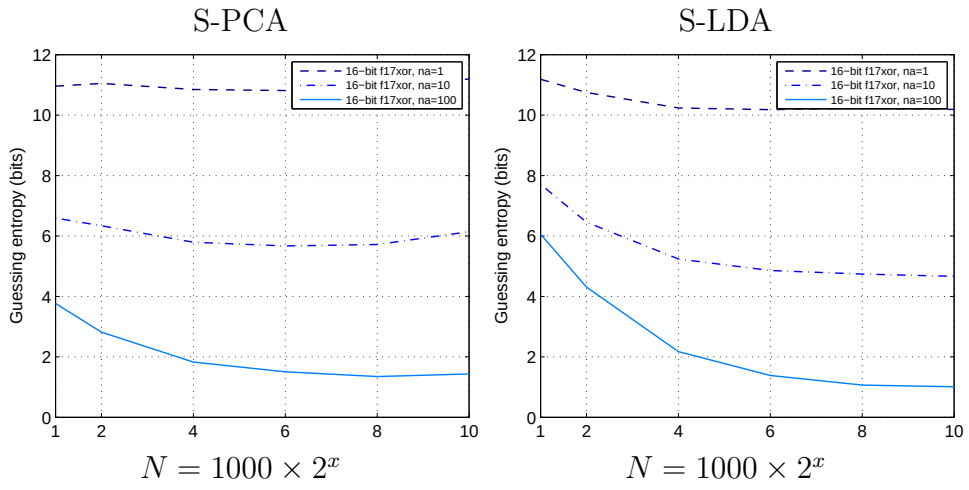

Figure 7.8: Results of SM attack using $\mathcal{F}_{17x}$ with S-PCA (left) and S-LDA (right).

Table 7.1: Approximate time required for the main steps of an evaluation using S-PCA on a 16-bit target with $N = 64000$

Step	**time**
Obtaining $\mathbf{V}$ on raw data (Algorithm 9, line 2)	40 s
Approximating raw mean vectors $\hat{\mathbf{x}}^{\mathrm{r}}_{\mathbf{k}}$ (Algorithm 9, lines 3–5)	32 s
Computing PCA parameters (Algorithm 8, line 2)	2 s
Obtaining $\mathbf{V}$ on compressed data (Algorithm 8, line 5)	38 s
Obtaining $\hat{\mathbf{x}}_{\mathbf{k}}$ for all k (Algorithm 8, lines 6–8)	28 s
Obtaining $\hat{\mathbf{S}}$ (Algorithm 8, lines 9–13)	33 s
Compute partial guessing entropy using $\lvert\mathcal{S}_s\rvert = 256$ (m of the same order as n_{a})	210 s

In Table 7.1, I show the execution times for the main steps of an evaluation using S-PCA on the 16-bit target. This table shows that SM attacks are feasible, at least computationally, on 16-bit data. All the steps can be extended for 32-bit data and more. The only steps that depend on the number of bits are the computation of the raw vectors $\hat{\mathbf{x}}^{\mathrm{r}}_{\mathbf{k}}$ and the compressed vectors $\hat{\mathbf{x}}_{\mathbf{k}}$ for all $k \in \mathcal{S}$, and the computation of the partial guessing entropy. These steps depend *linearly* on k, so a straight-forward extension to 32-bit may require 65536 times more time. That means that, for an attacker who only wants to find the most likely target value $k\star$, the attacks would take 24 days for the computation of the raw vectors $\hat{\mathbf{x}}^{\mathrm{r}}_{\mathbf{k}}$, 21 days for the computation of the compressed vectors $\hat{\mathbf{x}}_{\mathbf{k}}$ and 15 hours for the attack step. However, it seems that for an evaluator it would be impractical to compute the partial guessing entropy on 32-bit data for large $\lvert\mathcal{S}_s\rvert$.

Chapter 8

Conclusions

In this thesis, I provided an extensive evaluation of template attacks, which are considered among the most powerful side-channel attacks. They can be used to extract secret information (e.g. cryptographic keys) from tamper-resistant devices, such as the smartcards used in the banking or pay-TV industries to authenticate their customers.

While most of the previous publications have focused on attacking a particular cryptographic algorithm, such as DES, RC4, or AES, my focus has been to explore efficient multivariate statistical techniques for implementing the template attacks in a general scenario, independent of any algorithmic assumptions. For this reason, across my evaluations I targeted a *fixed* value, manipulated by a single, or at most two LOAD instructions. While this may seem a very simple scenario, in fact it is more difficult to extract the value processed by a single instruction, than extracting the key used with a software implementation of AES, because a typical implementation will process the target value (e.g. a key byte) using several instructions, therefore providing more leakage information to the attacker. Furthermore, operations of cryptographic algorithms, such as the XOR between a key byte and a known byte (e.g. plaintext), allow the attacker to obtain leakage traces for different values (the result of the XOR), which in turn permits him to determine the unknown key byte much easier, because some values are easier to identify based on their leakage than others (e.g. there is a single value, 0, with Hamming weight 0).

When I started to look at publications on template attacks, I observed that most evaluations were implementing the attack with a very small number of leakage samples, motivating this due to problems with the matrix inversion that is necessary during the attack. Furthermore, one publication mentioned that we do not need to consider more than one leakage sample per clock cycle, since additional leakage samples per clock provide no additional information. By implementing the attack using many different parameters (number of leakage traces for profiling and attack, number of leakage samples, compression method), I found that indeed there are some numerical issues, but not necessarily related to the matrix inversion. In Chapter 4, I presented in detail the numerical obstacles that can appear when implementing template attacks, and I provided several solutions

to overcome them. In that chapter, I also presented the main results of my extensive evaluations, showing that Principal Component Analysis (PCA) and Fisher's Linear Discriminant Analysis (LDA) can increase considerably the success of template attacks, while also reducing dramatically their computation cost. I also showed that, using good methods, we can actually implement the template attacks with a large number of leakage samples, and obtain better results than using fewer samples.

Before 2011, there were practically no publications of template attacks using different devices for the profiling and attack steps, although these attacks were specifically aimed for situations where an attacker could use one training device at will, but would like to extract some secret data from another device, that he cannot profile. For this reason, I decided to study the impact of using different devices on template attacks. The results of this study, where I used four different devices and tested five different implementations of the template attacks, were presented in Chapter 5. There, I showed that using different devices reduces tremendously the success of template attacks, as noticed also by two other publications. However, what I observed, and was missed in those two studies, is that LDA can help template attacks to perform very well even in this situation, by avoiding the variability that I observed between devices and between different acquisition campaigns on the same device. Based on these observations, I explained how to use both PCA and LDA to improve the results of template attacks with different devices.

While performing my evaluations, I noticed that changing the type of power supply, between batteries and a benchtop power supply, as well as changing the bandwidth of the oscilloscope which I used for my measurements, had a considerable effect on the correlation between leakage samples. I knew that a high degree of correlation will impede the use of a large number of samples, since in this case the covariance matrix of the leakage samples may become close to singular (i.e. not invertible). Therefore, I decided to evaluate more thoroughly the effect of these acquisition parameters (power supply, bandwidth) on the correlation between leakage samples and template attacks. The results of these evaluations were presented in Chapter 6. Using factor analysis, I showed that we can identify and isolate the effect of the acquisition parameters, and we can also identify another intrinsic factor affecting the correlation, which was present in my traces regardless of the acquisition setup. I also showed that, using factor analysis, we can synthesize arbitrary covariance matrices that can be used with template attacks. This can help simulating leakage traces having a similar structure as real leakage traces.

Finally, in Chapter 7, I showed several efficient ways to implement PCA and LDA with stochastic models, which were demonstrated as a very effective method to improve the profiling step of template attacks, by expressing the leakage samples as a linear combination of the target's bits. While PCA and LDA have been applied to classic template attacks since 2006 and 2008, respectively, there was a single attempt, in 2012, to implement PCA with stochastic models, but it was not very efficient. Using my PCA and LDA implementations, I performed an extensive comparison of stochastic models and classic

template attacks, and showed that these implementations provide the best of profiling attacks, by combining the profiling efficiency of stochastic models with the compression efficiency of PCA and LDA. Given the availability of many microcontrollers with 16 or 32 bit buses, the question arose, whether such template attacks are possible when targeting words larger than 8 bit. Therefore, in the same chapter, I presented an evaluation of stochastic models targeting a 16-bit value, and showed that, in this case, the PCA and LDA implementations are also particularly helpful, both in terms of attack success, as well as time required to perform the attack.

8.1 Directions for future research

In the past two and a half years of working on template attacks, I tried to explore many techniques to improve their efficiency, but there remain many ideas that still deserve to be explored further. I provide examples below:

- *Masking:* throughout my experiments I did not consider countermeasures, such as masking. However, this is a very important subject, and a vast amount of recent publications deal with this topic. Many of the current attacks on masking require to find particular combinations of leakage traces that allow a successful attack, but finding these combinations often requires very good knowledge of the target device, or an almost impractical computational cost. Therefore, it would be interesting to determine if we can use more efficient multivariate statistical techniques to attack masking countermeasures, and perhaps more importantly, it would be useful to find what are the real limits of such techniques, in order to determine the actual security of a device protected by masking.

- *Factor analysis:* in Chapter 6, I showed how factor analysis can be used to improve template attacks in some cases. However, the results also showed that factor analysis did not work well together with PCA. Therefore, it would be interesting to explore the causes of this result, and whether we can find better uses or implementations of factor analysis for template attacks.

- *Attacks on pipelining:* in Chapter 7, I presented template attacks, implemented using stochastic models, targeting two consecutive LOAD instructions. I showed that attacks which target the full 16-bit value (the bytes processed by the two instructions) perform worse than combining two attacks which target the two instructions separately (in each case, the values processed by the other instruction are considered as noise). One main factor for this may be the overlapping of the leakage from the two bytes during some clock cycles. However, it was clear from both attacks, that an attack in the presence of pipelining performs considerably worse compared to the

lack of it. This means that, with standard methods, attacks on single bytes, such as those that I presented, or the attacks on key scheduling presented by Oswald and Paar [83] cannot be directly extended to attacking multiple consecutive bytes. Therefore, it would be interesting to determine if it is possible to find some efficient method to remove the effect of pipelining on the leakage traces.

- *Attacks on more than 8 data lines:* it is generally possible to target only 8 bits at a time, regardless of the architecture of our target device, and consider the other parts of the device as noise. However, if we can use better equipment, providing enough resolution, and better attack methods, I think that targeting more bits will provide better results. In Chapter 7, I discussed some of the computational aspects involved when targeting more than 8 bits at once with template attacks, but my target device only had an 8-bit bus. Therefore, it would be interesting to determine the actual limits of template attacks when targeting more than 8 bits in platforms with larger data buses.

8.2 Final remarks

Overall, the results presented in these chapters represent the essence of a very large number of experiments, which I carried out in order to evaluate template attacks in different acquisition conditions (different devices, different acquisition setups), using different attack parameters (number of profiling traces, number of attack traces), trying different attack implementations (multivariate normal distribution, linear discriminant, factor analysis, stochastic model) and different targets (single LOAD instruction, two LOAD instructions with pipeline, hardware AES engine). These results show that using efficient implementations of the template attack, such as those that I presented in the previous chapters, we can determine almost perfectly an 8-bit value manipulated by a microcontroller in a single instruction. Even when using different devices, I described methods that allow us to reduce the entropy of such 8-bit value to below 1.5 bits, which is far below what was expected from Hamming weight leakage only. Therefore, with efficient methods, we can basically *"see"*, through side channels, the individual bus lines, and not just their Hamming weight.

Bibliography

[1] Data Encryption Standard. Technical Report FIPS PUB 46-3, National Institute of Standards and Technology, October 1999.

[2] Advanced Encryption Standard. Technical Report FIPS PUB 197, National Institute of Standards and Technology, November 2001.

[3] IEC/IEEE Standard for Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation – Part 1: General (Adoption of IEEE Std 488.1-2003). *IEC 60488-1 / IEEE 488.1*, pages 1–158, 2004.

[4] Common Criteria, 2014. `https://www.commoncriteriaportal.org/`.

[5] LAN eXtensions for Instrumentation. Website, 2014. `http://www.lxistandard.org/`.

[6] Dakshi Agrawal, Josyula Rao, Pankaj Rohatgi, and Kai Schramm. Templates as master keys. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin / Heidelberg, 2005.

[7] M. Aigner, S. Mangard, F. Menichelli, R. Menicocci, M. Olivieri, T. Popp, G. Scotti, and A Trifiletti. Side channel analysis resistant design flow. In *2006 IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006. Proceedings*, pages 2009–2912, 2006.

[8] Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, and Didier Moyart. Power Analysis, What Is Now Possible... In *Advances in Cryptology ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 489–502. Springer Berlin / Heidelberg, 2000.

[9] C. Archambeau, E. Peeters, F. Standaert, and J. Quisquater. Template attacks in principal subspaces. In *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 2006.

[10] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *2004 IEEE Symposium on Security and Privacy, 2004. Proceedings*, pages 3–11, 2004.

[11] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and practice of a leakage resilient masking scheme. In *Advances in Cryptology ASIACRYPT 2012*, number 7658 in Lecture Notes in Computer Science, pages 758–775. Springer Berlin Heidelberg, 2012.

[12] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In *Smart Card Research and Advanced Applications*, number 7771 in Lecture Notes in Computer Science, pages 263–276. Springer Berlin Heidelberg, 2013.

[13] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Comparative evaluation of rank correlation based DPA on an AES prototype chip. In *Information Security*, number 5222 in Lecture Notes in Computer Science, pages 341–354. Springer Berlin Heidelberg, 2008.

[14] Arthur O Bauer. Some aspects of military line communications as deployed by the german armed forces prior to 1945. In *The History of Military Communications, Proc. 5th Annual Colloquium*, 1999.

[15] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[16] Mike Bond, Omar Choudary, Steven J. Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and Skim: cloning EMV cards with the pre-play attack. In *IEEE Symposium on Security and Privacy (Oakland)*, May 2014.

[17] G. E. P. Box. Problems in the analysis of growth and wear curves. *Biometrics*, 6(4):362, 1950.

[18] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 135–152. Springer Berlin / Heidelberg, 2004.

[19] Christian Cachin. *Entropy measures and unconditional security in cryptography.* ETH University, Zürich, 1997.

[20] Dean Camera. Lightweight usb framework for avrs. Website, April 2014. `http://www.fourwalledcubicle.com/LUFA.php`.

[21] George Casella and Roger Berger. *Statistical Inference.* Duxbury Press, 2nd edition, 2001.

[22] Suresh Chari, Jutla Charanjit, Josyula Rao, and Pankaj Rohatgi. A cautionary note regarding evaluation of AES candidates on smart-cards. In *NIST AES Round 1*, 1999.

[23] Suresh Chari, Charanjit Jutla, Josyula Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1999.

[24] Suresh Chari, Josyula Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 51–62. Springer Berlin / Heidelberg, 2003.

[25] Marios O. Choudary and Markus G. Kuhn. Efficient stochastic methods: profiled attacks beyond 8 bits. In *(CARDIS) 2014*, number 8968 in Lecture Notes in Computer Science, pages 85–103. Springer Berlin Heidelberg, 2014. `http://eprint.iacr.org/2014/885/`.

[26] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In *Smart Card Research and Advanced Applications – CARDIS 2013*, number 8419 in Lecture Notes in Computer Science, pages 253–270. Springer Berlin Heidelberg, 2013. `http://eprint.iacr.org/2013/770/`.

[27] Omar Choudary and Markus G. Kuhn. Template attacks on different devices. In *Workshop on Constructive Side Channel Analysis and Secure Design (CODADE) 2014*, number 8622 in Lecture Notes in Computer Science, pages 179–198. Springer Berlin Heidelberg, 2014. `http://eprint.iacr.org/2014/459/`.

[28] Common Criteria. Security IC Platform Protection Profile, Version 1.0, 2007. `https://www.commoncriteriaportal.org/files/ppfiles/pp0035b.pdf`.

[29] Common Criteria. Infineon Technologies Smart Card IC (Security Controller) M9900 A22 and G11. Certification Report BSI-DSZ-CC-0827-V2-2014, 2014. `https://www.commoncriteriaportal.org/files/epfiles/0827V2a_pdf.pdf`.

[30] Jean-Sébastien Coron and Louis Goubin. On boolean and arithmetic masking against differential power analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, number 1965 in Lecture Notes in Computer Science, pages 231–237. Springer Berlin Heidelberg, 2000.

[31] Atmel Corporation. AVR XMEGA Microcontrollers, March 2014. `http://www.atmel.com/products/microcontrollers/avr/avr_xmega.aspx`.

[32] Renzo Derom-Franceschetto. Conception d'un simulateur d'état des registres et de consommation pour microcontrôleur. Master thesis, UCL, Louvain, August 2012.

[33] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *Smart Card Research and Applications*, number 1820 in Lecture Notes in Computer Science, pages 167–182. Springer Berlin Heidelberg, January 2000.

[34] Tim Dierks. The transport layer security (TLS) protocol version 1.2. 2008. RFC 5246.

[35] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[36] Thomas Eisenbarth, Christof Paar, and Björn Weghenkel. Building a side channel based disassembler. In *Transactions on Computational Science X*, volume 6340 of *Lecture Notes in Computer Science*, pages 78–99. Springer Berlin / Heidelberg, 2010.

[37] M. Abdelaziz Elaabid and Sylvain Guilley. Portability of templates. *Journal of Cryptographic Engineering*, 2(1):63–74, 2012.

[38] EMVCo, LLC. EMV specifications. Website, 2014. `http://www.emvco.com/`.

[39] J. Ferrigno and M. Hlavac. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[40] R. A. Fisher. The statistical utilization of multiple measurements. *Annals of Eugenics*, 8(4):376–386, 1938.

[41] Jerome H. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, 84(405):165–175, 1989.

[42] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, number 2162 in Lecture Notes in Computer Science, pages 251–261. Springer Berlin Heidelberg, 2001.

[43] C. F. Gauss. Theoria motus corporum coelestium. *Sectionibus Conicis Solem Ambientum*, 1809.

[44] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. Cryptology ePrint Archive, Report 2013/857, 2013. `http://eprint.iacr.org/2013/857/`.

[45] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2008*,

volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer Berlin / Heidelberg, 2008.

[46] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin / Heidelberg, 2006.

[47] Jacopo Giorgetti, Giuseppe Scotti, Andrea Simonetti, and Alessandro Trifiletti. Analysis of data dependence of leakage current in CMOS cryptographic hardware. In *Proceedings of the 17th ACM Great Lakes Symposium on VLSI*, GLSVLSI '07, page 7883, New York, NY, USA, 2007. ACM.

[48] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792807, 1986.

[49] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[50] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis – The Duplication Method. In *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1999.

[51] Sylvain Guilley, Philippe Hoogvorst, and Renaud Pacalet. Differential power analysis model and some results. In *Smart Card Research and Advanced Applications VI*, number 153 in IFIP International Federation for Information Processing, pages 127–142. Springer US, 2004.

[52] Annelie Heuser, Michael Kasper, Werner Schindler, and Marc Stttinger. A new difference method for side-channel analysis with high-dimensional leakage models. In *Topics in Cryptology CT-RSA 2012*, number 7178 in Lecture Notes in Computer Science, pages 365–382. Springer Berlin Heidelberg, 2012.

[53] Naofumi Homma, Sei Nagashima, Yuichi Imai, Takafumi Aoki, and Akashi Satoh. High-resolution side-channel attack using phase-based waveform matching. In *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 187–200. Springer Berlin / Heidelberg, 2006.

[54] Richard Johnson and Dean Wichern. *Applied Multivariate Statistical Analysis*. Pearson, 6th edition, 2007.

[55] Joint Interpretation Library. Application of Attack Potential to Smartcards, Version 2.9, 2013. `https://www.commoncriteriaportal.org/files/ppfiles/pp0035b.pdf`.

[56] Ian Jolliffe. *Principal Component Analysis*. John Wiley & Sons, Ltd, 2005.

[57] David Kahn. *The codebreakers: the story of secret writing*. Macmillan, 1967.

[58] Peter Karsmakers, Benedikt Gierlichs, Kristiaan Pelckmans, Katrien De Cock, Johan Suykens, Bart Preneel, and Bart De Moor. Side channel attacks on cryptographic devices as a classification problem. Technical Report 07/36, KU Leuven, 2007.

[59] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography: Principles and Protocols*. Chapman and Hall/CRC, 2007.

[60] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In *Computer Security ESORICS 98*, number 1485 in Lecture Notes in Computer Science, pages 97–110. Springer Berlin Heidelberg, 1998.

[61] Ilya Kizhvatov. Side channel analysis of avr xmega crypto engine. In *Proceedings of the 4th Workshop on Embedded Systems Security*, WESS '09, pages 8:1–8:7, New York, NY, USA, 2009. ACM.

[62] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 789–789. Springer Berlin / Heidelberg, 1999. First published in 1998. `http://www.cryptography.com/public/pdf/DPA.pdf`.

[63] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology – CRYPTO 96*, number 1109 in Lecture Notes in Computer Science, pages 104–113. Springer Berlin Heidelberg, 1996.

[64] Boris Köpf and David Basin. An Information-theoretic Model for Adaptive Side-channel Attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 286–296. ACM, 2007.

[65] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In *Advances in Cryptology CRYPTO 2013*, number 8042 in Lecture Notes in Computer Science, pages 429–448. Springer Berlin Heidelberg, 2013.

[66] Markus G Kuhn. Optical time-domain eavesdropping risks of crt displays. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 3–18. IEEE, 2002.

[67] Markus G. Kuhn. Compromising emanations: eavesdropping risks of computer displays. Technical Report UCAM-CL-TR-577, University of Cambridge, Computer Laboratory, December 2003.

[68] Olivier Ledoit and Michael Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, 88(2):365–411, 2004.

[69] Kerstin Lemke-Rust and Christof Paar. Analyzing side channel leakage of masked implementations with stochastic methods. In *Computer Security ESORICS 2007*, number 4734 in Lecture Notes in Computer Science, pages 454–468. Springer Berlin Heidelberg, 2007.

[70] Victor Lomné, Emmanuel Prouff, and Thomas Roche. Behind the scene of side channel attacks. In *Advances in Cryptology – ASIACRYPT 2013*, number 8269 in Lecture Notes in Computer Science, pages 506–525. Springer Berlin Heidelberg, 2013.

[71] Michael Luby and Charles Rackoff. How to construct pseudo-random permutations from pseudo-random functions. In *Advances in Cryptology CRYPTO 85 Proceedings*, number 218 in Lecture Notes in Computer Science, pages 447–447. Springer Berlin Heidelberg, 1986.

[72] P. C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[73] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[74] A. Theodore Markettos. Active electromagnetic attacks on secure hardware. Technical Report UCAM-CL-TR-811, University of Cambridge, Computer Laboratory, December 2011.

[75] J.L. Massey. Guessing and entropy. In *IEEE International Symposium on Information Theory*, page 204, 1994.

[76] James Clerk Maxwell. V. Illustrations of the dynamical theory of gases. – Part I. On the motions and collisions of perfectly elastic spheres. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 19(124):19–32, 1860.

[77] David McGrew and John Viega. The Galois/Counter mode of operation (GCM). *Submission to NIST Modes of Operation Process*, 2004.

[78] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards super-exponential side-channel security with efficient leakage-resilient PRFs. In *CHES 2012*, Leuven, Belgium, 2012. Springer-Verlag.

[79] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of power analysis attacks on smartcards. In *Proceedings of the USENIX Workshop on Smartcard Technology*. USENIX Association, 1999.

[80] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *Theory of Cryptography*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer Berlin / Heidelberg, 2004.

[81] Amir Moradi. Side-channel leakage through static power should we care about in practice? . Technical Report 025, 2014.

[82] David Oswald. *Implementation Attacks: From Theory To Practice*. PhD Dissertation. Bochum, Germany, 2013.

[83] David Oswald and Christof Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 207–222. Springer Berlin / Heidelberg, 2011.

[84] Christof Paar and Jan Pelzl. *Understanding Cryptography*. Springer Heidelberg, 1st edition, 2010.

[85] Telecom ParisTech. DPA Contest v4. Website, April 2014. `http://www.dpacontest.org/v4/`.

[86] Clayton R. Paul. *Introduction to Electromagnetic Compatibility*. John Wiley & Sons, 2nd edition, January 2006.

[87] Karl Pearson. Mathematical contributions to the theory of evolution.on a form of spurious correlation which may arise when indices are used in the measurement of organs. *Proceedings of the royal society of london*, 60(359-367):489498, 1896.

[88] Karl Pearson. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

[89] Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration, the VLSI Journal*, 40(1):52–60, 2007.

[90] E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *IEEE Transactions on Computers*, 58(6):799–811, 2009.

[91] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In *Smart Card Programming and Security*, number 2140 in Lecture Notes in Computer Science, pages 200–210. Springer Berlin Heidelberg, 2001.

[92] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In *Information Security Applications*, volume 3325 of *Lecture Notes in Computer Science*, pages 440–456. Springer Berlin / Heidelberg, 2005.

[93] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In *Advances in Cryptology EUROCRYPT 2011*, number 6632 in Lecture Notes in Computer Science, pages 109–128. Springer Berlin Heidelberg, 2011.

[94] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120126, February 1978.

[95] Ron Rivest. RC4. Wikipedia page, 1987. `http://en.wikipedia.org/wiki/RC4`.

[96] Ron Rivest. *Cryptography.* Handbook of Theoretical Computer Science: Algorithms and complexity. Elsevier, 1990. Chapter 13. `http://people.csail.mit.edu/rivest/Rivest-Cryptography.pdf`.

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

[98] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[99] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[100] SourceForge. Ngspice. Website, April 2014. `http://ngspice.sourceforge.net`.

[101] François-Xavier Standaert and Cedric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer Berlin / Heidelberg, 2008.

[102] François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In *Information Security and Cryptology ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267. Springer Berlin / Heidelberg, 2009.

[103] François-Xavier Standaert, François Koeune, and Werner Schindler. How to compare profiled side-channel attacks? In *Applied Cryptography and Network Security*, number 5536 in Lecture Notes in Computer Science, pages 485–498. Springer Berlin Heidelberg, 2009.

[104] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 443–461, Berlin, Heidelberg, 2009. Springer-Verlag.

[105] 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 Berlin Heidelberg, 2010.

[106] François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In *Advances in Cryptology – ASIACRYPT 2010*, number 6477 in Lecture Notes in Computer Science, pages 112–129. Springer Berlin Heidelberg, 2010.

[107] Craig Steiner. *The 8051/8052 Microcontroller: Architecture, Assembly Language, and Hardware Interfacing.* Universal Publishers, 2005.

[108] Neil Storey. *Electronics: a systems approach.* Pearson Education Limited, 4th edition, 2009.

[109] Simon Teran and Jakas. 8051 core design. Website, 2014. `http://opencores.org/project,8051,overview`.

[110] CadSoft USA. Eagle. Website, April 2014. `http://www.cadsoftusa.com/`.

[111] Wim van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[112] Peter Wright and Paul Greengrass. *Spycatcher: The candid autobiography of a senior intelligence officer.* Dell, 1988.

[113] Ping Xu, Guy N. Brock, and Rudolph S. Parrish. Modified linear discriminant analysis approaches for classification of high-dimensional microarray data. *Computational Statistics & Data Analysis*, 53(5):1674–1687, 2009.

Appendix A

EMV

EMV is a set of cryptographic protocols used for the commercial transactions involving a banking smartcard (which contains a secure microcontroller), a terminal (card reader), as well as several other parties, including the bank that issued the smartcard to its customer.

While the entire specification of EMV [38] is quite complex, covering several volumes that detail most aspects of a transaction, I now provide merely a brief overview of some of the cryptographic operations that are used during a transaction, in order to understand why such cards must be well protected against side-channel attacks.

In a standard EMV transaction, there are three main steps:

- *card authentication*: the terminal reads some data from the card and verifies a *static* RSA signature over this data. In addition, the terminal may send an unpredictable number (32-bit value) to the card, which in turn performs a *dynamic* RSA signature over some data, including the unpredictable number, and then returns this to the card. This is known as *Dynamic Data Authentication* (DDA).

- *cardholder verification*: the terminal sends the customer's PIN to the card, which then verifies if the PIN is correct, and finally returns a result to the terminal stating whether the PIN verification was successful or not. The PIN verification is not described in the EMV specification, but most likely it is performed by first computing a one-way function, such as a secure hash algorithm, and then comparing this with a stored value of the correct result. The terminal may choose to encrypt the PIN that is sent to the card, by using an RSA public key given by the card.

- *transaction authorisation*: in the last part of the protocol, the card generates a cryptographic Message Authentication Code (MAC), also known as transaction certificate, over the transaction data, which includes the amount, date, currency, and other details of the transaction, as well as an unpredictable number (UN) that aims to provide freshness of the transaction.

A.1 Side-channel targets for EMV

During the DDA step of the card authentication, the card generates an RSA signature (basically encrypting some data with the RSA private key). Therefore, a naive and unprotected implementation of this algorithm will be vulnerable to the timing side-channel attacks described by Kocher [63].

Similarly, if the terminal encrypts the PIN that is sent to the card during the cardholder verification step, the card may first perform an RSA decryption, which again could be subject to side-channel attacks.

In the last step, for the generation of the cryptographic MAC, there are several operations that could be targeted by a side-channel attack. Firstly, there is a key derivation procedure, in which a *session key* is derived from a *master key* that is permanently stored in the card. This derivation is performed using a deterministic and publicly known algorithm. Secondly, the card uses the session key with 3DES (three consecutive encryptions of DES, using two different keys – basically the two halves of the session key), to compute a MAC over the transaction data. If an attacker can recover the session key or the master key, e.g. if the keys are being transferred over a bus during this operation (see my attacks on loading data from Section 4.6), or during the key derivation procedure or MAC generation, then he can basically create a *cloned* card, and afterwards generate valid certificates for any desired transaction.

A.2 Practical security of EMV cards

As I showed above, there are many parts of the EMV protocol that may be targeted by side-channel attacks. For this reason, current EMV cards are often CC-certified using *EAL4+* or higher levels (see Section 2.1.6), which aims to give some confidence that such cards are well protected against known side-channel attacks. However, more efficient side-channel attacks, such as those presented in previous chapters, or new methods, may appear. For this reason, it is important that such cards are regularly replaced with new ones, that protect against the latest known attacks.

While today's banking cards may already be quite secure against side-channel attacks, it is also important to consider the overall implementation security of EMV. Recently, together with Bond, Murdoch, Skorobogatov and Anderson [16], we published a serious threat against the security of EMV, where deficiencies in the generation of random numbers at the terminal side allows an attacker to generate a valid transaction certificate that may be used in unlawful commercial transactions. Therefore, in such high-risk applications, it is important to provide security at all levels, including resistance to side-channel attacks, but also to many other possible threats.

Appendix B

CPA with unknown bus values

In Section 2.1.3, I explained how CPA can be used to determine the key value used with an encryption algorithm such as AES, by computing the correlation between the amplitude of the side-channel leakage and the Hamming weight of some intermediate value processed by the algorithm, such as the values $u = p \oplus k$, which are the inputs to the AES S-box.

However, I show here that this particular target value ($u = p \oplus k$) is not the best choice for CPA, and we should better use the output of the S-box $v = \text{Sbox}(p \oplus k)$.

Let's start with a simple case, where the leakage corresponding to processing the value $u = p \oplus k$ does correspond mostly to the Hamming weight: $L_u^1 \approx \text{HW}(u) = \text{HW}(p \oplus k)$. For the correct key hypothesis, $k' = k$, a CPA attack will compute the correlation $r_{L_u^1, \text{HW}(p \oplus k)}$ between L_u^1 and $\text{HW}(p \oplus k)$, which will be very close to 1. On the other hand, for a wrong key hypothesis, $k' \neq k$, we will compute the correlation $r_{L_u^1, \text{HW}(p \oplus k')} \approx r_{\text{HW}(p \oplus k), \text{HW}(p \oplus k')}$. We can write $p \oplus k' = p \oplus k \oplus k \oplus k'$. If we let X denote the random variable representing the values $p \oplus k$, and let $d = k \oplus k'$ for fixed k and k', we can see that in this case a CPA attack will compute the correlation between $\text{HW}(X)$ and $\text{HW}(X \oplus d)$, which will be high for many values d, hence making it difficult to determine the correct k. Moreover, when $d = 255$ (i.e. all bits 1), k' is the complement of k and we shall obtain a correlation very close to -1, which again can be misleading.

Let us now consider a CPA attack on the AES Sbox output v, where the leakage is well modeled by the Hamming weight of v: $L_v^1 \approx \text{HW}(v) = \text{HW}(\text{Sbox}(p \oplus k))$. For the correct key hypothesis, $k' = k$, the correlation $r_{L_u^2, \text{HW}(\text{Sbox}(p \oplus k))}$ will be again close to 1. However, for any wrong key hypothesis $k' \neq k$, the CPA attack will compute the correlation $r_{L_u^2, \text{HW}(\text{Sbox}(p \oplus k'))} \approx r_{\text{HW}(\text{Sbox}(p \oplus k)), \text{HW}(\text{Sbox}(p \oplus k'))}$. Due to the non-linearity property of the AES Sbox ($\text{Sbox}(a \oplus b) \neq \text{Sbox}(a) \oplus \text{Sbox}(b)$), this correlation will be close to zero, allowing us to distinguish easily the correct key value.

To illustrate these ideas, I simulated $N = 10000$ leakage values corresponding to $\text{HW}(p \oplus k)$ and to $\text{HW}(\text{Sbox}(p \oplus k))$, for $k = 39$ and random plaintexts, by adding some random noise to each value, obtaining a variance around 0.08 per intermediate value. In Fig-

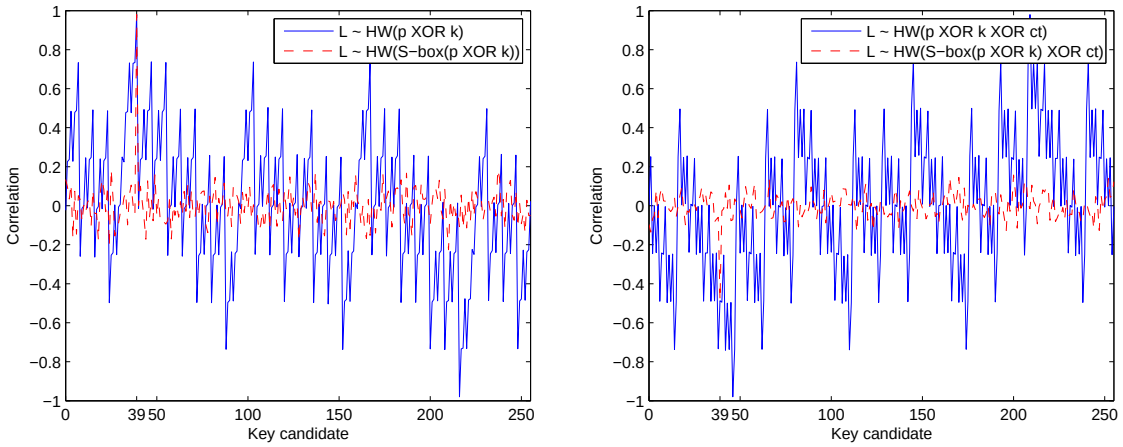

Figure B.1: Correlation after CPA attack, when the correct key is $k = 39$. Left: results when leakage depends on Hamming weight of intermediate value. Right: results when leakage depends also on some unknown constant.

ure B.1 (left), I show the correlations obtained for these two target values, using all 256 candidates k'.

In the examples above, the leakage depended directly on the intermediate values $u = p \oplus k$ or $v = \text{Sbox}(p \oplus k)$. However, it is also possible that the leakage depends on the Hamming distance $\text{HD}(u, c) = \text{HW}(u \oplus c)$ or $\text{HD}(v, c) = \text{HW}(v \oplus c)$ between the intermediate target value and some unknown constant value c, that is processed by the microcontroller before our target value. In this case, when targeting the intermediate value $u = p \oplus k$, the leakage will actually be modelled by $L_u^2 \approx \text{HW}(u \oplus c) = \text{HW}(p \oplus k \oplus c)$. For the correct key hypothesis, a CPA attack will compute the correlation $r_{L_u^2,\text{HW}(p\oplus k)} \approx r_{\text{HW}(p\oplus k\oplus c),\text{HW}(p\oplus k)}$. Considering again X the random variable for $p \oplus k$, we see that in this case we obtain the correlation between $\text{HW}(X)$ and $\text{HW}(X \oplus c)$, which depends on c. For the incorrect key hypothesis $k' \neq k$, we have the correlation $r_{L_u^2,\text{HW}(p\oplus k')} \approx r_{\text{HW}(p\oplus k\oplus c),\text{HW}(p\oplus k')}$. As before, we can write $p \oplus k' = p \oplus k \oplus c \oplus k \oplus c \oplus k'$. If we let $d = k \oplus c \oplus k'$ for fixed k, c, and k', we see that in this case we are also computing the correlation between a random variable X (corresponding to $p \oplus k \oplus c$) and $X \oplus d$, which will lead to arbitrary values depending on d. Therefore, with high probability we will not be able to identify the correct key value.

When targeting the Sbox, the leakage will be modelled by $L_v^2 \approx \text{HW}(v \oplus c) = \text{HW}(\text{Sbox}(p \oplus k) \oplus c)$. For the correct key hypothesis, we obtain the correlation $r_{L_v^2,\text{HW}(\text{Sbox}(p\oplus k))} \approx r_{\text{HW}(\text{Sbox}(p\oplus k)\oplus c),\text{HW}(\text{Sbox}(p\oplus k))}$, which will be non-zero for many values c. However, when using an incorrect key hypothesis $k' \neq k$, we obtain the correlation $r_{L_v^2,\text{HW}(\text{Sbox}(p\oplus k'))} \approx r_{\text{HW}(\text{Sbox}(p\oplus k)\oplus c),\text{HW}(\text{Sbox}(p\oplus k'))}$. Again, due to the non-linearity of the Sbox, the values $\text{Sbox}(p \oplus k)$ and $\text{Sbox}(p \oplus k')$ are uncorrelated, leading to a correlation close to zero in this case, allowing us to determine the correct key value. This is illustrated in Figure B.1 (right), where I show the results of CPA in this case, when targeting both the intermediate values $u = p \oplus k$ and $v = \text{Sbox}(p \oplus k)$.

Appendix C

Evaluation of normality

In this Appendix, I provide an evaluation of the assumption of normality for the dataset *Grizzly* (see Section 2.3.1), in order to show that such side-channel data can be considered approximately normal, and therefore the methods presented throughout this thesis that rely on this assumption can be applied. For this purpose, I shall use histograms, Q-Q plots, chi-square plots and scatter plots [54, Section 4.6]. For all the following plots, I use data from loading the fixed value $k = 208$ and $n_{\mathrm{p}} = 2000$ traces.

I start my analysis with univariate checks of normality, where the goal is to verify that the data follows the normal distribution. In Figure C.1, I show the histograms of the power consumption for the three leakage samples having the highest SNR (samples 884, 1133 and 1383).

Then, in Figure C.2, I show Q-Q plots of the same samples. These plots show the correlation of my *Grizzly* data with a standard normal distribution. If the data deviates greatly from the plot of the standard normal distribution, then we may consider the possibility of it not being normal, but these figures show that my data follows well a normal distribution.

Besides univariate checks, we can also evaluate how close our data follows a multivariate normal distribution. For this, I first show in Figure C.3 scatter plots for pairs of the three samples used earlier. Most of these points should be within an ellipse contour, as explained in Section 3.6.

Finally, we can use the statistical distance from Section 3.4 to test the normality of an arbitrary large group of samples. For this purpose, I computed the statistical distance of the $n_{\mathrm{p}} = 2000$ traces, using only the five samples with highest SNR (samples 883, 1133, 1383, 1884 and 2133). The histogram of this distance is shown in Figure C.4 (left). To check for normality, we can produce a Q-Q plot, similar to the univariate scenario, but here we compare the quantiles of the statistical distances with the chi-square distribution. This plot, known as a *chi-square plot*, is shown in Figure C.4 (right). A plot of multivariate data should resemble a straight line, passing through the origin, and having a slope 1.

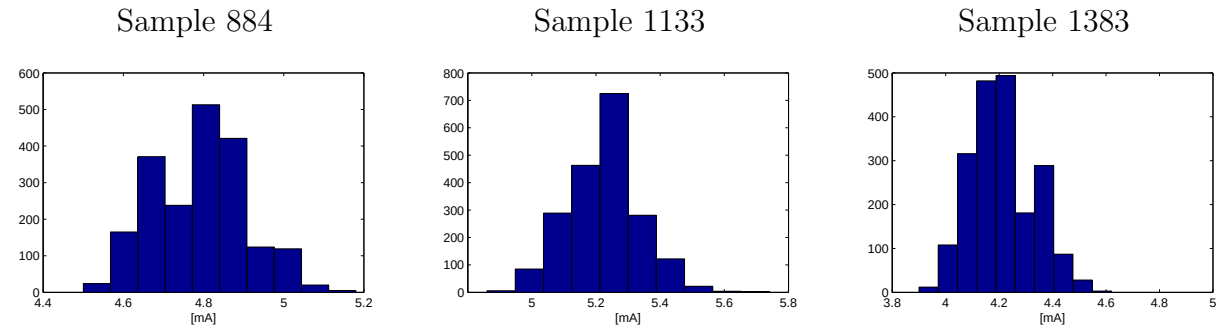

Figure C.1: Histograms of power consumption for some samples.

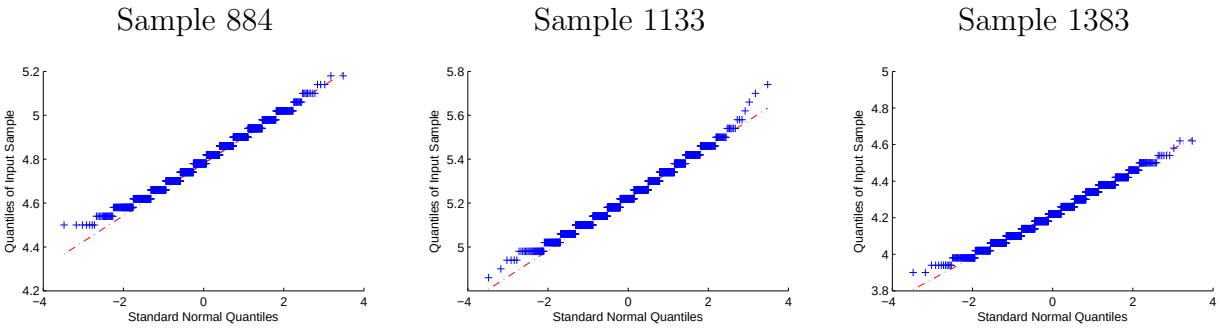

Figure C.2: Q-Q plots for some samples.

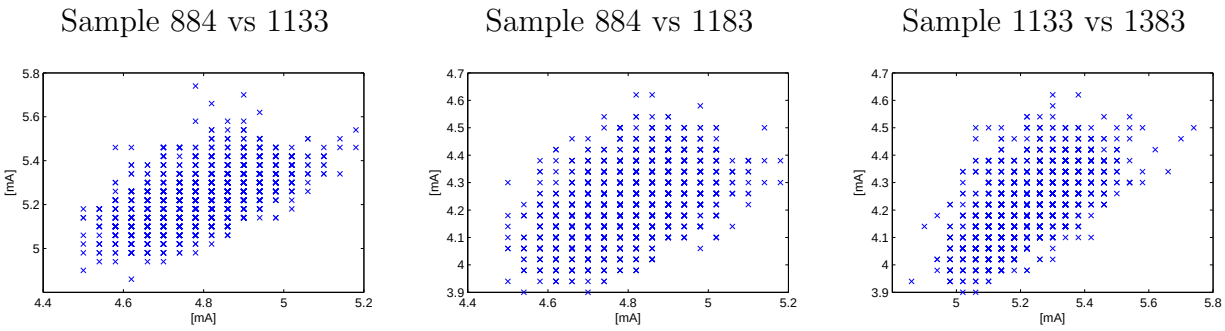

Figure C.3: Scatter plots for pairs of samples.

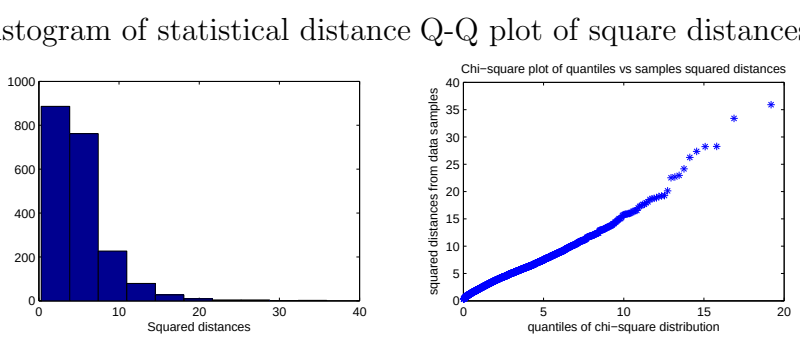

Figure C.4: Using the statistical distance to check the normality of multivariate data. Left: histogram of squared statistical distance. Right: chi-square plot. For these two plots I used 5 leakage samples.