CHANNEL ESTIMATION TO IMPROVE THE SCALABILITY OF POWER LEAKAGE BASED SIDE-CHANNEL ATTACKS ON CRYPTOGRAPHIC SYSTEMS

A Dissertation

by

SHAN JIN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Riccardo Bettati
Co-Chair of Committee,	A. L. Narasimha Reddy
Committee Members,	Paul Gratz
	Scott L. Miller
Head of Department,	Miroslav M. Begovic

May 2020

Major Subject: Computer Engineering

ABSTRACT

Side channel attacks exploit physical information that leaks from a cryptographic device in order to extract secret information, such as secret keys, passwords, or instructions that may be stored inside the device. The physical information used in side-channels can be electromagnetic or acoustic emanations, timing, power consumption, or others. A widely used form of side channels relies on the analysis of power consumption. The exploited physical information in these forms of side channel attacks is the leakage traces of the power consumed during a computation. This dissertation focusses on studying power-analysis based side-channel attacks to better understand this threat to modern cryptographic devices and their implementations.

The effectiveness of side-channel attacks is based on the fact that the physical leakages are dependent on the internal state of the device. This dependency is represented by a leakage model or leakage function. To better understand the leakage model in side-channel attacks, we propose to model the side channel as a communication channel in the traditional sense. This allows us to use a weighted leakage model and then to propose an ℓ_2-norm based re-weighted algorithm to further tune the leakage model. Compared to previous methods, our algorithm shows significant improvements in key recovery performance. Typically, secrets in cryptographic systems have a large number of bits, for example 128 bits in AES 128. Therefore, directly applying side-channel attacks that have proven effective for small secret with 8 or 16 bits, such as the Template Attack or the Stochastic Model, is computationally impossible. Most of the side-channel attacks typically apply a divide-and-conquer strategy to attempt to scale to larger number of bits. However, how to efficiently implement the Stochastic Model using divide-and-conquer is not obvious. This dissertation proposes two models to explore how to efficiently extend the Stochastic Model to non-linear cryptographic systems. The experimental results illustrate that our proposed methods show significant improvements in key recovery. Finally, how to efficiently exploit the samples in the leakage traces is always an important problem in side channel attacks. In the case of AES, side channel attacks are usually launched on either the first round or the last round of the AES encryption. We

propose an algorithm that exploits the information during both rounds, which significantly improves the key recovery. Compared to previous methods that attempt to integrate information from multiple AES rounds, such as the Algebraic Side-Channel Attacks and the Soft Analytical Side-Channel Attacks, our method brings saving in computing cost and complexity due to our pragmatic implementation.

DEDICATION

To my dear parents, dear wife, and many other important people in my life.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Professor Riccardo Bettati, without whom this dissertation would not have been possible. During the past five years, I learned from him not only the academic knowledge, but also the the attitude towards work and research. His insight into research, rigorous ethic of the work, pursuit of excellence have been of great importance to me. Without his continuous guidance, support, and encouragement, it is almost impossible for me to complete the program. It's my great honor to be his PhD student and work with him over the years.

I would like to thank Professor A. L. Narasimha Reddy for his guidance and help with my research. The advices and comments from him inspired me a lot.

I would like to thank Dr. Yang Liu in High Performance Research Computing Group of Texas A&M University. The support from him is also very important to me for continuing my PhD study at the early stage.

I would also like to thank the other committee members, Professor Paul Gratz and Professor Scott L. Miller, for their time and effort in serving my committee, and also their suggestions on my research and dissertation.

Thanks to my mates in Texas A&M University, Jiafan Wang, Hao He, Ruiqiu Miao, Lei Xu, Haifeng Jin, Wenbin Xu, Yi Cui, Di Xiao, Jiayi Huang, and Zhengcong Yin. I am always grateful for their wonderful friendship during my studies in College Station.

I would like to express my deepest thanks to my parents for their support, encouragement, and endless love in my whole life.

Last but not the least, I want to express my deepest gratitude to my lovely wife, Hongjin Long. She gave up her own promising career and has been to US to stay with me. With her companion and encouragement, I never feel lonely even at my most difficult and helpless moment.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Riccardo Bettati of the Department of Computer Science & Engineering, and Professor A. L. Narasimha Reddy, Paul Gratz and Scott L. Miller of the Department of Electrical & Computer Engineering.

The experimental data comes from open source, which includes the Grizzly datasets [1] and TeSCASE datasets [2].

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

My graduate study and research was partially supported by the Qatar National Research Foundation 9th Cycle under Grant no. 9-069-1-018.

NOMENCLATURE

AES	Advanced Encryption Standard
ASCA	Algebraic Side-Channel Attacks
BP	Belief Propagation
CPA	Correlation Power Analysis
CPU	Central Processing Unit
DC	Divide and Conquer
DPA	Differential Power Analysis
HD	Hamming Distance
HW	Hamming Weight
IoT	Internet of Things
LM	Leakage Model
LR	Least Regression
LS	Least Squares
MMSE	Minimum Mean Square Error
MSE	Mean Squared Error
PDF	Probability Density Function
RB	Ridge Regression
SASCA	Soft Analytical Side-Channel Attacks
SCA	Side-Channel Attack
TA	Template Attack

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

Over the years, side-channel attacks have increasingly shown to be successful in breaking a large number of cryptographic algorithms, implementations of cryptographic algorithms, or security and privacy measures in general. In a side-channel attack, the adversary infers a secret (the *key*) based on the observable physical information that is leaked from the cryptographic implementation. Such physical information can be electromagnetic [4] or acoustic [5] emanations, power consumption [6], or others. The effectiveness of these attacks is based on the fact that the observable physical leakages are dependent on the internal state of the system implementation. This dependency is represented as a *leakage model* or *leakage function* [7].

Power consumption models [6], which analyze the power consumed during the computation, are widely studied both in profiled and non-profiled side-channel attacks. Power consumption attacks measure the power consumption of the attacked device as it performs the computation. Fig. 5.1 shows an example of the power consumption trace of an AES-128 encryption device. It is encrypting a plaintext with a given key. The side channel attack in this case attempts to infer the key from the collected power consumption traces. In the analysis of power consumption, two a-priori models have gained prominence: they are the *Hamming Weight* (HW) [7] model and the *Hamming Distance* (HD) [8] model. The importance of these two models stems from the fact that in many systems the computation cost (and thus the power consumption) depends to some extend on the number of "ones" in the secret values, or number of bit flips in two consecutive states during the computation. These two models therefore can be applied in non-profiled attacks. In some cases, the leakage models are known a-priori, and the attacker does not need to learn these by training on the target device in advance; this kind of attack is also called *non-profiled* side channel attacks. In so-called *profiled* side channel attacks, the attacker learns the leakage model by training on the identical or on a similar target device. In profiled side channel attacks, there are two widely-used techniques: the Template Attack [9, 10] and the Stochastic Model [11]. In the *Template Attack*, the attacker develops, for each key, a special model, called a *template*, typically in form of a Gaussian

Model. The key extraction then becomes a problem of selecting the key with maximum likelihood, where the power consumption during computation with the secret key is matched against all the templates. In the *Stochastic Model*, it is assumed that the side-channel leakage can be represented as a linear model: each time-sample point of a leakage trace consists of a deterministic part and a random part, where the latter is usually considered as independent noise. The deterministic part is a linear combination of a base function and a coefficient vector. Hence, profiling the leakage function is equivalent to estimating the coefficient vector at each time-sample point.

For improving the attack's performance, a variety of approaches have been proposed to model the side channel attacks from the communication theory. In [12], it proposes a general framework to integrate the side channel attacks and communication theory together. In [13], the authors focus on modeling the side channels as AWGN channels. In the first part of the dissertation, we also focus on the specific issue of how to model a side channel as a communication channel and how to profile it. However, rather than treating the side channel model as a *noisy* channel, as in [12, 13], we treat the side channel as a *fading* channel [14]. This allows us to treat the leakage model's coefficients as channel gains and to construct a *weighted* leakage model. Then we formulate the *leakage model profiling problem* in side channel attacks as a *channel estimation problem* in communication theory. We then propose an ℓ_2-norm based re-weighted algorithm to estimate the leakage model in the second part of the dissertation. The experimental results show that our re-weighted iterative algorithm leads to better key recovery compared to other stat-of-the-art methods, such as the Least Squares [11] and the Ridge-Based method [15].

Since the secret in cryptographic system usually has a large number of bits, such as AES 128, the majority of side-channel attacks usually apply a divide-and-conquer strategy to separately recover portion of the secret key, like in [16, 17]. Under this strategy, each phase of the attack only focuses on a portion of the key, typically one byte and it treats the other portions as noise. At the end, the partial results are combined to get the key. In the context of AES, where encryption operations execute using one byte of the round key at a time, such DC attacks come naturally. We call these "independent" attacks because they treat each byte of the key independently. However,

this comes with a problem that under these independent attacks, the measurements on each byte are corrupted by both the noise and the leakages from other bytes, especially when the number of the profiling traces is limited. In the third part of this dissertation, we study the problem of how to efficiently extend the stochastic model, to non-linear cryptographic systems, such as AES [18]. More specifically, we explore how to build leakage models for the AES implementations that improve on the previous independent model. Our first method is based on the idea of taking the average of the leakage models which are built for each byte independently. This will obviously reduce the noise corruption of the leakage model estimation. We call this first model the *average measurement* model. A second method is based on a long linear leakage model. It assumes that the leakage model's coefficient for the same bit position are approximately the same for the different byte. Based on this assumption, we develop the second model, which we call the *approximated* model. The experimental results show that our models have significantly improvement in terms of key guessing compared to the state-of-the-art methods.

For the particular case of attacks on AES, divide-and-conquer approaches have proven to be effective. They are, however, realized typically in forms that do not take advantage of the power consumption data that can be collected during the entire encryption process. Rather, they attack either the early stage of the encryption, when the unencrypted text (the plaintext) may be available, or the late stages, in the more common cases where the encrypted text (the so-called ciphertext) is known. We investigate an approach to efficiently use the leakage samples from *multiple* AES rounds to improve the attack performance [1]. We set out to investigate if we can find a pragmatic way that can leverage the leakage samples from multiple rounds but at the same time also limit the computing cost. This leads us to develop what we call *reinforcement strategies* for side-channel attacks. By keeping track of possible key candidate for different rounds, and by reconciling theses sets across rounds, we enhance the power of the attack.

The rest of this dissertation is organized as follows. In Chapter 2, we introduce the power side channels. In Chapter 3, we discuss how to model a side channel as a communication channel.

[1] Some previous work has already considered this problem, such as Algebraic Side-Channel Attacks (ASCA) [19] and Soft Analytical Side-Channel Attacks (SASCA) [20]. We will discuss these approaches in Chapter 6.

Chapter 4 presents how to use the ℓ_2-norm based re-weighted algorithm to estimate the leakage model. Chapter 5 explores how to efficiently extend the Stochastic Model to the non-linear cryptographic systems. In Chapter 6, we present the reinforcement strategies based algorithm for efficiently breaking the secret in cryptographic system. We present our conclusion and lay out a path for future work in Chapter 7.

2. POWER SIDE CHANNELS

2.1 Introduction

In a power side-channel attack, the attacker tries to reconstruct a secret value (the "key") by leveraging information about power consumption that is leaked by the particular device that contains the secret. We assume that the secret key is B bit long. Hence, the set of all possible keys is $\mathcal{K}$, where $|\mathcal{K}| = 2^B$. When a secret key k is manipulated during the operation, the power traces that leak from the target device are referred to as *raw leakage traces* (vectors), which we denote as $\boldsymbol{x}^r_{ki} = [x^r_{ki}(1), x^r_{ki}(2), \ldots, x^r_{ki}(M)] \in \mathbb{R}^M$ (here r stands for 'raw'), where M is the length of the trace in time samples, and i stands for the i-th trace [9, 10, 21, 22]. Fig. 2.1 displays a sample power trace that comes in form of the current that is drawn from a 8-bit micro-controller. This sample is part of the **Grizzly** dataset [1]. We will describe the Grizzly dataset in more details in Chapter 4. In this figure, we represent the power consumption.

Due to the high sampling rates used during the data acquisition, M is usually a very large number. Often, only a few sample points, primarily where machine instruction are triggered by fixed Input/Output (I/O) activities, contain useful leakage information [10, 23, 24]. Hence, in order to extract the valid information and to save computing cost, attackers need to *compress* the length of the raw leakage vectors before further processing. We will elaborate on compression techniques in Chapter 4. After data compression, we obtain the *leakage vector* from the i-th trace of key k, which we refer to as $\boldsymbol{x}_{ki} = [x_{ki}(1), x_{ki}(2), \ldots, x_{ki}(m)] \in \mathbb{R}^m$ where m is the length of the (compressed) leakage vector.

In non-profiled side channel attacks, the attacker does not need to learn the leakage model sice it has already known. In profiled side channel attacks, the attacker needs to build a leakage model using an identical or a similar device in advance. Suppose that the attacker has recorded n_p leakage vectors for each key during the leakage model profiling. She constructs the $\boldsymbol{X}_k \in \mathbb{R}^{n_p \times m}$ *leakage matrix* by combining all of the vectors $\boldsymbol{x}_{ki}$ together. Finally, she builds the leakage model based

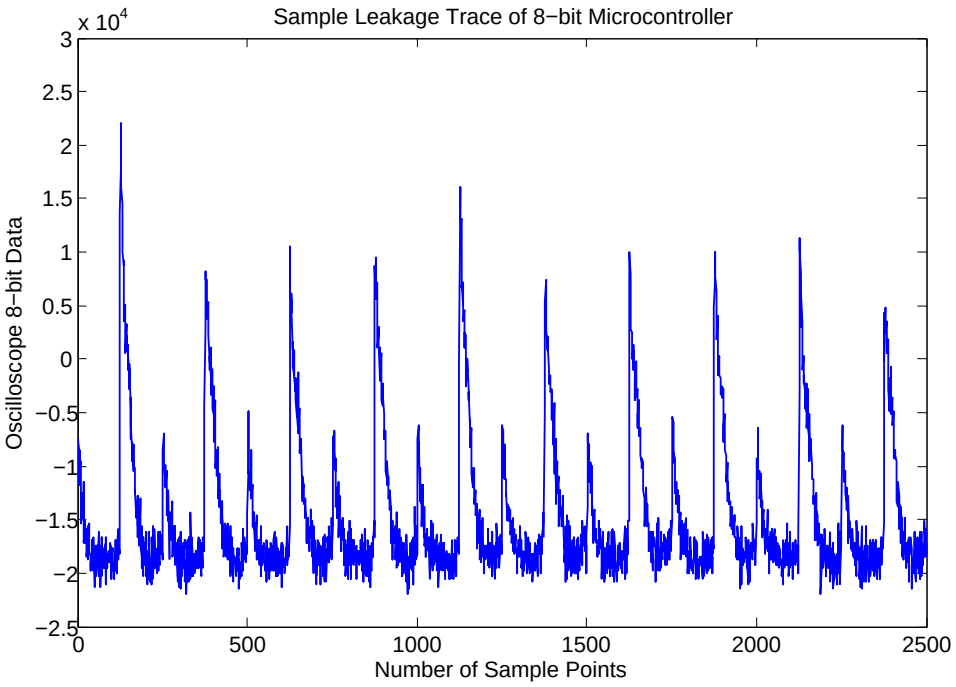

Figure 2.1: Sample of a Power Trace (Derived from the Dataset in [1])

on the leakage matrix. More details about how to learn the leakage model will be introduced in Chapter 4.

The objective of the attack is to recover the secret key from the leakage traces. Typically an attack is structured as follows:

1. The attacker has access to a set of leakage traces $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n_a}$. Each trace $\boldsymbol{x}_i$ is of the form $\boldsymbol{x}_i = [x_i(1), x_i(2), \ldots, x_i(m)] \in \mathbb{R}^m$, where m is the length of the leakage trace (in number of discrete samples), and n_a is the number of recorded traces.

2. The attacker knows that the secret key is one of the keys in $k_1, \ldots, k_{2^B}$, where B is the length of the key in bit.

3. The attacker estimates the secret key by generating an order (in decreasing likelihood) of the possible keys. The attacker succeeds if the actual key is among the most likely keys as determined by the attack.

2.2 Power Models

The likelihood of a particular key k_i in the generic attack described above is determined by how well the power model key k_i matches the leakage trace that was measured on the device for the secret key.

Two models have gained prominence: they are the *Hamming Weight* (HW) model [7], which counts the number of non-zeros in the secret string that represents the key, and the *Hamming Distance* (HD) model [8], which counts the number of bit changes between the current secret key and the reference state (the previous secret key). The importance of these two models stems from the fact that in many systems the computation cost (and thus the power consumption) depends to some extend on the number of "ones" in the values, including any secret, or on the number of bit flips in two consecutive states, that are used for the computation. In practice, however, purely HW-based or HD-based models fail to accurately reflect the available information that leaks through the side channel. In addition, both HW and HD models are limited to identify classes modulo Hamming Weight. For example, key=1 and key=128 both have a HW value of 1, and therefore an attack that is based purely on HD model would fail to distinguish these two key values. Means must be found to leverage the additional information in the leakage to identify keys beyond their HW. Thus, there is still an expectation to build more effective leakage models and also find more powerful key-extraction approaches to be used during attacks.

In recent years, profiled side-channel attacks, such as the Template Attack [9] and its advanced version, the Efficient Template Attack [10, 25], have shown excellent performance in breaking implementations of cryptographic algorithms. By creating a power model (template) for each key, typically in form of a multivariate Gaussian model, the key extraction becomes a problem of selecting the key with maximum likelihood [9, 21, 22].

In [11], rather than making *a priori* assumptions about the leakage model, the authors proposed a general approach to *learn* the leakage function based on the recorded leakage traces during profiling. The underlying assumption is that the leakage traces can be linearly represented: each time-sample point of a leakage trace consists of a deterministic part and a random part (considered

as independent noise), which can be represented as follows:

$$x_{ki}(j) = \delta_j(k) + \rho_j \ , \tag{2.1}$$

where $x_{ki}(j)$ is the j-th sample point of trace $\boldsymbol{x}_{ki}$ with $1 \leq j \leq m$, $\delta_j(k)$ is the deterministic part, and ρ_j is the random part. The random part is a random variable that is independent of the secret key and the plaintext, and it usually is modeled as noise. The deterministic part is modeled as a linear combination of a set of *base functions*: $\boldsymbol{g}_j(k) = [g_{j0}(k), g_{j1}(k), \ldots, g_{jB}(k)]$ and a *coefficients vector*: $\boldsymbol{\beta}_j = [\beta_{j0}, \beta_{j1}, \ldots, \beta_{jB}]$, which is expressed as:

$$\delta_j(k) = \sum_{b=0}^{B} \beta_{jb} \cdot g_{jb}(k) \ . \tag{2.2}$$

A common definition of the base function is:

$$g_{jb}(k) = \begin{cases} 1, \text{ where } b = 0 \\ \text{bit}_b(k), \text{ where } 0 < b \leq B \ , \end{cases} \tag{2.3}$$

where $\text{bit}_b(k)$ is the b-th bit value of the binary string of key k. In addition, in order to ensure that the measurement matrix $\boldsymbol{F}$ in Eq. (4.3), which we will introduce later, does not become singular, $g_{jb}(k) = 1$ when $b = 0$. The coefficients vector reflects the leakage model's behaviour and is unknown to the adversary before profiling. As a result, profiling the leakage function is equivalent to estimating the coefficient vector in each time-sample point. The authors in [11] call this approach the *Stochastic Model*.

The authors in [22, 26] give a solution to combine the Template Attack approach with the Stochastic Model and so improve the attack's performance. As we will show in the following sections, there is still room for improvement by focusing on the modeling of the leakage information, and the further profiling and attacking schemes based on it.

2.3 Conclusion

In this chapter, we first introduce what is the power side-channel attack and then present the generic procedure to perform it. Furthermore, we discuss the power models in both non-profiled and profiled side channel attacks. As examples of non-profiled side-channel attacks, we introduce the HW/HD models. As examples of profiled side-channel attacks, we present the Template Attack and the Stochastic Model.

3. SIDE CHANNELS AS CLASSICAL COMMUNICATION CHANNELS*

3.1 Introduction

A variety of approaches have been proposed to build the connection between side-channel attacks and traditional communication theory. The authors in [12] present a *general* uniform framework that integrates side channel attacks and classical communication theory. This framework allows to quantify the effect of a modeled leakage function. The attackers also measure the strength of the adversary through the use of information theoretic metrics, such as success rate, guessing entropy, or mutual information. The authors in [13] focus on the design of optimal side-channel distinguishers by viewing the side-channel model as a communication channel. The authors focus on side channels that can be represented as AWGN (Additive White Gaussian Noise) channels in communication theory, and the main analysis results are based on the *HW* model. The authors in [27] propose a side-channel modelling procedure for CPU-type system by incorporating the linear regression model at the instruction level. They focus on the leakage model mapping between the leakage traces and the instructions matrix, which is concatenated by the matrix of operand bits and the matrix of bit transitions that go across the buses. The authors also show that this leakage characterisation methodology is applicable to ARM Cortex M0 and M4, which are relevant for IoT (Internet of Things) setting.

In this dissertation, similarity to the previous work, we also employ the idea of formulating the side-channel model as a communication channel model in the traditional sense. However, different from those previous works, we focus on the specific issues of how to model the side channel as a communication channel and also how to do the profiling based on the proposed model. Specifically, rather than treating the side-channel model as a kind of AWGN channel, as in [12, 13], we treat the side channel as a *fading* channel [14]. Hence, the leakage model coefficients in the side channel are the channel gains in the fading channel. This allows us to treat the model coefficients as the

*Parts of this chapter are adapted with permission from "Adaptive Channel Estimation in Side Channel Attacks" by Shan Jin and Riccardo Bettati, 2018. *2018 IEEE International Workshop on Information Forensics and Security (WIFS'18)*, Dec 2018, ©2018 IEEE.

weights (gains) of the power leakage and to construct the side channel as a *weighted* leakage model. This in turn enable us to formulate the leakage model *profiling problem* in side channel attacks as a *channel estimation problem* in communication. We then perform the attack based on the profiled leakage model. In Chapter 4, we will show that this scheme has a number of benefits:

1. By mapping the leakage model to a communication channel model, the adversary can use channel-estimation (signal estimation) techniques, which have been validated in the communication research community, to do the profiling. We will show that this results in higher key-identification probabilities over other techniques, especially compared to the standard profiled side-channel attack [9] as well as more sophisticated approaches, such as [10].

2. The weighted leakage model is particularly well-suited for situations where the key space is too large to allow for an exhaustive profiling for all possible keys. The experimental results show that the weighted leakage model performs extremely well for even very small numbers of profile runs, such as small multiples of B in case of 2^B keys (B is the number of bits in the target device).

In the following, we will first introduce how to model the side channel as a communication channel. We then discuss how to construct the side channel as a weighted leakage model from the view of the communication theory in traditional sense.

3.2 Weighted Leakage Model

In this section, we generalize the leakage model proposed in [11]. To be more specific, we build a *weighted* leakage model of the power leakage information. Similarly to the previous work, described earlier, our work model the side channel as communication channel. However, different from the previous works in [12] and [13], our work uses a different view to treat the side channel by modeling it as a fading channel, rather than a general noisy channel. This has two effects: First, the coefficients of the leakage model become the gains of the channel, which are equivalent to the weights of the power leakage. Second, the coefficients profiling problem in side channel attacks

translates to a channel estimation (signals estimation) problem in communication. As a result, we can use more efficient signal estimation technologies when we do the profiling.

3.2.1 Modeling Side Channels

Figure 3.1 illustrates how side channel attacks compare to traditional communication channels. In our model, the target register is viewed as the *transmitter*, and the secret key k is the *input message*. Since the internal state of the target register is based on the state of the encryption operation $\text{Enc}_k(p)$ (for example, exclusive-or for the first round encryption in AES [18]) between key k and the (partial) plaintext p, the plaintext can be viewed as the *code* and the encryption operation as the *encoder*. However, since the adversaries' target is the internal state of the secret device *after* the encryption operation, we treat the plaintext as a constant machine word and hence use k to represent the current *internal state* in the secret device, where we treat $k := \text{Enc}_k(p)$. In this case, k can be viewed as the *transmitted signal* after the front-end processing in the transmitter. The *communication channel* is the side channel, which is represented by the leakage function. The adversary is the *receiver*, and each recorded trace is the *received signal* added with the *noise*. The adversary uses the distinguisher, which can be thought of as the *decoder*, to guess the target key. Finally, the target key is the same as the *decoded signal/decoded message*.

Note that the representation of the modeling of side channel through communication channel in Figure 3.1 differs from the description in [13], which also depicts side channel attacks in the context of general communication systems. In our case, we treat the coefficients of the leakage model (leakage function) as the gains of the communication channel and make our side channel model becomes a fading channel, not an AWGN as in [13]. This also allows us to view the coefficients as the weights of the leakage power.

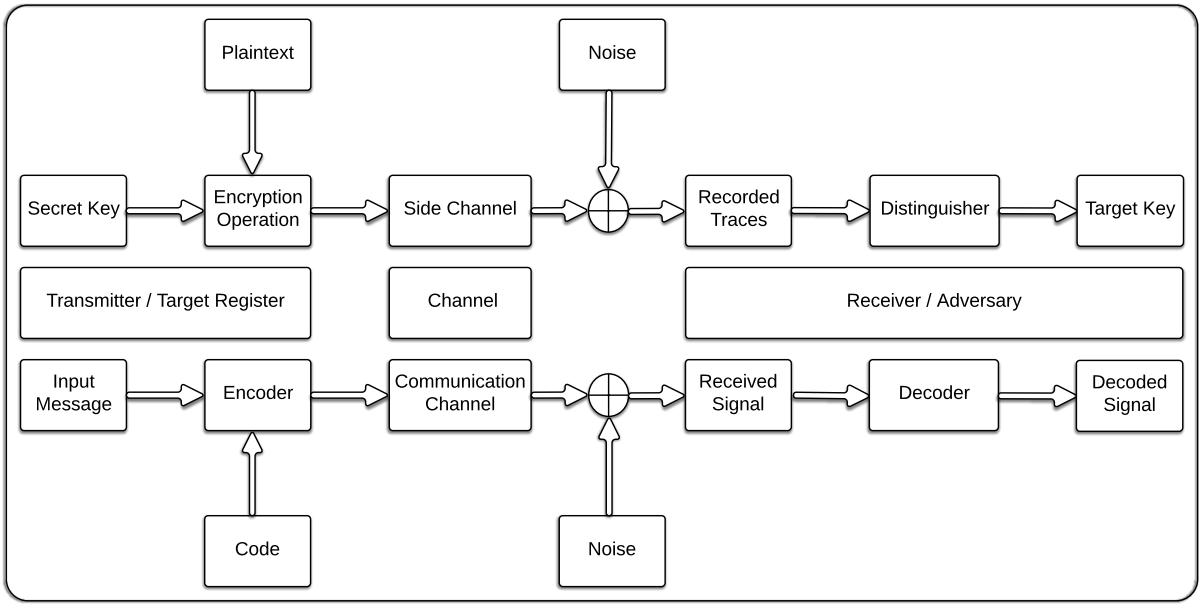

Figure 3.1: Viewing the Side Channel Model as the Communication Channel Model (Reprinted with permission from [3])

3.2.2 Mathematical Model

Based on the representation of the side channel model as a communication channel in Figure 3.1, we develop a generalized *weighted leakage model*, which we express as:

$$\boldsymbol{L} = \boldsymbol{A}\boldsymbol{H}\boldsymbol{S}_b(k) + \boldsymbol{R} \ , \tag{3.1}$$

where $\boldsymbol{L} \in \mathbb{R}^m$ is the leakage trace vector that is recorded by the adversary, $\boldsymbol{R} \in \mathbb{R}^m$ is the independent random part, which is caused by the noise, and $\boldsymbol{S}_b(k) \in \mathbb{R}^{B+1}$ is the advanced base function, which is equivalent to the transmitted signal and is defined as

$$\boldsymbol{S}_b(k) = \begin{cases} c, \text{ where } b = 0 \\ \mathcal{F}_b(k), \text{ where } 1 \leq b \leq B \ , \end{cases}$$

where c is a constant number. The definition of $\mathcal{F}_b(k)$ generalizes that of $\text{bit}_b(k)$ in Eq. (2.3) and can be adapted to different application scenarios. For example, $\mathcal{F}_b(k)$ could be ± 1 for representing a positive/negative level. We therefore call $\mathcal{F}_b(k)$ the *generalized* base function. For simplicity, we will use $\mathcal{F}_b(k) = \text{bit}_b(k)$ in the rest of this paper. $\boldsymbol{H}$ is the weight matrix of the data leakage, which is expressed as:

$$\boldsymbol{H} = \begin{bmatrix} h_{10} & h_{11} & h_{12} & \dots & h_{1B} \\ h_{20} & h_{21} & h_{22} & \vdots & h_{2B} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{m0} & h_{m1} & h_{m2} & \dots & h_{mB} \end{bmatrix} \tag{3.2}$$

where h_{ij} $(1 \leq i \leq m, 0 \leq j \leq B)$ is the weight coefficient for the data leakage. The linear combination of $\boldsymbol{S}_b(k)$ and $\boldsymbol{H}$ reflects that the data leakage of key k depends on not only *how many* number of bits are switching but also *which* specific bits are switching. $\boldsymbol{A}$ is the scalar gain matrix, which plays the same role as the *amplifier* in communication systems. It is expressed as:

$$\boldsymbol{A} = \begin{bmatrix} \alpha & 0 & 0 & \dots & 0 \\ 0 & \alpha & 0 & \vdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \alpha \end{bmatrix} \tag{3.3}$$

where α is the gain between the data leakage in each sample point and the leakage trace $\boldsymbol{L}$'s power consumption in the real circuit [*]. Since the adversary is only interested in the extraction of the secret keys based on the recorded leakage traces, the value of the gain α is of no interest. Instead, the result of the scalar gain matrix $\boldsymbol{A}$ multiplied by the weight matrix of the data leakage $\boldsymbol{H}$ can be viewed as a unified weight matrix for the leakage, which finally represents the communication

[*]Here we consider that all sample points have the same gain value. If α is different for each sample point in some circuit designs, $\boldsymbol{A}$ becomes a non-identical diagonal matrix

channel. Hence, Eq. (3.1) could be reformulated as:

$$\boldsymbol{L} = \boldsymbol{W}\boldsymbol{S}_b(k) + \boldsymbol{R} \tag{3.4}$$

where

$$\boldsymbol{W} = \boldsymbol{A}\boldsymbol{H} = \begin{bmatrix} \alpha h_{10} & \alpha h_{11} & \alpha h_{12} & \dots & \alpha h_{1B} \\ \alpha h_{20} & \alpha h_{21} & \alpha h_{22} & \vdots & \alpha h_{2B} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha h_{m0} & \alpha h_{m1} & \alpha h_{m2} & \dots & \alpha h_{mB} \end{bmatrix}$$

$$= \begin{bmatrix} w_{10} & w_{11} & w_{12} & \dots & w_{1B} \\ w_{20} & w_{21} & w_{22} & \vdots & w_{2B} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m0} & w_{m1} & w_{m2} & \dots & w_{mB} \end{bmatrix}$$

and w_{ij} ($1 \leq i \leq m, 0 \leq j \leq B$) is the weight coefficient at sample point i for the leakage brought by bit $j + 1$.

Given this model, the question remains how to determine its parameters w_{ij}, and thus how to compute the matrix $\boldsymbol{W}$. If we compare Eq. (3.4) with the formula of the fading channel in [14], this question is equivalent to determining the channel gains in the communication channel, a problem that is known as *channel estimation* [28]. The *profiling problem* therefore becomes a *channel estimation problem*, for which powerful systematic approaches can be used.

In the following, we will describe the applications of channel estimation (signals estimation) in side channel attacks. The details of the estimation algorithm will be discussed in Chapter 4.

3.3 Conclusion

In this chapter, by extending previous work on leakage model building, we generalize the weighted leakage model to describe the mathematical relationship between the measurable physical leakages and the internal state of the cryptographic implementation. Although our work is also

based on the idea of formulating the side channel as a communication channel, we use a different view to model the side channel and treat the model coefficients as the weights of the power leakage, similarity to the gains of a communication channel. This allows us to treat the profiling problem in side-channel attack as a *channel estimation problem* in communication.

4. PROFILING AS A CHANNEL ESTIMATION PROBLEM*

4.1 Introduction

Since the prior information of the side channel is unknown to the adversary, the profiling becomes a *blind estimation problem*. Hence, we propose a ℓ_2-norm based re-weighted algorithm to profile the leakage model in this paper. The authors in [11] also propose a Least Squares (LS) method to do the profiling. This method has been widely used in side channel attacks, such as Stochastic Model based Template Attack [26] and also the Linear-Regression (LR) based DPA (Differential Power Analysis) [29, 30]. Compared to these Least Square (Linear Regression)-based algorithms, our algorithm takes the noisy corruption or distortion among the leakage traces into consideration and hence obtains excellent performance. We also compared our approach with the recently proposed Ridge Regression-based (RB) method [15], and we found that our re-weighted iterative algorithm leads to a more efficient leakage function estimation, which also results in better key recovery. Our method is also easier to deploy, since it does not require a special step for parameter pre-training. Instead, the re-weighted algorithm adaptively obtains the penalty parameters that are used for the side-channel estimation *at run time*, compared to multiple off-line goodness-of-fit trials in the Ridge-based method. The choice of the set of parameters used during these goodness-of-fit trials has a significant impact on the final recovery performance. In addition, the re-weighted algorithm does away with the need to estimate a reasonable set for candidates of penalty parameters when profiling a new device.

Given that our algorithm is iterative in nature, we will give an analysis of its convergence. We will also show how to build the templates based on the profiled leakage model. Finally, experimental results are provided to show the effectiveness of the proposed scheme.

In the following, we will describe the profiling problem and introduce the side channel estimation algorithm. We will then describe the attack and show the advantage of our scheme under the

*Parts of this chapter are adapted with permission from "Adaptive Channel Estimation in Side Channel Attacks" by Shan Jin and Riccardo Bettati, 2018. *2018 IEEE International Workshop on Information Forensics and Security (WIFS'18)*, Dec 2018, ©2018 IEEE.

case of limited traces recorded during profiling. Finally, we present the experimental results for our proposed scheme.

4.2 Data Processing Technologies

One immediate problem that comes up during profiling is the large amount of data that needs to be processed due to the individual samples usually being very large. A variety of compression technologies has been proposed to reduce the length of the leakage samples. Based on how the samples are processed, the proposed methods can be divided into two categories:

- Sample Selection: The number of samples is reduced by considering only a subset of all recorded samples. Samples are selected using Difference of Means (DoM) [9], Sum of Squared Pairwise T-Differences (SOST) [31], or Signal-to-Noise Ratio (SNR) [32].
- Linear Combination: Using a linear transformation for projecting the raw samples onto a low-dimensional subspace. This is done by using Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) [10, 24, 33, 34].

One question is: Does the linear model from Eq. (3.4) hold after the samples have been compressed? This is clearly the case for the Sample Selection approach, since this method is directly picking valid sample points from the raw data. On the other hand, any of the Linear Combination approaches, whether LDA or PCA, use a projection matrix, call it $\boldsymbol{U}$, to transform the raw leakage vector into a lower-dimensional vector. The transformation process can be represented as:

$$\widetilde{\boldsymbol{L}} = \boldsymbol{U}\boldsymbol{L} = \boldsymbol{U}\boldsymbol{W}\boldsymbol{S}_b(k) + \boldsymbol{U}\boldsymbol{R} = \widetilde{\boldsymbol{W}}\boldsymbol{S}_b(k) + \widetilde{\boldsymbol{R}} \ . \tag{4.1}$$

Thus, the linear relationship between the compressed leakage vector $\widetilde{\boldsymbol{L}}$ and the internal state of the target register still holds after the PCA/LDA processing. More details about the analysis of the linear relationship between the data sampling and leakage measures in PCA/LDA can be found in [35]. For sake of simplicity of the following discussion, we will continue to use Eq. (3.4) as the general model to describe the leakages independently of whether the leakage data has been

processed or not.

In the experimental part, the compression methods that we will use include: 20ppc (DoM), PCA, and LDA. More details about how to efficiently apply these compression technologies in side channel attacks are in [10].

4.3 Weighted Leakage Model based Profiling

4.3.1 Side Channel Estimation

As we described in Chapter 3, we can treat the profiling problem as a side channel estimation problem. The procedure goes as follows: First, the attacker records a number N_a of leakage traces $\boldsymbol{x}_i \in \mathbb{R}^m$ ($1 \leq i \leq N_a$). These traces come from a uniform distribution of the set of keys $\mathcal{K}$. The size of the set is N, and hence $N_a = n_p N$. Finally, all the combined traces that come from the N profiled keys are considered to generate the *trace matrix* $\boldsymbol{X} \in \mathbb{R}^{N \times m}$, whose i-th row belongs to key k^i used for profiling.

Then, based on Eq. (3.4), we have

$$\begin{aligned} \boldsymbol{X} &= (\boldsymbol{W}[\boldsymbol{S}_b(k^1)\ \boldsymbol{S}_b(k^2) \dots\ \boldsymbol{S}_b(k^N)] + [\boldsymbol{R}_1\ \boldsymbol{R}_2 \dots\ \boldsymbol{R}_N])^T \\ &= \boldsymbol{F}\boldsymbol{W}^T + \boldsymbol{\Phi} \end{aligned} \tag{4.2}$$

where $\boldsymbol{\Phi} = [\boldsymbol{R}_1\ \boldsymbol{R}_2 \dots\ \boldsymbol{R}_N]^T$ is the noise matrix, and $\boldsymbol{F}$ acts as the measurement matrix, which is expressed as:

$$\begin{aligned} \boldsymbol{F} &= [\boldsymbol{S}_b(k^1)\ \boldsymbol{S}_b(k^2) \dots\ \boldsymbol{S}_b(k^N)]^T \\ &= \begin{bmatrix} c & \mathcal{F}_1(k^1) & \dots & \mathcal{F}_B(k^1) \\ c & \mathcal{F}_1(k^2) & \vdots & \mathcal{F}_B(k^2) \\ \vdots & \vdots & \ddots & \vdots \\ c & \mathcal{F}_1(k^N) & \dots & \mathcal{F}_B(k^N) \end{bmatrix} . \end{aligned} \tag{4.3}$$

For each column of $\boldsymbol{X}$ in Eq. (4.2), we have

$$\boldsymbol{X}_{:,j} = \boldsymbol{F}\boldsymbol{w}_j + \boldsymbol{\phi}_j \tag{4.4}$$

where $\boldsymbol{w}_j$ $(1 \leq j \leq m)$ is the j-th row of matrix $\boldsymbol{W}$ and $\boldsymbol{\phi}_j$ $(1 \leq j \leq m)$ is the j-th column of matrix $\boldsymbol{\Phi}$. Since the coefficient vector $\boldsymbol{w}_j$ (we call it *agent* here) in sample point j is independent of other coefficient vectors, the weight matrix $\boldsymbol{W}$ can be computed by solving m linear equations from Eq. (4.4).

Previous work used the Least Squares method to get the solutions in Eq. (4.4) under the Stochastic Model [11, 26]. A similar idea has also been used in the work on so called Linear-Regression based DPA [29, 30]. In this work, the Pearson coefficient is obtained by computing the correlation between the actual traces and the predictions (templates) with the use of the linear regression method to profiling the leakage model. Previous results in channel estimation and signals estimation show that Least Squares (Linear Regression) does not show stable performance, especially in the case when measurements (received signals) are corrupted by noise or other distortions [28]. We will illustrate this in our experiments by comparing the LS (LR) algorithm with our proposed scheme in the case of strong corruption caused by noise, for example when the number of profiling traces is small.

Hence, we formulate the estimation of the weight coefficients as the following decentralized optimization problem:

$$\min_{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m} \sum_{j=1}^{m} G_j(\boldsymbol{w}_j) \tag{4.5}$$

where

$$\sum_{j=1}^{m} G_j(\boldsymbol{w}_j) = \sum_{j=1}^{m} \|\boldsymbol{F}\boldsymbol{w}_j - \boldsymbol{X}_{:,j}\|_{\ell_2}^2 + \sum_{j=1}^{m} \lambda_j J(\boldsymbol{w}_j) \tag{4.6}$$

and

$$J(\boldsymbol{w}_j) \triangleq \sum_{l=0}^{B} q_{jl}|w_{jl}|^2 \ . \tag{4.7}$$

We denote by $\boldsymbol{q}_j = [q_{j1}, q_{j2}, \ldots, q_{jB}]$ $(1 \leq j \leq m)$ the weight associated with agent $\boldsymbol{w}_j$, and by $J(\boldsymbol{w}_j)$ the weighted ℓ_2-norm (diversity measure).

Finding the solution of Eq. (4.5) becomes a *weighted ℓ_2-norm minimization problem*. In general, weighted ℓ_p-norm minimization $(0 \leq p \leq 2)$ [36, 37] and ℓ_p diversity measure optimization [38] are very common approaches in data recovery. For example, weighted ℓ_2-norm minimization is efficient in solving the nuclear norm minimization (NNM) problem [39]. Similarly, ℓ_p diversity measure optimization is widely used to solve underdetermined system problems [38]. For our particular side channel problem, we choose the *weighted* ℓ_2-norm rather than traditional ℓ_2-norm in order to attain more focal estimations.

Wang *et al.* recently proposed a Ridge-Regression based algorithm for DPA [15]. This method determines the value of the *penalty parameter* λ by selecting the "best" value from a predefined set of candidates. The selection is done by running multiple goodness-to-fit trials. The choice of the set of candidates also has a significant impact on the final recovery performance. The estimation of the coefficients is computed by putting the selected final penalty parameter in the ridge regression estimator.

In this paper, we propose an iterative, self-adaptive algorithm: As the values in $\boldsymbol{W}$ are unknown *a priori*, the recovery algorithm has to proceed iteratively. This includes updating the value of the weight $\boldsymbol{q}_j$ and penalty parameters as well. Suppose now we have the solution for the j-th agent $\boldsymbol{w}_j^{(t)}$ at iteration step t, then the value for $\boldsymbol{q}_j$ to be used for step $t+1$ is

$$q_{jl}^{(t+1)} = \frac{1}{|w_{jl}^{(t)}|^2},\ l = 0, 1, \ldots, B \ . \tag{4.8}$$

This gives rise to the diagonal matrix $\boldsymbol{Q}_j^{(t+1)}$, where

$$\boldsymbol{Q}_j^{(t+1)} = \text{diag}(\boldsymbol{q}_j^{(t+1)}) = \begin{bmatrix} q_{j0}^{(t+1)} & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & q_{jB}^{(t+1)} \end{bmatrix} \ . \tag{4.9}$$

The algorithm iterates until the optimal point of Eq. (4.5) is reached, which we denote as

$$\boldsymbol{W}^* = [\boldsymbol{w}_1^* \, \boldsymbol{w}_2^* \dots \, \boldsymbol{w}_m^*] \ . \tag{4.10}$$

In the following sections, we will describe the algorithm to find $\boldsymbol{W}^*$ and then prove its convergence.

We notice that the Template Attack proposed in [9] and also [10] used all 2^B keys in profiling. This would require the attackers to collect a very large number of time samples. The number is $N_a = n_p 2^B$ where n_p is the number of recorded samples per key during profiling. To reduce the profiling overload, in our scheme, the attacker can select only a subset keys for profiling, for example keys with Hamming Weight (HW) = 1. Now, for an N-bit key (e.g., $N = 8$), one could close to profile with $k \in \{1, 2, 4, \dots, 2^N\}$, which would reduce the profiling cost by a factor of $\frac{2^N}{N}$ while achieving acceptable key recovery rates.

4.3.2 Iterative ℓ_2-norm based Re-weighted Algorithm

In this section, we describe an iterative algorithm for estimating the weight matrix $\boldsymbol{W}$. In Eq. (4.5), after applying

$$\frac{\partial[\sum_{j=1}^m G_j(\boldsymbol{w}_j)]}{\partial \boldsymbol{w}_j} = \frac{\partial G_j(\boldsymbol{w}_j)}{\partial \boldsymbol{w}_j} = 0 \ ,$$

a necessary condition of the optimal solution for each agent $\boldsymbol{w}_j$ is that

$$\boldsymbol{w}_j^* = (\boldsymbol{F}^T \boldsymbol{F} + \lambda_j \boldsymbol{Q}_j)^{-1} \boldsymbol{F}^T \boldsymbol{X}_{:,j} \tag{4.11}$$

where $j = 1, 2, \ldots, m$.

The right-hand side the of above equation is equivalent to

$$\begin{aligned}(\boldsymbol{F}^T\boldsymbol{F} + \lambda_j \boldsymbol{Q}_j)^{-1}\boldsymbol{F}^T &= \boldsymbol{Q}_j^{-1/2}(\boldsymbol{Q}_j^{-1/2}\boldsymbol{F}^T\boldsymbol{F}\boldsymbol{Q}_j^{-1/2} + \\ &\quad \lambda_j \boldsymbol{I})^{-1}\boldsymbol{Q}_j^{-1/2}\boldsymbol{F}^T \\ &= \boldsymbol{Q}_j^{-1}\boldsymbol{F}^T(\boldsymbol{F}\boldsymbol{Q}_j^{-1}\boldsymbol{F}^T + \lambda_j \boldsymbol{I})^{-1} \ , \end{aligned} \tag{4.12}$$

where λ_j is the *penalty parameter*, which is controlled by the noise's energy. At the beginning of the side-channel estimation procedure (i.e., at $t = 0$), we don't have any information about $\boldsymbol{W}$, and each $\boldsymbol{Q}_j^{(0)}$ is an empty matrix. This reduces Eq. (4.11) to be a LS estimator. After the first iteration on Eq. (4.11), we obtain the initial estimation $\boldsymbol{w}_j^{(1)}$, which is then used for computing $\boldsymbol{Q}_j^{(1)}$. This procedure iterates until the algorithm converges.

The computing of λ_j is important since it has a direct impact on the performance of recovery [40]. Given that the noise energy is unknown to the attacker, the penalty parameter λ_j can not be limited to a fixed value across all iterations. Rather, a value that is dependent on the iteration is more suitable. To ease the implementation of our algorithm, different to the work in [40] and [38], we give a simple approach that has been previously validated before [41, 42], and which also works in this application: We set $\lambda_j = \beta(\delta_j)^{-1}$ where we compute δ_j as follows:

$$\delta_j = \|\boldsymbol{F}\boldsymbol{w}_j^{(t)} - \boldsymbol{X}_{:,j}\|_2^2$$

by iteratively estimating the error. The value for β is selected based on the choice of the compression method. We will describe the details of how to select β in the experimental part.

The framework for our algorithm is illustrated in **Algorithm 1**. The algorithm takes as input the traces matrix $\boldsymbol{X}$ and the measurement matrix $\boldsymbol{F}$. It initializes the index set $\boldsymbol{\psi}$. It then iteratively computes the weight matrix $\boldsymbol{Q}_j^{(t)}$ through Eq. (4.9). Once computed, $\boldsymbol{Q}_j^{(t)}$ is used to update $\boldsymbol{w}_j^{(t)}$. This iterative process stops for agent $\boldsymbol{w}_j$ once that particular agent converges to the optimal point, that is, once the difference between $\boldsymbol{w}_j^{(t+1)}$ and $\boldsymbol{w}_j^{(t)}$ is less or equal to the threshold θ. If this is the

case, the index j is removed from the index set $\boldsymbol{\psi}$. Once $\boldsymbol{\psi}$ becomes an empty set, the algorithm comes to an end. It then combines each agent's optimal point together and outputs $\boldsymbol{W}^*$. As we mentioned earlier, the $\boldsymbol{w}_j$'s are mutually independent, which makes the algorithm parallelizable: each agent $\boldsymbol{w}_j$ can be computed in parallel.

Algorithm 1 Side Channel Estimation Algorithm

Input: $\boldsymbol{X}, \boldsymbol{F}$
  initialize $\boldsymbol{\psi} = \{1, 2, \ldots, m\}, j \in \boldsymbol{\psi},\ t = 0$
  repeat
    $\boldsymbol{w}_j^{(t+1)} = (\boldsymbol{Q}_j^{(t)})^{-1}\boldsymbol{F}^T(\boldsymbol{F}(\boldsymbol{Q}_j^{(t)})^{-1}\boldsymbol{F}^T + \lambda_j \boldsymbol{I})^{-1}\boldsymbol{X}_{:,j}$
    update $\boldsymbol{Q}_j^{(t+1)}$ **and** λ_j
    $t = t + 1$;
    If $\left\|\boldsymbol{w}_j^{(t)} - \boldsymbol{w}_j^{(t-1)}\right\|_2 < \theta$
    $\boldsymbol{w}_j^* = \boldsymbol{w}_j^{(t)}$;
    $\boldsymbol{\psi} = \boldsymbol{\psi} \backslash j$;
    end
  until $\boldsymbol{\psi} = \boldsymbol{\emptyset}$
Output: $\boldsymbol{W}^* = [\boldsymbol{w}_1^*\ \boldsymbol{w}_2^* \ldots\ \boldsymbol{w}_m^*]^T$

The threshold value θ that controls the convergence can be adapted to different application scenarios. In some situations, we also suggest to use the relative error rather than the absolute error to control the iterative process, which is

$$\frac{\left\|\boldsymbol{w}_j^{(t)} - \boldsymbol{w}_j^{(t-1)}\right\|_2}{\left\|\boldsymbol{w}_j^{(t-1)}\right\|_2} < \theta \ .$$

In addition, the attacker can also bound the computation cost by putting a limit on the number of iterations as the halting condition. In this paper, we fix the number of the iterations as 20 in the experiments. We note that, once the traces used for profiling have been recorded, the leakage model estimation can be done off-line. In this case, the constant-factor changes to the computation cost caused by modifications to the number of iterations are not critical.

4.3.3 Convergence Analysis for the Algorithm

In order to demonstrate the convergence of **Algorithm 1**, we define the sum of each agent's objective function as follows:

$$G(\boldsymbol{W}) = \sum_{j=1}^{m} G_j(\boldsymbol{w}_j) \ . \tag{4.13}$$

The convergence then follows from this theorem:

Theorem 1. *For **Algorithm 1**, if $\boldsymbol{W}^{(t)} \neq \boldsymbol{W}^{(t-1)}$, the objective function $G(\boldsymbol{W})$ is strictly monotonically decreasing, i.e., $G(\boldsymbol{W}^{(t)}) < G(\boldsymbol{W}^{(t-1)})$.*

Proof. Based on Eq. (4.5) and Eq. (4.8), we have

$$\begin{aligned}
&G(\boldsymbol{W}^{(t)}) - G(\boldsymbol{W}^{(t-1)}) \\
&= \sum_{j=1}^{m} [\|\boldsymbol{F}\boldsymbol{w}_j^{(t)} - \boldsymbol{X}_{:,j}\|_{\ell_2}^2 + \lambda_j \|[\boldsymbol{Q}_j^{(t)}]^{\frac{1}{2}} \boldsymbol{w}_j^{(t)}\|_{\ell_2}^2 \\
&\quad - \|\boldsymbol{F}\boldsymbol{w}_j^{(t-1)} - \boldsymbol{X}_{:,j}\|_{\ell_2}^2 + \lambda_j \|[\boldsymbol{Q}_j^{(t)}]^{\frac{1}{2}} \boldsymbol{w}_j^{(t-1)}\|_{\ell_2}^2] \ .
\end{aligned}$$

We define

$$f_j^{(t)}(\boldsymbol{w}_j) = \|\boldsymbol{F}\boldsymbol{w}_j - \boldsymbol{X}_{:,j}\|_{\ell_2}^2 + \lambda_j \|[\boldsymbol{Q}_j^{(t)}]^{\frac{1}{2}} \boldsymbol{w}_j\|_{\ell_2}^2 \ .$$

When t is determined, the function $f_j^{(t)}(\boldsymbol{w}_j)$ is a $\mathcal{L}^2$-function. It is also continuously differentiable. In this case, the minimum of $f_j^{(t)}(\boldsymbol{w}_j)$ can be obtained and is also unique. Based on Eq. (4.11), the solution of this minimization problem, which is

$$\arg\min_{\boldsymbol{w}_j} f_j^{(t)}(\boldsymbol{w}_j) \ , \tag{4.14}$$

is $\boldsymbol{w}_j^{(t)}$. This then leads to

$$f_j^{(t)}(\boldsymbol{w}_j^{(t)}) - f_j^{(t)}(\boldsymbol{w}_j^{(t-1)}) < 0$$

and hence for all agents

$$\sum_{j=1}^{m}[f_j^{(t)}(\boldsymbol{w}_j^{(t)}) - f_j^{(t)}(\boldsymbol{w}_j^{(t-1)})] < 0 \ .$$

Finally, we have

$$G(\boldsymbol{W}^{(t)}) - G(\boldsymbol{W}^{(t-1)}) < 0 \ .$$

Thus, $G(\boldsymbol{W})$ is decreasing at every iteration step of our algorithm. □

4.4 Using the Weighted Leakage Model in An Attack

Once we get the weight matrix $\boldsymbol{W}$ in the profiling phase, the template parameters for each key k can be easily obtained by computing the stochastic mean vector $\bar{\boldsymbol{x}}_k$ and the covariance matrix $\boldsymbol{C}_k$ (same as [9]), which are

$$\bar{\boldsymbol{x}}_k = \boldsymbol{W}\boldsymbol{S}_b(k) \tag{4.15}$$

and

$$\boldsymbol{C}_k = \frac{1}{n_p - 1}(\boldsymbol{X}_k - \boldsymbol{F}_k^+\boldsymbol{W}^T)^T(\boldsymbol{X}_k^T - \boldsymbol{F}_k^+\boldsymbol{W}^T) \ , \tag{4.16}$$

where

$$\boldsymbol{F}_k^+ = \boldsymbol{I}_{1\times n_p}\boldsymbol{S}_b(k)^T \ , \tag{4.17}$$

and $\boldsymbol{I}_{1\times n_p} = [1, 1, \ldots, 1]^T \in \mathbb{R}^{n_p \times 1}$. Here we consider that the adversary has obtained the traces for all the keys in $k \in \mathcal{K}$ for profiling and hence can generate the $\boldsymbol{C}_k$ for each k. If the adversary only gets the traces from only a subset of $\mathcal{K}$, denoted by $\mathcal{K}_s$, we use a *pooled* covariance matrix $\boldsymbol{C}_{pool}$, which is introduced in [10], for all keys to replace each key's covariance matrix. It is expressed as

$$\boldsymbol{C}_{pool} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{C}_{k^i} \ , \tag{4.18}$$

where now $N = |\mathcal{K}_s|$.

As described in [10], the computing result of Eq. (4.18) is equivalent to the computing of

$$\boldsymbol{C}_{pool} = \frac{\boldsymbol{Z}^T\boldsymbol{Z}}{N * n_p - 1} \ , \tag{4.19}$$

where

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{X}_{k^1} - \boldsymbol{F}^{+}_{k^1}\boldsymbol{W}^T \\ \vdots \\ \boldsymbol{X}_{k^N} - \boldsymbol{F}^{+}_{k^N}\boldsymbol{W}^T \end{bmatrix} \ .$$

Compared to the standard profiled side channel attack, our model simplifies the computation. For example, in our model, the adversary can use the traces from a subset of $\mathcal{K}$ to profile the leakage model $\boldsymbol{W}$ and obtain the mean vector for all k by computing Eq. (4.15). This saves time compared to the previous Template Attack, where the adversary has to use all n_p traces to compute the mean vector for each k. In the following, we use $\boldsymbol{\Omega_k}$ to represent the template for each key k as

$$\boldsymbol{\Omega_k} := \{\bar{\boldsymbol{x}}_k, \boldsymbol{C}_k(\boldsymbol{C}_{pool})\} \ . \tag{4.20}$$

From [9], we know that the side channel leakage can be modeled by a multivariate normal dis-

tribution. Hence, based on the template parameters, the probability density function (pdf) $f(\boldsymbol{x}|\boldsymbol{\Omega_k})$ of a given leakage trace vector $\boldsymbol{x}$ in k is

$$f(\boldsymbol{x}|\boldsymbol{\Omega_k}) = \frac{1}{\sqrt{(2\pi)^m|\boldsymbol{C}_k|}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}}_k)^T(\boldsymbol{C}_k)^{-1}(\boldsymbol{x}-\bar{\boldsymbol{x}}_k)\right) \quad . \tag{4.21}$$

During the attacks phase, the adversary guesses the unknown target key $k^* \in \mathcal{K}$, which is executed in the cryptographic implementation, through the analysis of the recorded leakage traces. Suppose that the adversary has recorded n_a raw leakage traces for the target key k^*. After applying the same data processing techniques (see Section 4.2) as during the profiling phase, the adversary obtains the leakage matrix $\boldsymbol{X}_{k^*} \in \mathbb{R}^{n_a \times m}$. From [10], we can see that according to the Bayes' rule, the *likelihood* for a given trace (row) $\boldsymbol{x}_{k^*i}$ $(1 \leq i \leq n_a)$ in matrix $\boldsymbol{X}_{k^*}$ to come from secret key k is

$$\boldsymbol{l}(k|\boldsymbol{x}_{k^*i}) = \boldsymbol{d}(k|\boldsymbol{x}_{k^*i}) = \boldsymbol{l}(\boldsymbol{\Omega_k}|\boldsymbol{x}_{k^*i}) = \boldsymbol{f}(\boldsymbol{x}_{k^*i}|\boldsymbol{\Omega_k}) \tag{4.22}$$

where $\boldsymbol{d}(\cdot|\cdot)$ is the function of the discriminant score. Since usually n_a is a large number, to efficiently implement a template attack, the attacker needs to combine all n_a likelihood values $\boldsymbol{l}(k|\boldsymbol{x}_{k^*i})$ together to get **a final** likelihood value (discriminant score). This combination can be done through average/joint rule, as described in [10]. Hence, for each $k \in \mathcal{K}$, the attacker applies the combination rule and obtains a final likelihood value $\boldsymbol{l}(k|\boldsymbol{X}_{k^*})$ through Eq. (4.22).

We sort all final likelihood values and find the correct key k^* by picking the one with the maximum value. We observe that now the maximum value finding works as the *argmax* distinguisher in our model, and k^* is the target key in Figure 3.1. The main principle of our attack algorithm is based on the Template Attack [9]. To better illustrate the attacking procedure, we summarize the attacking algorithm in **Algorithm 2**, shown in the following.

More details about how to implement the combination of the likelihood value of each trace and how to efficiently calculate the final likelihood value can be found in [10].

Algorithm 2 Gaussian Model based Attacking Algorithm

Input: $\boldsymbol{X}_{k^*}, \{\boldsymbol{\Omega_i}\}_{i=0}^{2^B-1}$
  For $j = 0, \ldots, 2^B - 1$
    Computes $\boldsymbol{l}(j|\boldsymbol{X}_{k^*})$ **through** Eq. (4.22);
  End
  Computes $j^* := \text{argmax}_{\{j=0\ldots,2^B-1\}} \boldsymbol{l}(j|\boldsymbol{X}_{k^*})$;
Output: $k^* = j^*$

4.5 Advantage of the Proposed Scheme for Limited Traces

The advantage of the linear model compared to the Template Attack is that the construction of the templates for all keys only needs the leakage traces that have been collected from a *subset* of $\mathcal{K}$. For example, the adversary may attack a secret implementation with an 8-bit key ($B = 8$). To speed up the profiling, the adversary only collects leakage traces for $k = 0$ and keys in the group with HW=1. This makes for 9 keys in total. From Eq. (4.11), the measurement matrix now is

$$\boldsymbol{F}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & \vdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 0 \end{bmatrix} . \tag{4.23}$$

We point out that in comparison to Eq. (4.2), c in the fist column is set to 1.

$\boldsymbol{F}'$ is a square nonsingular matrix. As a result, theoretically we should obtain a unique solution for each $\boldsymbol{w}_j$ in Eq. (4.4). After acquiring the weight matrix $\boldsymbol{W}$, the attackers can compute the templates for **all** $k \in \mathcal{K}$: the mean vector through the use of Eq. (4.15) and the pooled covariance matrix through Eq. (4.18). Although those templates may not be very accurate due to the limited amount of data collected during profiling, the accuracy is still sufficient for the adversary to find the correct key by applying the profiled attack. Standard profiled side channel attacks like Template Attack, in comparison, do not allow for the approximate computation of templates. Thus, if the adversary only gets 9 templates during profiling, only 9 classifiers can be used for classifying the

unknown leakage traces. At best, the standard Template Attack can recover 9 keys in total.

In general, the attacker faces a trade-off between the attacking *performance* (key's recovery rate, mutual information) and the profiling *resource cost* (profiling traces, profiling time, keys used). As the profiling resource cost increases, so does the performance of the attack. However, in many application scenarios, there are a variety of constraints on the profiling resources that the attackers may be able to afford. Hence, how to find a smart strategy to allocate the limited resources for getting optimal attacking performance goes beyond the scope of this dissertation but is a problem of practical important that is well worth studying.

4.6 Experimental Results

We evaluate our proposed algorithm and compare it to the other state-of-the-art algorithms by using it against two targets. One target is the data bus of the Atmel XMEGA 256 A3U [43], a widely used 8-bit micro-controller. The other target is the implementation of the AES 128 algorithm [18] on the SASEBO-GII board [44].

In the following performance figures, we use the term HW to denote the Hamming Weight model, TA to represent the Template Attack, LS to denote Least Square, RB for Ridge-based, and L2-Re to denote our proposed algorithm. We notice that in [15], the Ridge-based method uses B bits instead of $B + 1$ bits to construct the measurement matrix $\boldsymbol{F}$, which could easily render $\boldsymbol{F}$ singular. Hence, in order to fairly compare our algorithm with the Ridge-based method, we also present the experiments that use the Ridge-based method under our model with $B + 1$ base functions. We use the term RB-U9 to denote the experiment results for this notation of the Ridge-based method.

4.6.1 Results on 8-Bit Micro-controller

For the experiment on the Atmel 8-bit processor, we use the **Grizzly** benchmark datasets, which were collected by Choudary and Kuhn [10, 26] and shared in [1], to compare the attacking performance between the standard Template Attack, Least Squares, Ridge-Based, and Re-weighted ℓ_2-norm algorithms under our linear model with three different compression methods: 20ppc, PCA,

and LDA.

In the experiments, we use the *successful guessing rate* as the measure of merit to compare all methods. This rate is defined as the average of the successful guessing probability for **all** keys, that is:

$$\text{Successful Guessing Rate} = \frac{\sum_{k=0}^{2^B-1} P_{attack}(k)}{2^B}$$

where $P_{attack}(k) = \frac{N_{hit}(k)}{N_T}$. Here N_T is the number of tests and $N_{hit}(k)$ is how many times that key k is successfully guessed in all N_T tests.

For the constant number c in $\boldsymbol{F}$, we choose 1 in all our experiments. We encourage readers to choose different values for c when doing their own testing. The choice of β is based on the choice of the compression methods. The authors in [36] describe a practical approach to updating β in each iteration. The parameter β plays the role of trade-off between the effect of trial and error, and the iteration speed. The value of β will decrease across iterations, and the minimum value is decided by the power of the leakage data, which could be easily observed once the data is recorded by the adversary. In our experiment, we recommend $\beta = \max\{(0.8)^t, \frac{1}{(B+1)|N|}\}$ for 20ppc case and $\beta = \max\{\frac{(0.92)^t}{9}, \frac{1}{(B+1)|N|}\}$ for PCA/LDA. To fairly test our algorithm, we fix the number of the iterations in **Algorithm 1** to 20 for absolute error control case and $\theta = 10^{-3}$ for relative error control case in all experiments. Here we should note that when we set $\beta = 1$ and the iteration number to 1, our algorithm becomes the joint algorithm of MMSE [28]. In the attacking phase, for each k, we independently run the attacking 100 times by randomly picking the leakage traces from the *attacking* set.

As described in [10], the attacker's target in 8-bit **Grizzly** is a 8-bit CPU Atmel XMEGA 256 A3U micro-controller (B=8 in this case). For each key $k \in \{0, 1, \ldots, 255\}$, 3072 raw traces $\boldsymbol{x}_{ki}^r$ ($1 \leq i \leq 3072$) are recorded. These traces are divided into two sets: a *profiling* set and an *attacking* set. Each raw trace $\boldsymbol{x}_{ki}^r$ has 2500 samples, which is the total current consumption in all CPU ground pins. The samples are recorded when the target micro-controller executes

the same sequence of instructions: a MOV instruction and followed by several consecutive LOAD instructions. One of these LOAD instruction loads the secret k, and the others load the constant value 0. This guards against variability of the recorded traces that would be caused by the data that was loaded in the nearby instructions.

Note the term "Key=All" to indicated that all 2^B keys have been chosen for profiling (For example $B = 8$, Key=256). The term "Key=HW1" indicates that only keys with the HW value of 1, in addition to $k = 0$ and $k = 2^B - 1$, have been chosen for profiling. For example, if $B = 8$, Key=HW1 is the group of the keys $\{0, 1, 2, 4, 8, 16, 32, 64, 128, 255\}$. The same notation holds for Key=HW2 and Key=HW4, etc.

In Fig. 4.1, we show the successful guessing rate with different numbers of profiling traces under different profiling schemes. We use 20ppc as the compression approach here. The number of attacking traces n_a is fixed to 10, 1000, and 1000 respectively, and the number of profiling traces n_p varies from 10 to 2000 in each figure. From the figure we can see that, for the case of Key=All (Key=256), our algorithm is generically better than all other methods: the basic Template Attack, the Least Squares method, the Ridge-based method and the RB-U9, the Ridge-based method fitted to our mode, in all cases of attacking. When the number of profiling traces is limited, for example $n_p = 50$, compared to Template Attack, our method has nearly 15% higher guessing rate when $n_a = 10$, nearly 40% higher guessing rate when $n_a = 100$, and almost 55% higher guessing rate when $n_a = 1000$. Even when the number of profiling traces increase, our method outperforms the Template Attack. Compared to Least Square, our method also shows better performance, especially when the adversary only has a small number of profiling traces, where the noise is strong. Our algorithm outperforms the other two Ridge-based schemes. When $n_a = 1000$ and $n_p = 50$, our method has at least 13 more keys successfully guessed compared to the Ridge-based and the Ridge-based-U9 methods. From all the experimental results, we observe that our method always shows particular better performance.

In Fig. 4.2, we present the successful guessing rate when n_a is fixed to 1000 and n_p again varies from 50 to 2000. The compression approach is still 20ppc. The keys used for profiling come from

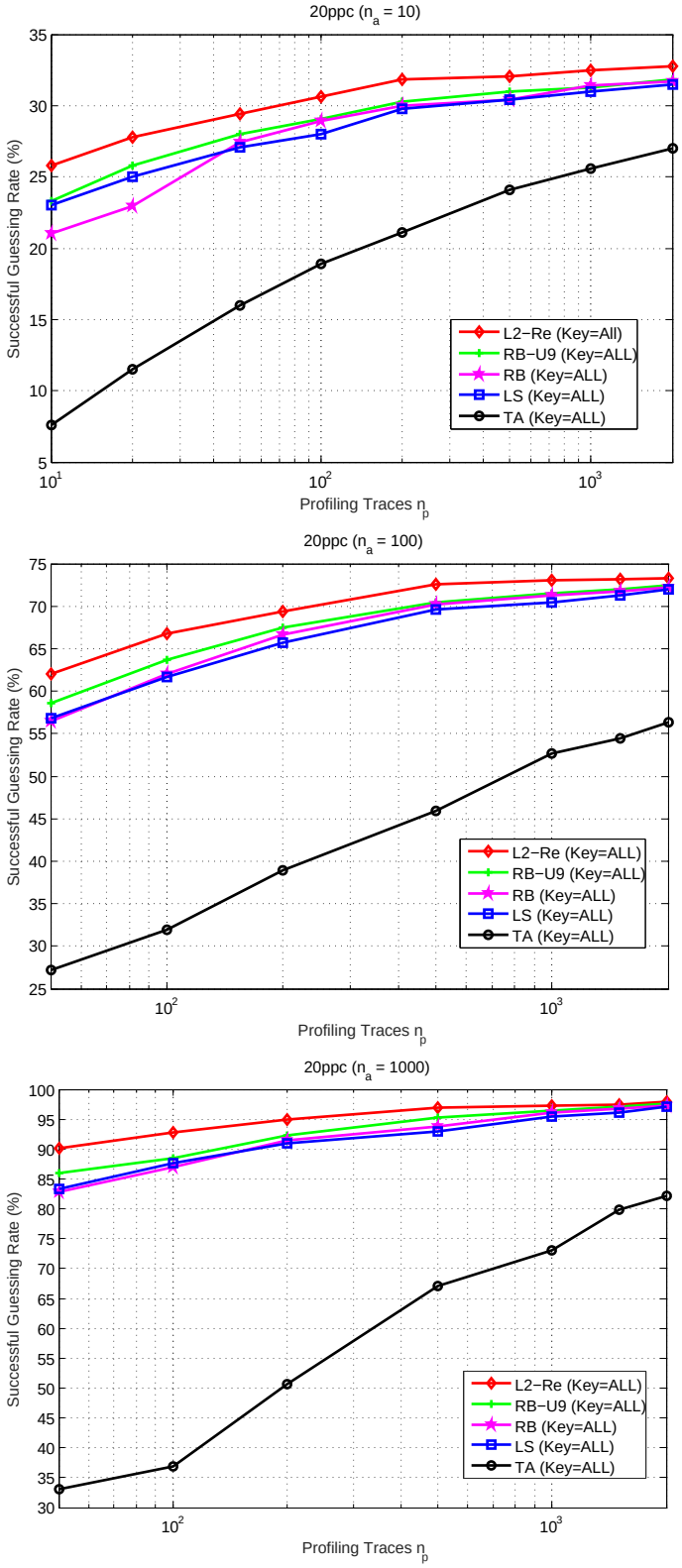

Figure 4.1: Probability of Successful Guessing versus Different Profiling Traces n_p with Different Attack Traces $n_a = 10$, $n_a = 100$, and $n_a = 1000$ under 20ppc Compression (Adapted with permission from [3])

HW2 group and HW4 group respectively. In general, our method still shows great performance for both cases where we only use subsets of keys to do the profiling. If we choose the subset of keys whose HW value is no less than 2, our method performs better than even the Template Attack, which uses all 256 keys. When Key=HW2 and Key=HW4, our method outperforms the Least Square with the same subset of keys in all cases. Compared to Ridge-based and Ridge-based U9 with the same subset of keys during profiling, our method still wins in almost all cases. When the number of profiling traces is small, for example $n_p = 100$, our method has a nearly 8% higher guessing rate than RB-U9 and at least 10% higher guessing rate than RB under Key=HW4. This also shows that our algorithm is applicable in the case when the selection of profiling traces and profiled keys are very limited.

In Fig. 4.3, the guessing rate is computed with $n_a = 1000$, and the values for n_p are the same as in the previous experiment in Fig. 4.2. All experimental data is still compressed using the 20ppc approach. For all four profiling methods, LS, RB, RB-U9,and L2-Re, the profiling traces come from these different subsets of keys: HW1 group, HW2 group and HW4 group. The results show that by using the linear model, selecting only a subset of keys to do the profiling will also bring a huge performance improvement compared the standard Template Attack with all keys used (Key=256). Note that when the adversary only uses Key=HW1, our method is a bit worse than the Template Attack with Key = 256. This comes as no surprise, given that the profiling cost is only a small fraction compared to that of the Template Attack. In fact, we now use only $B + 2 = 10$ keys for profiling instead of $2^B = 256$. Compared with the huge saving in profiling cost, such performance loss is indeed acceptable. When the keys that are used for profiling come from HW2 group (i.e., Key=HW2), or the HW=4 group (i.e., Key=HW4), even the Least Square method can get higher recovery rate compared to Template Attack, let alone our method.

In Fig. 4.4, we test the successful guessing rate for two compression methods: PCA and LDA. We choose $n_a = 100$ and use the same values for n_p as in the previous experiments. We use all 256 keys for profiling. We see from the figure that for both LDA and PCA, generally our method shows excellent performance compared to all other methods. For LDA, when $n_p = 100$, our method has

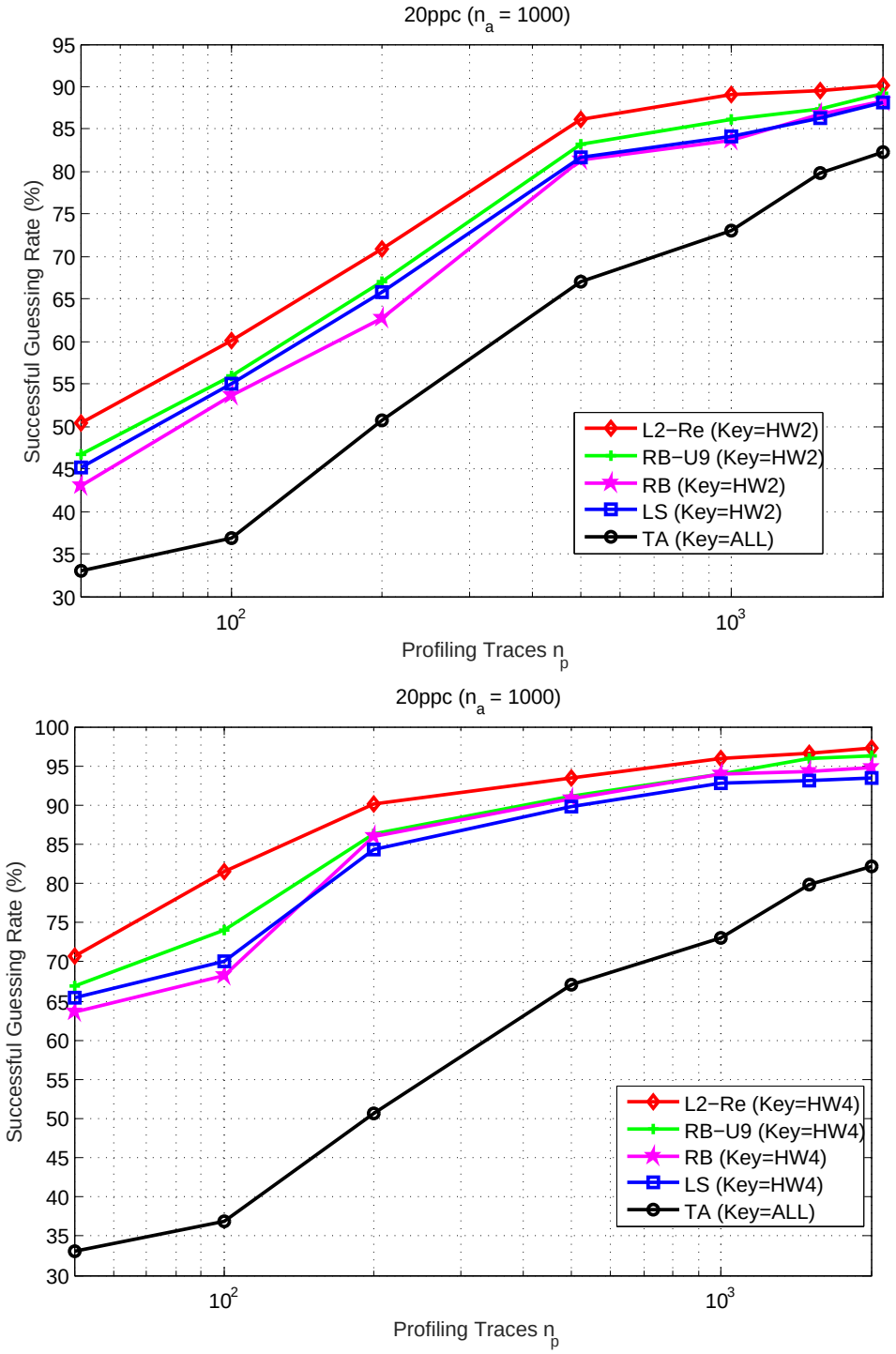

Figure 4.2: Probability of Successful Guessing versus Different Profiling Traces n_p with Attack Traces $=$ 1000 under 20ppc Compression, to Different HW Groups (Adapted with permission from [3])

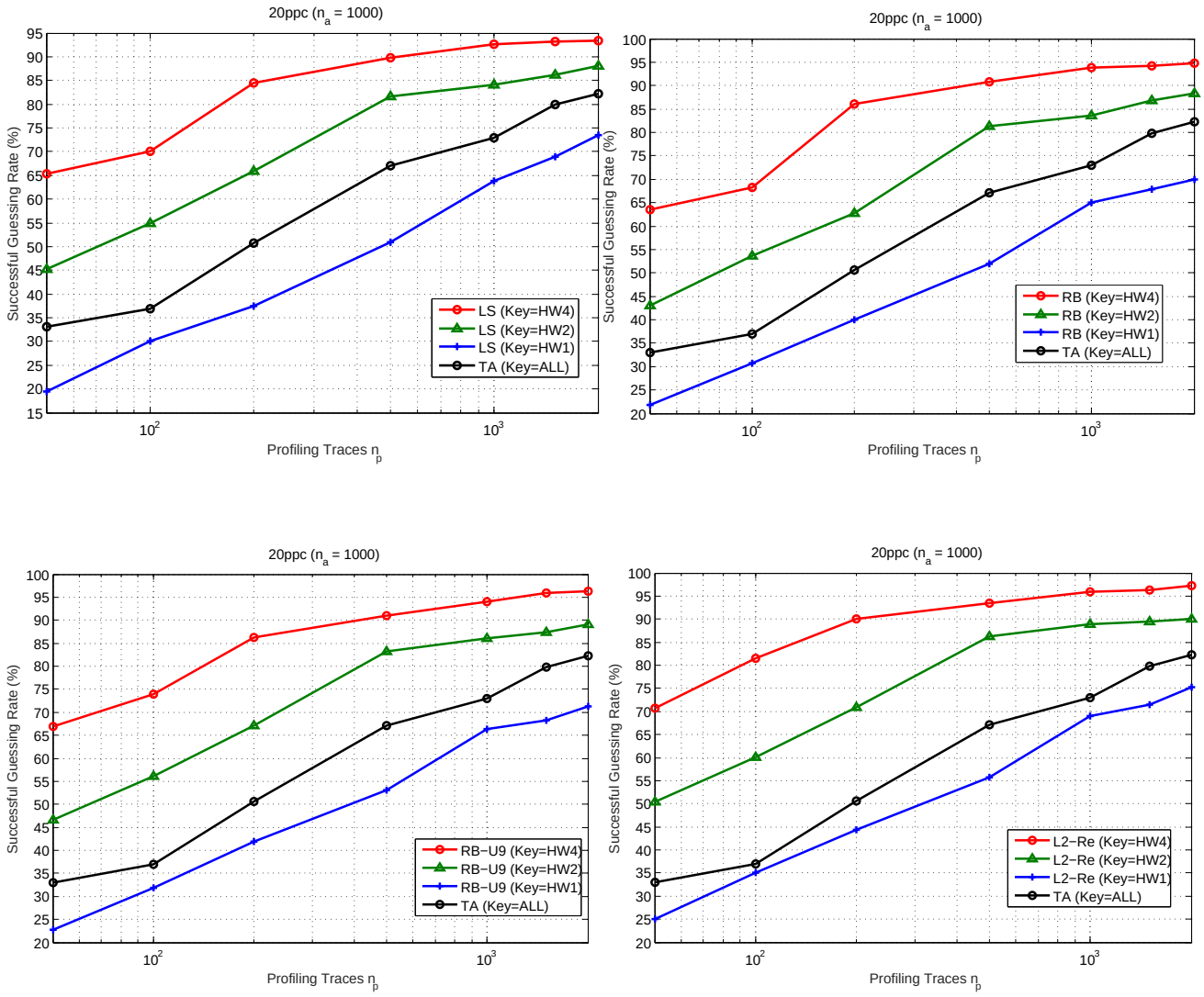

Figure 4.3: Probability of Successful Guessing versus Different Profiling Traces n_p with Attack Traces $= 1000$ under 20ppc Compression, to Each Profiling Methods: Least Square; Ridge-based; Ridge-based-U9; L2-Re-weighted. (Adapted with permission from [3])

nearly a 20% performance improvement compared to the Template Attack. For PCA, we also have a 15% improvement in the case of $n_p = 100$. Compared to Least Squares, our method shows 5% better performance in both LDA and PCA for $n_p = 100$. Compared to Ridge-based schemes, our method still performs better in both LDA and PCA. For LDA with $n_p = 200$, our method shows nearly 5% performance improvement compared to RB-U9 and also far more compared to RB.

4.6.2 Result on 128-Bit AES Implementation

For the experiment on long keys ($B = 128$), our target is the implementation of an unmasked AES 128 algorithm. The details of the implementation, and also the power traces, are introduced in [45] and shared in TeSCASE [2]. The attacker's target is a AES 128 core on the Xilinx Virtex-5 LX30 FPGA with a clock signal at 24MHz. The data contains 50000 raw traces, each of which corresponds to a different ciphertext (that is, n_p=1) with 3124 sample points each. All traces are from the same secret key, which is "00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F". The attack targets on the last round operations in AES 128, which means that we want to guess the last-round key and subsequently get the AES secret key by reversing the AES key expansion. We divided the traces into two part: a *profiling* set and an *attacking* set. In the following figures, we use N_p to represent the total number of profiling traces and N_a as the total number of attack traces.

Since AES 128 has 128 bits, we can not directly apply the Template Attack on the AES implementation due to the huge computing cost (now the number of templates is 2^{128}). Instead, we use the Correlation Power Analysis [7] based on the divide-and-conquer (DC) strategy. Under this strategy, in the profiling phase, each time we only focus on a single byte of the round key, which we call *subkey*, and treat the other bytes as independent noise. We then train the leakage model on each byte independently; in the attack phase, we still attack each byte independently and then combine all the guessed subkeys to construct the round key.

We use the *successful-recovery rate* as the measure for evaluation, which is defined as the

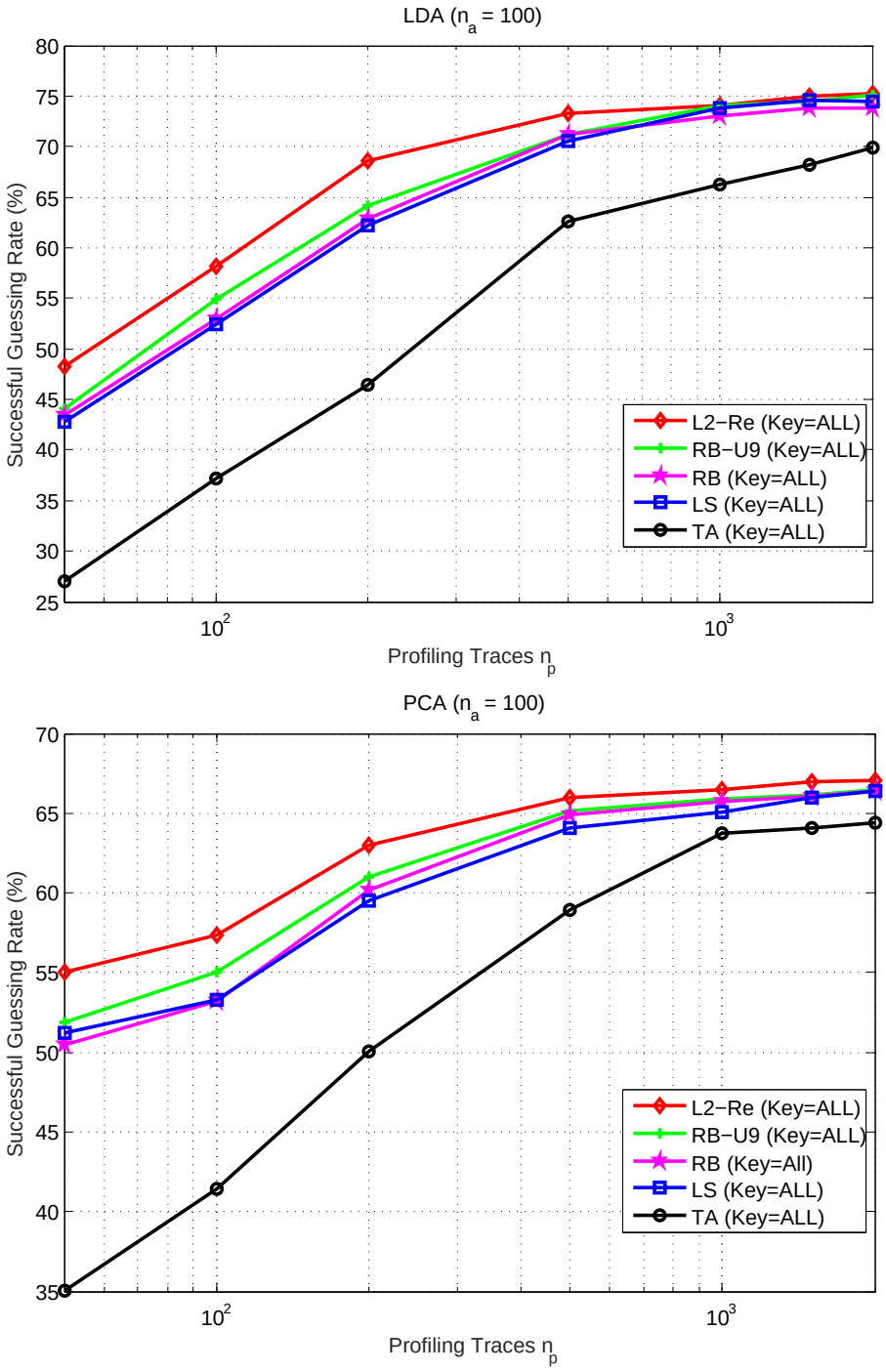

Figure 4.4: Probability of Successful Guessing versus Different Profiling Traces n_p with Attack races $n_a = 100$ under LDA and PCA Compression (Reprinted with permission from [3])

average of the successful recovery probability for attacking **all** subkyes:

$$\text{Successful-Recovery Rate} = \frac{\sum_{i=1}^{N_e} P_{hit}(i)}{N_e} \ , \tag{4.24}$$

where

$$P_{hit}(i) = \frac{N_{hit}(i)}{I} \ .$$

Here I is the number of bytes, N_e is the number of the independent experiments, and $N_{hit}(i)$ is how many bytes that have been successfully guessed in the i-th experiment.

We use the HW-model based attack as the baseline to compare with the linear-model based methods: LS, RB-U9, and our L2-Re. Since the HW-model based attack is a non-profiled side channel attack, in the experiments, we report its successful-recovery rate by averaging the recovery rate on all the cases where N_a is fixed but N_p is different. This is valid because for any fixed number of attack traces, the successful-recovery rate of HW-model will not change when the number of profiling traces changes.

In Fig. 4.5, we present the results of the successful recovery rate for different numbers of profiling traces. The number of attack traces N_a is fixed to 3000 and 4000, respectively, and the number of profiling traces N_p varies from 100 to 8000 in each figure. From the two sub-figures, we can see that when the number of profiling traces is small, for example when N_p is less than 2000, the HW based method works better than the other, profile-based, methods. This is because for limited training resources, the estimation of the leakage model is not accurate, due to the strong noise corruption. As a result, profiled side-channel attacks can not show their full power. However, since we can do the training offline, we are able to spend more resource for improving the profiling performance. With more traces collected for profiling, for example $N_p > 3500$, we can see all three profile based methods are significantly better than the HW model, because more accurate leakage models have been built at this time. When $N_a = 3000$ and $N_p = 6000$, the recovery rate of our reweighted algorithm is about 77%, LS is near 71%, RB-U9 is 72.6% and HW is only

around 55%. Besides, we can see in general, our reweighted algorithm always outperforms other two profile-based methods: LS and RB-U9. As a example, when $N_a = 3000$ and $N_p = 3000$, our method has about 8% performance improvement compared to LS, and around 5% performance improvement compared to RB-U9. The experimental results in AES implementation also support the effectiveness of our algorithm in leakage model building.

From the experimental results in Fig. 4.5, we can see that when the number of profiling traces is small, the performance of the profile-based methods are poor. This is because when we use the DC-based attack, each time we only focus on an independent part of the key and treat others as noise. This leads to additional noise on each byte. As a result, each byte has to encounter a strong noise corruption, especially when the number of profiling traces is small. Hence, we need to investigate whether there are methods that can improve the profiling performance of the leakage model under the DC strategy. We will discuss this in next chapter.

4.7 Conclusion

In this chapter, we proposed a ℓ_2-norm based re-weighted algorithm to estimate the leakage model in profiling phase, which outperforms the previous LS method and Ridge-based method as illustrated by the experimental results. We also discussed the convergence behavior of our algorithm. In the attacking phase, we showed how to build the templates based on the proposed model. We also discussed the advantage of our scheme when attackers only obtain leakage traces that come from a subset keys compared to the standard profiled side-channel attack. Finally, experimental results are provided to show the effectiveness of our proposed scheme.

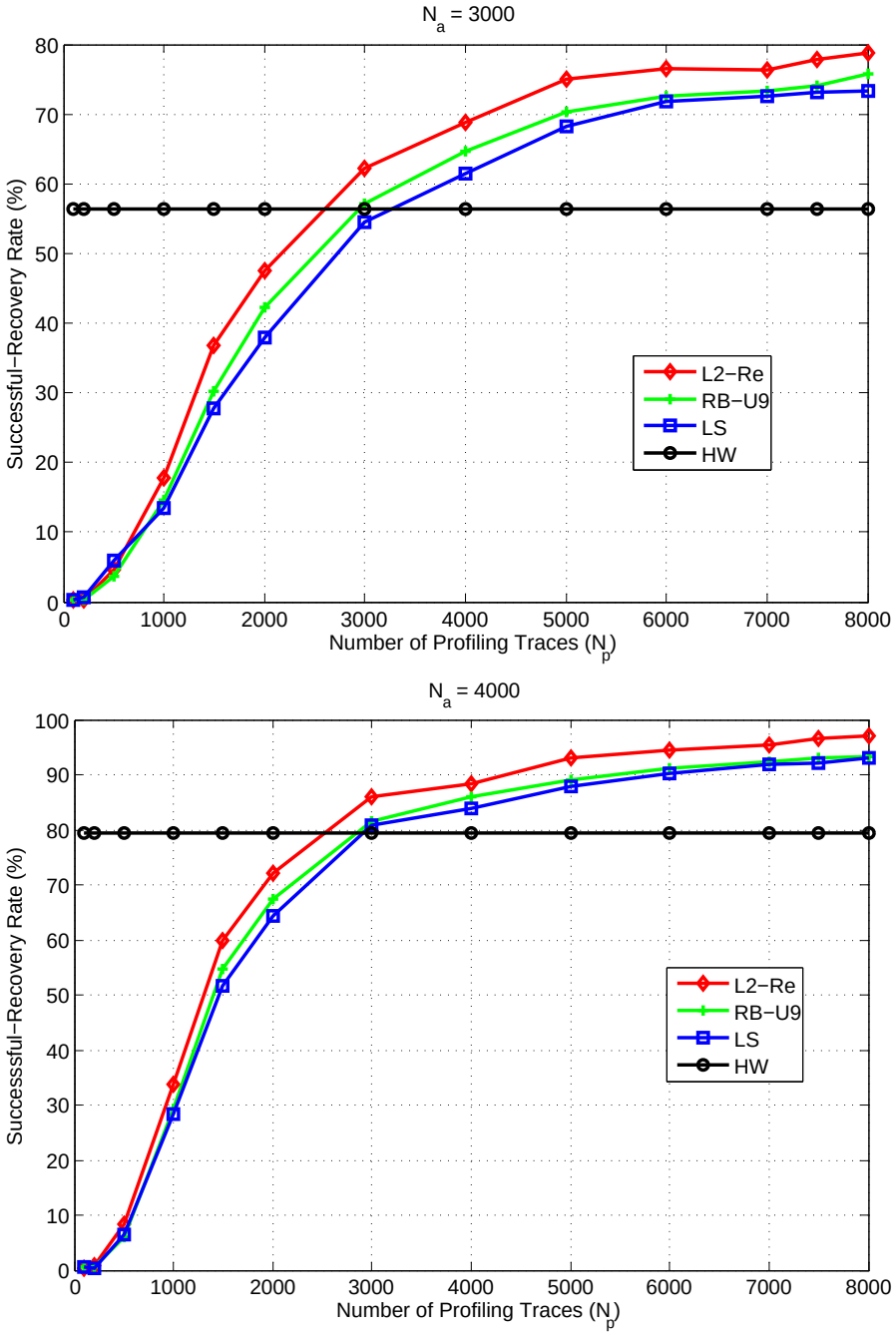

Figure 4.5: Probability of Successful Recovery versus Different Profiling Traces N_p with Different Attack Traces $N_a = 3000$, and $N_a = 4000$

5. EXTENDING THE STOCHASTIC MODEL TO NONLINEAR SYSTEMS

5.1 Introduction

As we discussed in Chapter 4, profiled side-channel attack have shown to be successful in breaking a number of cryptographic algorithms. Their performance, however, relies on the quality of the profiling to get the leakage function. The scalability (data complexity) of profiling is also a key point to efficiently implementing the profiled side-channel attack, especially for the Template Attack. For small-size system, for example 8-bit register, the profiling cost of the Template Attack is acceptable since there are only 256 templates that need to be built. Many systems have a much larger key size. AES-based systems [18], for example, have key sizes that can be 128-bit long (for AES 128) or even 256-bit or longer. In the case of AES 128, directly applying the Template Attack on the AES implementations will exponentially increase the computing cost (now the number of templates is 2^{128}), which is a hard task. Although the Stochastic Model can lead to significant saving in profiling cost by using a small number of keys to build the leakage function as we discussed in Chapter 4, attacking the full key in an AES system at once is also not a wise idea, since based on the problem of selecting the key with maximum likelihood [9, 21, 22], now the guessed key is selected from all 2^{128} possible candidates. Hence, how to efficiently implement side-channel attacks in breaking complex cryptographic systems such as AES is non-trivial.

To address the above issues, the majority of side-channel attacks usually apply a *divide-and-conquer* (DC) strategy to separately recover the partial secrets and then combine them to form the full secret key, such as in [16, 17, 46]. In this strategy, the attackers focus on attacking independent parts of the key (these are called subkeys) separately, and then combine all the subkeys to construct the key. One successful example of applying DC for side channel attacks is the Correlation Power Analysis (CPA) [7]. In CPA, the attacker first collects a set of power traces using different plaintext. Then, for each byte, the attacker generates a set of hypothesis power vectors, which are based on the leakage model and the candidates for the subkey, and computes the correlation coefficient

between the hypothesis power vector and the vector of the real power data. The attacker then picks that guessed subkey among the candidates which has the maximum correlation value. Finally, combing the subkeys from each byte yields the full key.

We note that in the DC strategy the attacker only focuses on one byte value at a time. For the special case of the Stochastic Model this means that during profiling, the attacker will only consider the value of the target byte, while at the same time treating the other bytes as independent noise. This has the problem that if the the number of profiling traces is small, the measurements for each byte will be corrupted by strong noise, which comes from the real independent noise in the power signal, as well as the leakage generated from other bytes during computation. We also note that using a DC strategy in combination with the Stochastic Model also results in having to generate a separate leakage model for each target byte. In the case of AES 128 the attacker would have to generate and manage 16 leakage models. This increases the data complexity for storing the leakage models on the attackers' side. Hence, finding approaches to construct a general leakage model that can be applied to all bytes, and also improve the profiling performance by reducing the overall noise corruption on leakage model building, will be an interesting and also important problem.

In this chapter, we will discuss how to efficiently extend the Stochastic Model to a class of non-linear cryptographic systems, such as the implementations of AES. More specifically, we explore methods to build leakage models that perform better than the previous widely-used independent model for breaking the AES implementations. Our first method is based on the idea that taking the average of the independent models reduces the noise corruption on the general model and leads to what we call the average measurement model. Another method is based on the assumption that the leakage model coefficient for the same bit position of each byte can be approximated, which develops the approximated model from the long linear model [*]. We compare models with the HW model for non-profiled side-channel attacks, and also the independent model for profiled

[*]The long linear model is defined as a linear model which considers all bits in an AES implementation. For example, in AES 128, the number of the coefficients in the long linear model is 128 (or 129, which considers an additional bit as we introduced in Section 3.2.2).

side-channel attacks. The experimental results show that our models has significant performance improvements in terms of key recovery rate compared to the state-of-the-art methods.

In the following, we will first introduce the AES algorithm and its vulnerabilities with regards to side-channel attacks. After recalling the mathematical model of the Stochastic Model, we then discuss how to efficiently extend the Stochastic Model to the case of AES implementations. Finally, we present the experimental results for our proposed scheme.

5.2 Implementation of AES: Overview

The Advanced Encryption Standard (AES) [18] is a widely-used symmetric encryption algorithm [47]. Fig. 5.1 illustrates the framework of the *encryption* algorithm of the 128-bit version of AES, called AES 128. At the very beginning of the encryption, the *secret key* (we call it K_0) is expanded into a number of so-called *round keys*, which are $K_1, K_2, \ldots, K_{10}$, by using the so-called Rijndael's key schedule [18] [†]. In the following, for simplicity, we enumerate the rounds by calling the first AES round the *0-th round* and the last AES round the *10-th round*.

In many standard AES implementations, the secret key and the plaintext are represented as a square matrix, which in the case of AES 128 is of size 4×4 byte. The first step of the encryption is the computation of the exclusive-or between the plaintext and the secret key. The result of the exclusive-or is called the *state*. Then, the state is sent into the following operations, which will be repeated 9 times:

1. *SubBytes*: Each byte in the state is replaced with another byte from a 8-bit substitution box.

2. *ShiftRows*: For each row of the state, the bytes in the row are cyclically shifted by a number of positions that is different for each row.

3. *MixColumns*: For each column of the state, a linear transformation is computed with the matrix whose elements are generated from $\mathbf{GF}(2^8)$.

4. *AddRoundKey*: Each byte in the state is combined with the corresponding subkey of current round key through bitwise XOR.

[†]The 256-bit version of AES, called AES 256, uses 12 round keys, $K_1, K_2, \ldots, K_{12}$.

In the last round, the above operations are still performed, but the *MixColumns* is excluded. We call the output of the encryption algorithm the *ciphertext*.

Fig. 5.2 illustrates the framework of the AES 128 *decryption*. The steps in AES decryption are exactly the reverse of those in the AES encryption. The operations for decryption are the inverse versions of the operations used in encryption: *InvSubBytes*, *InvShiftRows*, *InvMixColumns*, while the inverse of *AddRoundKey* is just itself.

Previous work shows that side-channel attacks on AES can be launched by exploiting the power consumption leakage from a hardware structure, such as a register or a data bus. This attack is typically performed on either the first round, where the plaintext is known, or the last round, where the ciphertext is obtained, during the AES encryption [45]. The power consumption is dependent on the internal state $\boldsymbol{S}$ in the AES implementation, which is the result of the XOR operation between the state of the hardware before and after the cryptographic operations, respectively, and is represented as

$$\boldsymbol{S} = \mathbf{XOR}(\boldsymbol{S}_{pre}, \boldsymbol{S}_{post}) = \boldsymbol{S}_{pre} \oplus \boldsymbol{S}_{post} \ , \tag{5.1}$$

where **XOR** stands for the exclusive-or operation, $\boldsymbol{S}_{pre}$ is the state before the operations, and $\boldsymbol{S}_{post}$ is the state after the operations. The computation in each round is broken down into multiple blocks (bytes), and the operations on each byte are also independent to each other. Hence, all the states in Eq. (5.1) are on one *byte*.

In typical non-profiled version of side channel attacks, the power consumption is modeled as the Hamming Weight of the internal state [7], which is:

$$\boldsymbol{L} = \alpha \mathbf{HW}(\boldsymbol{S}) + \boldsymbol{R} \ , \tag{5.2}$$

where $\boldsymbol{L} \in \mathbb{R}^m$ is a vector of leakage samples with the size of m, $\boldsymbol{R} \in \mathbb{R}^m$ is the noise vector, **HW** is the Hamming Weight function, and α is the scalar gain. In comparison, in profiled side-channel attacks, the relationship between the power consumption and the state is still unknown

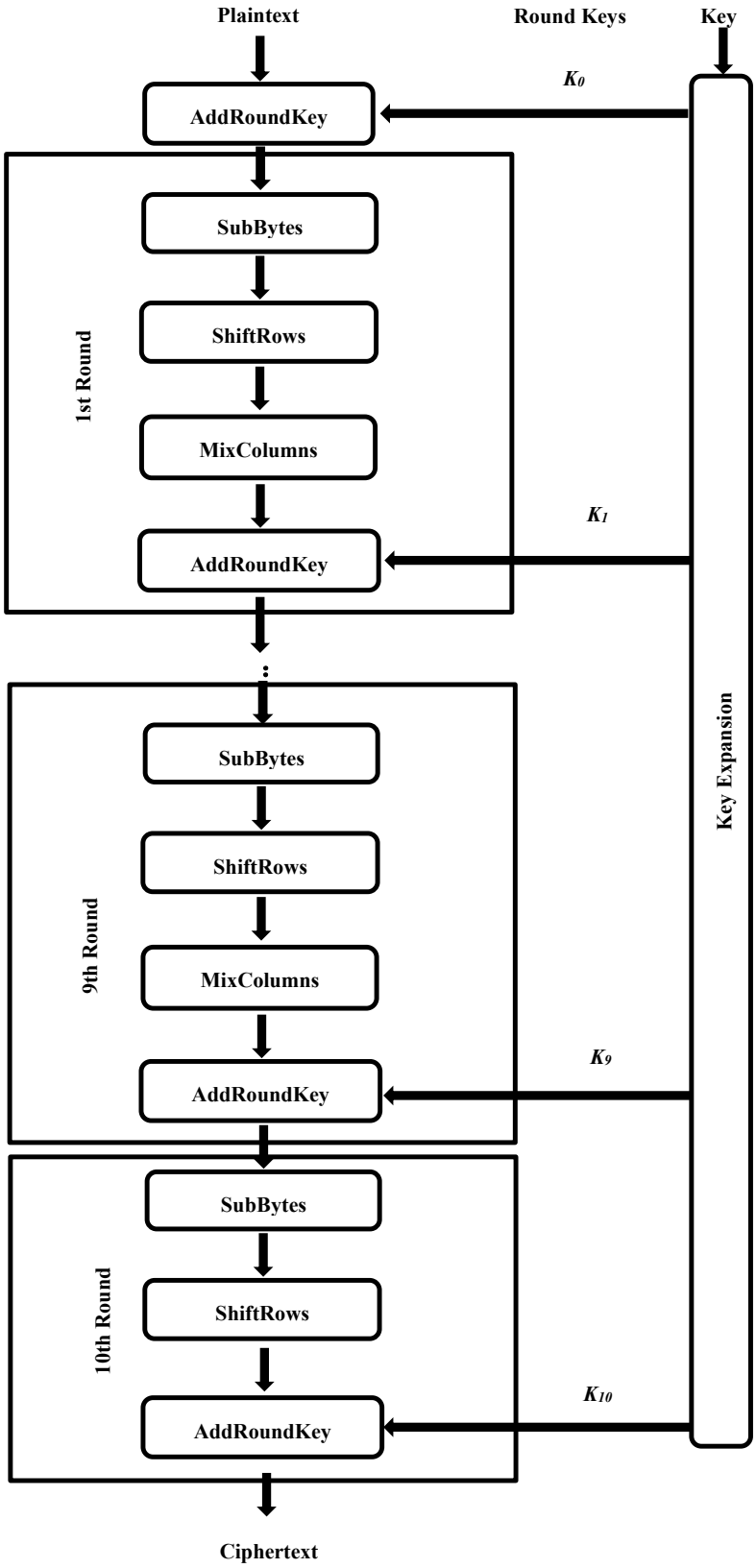

Figure 5.1: The Framework for AES Encryption

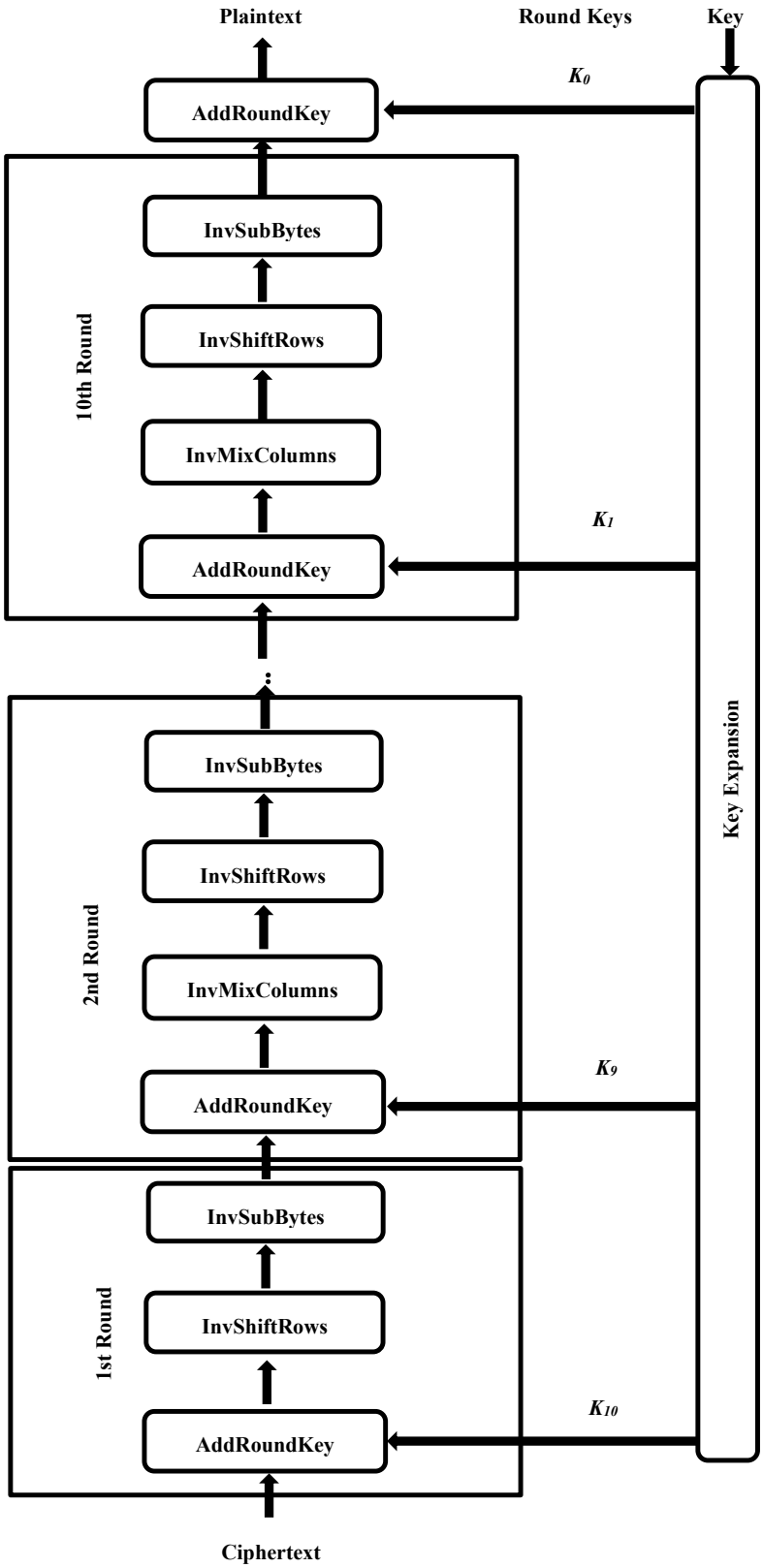

Figure 5.2: The Framework for AES Decryption

before profiling.

In this dissertation, for our study of side channel attack on AES implementations, we target the leakages and cryptographic operations in the *last* round. In the last round of the AES, the internal state is the result of the changes between the state in the 9-th round and the state in the 10-th round, which is the ciphertext itself. In next section, we will introduce how to build the leakage model between the power leakage trace and the internal state using the Stochastic Model.

5.3 Background for Stochastic Model

From our work in Chapter 3, we know that the leakage function can be modeled as a weighted linear model. Hence, by applying Eq. (5.1) to the weighted model in Eq. (3.4), we can obtain a linear relationship between the power consumption and the internal state, which is:

$$\boldsymbol{L} = \boldsymbol{W} F_b(\boldsymbol{S}) + \boldsymbol{R} \ , \tag{5.3}$$

where m is the sample point, $\boldsymbol{L} \in \mathbb{R}^m$ is the samples vector of a leakage trace, $\boldsymbol{R} \in \mathbb{R}^m$ is the independent random part, which is modeled as noise, and $F_b(\boldsymbol{S}) \in \mathbb{R}^{N \times B+1}$ is the vector of a binary string, which is defined as

$$F_b(\boldsymbol{S}) = \begin{cases} c, \text{ where } b = 0 \\ \text{bit}_b(\boldsymbol{S}), \text{ where } 1 \leq b \leq B \ , \end{cases}$$

where c is a constant number, and B is the number of bits in the structure (now for one byte, it is 8). The definition of $\text{bit}_b(\boldsymbol{S})$ is the b-th bit value of the binary string of the byte state $\boldsymbol{S}$. $\boldsymbol{W} \in \mathbb{R}^{m \times (B+1)}$ is the weight matrix of the power leakage, where each element w_{ij} ($1 \leq i \leq m, 0 \leq j \leq B$) is the weight coefficient at sample point i for the leakage caused by bit $j+1$.

5.4 The Efficient Implementations of Stochastic Model on AES

We discussed in Chapter 4 how the Stochastic Model works very well in systems with small-size keys, like the case of the 8-bit CPU Atmel XMEGA 256 A3U micro-controller [26, 3]. We

also discussed that directly applying the Stochastic Model to more complex systems such as AES is very costly: while the Stochastic Model significantly reduces the cost of *profiling*, the cost of *attacking* remains unchanged. In the case of AES 128, the attacker computes the likelihood values for all 2^{128} keys for one attack. Hence, we need to find ways to efficiently implement the Stochastic Model in side-channel attacks in ways that are particularly effective against AES implementations.

5.4.1 Profiling Phase for Stochastic Model for Side-Channel Attacks

Before discussing the implementation of the Stochastic Model, first we introduce how the adversary sets up the profiling of the AES module. In the profiling phase, the adversary first records a number N_p of leakage traces $\boldsymbol{x}_i \in \mathbb{R}^m$ ($1 \leq i \leq N_p$). These traces come from a fixed secret key and a set of uniform distributed plaintexts. The size of this set is N and hence $N_p = n_p N$ where n_p is the number of traces collected per plaintext. For our case, each ciphertext only generates a single trace, which leads to $n_p = 1$. Finally, all the traces that come from the N profiled plaintexts are combined together, which leads to a *traces matrix* $\boldsymbol{X} \in \mathbb{R}^{N \times m}$, whose i-th row belongs to plaintext P_i.

Based on Eq. (5.3), for the i-th byte, we have

$$\begin{aligned} \boldsymbol{X} &= (\boldsymbol{W}_i[F_b(\boldsymbol{S}_i^{(1)}) \dots \ F_b(\boldsymbol{S}_i^{(N)})] + [\boldsymbol{R}_i^{(1)} \dots \ \boldsymbol{R}_i^{(N)}])^T \\ &= \boldsymbol{F}_i \boldsymbol{W}_i^T + \boldsymbol{\Phi}_i \end{aligned} \tag{5.4}$$

where $\boldsymbol{S}_i^{(j)}$ is the internal state for the j-th plaintext in the i-th byte with $1 \leq i \leq I$ and $1 \leq j \leq N$, I is the total number of bytes in the system (for AES 128, $I = 16$), $\boldsymbol{W}_i$ is the leakage model on i-th byte, $\boldsymbol{\Phi}_i = [\boldsymbol{R}_i^{(1)} \dots \ \boldsymbol{R}_i^{(N)}]^T$ is the noise matrix, and $\boldsymbol{F}_i \in \mathbb{R}^{N \times B+1}$ acts as the measurement

matrix, which is expressed as:

$$\begin{aligned}\boldsymbol{F}_i &= [F_b(\boldsymbol{S}_i^{(1)}) \dots \ F_b(\boldsymbol{S}_i^{(N)})]^T \\ &= \begin{bmatrix} c & \text{bit}_1(\boldsymbol{S}_i^{(1)}) & \dots & \text{bit}_B(\boldsymbol{S}_i^{(1)}) \\ \vdots & \vdots & \vdots & \vdots \\ c & \text{bit}_1(\boldsymbol{S}_i^{(N)}) & \dots & \text{bit}_B(\boldsymbol{S}_i^{(N)}) \end{bmatrix} \ . \end{aligned} \tag{5.5}$$

Finally, based on the leakage traces matrix $\boldsymbol{X}$ and the measurement matrix $\boldsymbol{F}_i$, the leakage model on an i-th byte $\boldsymbol{W}_i$ can be obtained by solving the linear equations in Eq. (5.4).

5.4.2 Independent Model

The main idea of the implementation of the *independent model* is based on the divide-and-conquer strategy [16, 17, 45]: when the adversary performs the profiling, for each byte, she considers that the target byte is known, and the remaining bytes are represented by noise. That is, for each byte, the attacker solves the linear equation in Eq. (5.4), and repeat this procedure I times, as:

$$\begin{aligned} {\boldsymbol{W}_1}^T &= (\boldsymbol{F}_1)^{-1}\boldsymbol{X} \ , \\ {\boldsymbol{W}_2}^T &= (\boldsymbol{F}_2)^{-1}\boldsymbol{X} \ , \\ &\vdots \\ {\boldsymbol{W}_I}^T &= (\boldsymbol{F}_I)^{-1}\boldsymbol{X} \ . \end{aligned} \tag{5.6}$$

In the profiling phase, the attacker obtains the leakage model $\boldsymbol{W}_i$ $(1 \leq i \leq I)$ for each byte. In the subsequent attack phase, for each byte, the attacker applies the leakage model profiled for that byte on the recorded leakage traces.

We discussed earlier that an attacker using the DC strategy for profiling only considers the target byte, and treats the other bytes as independent noise. This will lead to noise corruption on the measurements for the target byte. In addition, errors in the profiling will compound: if the leakage model for one byte is inaccurate, the recovery of the entire key fails too. Hence, in the

following, we will explore more efficient methods to develop models that are more accurate than the independent model.

5.4.3 General Model: Average Measurement

The problem with independent model is that if any byte's model is inaccurate, the attack on that byte will fail and so will the attack on the full key. However, if we use a general model to represent all the independent leakage models, we can exploit the joint structure among all the independent models, and reduce the noise corruption on the leakage model's building. Hence, we can use an average model on all the bytes, instead of a separate model for each of them.

The expression of the **average model** can be easily obtained:

$$\overline{\boldsymbol{W}} = \frac{\sum_{i=1}^{I} \boldsymbol{W}_i}{I} \ . \tag{5.7}$$

In this way, the attacher can use a general model $\overline{\boldsymbol{W}}$ for all the bytes and attack each subkey of the master secret key through this general model.

Now, let's consider another question: if the average model works, it should also satisfy the linear relationship between the "averaged" internal state and the power consumption in the target hardware structure. Based on Eq. (4.2), let's assume there is a matrix $\overline{\boldsymbol{F}}$ that satisfies the following equation:

$$\boldsymbol{X} = \overline{\boldsymbol{F}} \cdot \overline{\boldsymbol{W}}^T + \overline{\boldsymbol{\Phi}} \ , \tag{5.8}$$

where $\overline{\boldsymbol{\Phi}}$ is assumed to be the average noise matrix across all bytes during the encryption operations.

Since from Eq. (5.4), we have

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{F}_1 {\boldsymbol{W}_1}^T + \boldsymbol{\Phi}_1 \ , \\ \boldsymbol{X} &= \boldsymbol{F}_2 {\boldsymbol{W}_2}^T + \boldsymbol{\Phi}_2 \ , \\ &\vdots \\ \boldsymbol{X} &= \boldsymbol{F}_I {\boldsymbol{W}_I}^T + \boldsymbol{\Phi}_I \ , \end{aligned} \tag{5.9}$$

where $\boldsymbol{F}_i$ is the measurement matrix for the i-th byte, and $\boldsymbol{\Phi}_i$ is the latter's noise matrix. Based on Eq. (5.8) and Eq. (5.9), we can obtain

$$\overline{\boldsymbol{F}} = \left[\frac{\sum_{i=1}^{I} \boldsymbol{F}_i^{-1}}{I} \right]^{-1} \ . \tag{5.10}$$

In this way, during the profiling phase, the attackers can obtain a general average model $\overline{\boldsymbol{W}}$ through the following equation:

$$\overline{\boldsymbol{W}}^T = (\overline{\boldsymbol{F}})^{-1} \boldsymbol{X} \ . \tag{5.11}$$

We call this method as **average measurement** method.

5.4.4 General Model: Approximated Long Linear Model

The approximated long linear model is also based on the main idea of the Stochastic Model. The rationale for this model is as follows: if we consider all the 128 bits of an AES 128 system at same time, the linear model between a leakage trace and the state that considering all bytes is:

$$\boldsymbol{L} = \sum_{i=0}^{128} \boldsymbol{W}_{:,i} \boldsymbol{S}_K(i) + \boldsymbol{R} \ , \tag{5.12}$$

where $\boldsymbol{S}_K(i)$ is the i-th bit of the full state array $\boldsymbol{S}_K$ (we set $\boldsymbol{S}_K(0) = 1$ here), $\boldsymbol{W}_{:,i}$ is the leakage model coefficients for the i-th column of $\boldsymbol{W}$, which is the leakage model for the 128-bit

system, and $\boldsymbol{R}$ is the noise matrix.

As we mentioned before, the computing cost for Eq. (5.12) is huge since there are 128 coefficients to be estimated during profiling and 2^{128} classifiers to be built for computing the likelihoods during the attack. Hence, to efficiently apply the linear model in long-key side channel attacks, we do some approximation of this long linear model.

First, from Eq. (5.12), we have:

$$\begin{aligned}\boldsymbol{L} &= \sum_{i=0}^{B_I} \boldsymbol{W}_{:,i}\boldsymbol{S}_K(i) + \boldsymbol{R} \ , \\ &= \boldsymbol{W}_{:,0} + \left(\sum_{i=1}^{I} \boldsymbol{W}^1_{:,i}\boldsymbol{S}_i(1)\right) + \left(\sum_{i=1}^{I} \boldsymbol{W}^2_{:,i}\boldsymbol{S}_i(2)\right) + \cdots + \left(\sum_{i=1}^{I} \boldsymbol{W}^B_{:,i}\boldsymbol{S}_i(B)\right) + \boldsymbol{R} \ ,\end{aligned}$$

where $B = 8$ for one byte, B_I is the total number of bits in the system ($B_I = 128$ when $I = 16$), $\boldsymbol{W}^j_{:,i}$ is the leakage model coefficient for the j-th bit in the i-th byte, and $\boldsymbol{S}_i(j)$ is the bit value of the state in the j-th bit on the i-th byte, with $1 \leq j \leq B$ and $1 \leq i \leq I$.

Since the bits in the same bit position should have a similar physical behavior across all bytes, we assume that the coefficients for the *same* bit position are *identical* for all bytes, that is:

$$\begin{aligned}\boldsymbol{W}^1_{:,1} = \boldsymbol{W}^1_{:,2} = \cdots = \boldsymbol{W}^1_{:,I} = \boldsymbol{w}^{(1)} \ , \\ \boldsymbol{W}^2_{:,1} = \boldsymbol{W}^2_{:,2} = \cdots = \boldsymbol{W}^2_{:,I} = \boldsymbol{w}^{(2)} \ , \\ \vdots \\ \boldsymbol{W}^B_{:,1} = \boldsymbol{W}^B_{:,2} = \cdots = \boldsymbol{W}^B_{:,I} = \boldsymbol{w}^{(B)} \ .\end{aligned}$$

Hence, we can model an approximated linear model to describe the leakage emitted through

the side channel as follows:

$$\begin{aligned}\boldsymbol{L} &= \boldsymbol{W}_{:,0} + \left(\boldsymbol{w}^{(1)} \sum_{i=1}^{I} \boldsymbol{S}_i(1)\right) + \left(\boldsymbol{w}^{(2)} \sum_{i=1}^{I} \boldsymbol{S}_i(2)\right) + \cdots + \left(\boldsymbol{w}^{(B)} \sum_{i=1}^{I} \boldsymbol{S}_i(B)\right) + \boldsymbol{R} \\ &= [\boldsymbol{w}^{(0)}\ \boldsymbol{w}^{(1)}\ \boldsymbol{w}^{(2)}\ \ldots\ \boldsymbol{w}^{(B)}] \begin{bmatrix} c \\ \sum_{i=1}^{I} \boldsymbol{S}_i(1) \\ \sum_{i=1}^{I} \boldsymbol{S}_i(2) \\ \vdots \\ \sum_{i=1}^{I} \boldsymbol{S}_i(B) \end{bmatrix} + \boldsymbol{R} \\ &= \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{S}_K} + \boldsymbol{R}\ , \end{aligned} \tag{5.13}$$

where we set $\boldsymbol{W}_{:,0} = \boldsymbol{w}^{(0)}$, $\widetilde{\boldsymbol{W}} = [\boldsymbol{w}^{(0)}\ \boldsymbol{w}^{(1)}\ \boldsymbol{w}^{(2)}\ \ldots\ \boldsymbol{w}^{(B)}]$, and

$$\widetilde{\boldsymbol{S}_K} = \begin{bmatrix} c \\ \sum_{i=1}^{I} \boldsymbol{S}_i(1) \\ \sum_{i=1}^{I} \boldsymbol{S}_i(2) \\ \vdots \\ \sum_{i=1}^{I} \boldsymbol{S}_i(B) \end{bmatrix}\ .$$

Now, based on Eq. (5.4), we can easily estimate the approximated linear model by computing:

$$\widetilde{\boldsymbol{W}}^T = (\widetilde{\boldsymbol{F}})^{-1}\boldsymbol{X}\ , \tag{5.14}$$

where

$$\begin{aligned}\widetilde{\boldsymbol{F}} &= \left[\widetilde{\boldsymbol{S}_K^{(1)}}\ \widetilde{\boldsymbol{S}_K^{(2)}} \ldots \widetilde{\boldsymbol{S}_K^{(N)}}\right] \\ &= \left[c\boldsymbol{I}_{N\times 1}\quad \sum_{i=1}^{I} \boldsymbol{F}_i^*\right]\ . \end{aligned} \tag{5.15}$$

In Eq. (5.15), $\boldsymbol{I}_{N\times 1} = [1, 1, \ldots, 1]^T \in \mathbb{R}^{N\times 1}$ and $\boldsymbol{F}_i^*$ is the submatrix of $\boldsymbol{F}_i$ excludes the first

column.

5.5 Correlation Power Analysis on AES Implementation

Once the attacker obtains the leakage model, she can proceed to perform the profiled side-channel attack. In the correlation power analysis, the attacker first records N_a leakage traces from N_a different ciphertexts but under the same key K^* on the target device. Since the attacker only targets the last round of the AES encryption algorithm, she only stores the sample points in the last round, which is expressed by $\boldsymbol{X}_{K^*} \in \mathbb{R}^{N_a \times m}$. The attacker also has to access to a ciphertext table $\boldsymbol{C}$, which has N_a rows with each row having I byte ciphers. Starting from here, the attacker computes the correlation between the power leakage traces and the hypothesis power vector, which is computed based on the candidate subkey and the power leakage model, where the leakage model could be the HW/HD model in the case of the non-profiled side channel attacks [45], or one of the implementations of the Stochastic Model in the case of profiled side channel attacks, which we summarized in Section 5.4.

Although the obtained traces have a large number of samples, only a small number of sample points has useful leakage information. Hence, usually attackers only collect a very small number of samples from each trace for further analysis. We follow the lead in [45] and choose the one sample in the trace of the AES last round with the largest power value. Note that this means that $m = 1$.

Since our target is the last round of the AES encryption, the input of the attack algorithm includes the ciphertext $\boldsymbol{C}$, the power traces $\boldsymbol{X}_{K^*}$, and the leakage model $\mathbf{LM}$, which is defined as:

$$\mathbf{LM} = \begin{cases} \mathbf{HW}, \text{ for Non-profiled Attack;} \\ \boldsymbol{W}, \text{ for Profiled Attack;} \end{cases}$$

here $\boldsymbol{W}$ could be independent leakage model, average measurement model, or the approximated model, as shown above.

The main idea of our attack algorithm is based on the DC-strategy based CPA, which is de-

scribed in [45]. In contrast to [45], which is a non-profiled attack where the leakage model is based on the HW/HD model, we use a profiled side-channel attack where the leakage model is trained during a profiling phase by the methods that we discussed in Chapter 4. To better illustrate the attack procedure, we go through each steps of the algorithm in the following.

The first step in the attack algorithm initializes a multi-dimensional vector $\boldsymbol{P}_{ts} \in \mathbb{R}^{I \times 2^B \times N_a \times m}$ which is used to store the hypothesis power vectors for each subkey candidate on each byte, and a matrix $\boldsymbol{R}_{ts} \in \mathbb{R}^{I \times 2^B}$, which stores the correlation coefficients between the real power traces and the hypothesis power vectors for each subkey candidate for each byte.

Once the initialization has been completed, the algorithm decrypts the ciphertext by applying the inverse of the AES last-round operations, as is shown in Fig. 5.2. Since the last-round key is the target and of course unknown to the attacker, the attacker has to decrypt the ciphertext byte with every possible hypothesis on the last-round key byte (256 possible guesses for B=8). As a result, for a given ciphertext, and for every byte, the attacker has its final state (also called as *post state*), which is the ciphertext byte, and also $2^B = 256$ possible input bytes (also called as *pre states*). As a result, the attacher can obtain 256 possible internal states from Eq. (5.1). By applying all 256 possible internal states to the leakage model $\mathbf{LM}$, the attacker obtains 256 hypothesis power values for one byte in a ciphertext. Since the attacker collects the traces from N_a ciphertexts, the above procedure repeats N_a times and finally fills all the values into the matrix $\boldsymbol{P}_{ts}$.

To recover the round key, the attacker follows the DC strategy and analyzes each byte individually. Since $\boldsymbol{P}_{ts}$ is already obtained, for each byte, the attacker goes through all 256 hypothesis power vectors, and iteratively computes the correlation between the chosen hypothesis power vector and the vector of the real power traces $\boldsymbol{X}_{K^*}$. The computing of the correlation uses the *Pearson Correlation Coefficient* [48] method. At the end, the attacker finds the guessed subkey by picking the one that has the highest correlation result. This process is repeated until all subkeys are recovered. The last-round key is the combination of the recovered subkeys. In order to get the AES secret key, the attacker only needs to reverse the key expansion schedule, which is a standard algorithm in AES decryption, and inputs the last-round key. The output will be the secret key, which is

the attacker's ultimate target. To better illustrate the attack procedure, the framework of the attacking algorithm is described in **Algorithm 3**. In this description, the function **Pearson** computes the Pearson Correlation Coefficient, the vector **RoundKey** is the guessed round key in the last round, the function **InvKeyExpansion** performs the reverse of the AES key expansion schedule, and the variable **AESKey** is the AES secret key after reversing the round key. More details about how to implement CPA to attack the 10-th round of AES 128 can be found in [45].

Algorithm 3 Correlation Power Analysis based Side Channel Attack (Derived from [45])

Input: $\boldsymbol{C}, \boldsymbol{X}_{K^*}$, **LM**
  Initialize: $\boldsymbol{P}_{ts}, \boldsymbol{R}_{ts}$

  For $i = 0, \ldots, N_a - 1$
    For $j = 0, \ldots, I - 1$
      PostState $= \boldsymbol{C}[i][j]$;
      For $k = 0, \ldots, 2^B - 1$
        PreStateID = **InvShiftRows**(j);
        PreState $= \boldsymbol{C}[i]$[PreStateID];
        PreState = **AddRoundKey**(PreState, k);
        PreState = **InvSubBytes**(PreState);
        $\boldsymbol{P}_{ts}[j][k][i]$ = **LM**(PreState $\oplus$ PostState);
      End
    End
  End

  For $j = 0, \ldots, I - 1$
    For $k = 0, \ldots, 2^B - 1$
      $\boldsymbol{R}_{ts}[j][k]$ = **Pearson**$(\boldsymbol{P}_{ts}[j][k], \boldsymbol{X}_{K^*})$;
    End
    Computes $k^* := \text{argmax}_{\{k=0\ldots,2^B-1\}} \boldsymbol{R}_{ts}[j][k]$;
    RoundKey$[j] = k^*$
  End
  AESKey=InvKeyExpansion(RoundKey);

Output: AESKey

5.6 Experimental Results

Our experiments target an unmasked implementation of the AES 128 algorithm. The data used in these experiments is the same as we used for the performance evaluation in Section 4.6.2, where the attacker's target is a AES 128 core on a Xilinx Virtex-5 LX30 FPGA with a clock signal at 24MHz. The data contains 50000 raw traces, each of which is corresponds to a ciphertext (that is, n_p=1) with 3124 sample points each, with a fixed secret key, which is "00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F". We divided the traces into two parts: a *profiling* set and an *attack* set. In the following description we use N_p to represent the total number of profiling traces and N_a the total number of attack traces.

We compare the recovery performance of the standard Hamming Weight model, the independent models, the average measurement model, and the approximated long linear model. We use the *successful-recovery rate* as the measure for evaluation. This rate is defined as the average of the successful recovery rate for attacking across **all** subkyes, as follows:

$$\text{Successful-Recovery Rate} = \frac{\sum_{i=1}^{N_e} P_{hit}(i)}{N_e} \ ,$$

where $P_{hit}(i) = \frac{N_{hit}(i)}{I}$. Here I is the number of bytes, N_e is the number of independent tests, and $N_{hit}(i)$ is how many bytes that are successfully recovered in the i-th test. During the attack phase, we independently perform the attacks 100 times by randomly picking the leakage traces from the *attack* set. Since the Hamming Weight model based attack is not profiled, the value of N_p is irrelevant. In order to compare the performance of this model with those of the other profile-based models, when in the case where N_a is fixed but N_p varies, we do the experiments $N_e \cdot T$ times under the fixed N_a, where T is the number of different N_p, and then take the average. Hence, the successful recovery rate of the Hamming Weight model is constant for different N_p at this time.

In the following, we use the term **Hamming Weight** to represent the Hamming weight model, **Independent Model** to denote the independent model, **Average Measurement** for the average measurement model, and **Approximated Model** to denote the approximated linear model. For

all the profile-based methods, in order to make fair comparisons, we use the Least Squares (LS) method to take the inverse of the measurement matrix for computing the coefficients of the leakage model.

In Fig. 5.3, we show the results of the successful recovery rate for different numbers of profiling traces. The number of attack traces, N_a, is fixed to 4000 and 5000 respectively, and the number of profiling traces N_p varies from 100 to 3000 in each figure. The result indicates that for very small numbers of profiling traces, such as $N_p = 100$, the profiled side-channel attacks are all worse than the, non-profiled, Hamming Weight based side channel attack. This is natural because when the number of profiling traces is small, the estimation of the leakage model is not accurate. We also note that even in this case, the recovery performance of the **Average Measurement** and the **Approximated Model** are still much better than that of the **Independent Model**. For example, when $N_p = 100$ and $N_a = 4000$, the recovery rate of the average measurement model is about 62%, that of the approximated model is near 64%, and that of the independent model is only 0.5%. As N_p increases, for example when $N_p \geq 500$, all the general models will have better recovery performance compared to the Hamming Weight model and the independent model. When $N_p = 1500$ and $N_a = 4000$, the recovery rates of both average measurement model and approximated are larger than 91%, compared to the Hamming Weight model, which is around 80% and the independent models at only about 51%. We also note that the recovery rate of independent model is always worse than the other three models, except for the case of $N_p = 3000$, where their performance is close to the Hamming Weight model. This illustrates the independent model's weak resistance against noise: In order to show the same recovery rate, the number of the profiling traces spent for the approximated model is about 200, where for the independent models it is 3000. For our general models, when $N_p = 3000$, they both show excellent recovery performance: near 99% recovery rate when $N_p = 3000$ and $N_a = 5000$.

In practice, an attacker operates under a number of constraints. For example, she may have only limited access to the device, and can therefore only do limited profiling. Similarly, the number of attack traces may be limited as well. We represent these constraints by giving the attacker a given

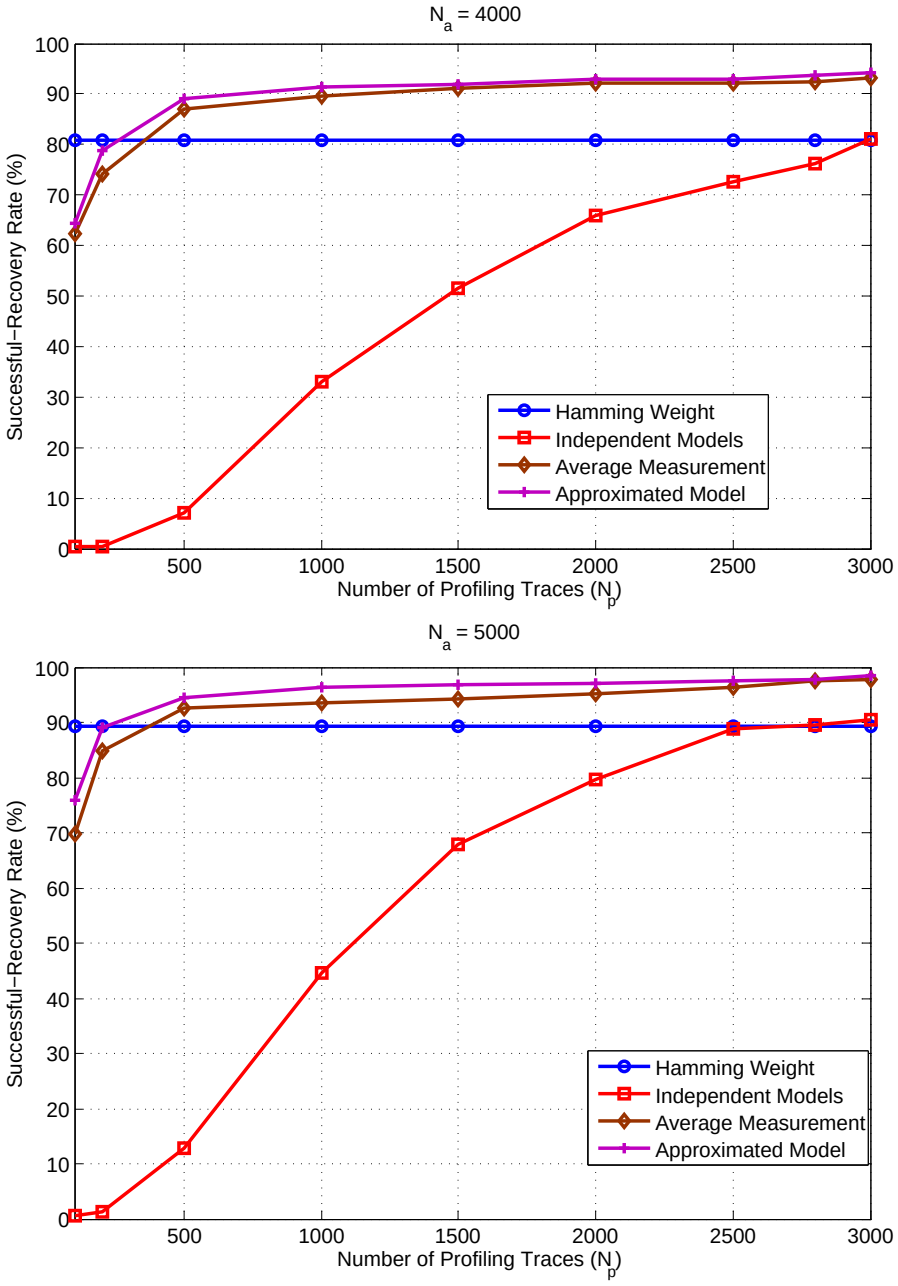

Figure 5.3: Probability of Successful Recovery versus Different Profiling Traces N_p with Different Attack Traces $N_a = 4000$, and $N_a = 5000$

budget, in terms of number of traces available, which the attacker can use either for profiling or for attacking. Fig. 5.4 shows the recovery result for the case where the attacker is given such fixed budgets for measurement. Given a budget, the attacker can decide what portion of it to use for profiling vs. how much for attacking. The question for the attacker is how to best allocate the given budget. We run two experiments, where the overall number traces (the budget) is fixed to 6000 and 8000, respectively. For the case of a 6000 traces budget, N_a varies from 100 to 5900, and hence N_p varies from 5900 to 100. When we increase the budget to 8000 traces, N_a varies from 100 to 7900, and hence N_p varies from 7900 to 100. From the figure we observe for very small value for N_a, such as $N_a < 1000$, the recovery rate of all models is poor. This is because the CPA based attack relies on the number of attack traces. Even though the attacker has a large number of profiling traces, the small number of attack traces leads to a bad performance. For larger numbers of attack traces, say N_a > 1000, the recovery performance of all models increases rapidly. This is because attacker has more traces to do the CPA at this time. However, we notice that when N_a increases to a certain stage (for example $N_a = 4000$ for 6000 total traces), the recovery rate of independent models starts to decrease. In comparison, the recovery rate of the two general models still keeps increasing as the budget get-transferred from profiling to attack. When most of the budget is allocated to profiling, the performance of the independent model starts to decrease. This means that the independent model is more susceptible to the decrease in the number of profiling traces. This also shows that the independent models have a higher requirement on the number of profiling traces in order to get a more accurate estimation of the leakage models. For small number of profiling traces, for example $N_p = 100$, the recovery rate of independent model is near zero, where the two general models still show high performance (the recovery rates for both the average measurement model and the approximated model are larger than 83% even when $N_p = 100$ and $N_a = 5900$). We also find that our general models outperform the independent model for all cases, and are almost better than the Hamming Weight model when $N_p > 200$. For example, when $N_a = 3000$ and $N_p = 3000$, the recovery rates for both average measurement model and approximated are larger than 74%, while at same time Hamming Weight model is

around 54.5% and the independent models is only about 58%. This experiment also shows that our general models can work very well in the case where the attacker only has a very limited traces for profiling the traces. For the case of 8000 overall traces, when $N_p = 100$, the recovery rates for both the average measurement model and approximated are larger than 90%.

5.7 Conclusion

In this chapter, we focused on the problem of how to efficiently apply the Stochastic Model in breaking non-linear cryptographic systems using profiled side-channel attacks. Since usually the cryptographic system has a large number of bits in the secret key, such as 128 bits in the case of AES 128, previous work applies a Divide-and-Conquer strategy to recover the key. Under this strategy, the attacker only focuses on one byte at a time, and treats the other bytes as noise during profiling. This leads to the problem that a model built for each independent byte will be corrupted by the noise that comes both from the independent noise and from the leakage from the other bytes, especially when the number of profiling traces is small. This inaccurate modeling for one single byte will definitely decrease the recovery performance for the whole key. To solve these problems, we propose two methods to build the leakage model for profiled side-channel attacks for the special case where the target is an AES implementation. One method, which we call the average measurement model, is based on the idea of averaging the noise from all the bytes. The second method is based on the assumption that the leakage model's coefficients are identical for each given bit position across all bytes, which leads to a more robust model. We call this the approximated model. Finally, experimental results show the effectiveness of our proposed methods.

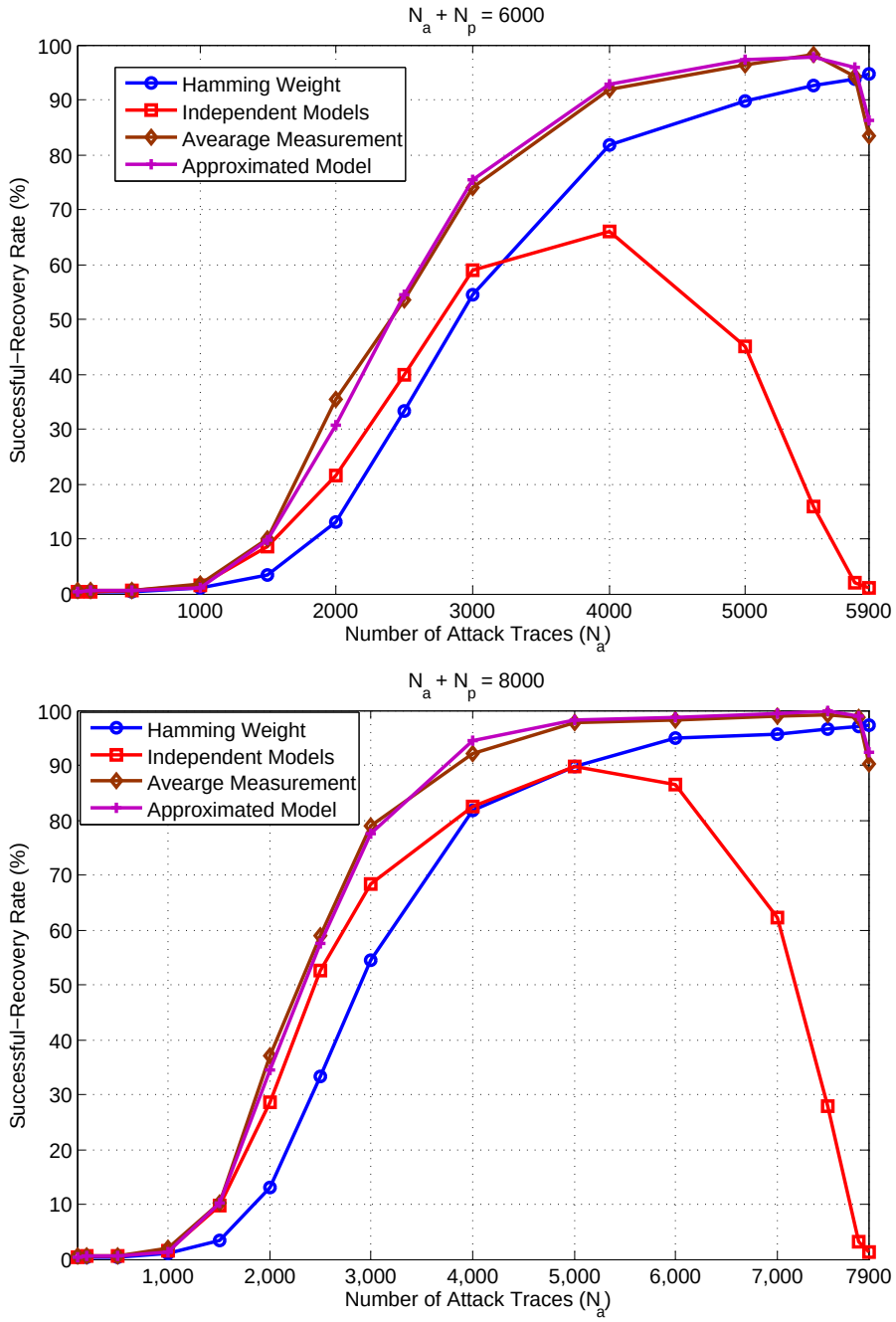

Figure 5.4: Probability of Successful Recovery versus Different Attack Traces N_a with Fixed Budget

6. IMPLEMENTING REINFORCEMENT STRATEGIES IN SIDE CHANNEL ATTACKS

6.1 Introduction

We discussed in the previous chapter that in power analysis, one common method is Correlation Power Analysis (CPA) [7], which computes the correlation between the power leakage traces and the hypothesis power vector, which is dependent on the key candidates and the power leakage model. We also observed that in many practical systems, such as AES [18], the number of bits in the secret can be very large (128 bits for AES 128). Using side-channel attacks to guess the full key at once is computationally impossible (for AES 128, 2^{128} candidates need to be generated). Hence, the majority of side-channel attacks use a divide-and-conquer (DC) strategy.

Although DC-strategy-based side-channel attacks are effective, one remaining question is if the DC strategy alone is sufficient for analyzing the security of the cryptographic system. In fact, a considerable amount of previous work has been proposed for exploring this question: In [19, 49, 50], Algebraic Side-Channel Attacks (ASCA) were introduced and further analyzed. Totally different from the DC strategy, ASCA has access to the full design of the system and so can attempt to build a fine-granularity power model. For the case of AES, the attacker then can proceed to exploit the samples in *all* rounds of the encryption process to build a system of equations describing the target cryptographic algorithm. The equations are solved by using a SAT solver [51]. As a result, ASCA is able to extract the key of an AES implementation from very few leakage traces (usually just a single trace). However, the computing cost of ASCA is huge, both in terms of computation and memory requirement. For example, the SAT representation for a single trace [52] has approximatively 18000 equations with 10000 variables. In addition, ASCA also displays a very weak resistance to noise. This led to the design of an attack framework called Soft Analytical Side-Channel Attacks (SASCA) which was introduced in [20]. Different from ASCA, which encodes the cryptographic implementation and all the leakages as equations, SACSA describes them as a code, which is eventually modeled as a factor graph [53]. To decode the leakage information,

the Belief Propagation (BP) [54] algorithm is used. Since SASCA can directly exploit the soft information, such as the posterior probabilities obtained during a Template Attack, it reduces the memory requirement, compared to ASCA. More recently, Local Random Probing Model (LRPM) was used to model the SASCA from a coding theory viewpoint in [55]. However, the conclusion in [56] shows that in general SASCA is still significantly more computationally intensive than DC-based side-channel attacks. We observe that, compared to the standard DC-based attacks, the profiling effort in SASCA is more expensive since it requires characterizing the leakages from *all* the intermediate values. In the case of AES 128, this means that in a worst-case scenario, leakage is characterized for all 11 AES encryption rounds. Hence, this leads to the interesting question whether it is possible to find a pragmatic method that can on one hand uses the leakage samples from multiple rounds, but on the other only requires a moderate computing cost.

In this chapter, we propose a method to address the above question. Our method still relies on the DC-based Correlation Power Analysis (CPA) that we described in Chapter 5. However, different from the previous DC-based methods, which use the leakage samples that are emitted from either the first round or the last round, our method leverages the samples that come from more than one round. The attack relies on CPA as follows:

During the attack phase, based on the received ciphertext, we apply CPA on the leakage samples in the *last* round for each byte independently. As a result, we obtain a matrix that records the correlation coefficients for each candidate subkey at each byte position. In typical CPA fashion, the attack ends once the roundkey has been found. The guessed subkey at each byte position is the one subkey that has the maximum correlation value. In our method, we use this same last-round attack to determines *a set of* candidate keys rather than a single candidate key. This is easy to do, since we already recorded the correlation values for all possible subkey guesses for each byte. By applying a simple key enumeration algorithm [57], we can obtain a list of candidates for the last round-key in order of the their overall likelihood as determined by the last-round attack. The attacker can vary the size of the list of candidate keys. For each of the last roundkey candidates, we can determine all round keys by reversing the schedule of the AES key expansion. Based on

this information, we can compute the likelihood between the samples and the hypothesis power values in the previous rounds. Once we get the likelihood values mapped to all the last roundkey candidates, we can find the most-likely last roundkey, which has the largest likelihood value or smallest reconstruction error. In other words, we use the result from previous rounds to validate the selection of keys in the last round thus reinforcing the selection of the key. We call this approach "Reinforcement Strategies" for the side-channel attack.

Compared to previous DC-based CPA approaches, our method improves the attack performance by exploiting the samples from multiple rounds. The experimental results also show that reinforcement leads to significant improvements in key-recovery performance. Different from ASCA and SASCA, we don't need to solve a large number of linear equations in the attack phase, or spend a large effort in profiling, which leads to significant computing cost saving.

In the following, we will briefly introduce how the side-channel attacks target the first and the last rounds of AES. After introducing the key enumeration algorithm, we present our Reinforcement strategy based Side Channel Attack algorithm. Finally, we present and analyze the experimental results for our proposed scheme.

6.2 Side Channel Attacks on AES Implementation

In Fig. 6.1, we display a sample power trace for one AES 128 encryption. The power trace comes from the TeSCASE dataset [2]. We will describe this dataset in more details in the experimental part. In this figure, we represent the power consumption in form of the voltage. We see from Fig. 6.1 how the voltage curve clearly indicates the 11 rounds of the AES encryption. From the previous chapter, we know that the power consumption in the AES implementation is dependent on the changes of the internal staet $\boldsymbol{S}$ during the operations, as shown in Eq. (5.1). Hence, by analyzing the power consumptions among the leakage traces, the internal state cane be guessed, and so can be the secret key.

As we described in Chapter 5, previous work shows that side-channel attacks can be launched on either the first round, where plaintext is supposed to be known, or the last round, where ciphertext is obtained after the AES encryption. In the following, we take the first round and the last

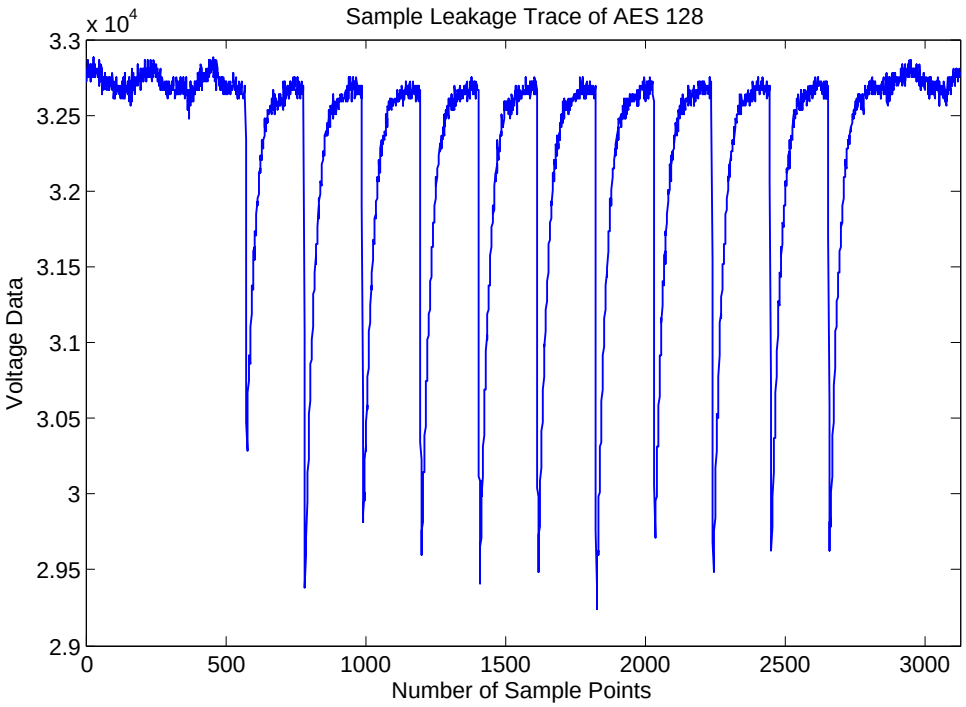

Figure 6.1: The Sample of AES 128 Trace (Derived from the Dataset in [2])

round as the case studies to show how side channel attacks break the AES in these rounds.

6.2.1 Side Channel Attacks on First-Round AES Implementation

In the first AES encryption round, the plaintext is XOR-ed with the secret key. Hence, if the plaintext P is known, guessing the secret key K is equivalent to guessing the intermediate value V. Given the simplicity of the XOR operation, it is tempting to recover the intermediate value V using a side channel attack.

Since AES 128 has 128-bit keys, directly using the Template Attack or the Stochastic Model on all the 16-bytes is computational impossible. Hence, the attack on the intermediate value V must use a DC strategy. If we can assume that the XOR operation is at byte level and linear in terms of power consumption, then we can assume that the power consumption of the XOR operations as applied to different bytes of the key are independent. This makes the first-round analysis an effective attack. In fact, a lot of previous work, such as [20, 25, 55], targets the AES first-round leakage as part of side-channel attacks. However, the success of the attack on the first round relies

on the assumption that *the plaintext is known to the attacker*, which severely limits the applicability of this attack. Hence, analyzing the traces in the last round is more reliable, since one can assume that the attacker has access to the ciphertext.

6.2.2 Side Channel Attacks on Last-Round AES Implementation

As we described in Chapter 5, for the last round of AES 128, the internal state is the changes between the states before and after the 10-th round which is the ciphertext. The DC strategy finds the subkey guess with the maximum correlation, for each byte, by generating 256 hypothesis power vectors and computing the correlation values with the leakage samples in the last round. This process is repeated 16 times, and all the guessed subkeys are combined together to produce the recovered last roundkey. The guessed AES secret key is obtained by reversing the AES key expansion on the guessed last roundkey.

Although the last-round attack is proven to be effective in this work as well as in previous work [45], we investigate whether exploiting more samples that come from other rounds would result in improved recovery performance. This is because samples from other rounds also contain information about the internal state in their round. For the AES algorithm, we know that the internal state in any previous round is also related to the internal state in the last round. Hence, if we can find a way that utilizes the samples from the previous rounds, especially the samples in the first round, the key-recovery performance would improve significantly. Our approach to leverage information from other rounds is to treat the last-round attack as the initial guessing for generating a set of candidates, and then use the other round attack to validate and further narrow the set of original candidates. The rationale for this approach is that the most-likely guess in the last round is not always correct, and that we need the information from other rounds to validate it. Hence, the attack on the other round will help the attacker to validate the selection of the candidate keys and finally pick the most-likely key from among the candidates.

6.3 Candidate Keys Selection

Before introducing the reinforcement-based DC strategy in detail, we describe how to select the candidate keys. Suppose our target is AES 128. In the last round-attack, for each byte, we compute the correlation between the real power vector and the hypothesis power vector, which in turn is defined by the power-leakage model and the subkey candidate. This will lead to 256 correlation values for the 256 subkey candidates. Hence, after computing these correlations for every byte position, the attacker will get a table of the correlation values, whose size is 16×256. To standard DC-based side channel attacks, the attacker independently sorts the correlation values for each byte position (i.e., 0 to 15) in this table and finds the most likely subkey by picking the one with the highest correlation value. Finally, the attacker obtains the most-likely full key by combing all the selected subkeys together.

In the reinforcement-based DC strategy, different from the previous methods, we do not directly guess the last-round key after we get the table of the correlation values. Instead, we want to get a list of the most-likely roundkey candidates, by using an enumeration algorithm, such as the one described in [57].

The objective of the key enumeration algorithm is to generate a sequence of keys in order of decreasing likelihood shifting with the most likely key. The main idea of the particular enumeration algorithm in [57] is the merging of a set of lists of subkeys based on the likelihood values in each list. The algorithm starts by merging two bytes together, based on the rule that for any pair of subkeys, for example $(k_1^{(i)}, k_2^{(j)})$, if the sum of their likelihoods is larger compared to another pair, for example $(k_1^{(l)}, k_2^{(t)})$, then pair $(k_1^{(i)}, k_2^{(j)})$ will be placed in front of the pair $(k_1^{(l)}, k_2^{(t)})$ in the merged list.

For the case of merging multiple lists of subkyes, the authors in [57] applied a recursive decomposition on this problem: every time, they only use the enumeration algorithm to merge two lists, and its outputs are used to form larger subkey lists which are in turn merged together. like the AES 128 case, the algorithm fist merges a pair of bytes together, which generates 8 groups now (each group stands for the merging result of two bytes). Then, The enumeration algorithm

works on the 8 groups again and get a 4 new groups. This process is repeated until only a single group remains. Theoretically, the algorithm will generate a table, which records the candidates of the full key, ordered by the likelihood values in the key enumeration. However, since usually it's unnecessary to get a full table of key candidates due to its huge size (2^{128} for AES 128), the algorithms also support the function that by inputting a number of expected candidates n, it only outputs the first n candidate keys in the final merged list.

In our algorithm, we send the table of correlation values and the number of candidate into the key enumeration algorithm, and we get a new table of the candidates of the most-likely last roundkey, which is ranked by the accumulation of the candidates' correlation values.

6.4 Reinforcement-based DC Strategy in Side Channel Attacks

In this section, we introduce our reinforcement-based DC strategy attack algorithm. At the beginning of the attack, the attacker records N_a leakage traces from different ciphertexts but for the same key K^* on the target device. Since now the attacker wants to exploit the samples from multiple rounds, it creates a space to store the sample points from the multiple rounds, which is represented by $\boldsymbol{X}_{K^*} \in \mathbb{R}^{N_a \times r_a \times m}$ where r_a stands for the number of rounds that the attacker wants to exploit, and m is the number of samples in each trace. For example, if we only use the samples from the first round and the last round, then $r_a = 2$. The samples from the last round are always stored in $\boldsymbol{X}_{K^*}[:][r_a - 1][:]$ and the samples from the previous rounds are stored from $\boldsymbol{X}_{K^*}[:][0][:]$ to $\boldsymbol{X}_{K^*}[:][r_a - 2][:]$ ordered by the round's index. The attacker also records a ciphertext table $\boldsymbol{C}$ who has N_a rows where each row has I byte ciphers.

Similarly to Section 5.5, we choose one sample (now $m = 1$) for each of the chosen AES round in a trace. The selected sample is the one who has the largest power value in the chosen round.

The input of our algorithm includes a ciphertext table $\boldsymbol{C}$, the power traces $\boldsymbol{X}_{K^*}$, the number of the roundkey candidates Num_c, the set of the indices of the previous rounds that the attacker exploits $\boldsymbol{r}$, and the leakage model **LM**, which is as same as in Section 5.5.

The algorithm first initializes a multi-dimensional vector $\boldsymbol{P}_{ts} \in \mathbb{R}^{I \times 2^B \times N_a \times m}$, which is used to store the hypothesis power vectors for each candidate subkey on each byte, a matrix $\boldsymbol{R}_{ts} \in \mathbb{R}^{I \times 2^B}$,

which stores the correlation coefficients between the real power vector and the hypothesis power vector for each candidate subkey under each byte, a table for recording the round key candidate $\boldsymbol{T}_c$, whose size is $I \times 2^B$, and a vector $\boldsymbol{L}_c \in \mathbb{R}^{Num_c \times |\boldsymbol{r}|}$, which is used to store the likelihood value for each roundkey candidate.

The first part of the algorithm decrypts the ciphertext by applying the inverse procedure on the AES last-round operations, as shown in Fig. 5.2, and also introduced in **Algorithm 3**. Here we choose AES 128 as case study, which gives us $I = 16$ and $B = 8$. At the end of the first part, we obtain the matrix $\boldsymbol{P}_{ts}$, which stores the hypothesis power values for 256 possible subkeys under each 16 bytes. Then, for each byte, the attacker goes through all 256 hypothesis power vectors and iteratively computes the correlation with the vector of the real power samples from the AES last round, which is stored in $\boldsymbol{X}_{K^*}[:][r_a - 1][:]$. We uses the *Pearson Correlation Coefficient* [48] to compute the correlation coefficients. After computing the correlation on all 16 bytes, the attacker obtains a table that stores the correlation values for each possible subkey under every byte, which is $\boldsymbol{R}_{ts}$.

The algorithm then goes to its second part. The attacker inputs $\boldsymbol{R}_{ts}$ into the key enumeration algorithm [57], which we described in Section 6.3, with the parameter Num_c which stands for how many last-round key candidates that the attacker want to use. Theoretically Num_c is in the range of $[1, 2^{128}]$. Once we get the output from the key enumeration algorithm, which is $\boldsymbol{T}_c$, we iteratively pick one last-round key candidate with the order in the ranking, and decode the roundkey in the r-th round where $r \in \boldsymbol{r}$, through the reversing of the AES key expansion. Since we already know the ciphertext and also the last-roundkey candidate, based on the r-th roundkey, it is easy to get the pre-state and post-state in the r-th round by the AES decryption. Then, we compute the mean squared error (MSE) between the vector of the real power samples and the hypothesis power vector in the r-th round. We specify the computing of the MSE value for a given round and a given candidate as follows:

We use $\boldsymbol{\theta}(K, C)$ to stand for the internal state of the all 16 bytes in a given round, with the

roundkey K and the ciphertext C, such that

$$\boldsymbol{\theta}(K, C) = \{\text{PreState}_j \oplus \text{PostState}_j\}_{j=0}^{I-1} \ ,$$

where PreState_j and PostState_j are the pre-state and post-state in the j-th byte. Here we want to note that for the first round, the internal state is equivalent to the first-round key itself.

Once we obtained the internal state in r-th round by applying the function $\boldsymbol{\theta}(\textbf{RoundKey}_r, C)$, where $\textbf{RoundKey}_r$ is the r-th roundkey, we compute the hypothesis power vector based on a general leakage model $\overline{\textbf{LM}}$ as follows:

$$\overline{\textbf{LM}}(\boldsymbol{\theta}(\textbf{RoundKey}_r, C)) = \begin{cases} \sum_{j=0}^{I-1} \textbf{LM}(\boldsymbol{\theta}(\textbf{RoundKey}_r, C)[j]), \text{ for Approximated Model } ; \\ \{\textbf{LM}(\boldsymbol{\theta}(\textbf{RoundKey}_r, C)[j])\}_{j=0}^{I-1}, \text{ for other Models } ; \end{cases}$$

where **LM** is always the same model that we trained in the last AES round. For the case of approximated model, we get a hypothesis power value for the full roundkey. For other models, we get a vector of hypothesis values one for each byte. After obtaining the hypothesis power values for all N_a traces, we compute the MSE between the hypothesis power vector $\overline{\boldsymbol{P}_r}$ and the real power samples in the r-th round, which is:

$$\textbf{MSE}(\overline{\boldsymbol{P}_r}, \boldsymbol{X}_{K^*}[:][r_i][:]) = \begin{cases} \left\| \overline{\boldsymbol{P}_r} - \boldsymbol{X}_{K^*}[:][r_i][:] \right\|_{\ell_2} , \text{ for Approximated Model } ; \\ \sum_{j=0}^{I-1} \left\| \overline{\boldsymbol{P}_r}[j] - \boldsymbol{X}_{K^*}[:][r_i][:] \right\|_{\ell_2} , \text{ for other Models } , \end{cases}$$

where r_i is the index of r in vector $\boldsymbol{r}$ with $0 \leq r_i \leq r_a - 2$, $\overline{\boldsymbol{P}_r}$ is a vector with size $N_a \times 1$ for the approximated model and a matrix with the size of $N_a \times 16$ for other models.

Once the MSE values of all $|\boldsymbol{r}|$ previous rounds for every candidate are obtained, we aggregate those values by taking the average and obtain an averaged MSE value for each candidate. After computing the averaged MSE for all Num_c roundkey candidates, we put the results into the vector $\overline{\boldsymbol{L}_c} \in \mathbb{R}^{Num_c} \times 1$ and find the one who has the minimal error. Finally, we pick the most-likely

roundkey, which leads us to the guessed AES key through the reversing of the AES key expansion. The framework for the attack algorithm is described in **Algorithm 4**, shown in the following. In the algorithm, **InvKeyExpansion**(K, r) stands for reversing the last-round key K to the r-th roundkey. If $r = 0$, this means that we need to get the AES secret key, and hence we just use **InvKeyExpansion**(K) to represent this operation. **Mean**$(\cdot)$ stands for taking the average on the vector.

Although our algorithm supports the implementation of multiple previous rounds for attacking, we want to note that here we take the case that only one additional round (the first round) is used except the last round (now $r_a = 2$) as case study in the experimental part. This also means now $|\boldsymbol{r}| = 1$, and $\boldsymbol{L}_c$ becomes a vector and replaces the role of $\overline{\boldsymbol{L}_c}$.

Besides, although here we use **MSE** to compute the error between the vector of the real power samples and the vector of hypothesis power for given previous round for each last-round candidate, and select the most-likely last roundkey from the candidates, which has the minimal error, our algorithm also supports to use the standard CPA to find the most possible last roundkey: after we obtain the internal state for a given previous round for each candidate, we compute the Pearson Correlation Coefficient [48] between the vector of the real power samples and the vector of hypothesis power on each byte and aggregate the correlation values on all bytes (for the case of AES 128, for a given round, we obtain 16 correlation values and take the aggregation), which means now we use the operation **Pearson** to replace the **MSE** in the algorithm, and matrix $\boldsymbol{L}_c$ stores the correlation values in all previous rounds for each candidate. $\overline{\boldsymbol{L}_c}$ stores the aggregated correlation values for each candidate. Finally, we select the most-likely last roundkey, which has the *maximum* correlation value in $\overline{\boldsymbol{L}_c}$.

6.5 Experimental Results

Our experiments target an unmasked implementation of the AES-128 algorithm. We use the same dataset as we used in Section 5.6. We compare the performance of the standard Hamming Weight model, the independent leakage models, the average measurement model, and the approximated long linear model. We use the *successful-recovery rate* as the measure for performance

Algorithm 4 Reinforcement Strategies based Side Channel Attack

Input: $\boldsymbol{C}, \boldsymbol{X}_{K^*}, \mathbf{LM}, Num_c, \boldsymbol{r}, r_a$

Initialize: $\boldsymbol{P}_{ts}, \boldsymbol{R}_{ts}, \boldsymbol{T}_c, \boldsymbol{L}_c$

For $i = 0, \ldots, N_a - 1$
  For $j = 0, \ldots, I - 1$
    PostState $= \boldsymbol{C}[i][j]$;
    For $k = 0, \ldots, 2^B - 1$
      PreStateID $=$ **InvShiftRows**(j);
      PreState $=$ **InvShiftRows**(PreState$, j$);
      PreState $=$ **AddRoundKey**(PreState$, k$);
      PreState $=$ **InvSubBytes**(PreState);
      $\boldsymbol{P}_{ts}[j][k][i] =$ **LM**(PreState $\oplus$ PostState);
    End
  End
End

For $j = 0, \ldots, I - 1$
  For $k = 0, \ldots, 2^B - 1$
    $\boldsymbol{R}_{ts}[j][k] =$ **Pearson**$(\boldsymbol{P}_{ts}[j][k], \boldsymbol{X}_{K^*}[:][r_a - 1][:])$;
  End
End
$\boldsymbol{T}_c =$ **KeyEnumeration**$(\boldsymbol{R}_{ts}, Num_c)$;

For $l = 0, \ldots, Num_c - 1$
  For $t = 0, \ldots, |\boldsymbol{r}| - 1$
    RoundKey$_r =$ **InvKeyExpansion**$(\boldsymbol{T}_c[l], \boldsymbol{r}[t])$;
    For $i = 0, \ldots, N_a - 1$
      $\overline{\boldsymbol{P}_r}[i] = \overline{\mathbf{LM}}(\boldsymbol{\theta}($**RoundKey**$_r))$;
    End
    $\boldsymbol{L}_c[l][t] =$ **MSE**$(\overline{\boldsymbol{P}_r}, \boldsymbol{X}_{K^*}[:][t][:])$;
  End
  $\overline{\boldsymbol{L}_c}[l] =$ **Mean**$(\boldsymbol{L}_c[l][:])$;
End
Computes $l^* := \text{argmin}_{\{l=0\ldots,Num_c\}} \overline{\boldsymbol{L}_c}[l]$;
RoundKey $= \boldsymbol{T}_c[l^*]$;
AESKey=InvKeyExpansion(RoundKey);

Output: AESKey

evaluation, which is also defined in Section 5.6. In the attack phase, we independently perform the attacking 100 times by randomly picking the leakage traces from the *attack* set.

We compare our reinforcement-based DC strategy with the standard CPA based attack descried in Section 5.5, which focuses on the 10-th round attack only. In our experiment, our algorithm only uses the samples in the first AES round to reinforce the result of the last AES round attack. In the following, for the last AES round attack, we use **HW** to represent the Hamming Weight model, **Independent Model** to denote the independent model, **Average Measurement** for the average measurement method, and **Approximated Model** to represent the approximated linear model. We use **Reinforcement** in front of each model to represent the using of the reinforcement-based DC strategy on that model. For all the profile-based methods, in order to make fair comparisons, we use the Least Squares (LS) method to take the inverse on the measurement matrix for computing the coefficients in the leakage model.

In Fig. 6.2, we present the experimental results for different numbers of attack traces, with different models. The number of roundkey candidates Num_c is set to 100 here. In each experiment, the number of profiling traces N_p is fixed to 3000 and the number of attack traces N_a varies from 100 to 5000. The results indicate that by using reinforcement-based DC strategy, each of the models show huge performance improvements compared to the results of only attacking 10-th round. For example, for the Hamming Weight model with $N_a = 2000$, the recovery rate for 10-th round attack is about 15% and for reinforcement-based DC strategy is near 56%. Besides, when $N_a \geq 4000$, all the models that using reinforcement-based DC strategy can achieve almost 100% recovery rate, which is an excellent result.

In Fig. 6.3, we show the successful recovery rate for reinforcement-based DC strategy for different numbers of roundkey candidates Num_c. The number of profiling traces N_p is fixed to 3000, the number of attack traces N_a is fixed to 4000 and 5000 respectively, and the number of roundkey candidates Num_c varies from 1 to 2000 in each figure. We want to note that for the special case where the number of candidates Num_c is one, and there is subsequently only one candidate in the table $\boldsymbol{T}_c$, our reinforcement-based DC strategy degrades to the standard DC

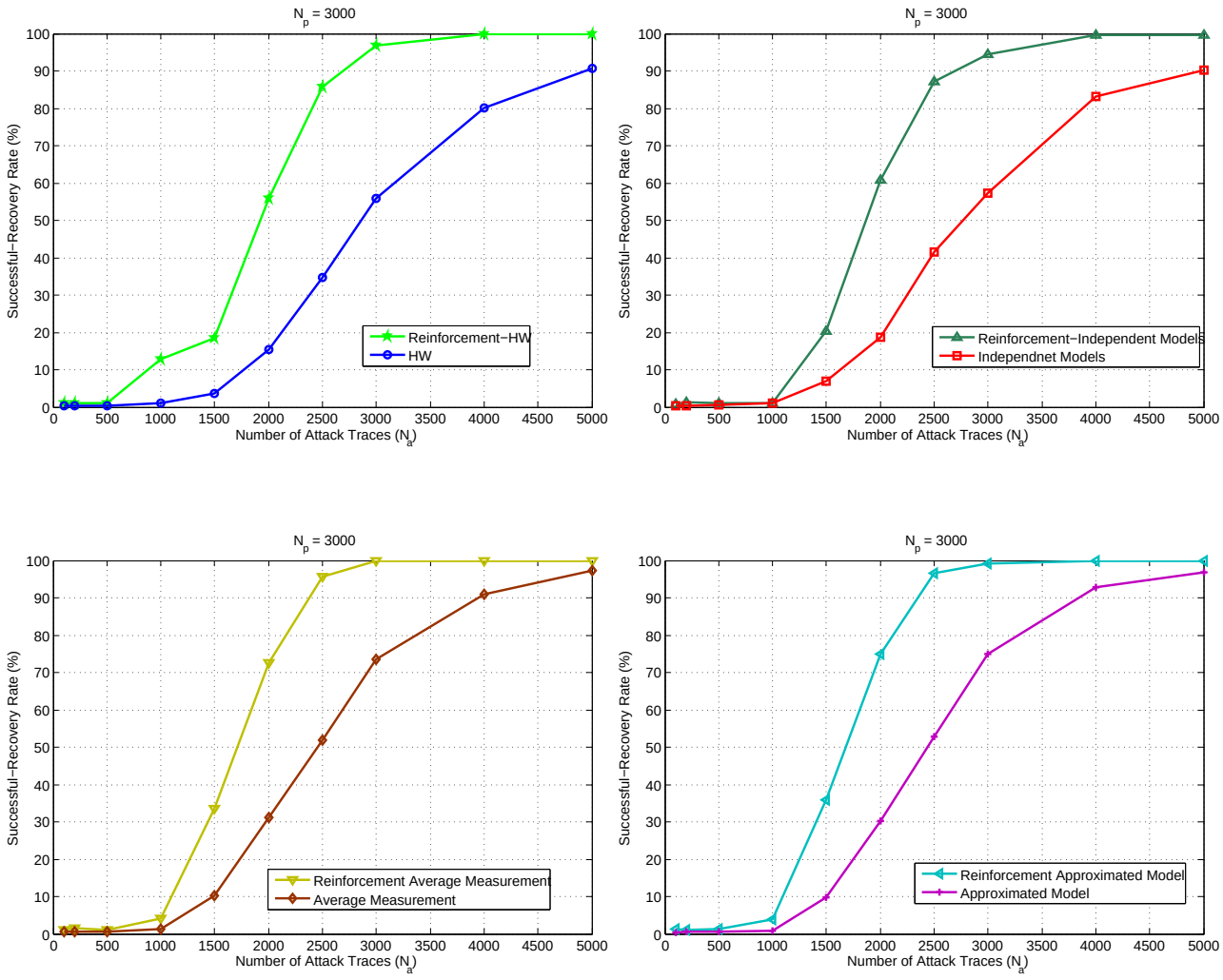

Figure 6.2: Successful Recovery Rate versus Different Attack Traces N_a with Profiling Traces $N_p = 3000$, to Each Leakage Model: Hamming Weight; Independent; Average Measurement; Approximated

strategy based CPA, since now there is only one candidate to select from the validation. She only uses the last-round's information for attacking. From the Fig. 6.3, we can see that when Num_c increases, so will be the recovery rate for all models. This is reasonable since when more candidates are in $\boldsymbol{T}_c$, more validations can be performed and hence more chances that the correct key will be picked. This also tells us that if attacker only gets a limited number of traces for attacking, increasing the number of roundkey candidates is another way to to improve the recovery performance in our Reinforcement strategy. For example, when $N_a = 2000$, the recovery rate for Hamming Weight model with $Num_c = 1$ (DC strategy based CPA) is about 15.8%, and with $Num_c = 2000$ is near 84%, which is a huge improvement.

6.6 Conclusion

In this chapter, we propose a Reinforcement strategies based side channel attacks. By using a key enumeration algorithm, we successfully combine the leakage information from different AES rounds together and hence enhance the power of the attack. Although our method is still based on the idea of Divide-and-Conquer, we exploit the samples from multiple rounds and hence get more information in order to improve the attack performance. Different to the previous methods, such as ASCA and SASCA, which also use the all samples during the entire AES operations, our method doesn't solve a large number of linear equations during the attack phase, or spend large efforts during the profiling phase, which reduces the computing cost. Finally, experimental results are provided to prove the effectiveness of our algorithm.

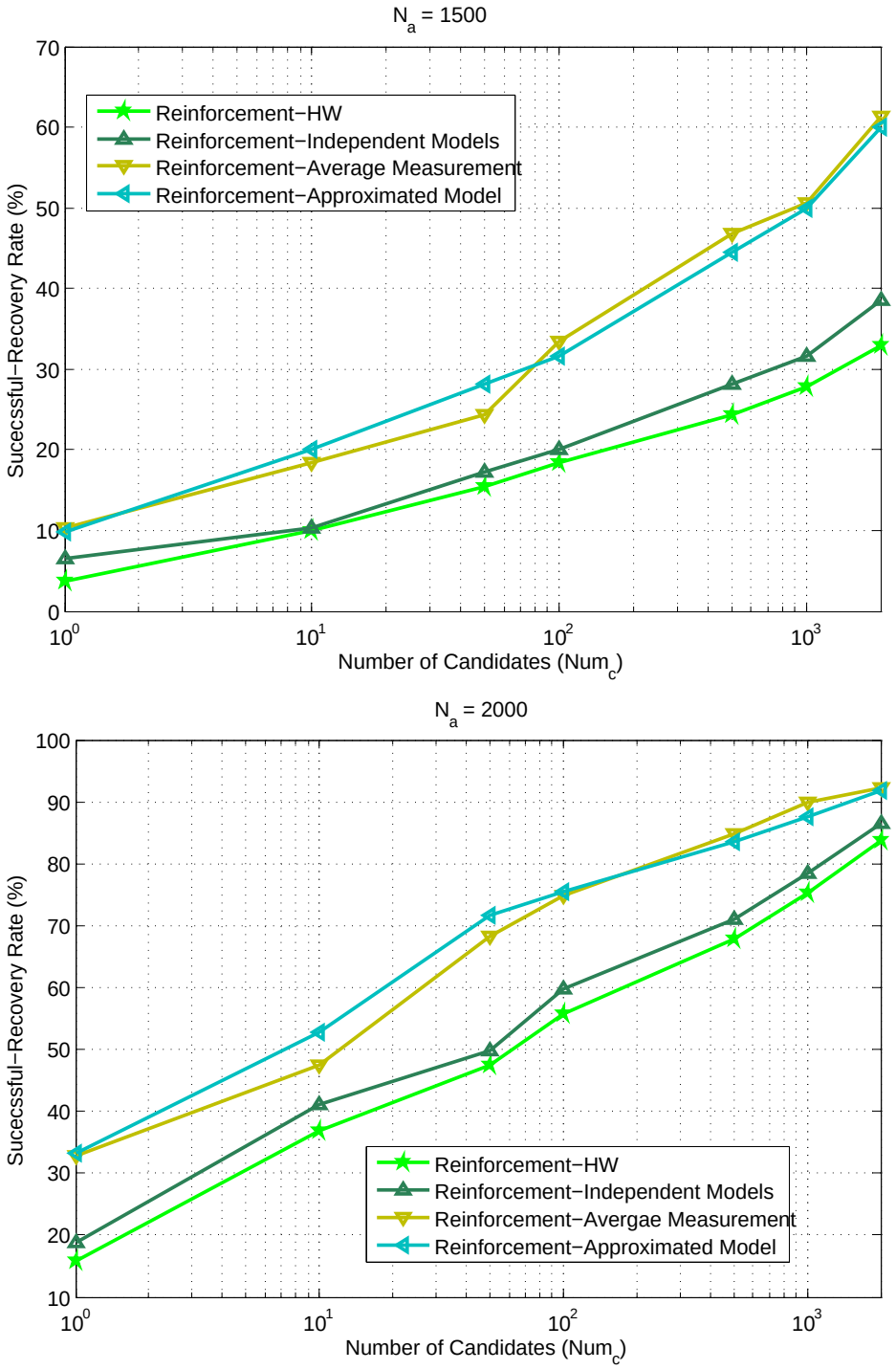

Figure 6.3: Successful Recovery Rate versus Different Number of Roundkey Candidates Num_c with Different Number of Attack Traces $N_a = 1500$, and $N_a = 2000$

7. SUMMARY AND CONCLUSIONS

In a side-channel attack, the adversary exploits physical information that leaks from particular system implementations to extract secret information, such as secret keys, instructions. Since the physical leakages are dependent on the internal state of the cryptographic implementation, profiling this dependency (which is also called leakage model or leakage function) and using it in attack is a powerful technique in side channel attacks.

Power analysis [6] is a widely-used method in side channel attacks, which analyzes the physical leakage information in form of power consumption of the cryptographic device. This power consumption based model relies on the fact that there is relationship between the power consumption among the data-dependent components in the circuits, and the bit flips/state transitions caused by the presumably secret key. Typically, these side-channel attacks can be divided into non-profiled and profiled side-channel attack, respectively. Non-profiled side-channel attacks, use some a priori information about the leakage model, such as *HW* or *HD* model. Profiled side-channel attacks, on the other hand, learn the leakage model by doing the training on an identical or similar target device. There are two widely used techniques in profiled side-channel attacks: the Template Attack and the Stochastic Model.

By extending previous work on leakage model building, in Chapter 3, we generalize the weighted leakage model to describe the mathematical relationship between the measurable physical leakages and the internal state of the cryptographic implementation. We model the side channel as a fading channel in communication theory and treat the leakage model coefficients as the gains of a communication channel. This also allows us to treat the profiling problem in side-channel attack as a *channel estimation problem* in communication. Furthermore, in Chapter 4, we proposed a ℓ_2-norm based re-weighted algorithm to estimate the leakage model in profiling phase, which outperforms the previous LS method and Ridge-based method as illustrated by the experimental results. We also discussed the convergence behavior of our algorithm. We also discussed the advantage of our scheme when only a limited profiling traces is obtained compared to the standard

profiled side channel attack.

In Chapter 5, we focus on the problem of how to efficiently apply the Stochastic Model to the non-linear cryptographic systems as part of profiled side-channel attack. Since usually the cryptographic system has a large number of bits in the secret register, such as AES 128, most of the side-channel attacks use a Divide-and-Conquer strategy to break the cryptographic algorithms. Under this strategy, the attacker only focus on one byte at a time, and treats the other bytes as noise during profiling. This leads to the problem that the model built for each independent byte will be corrupted by the noise comes from both the independent noise, and also the leakage traces generated by other bytes, especially under the case of only small number of profiling traces are given. To address this issue, we propose two methods to build the leakage model for profiled side-channel attacks targeted on AES implementations. One method is based on the idea of averaging the noise from all the bytes, which we call it average measurement model. Another method is based on the idea of using the joint structure for the same bits position, which we call it approximated model. Finally, experimental results are provided to show the effectiveness of our proposed methods.

In Chapter 6, we proposed a reinforcement-based DC strategy for side-channel attacks. By using a key enumeration algorithm, we successfully combine the leakage information from different AES rounds together and hence enhance the power of the side channel attack. Different from the previous methods such as ASCA and SASCA, which also use all leakage samples during the AES encryption, our method doesn't solve a large number of linear equations during the attack phase, or spend large effort during the profiling phase, since our algorithm is also based on the idea of DC strategy, which reduces the computing cost saving compared to those algebraic-based methods. The experimental results are provided to prove the effectiveness of our algorithm.

7.1 Further Study

Our work of exploring the relation between the secret keys and side channel signals leads us to believe that there are inherent structural characteristics of the system that express themselves in how side channel information is leaked. This is also supported by the effectiveness of linear leakage models, such as the Stochastic Model. We plan to further explore their relationships by

using a deep learning framework.

Some previous work have investigated the application of deep learning techniques in the context of side channel attacks. In [58], the authors conduct a comprehensive study of the application of deep learning theory in the context of side channel attacks. In [59], the authors propose a new deep learning architecture called SCANet, and compare it with other machine learning techniques, in side channel attacks. However, in both of these work, the attacker's target is only the HW value of the keys, rather than the real value of keys. This HW information is far from enough to be a threaten to the security of the target devices, especially for the AES based system. Besides, the architectures used in both of these works are complex, which lead to a huge computing cost during profiling. Hence, how to build efficient Convolutional Neural Networks (CNNs) architectures, which can directly guess the binary string of the secret key will be an interesting future work.

REFERENCES

[1] Grizzly, "http://www.cl.cam.ac.uk/research/security/datasets/grizzly/,"

[2] TeSCASE, "https://tescase.coe.neu.edu/?current_page=power_trace_link,"

[3] S. Jin and R. Bettati, "Adaptive Channel Estimation in Side Channel Attacks," in *2018 IEEE International Workshop on Information Forensics and Security (WIFS'18)*, pp. 1–7, IEEE, Dec 2018.

[4] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic Analysis: Concrete Results," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, pp. 251–261, Springer, 2001.

[5] A. Faruque, M. Abdullah, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic Side-Channel Attacks on Additive Manufacturing Systems," in *Proceedings of the 7th International Conference on Cyber-Physical Systems(ICCPS'16)*, pp. 1–10, IEEE, 2016.

[6] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in cryptology (CRYPTO'99)*, pp. 789–789, Springer, 1999.

[7] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*, pp. 16–29, Springer, 2004.

[8] J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert, "Univariate Side Channel Attacks and Leakage Modeling," *Journal of Cryptographic Engineering*, vol. 1, no. 2, pp. 123–144, 2011.

[9] S. Chari, J. R. Rao, and P. Rohatgi, "Template Attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, pp. 13–28, Springer, 2003.

[10] M. O. Choudary and M. G. Kuhn, "Efficient template attacks," in *International Conference on Smart Card Research and Advanced Applications (CARDIS'13)*, pp. 253–270, Springer, 2013.

[11] W. Schindler, K. Lemke, and C. Paar, "A Stochastic Model for Differential Side Channel Cryptanalysis," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'05)*, pp. 30–46, Springer, 2005.

[12] F.-X. Standaert, T. Malkin, and M. Yung, "A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks," in *28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt'09)*, pp. 443–461, Springer, 2009.

[13] A. Heuser, O. Rioul, and S. Guilley, "Good is Not Good Enough: Deriving Optimal Distinguishers from Communication Theory," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'14)*, pp. 55–74, Springer, 2014.

[14] M. K. Simon and M. S. Alouini, *Digital Communication over Fading Channels*. Wiley, 2005.

[15] W. Wang, Y. Yu, F. Standaert, J. Liu, Z. Guo, and D. Gu, "Ridge-Based DPA: Improvement of Differential Power Analysis For Nanoscale Chips," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1301–1316, 2018.

[16] B. Köpf and D. Basin, "An Information-Theoretic Model for Adaptive Side-Channel Attacks," in *Proceedings of the 14th ACM conference on Computer and Communications Security (CCS'07)*, pp. 286–296, ACM, 2007.

[17] F.-X. Standaert, B. Gierlichs, and I. Verbauwhede, "Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks Against Two Unprotected CMOS Devices," in *11th International Conference on Information Security and Cryptology (ICISC'08)*, pp. 253–267, Springer, 2009.

[18] J. Daemen and V. Rijmen, *The Design of Rijndael: AES-the Advanced Encryption Standard*. Springer, 2002.

[19] M. Renauld and F.-X. Standaert, "Algebraic Side-Channel Attacks," in *International Conference on Information Security and Cryptology (INSCRYPT '09)*, pp. 393–410, Springer, 2009.

[20] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert, "Soft Analytical Side-Channel Attack," in *The 20th Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'14)*, pp. 282–296, Springer, 2014.

[21] M. Rivain, "On the Exact Success Rate of Side Channel Analysis in the Gaussian Model," in *The 15th Annual Conference on Selected Areas in Cryptography (SAC'08)*, pp. 165–183, Springer, 2008.

[22] F.-X. Standaert, F. Koeune, and W. Schindler, "How to Compare Profiled Side-Channel Attacks?," in *7th International Conference on Applied Cryptography and Network Security (ACNS'09)*, pp. 485–498, Springer, 2009.

[23] C. Rechberger and E. Oswald, "Practical Template Attacks," in *International Workshop on Information Security Applications (WISA'04)*, pp. 440–456, Springer, 2004.

[24] C. Archambeau, E. Peeters, F.-X. Standaert, and J.-J. Quisquater, "Template Attacks in Principal Subspaces," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'06)*, pp. 1–14, Springer, 2006.

[25] M. O. Choudary and M. G. Kuhn, "Efficient, Portable Template Attacks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 2, pp. 490–501, 2018.

[26] M. O. Choudary and M. G. Kuhn, "Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits," in *International Conference on Smart Card Research and Advanced Applications (CARDIS'14)*, pp. 85–103, Springer, 2014.

[27] D. McCann, E. Oswald, and C. Whitnall, "Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages," in *26th USENIX Security Symposium (USENIX Security'17)*, pp. 199–216, USENIX, 2017.

[28] H. Meyr, M. Moeneclaey, and S. Fechtel, *Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing*. John Wiley & Sons, Inc., 1997.

[29] C. Whitnall, E. Oswald, and F. Standaert, "The Myth of Generic DPA...and the Magic of Learning," in *Topics in Cryptology - CT-RSA 2014*, pp. 183–205, Springer, 2014.

[30] C. Whitnall and E. Oswald, "Profiling DPA: Efficacy and Efficiency Trade-Offs," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'13)*, pp. 37–54, Springer, 2013.

[31] B. Gierlichs, K. Lemke-Rust, and C. Paar, "Templates vs. Stochastic Methods," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'06)*, pp. 15–29, Springer, 2006.

[32] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2008.

[33] F.-X. Standaert and C. Archambeau, "Using Subspace-based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'08)*, pp. 411–425, Springer, 2008.

[34] X. Zhou, C. Whitnall, E. Oswald, D. Sun, and Z. Wang, "A Novel Use of Kernel Discriminant Analysis as a Higher-Order Side-Channel Distinguisher," in *International Conference on Smart Card Research and Advanced Applications (CARDIS'17)*, pp. 70–87, Springer, 2017.

[35] N. Bruneau, S. Guilley, A. Heuser, D. Marion, and O. Rioul, "Less is More - Dimensionality Reduction from a Theoretical Perspective," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'05)*, pp. 22–41, Springer, 2015.

[36] R. Chartrand and W. Yin, "Iteratively Reweighted Algorithms for Compressive Sensing," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'08)*, pp. 3869–3872, IEEE, 2008.

[37] M. S. Asif and J. Romberg, "Fast and Accurate Algorithms for Re-Weighted ℓ_1-Norm Minimization," *IEEE Transactions on Signal Processing*, vol. 61, no. 23, pp. 5905–5916, 2013.

[38] B. D. Rao, K. Engan, S. F. Cotter, J. Palmer, and K. Kreutz-Delgado, "Subset Selection in Noise based on Diversity Measure Minimization," *IEEE Transactions on Signal Processing*, vol. 51, no. 3, pp. 760–770, 2003.

[39] Q. Xie, D. Meng, S. Gu, L. Zhang, W. Zuo, X. Feng, and Z. Xu, "On the Optimal Solution of Weighted Nuclear Norm Minimization," *arXiv preprint arXiv:1405.6012*, 2014.

[40] P. C. Hansen and D. P. O'Leary, "The Use of The L-curve in The Regularization of Discrete Lll-posed Problems," *SIAM Journal on Scientific Computing*, vol. 14, no. 6, pp. 1487–1503, 1993.

[41] M. E. Tipping, "Sparse Bayesian Learning and the Relevance Vector Machine," *The Journal of Machine Learning Research*, vol. 1, pp. 211–244, 2001.

[42] D. Wipf and S. Nagarajan, "Iterative Reweighted ℓ_1 and ℓ_2 Methods for Finding Sparse Solutions," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, no. 2, pp. 317–329, 2010.

[43] ATxmega256A3U, "https://www.microchip.com/wwwproducts/en/atxmega256a3u,"

[44] SASEBO, "http://www.rcis.aist.go.jp/special/sasebo,"

[45] T. Swamy, N. Shah, P. Luo, Y. Fei, and D. Kaeli, "Scalable and Efficient Implementation of Correlation Power Analysis Using Graphics Processing Units (GPUs)," in *The 3rd Workshop on Hardware and Architectural Support for Security and Privacy (HASP '14)*, pp. 1–8, ACM, 2014.

[46] M. A. Elaabid, O. Meynard, S. Guilley, and J.-L. Danger, "Combined Side-Channel Attacks," in *International Workshop on Information Security Applications*, pp. 175–190, Springer, 2010.

[47] M. Ebrahim, S. Khan, and U. Khalid, "Symmetric Algorithm Survey: A Comparative Analysis," *CoRR*, vol. abs/1405.0398, 2014.

[48] P. Sedgwick, "Pearson's Correlation Coefficient," *BMJ*.

[49] Y. Oren, M. Kirschbaum, T. Popp, and A. Wool, "Algebraic Side-Channel Analysis in the Presence of Errors," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'10)*, pp. 428–442, Springer, 2010.

[50] Y. Oren, M. Renauld, F.-X. Standaert, and A. Wool, "Algebraic Side-Channel Attacks Beyond the Hamming Weight Leakage Model," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'12)*, pp. 140–154, Springer, 2012.

[51] G. V. Bard, N. T. Courtois, and C. Jefferson, "Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers," *Cryptology ePrint Archive, Report 2007/024*, 2007.

[52] M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon, "Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'09)*, pp. 97–111, Springer, 2009.

[53] H.-A. Loeliger, "An Introduction to Factor Graphs," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 28–41, 2004.

[54] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding Belief Propagation and Its Generalizations," *Exploring Artificial Intelligence in the New Millennium*, pp. 239–269, 2003.

[55] Q. Guo, V. Grosso, and F.-X. Standaert, "Modeling Soft Analytical Side-Channel Attacks from a Coding Theory Viewpoint," *Cryptology ePrint Archive, Report 2018/498*, 2018.

[56] V. Grosso and F.-X. Standaert, "ASCA, SASCA and DPA with Enumeration: Which One Beats the Other and When?," in *The 21th Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'15)*, pp. 291–312, Springer, 2015.

[57] N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F.-X. Standaert, "An Optimal Key Enumeration Algorithm and its Application to Side-Channel Attacks," in *International Conference on Selected Areas in Cryptography (SAC'12)*, pp. 390–406, Springer, 2012.

[58] E. Prouff, R. Strullu, R. Benadjila, E. Cagli, and C. Dumas, "Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database," *Cryptology ePrint Archive, Report 2018/053*, 2018.

[59] S. Picek, I. P. Samiotis, A. Heuser, J. Kim, S. Bhasin, and A. Legay, “On the Performance of Deep Learning for Side-Channel Analysis,” *Cryptology ePrint Archive, Report 2018/004*, 2018.