Physical attacks on block ciphers

Jap, Dirmanto

2016

Jap, D. (2016). Physical attacks on block ciphers. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/69409>

<https://doi.org/10.32657/10356/69409>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

PHYSICAL ATTACKS ON BLOCK CIPHERS

DIRMANTO JAP

School of Physical and Mathematical Sciences

2016

PHYSICAL ATTACKS ON BLOCK CIPHERS

DIRMANTO JAP

School of Physical and Mathematical Sciences

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2016

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and great support of many people.

First and foremost, I would like to express my thanks to my main supervisor, Prof. Thomas Peyrin, for his inspired guidance and advice, his patience, encouragement and support during my PhD years. I am deeply indebted to him for his understanding and kindness, in both academic and personal aspects.

I would like to sincerely thank my previous supervisor, Dr. Axel Poschmann for his helps in shaping my foundation in this help, and also for setting up the PACE lab which help me a lot during my research.

I would like to give a thank to Dr. Marc Stöttinger and Dr. Shivam Bhasin, for their insightful suggestions, instructive explanations and comments. I am profoundly grateful for their enthusiasm and kindness, and am honored to have worked with them.

I would also like to thank my current PACE lab members for helpful discussions and cooperations: Dr. Chien-Ning Chen, Dr. Jakub Breier, and Dr. He Wei, as well as previous members of the lab, Sebastian Kutzner, Nisha Jacob, Sarote Saetang, Nguyen Phuong Ha, and Alexander Hermann. I also want to thank all the members of cryptographic group in SPMS, for their wonderful expertise and helps, and also my current and previous colleagues in SPMS, for the wonderful time together.

Last but not least, I want to gratefully thank my parents and my sisters, who provide moral support and guidance during my study, and their encouragement.

LIST OF WORKS

Below is the list of works that have been published during my PhD studies, in chronological order.

- [1] Dirmanto Jap, Jakub Breier: Overview of Machine Learning based Side-Channel Analysis Methods. *Integrated Circuits (ISIC), 2014 14th International Symposium on*, Singapore, pp. 38-41.
- [2] Jakub Breier, Dirmanto Jap: A Survey of State-of-the-Art Fault Attacks. *Integrated Circuits (ISIC), 2014 14th International Symposium on*, Singapore, pp. 152-155.
- [3] Jakub Breier, Dirmanto Jap, Chien-Ning. Chen: Laser Profiling for the Back-Side Fault Attacks: With a Practical Laser Skip Instruction Attack on AES. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security (CPSS '15)*. ACM, Singapore, 99-103.
- [4] Dirmanto Jap, Marc Stöttinger, Shivam Bhasin: Support Vector Regression: Exploiting Machine Learning Techniques for Leakage Modeling. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy (HASP '15)*. ACM, New York, NY, USA, , Article 2 , 8 pages.
- [5] Wei He, Dirmanto Jap: Dual Rail Active Protection System Against Side-Channel Analysis in FPGAs. *Application-specific Systems, Architectures and*

-
- Processors (ASAP)*, 2015 IEEE 26th International Conference on, Toronto, ON, 64-65.
- [6] Dirmanto Jap, Jakub Breier: Differential Fault Attack on LEA. *Information and Communication Technology - Third IFIP TC 5/8 International Conference, ICT-EurAsia 2015, and 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, Held as Part of WCC, 2015 LNCS*, Daejeon, Korea, 265-274.
- [7] Jakub Breier, Dirmanto Jap: Testing Feasibility of Back-Side Laser Fault Injection on Microcontroller. In *Proceedings of the WESS'15: Workshop on Embedded Systems Security (WESS'15)*. ACM, Amsterdam, The Netherlands, 5 pages.
- [8] Wei He, Dirmanto Jap, Alexander Hermann: Process Variation Evaluation using RO PUF for Enhancing SCA-Resistant Dual-Rail Implementation. *Information Security Applications - 16th International Workshop, WISA 2015*, Jeju Island, Korea, 16-27.
- [9] Jakub Breier, Dirmanto Jap, Shivam Bhasin: The Other Side of The Coin: Analyzing Software Encoding Schemes Against Fault Injection Attacks. *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016*, McLean, VA, USA, 209-216.
- [10] Dirmanto Jap, Wei He, Shivam Bhasin: Supervised and Unsupervised Machine Learning for Side-Channel Trojan Detection. *IEEE ASAP 2016*: to appear.
- [11] Wei He, Jakub Breier, Shivam Bhasin, Dirmanto Jap, Hock Guan Ong, Chee Lip Gan: Comprehensive Laser Sensitivity Profiling and Data Register Bit-Flips for Cryptographic Fault Attacks in 65 nm FPGA. *LNCS SPACE 2016*: to appear.

Contents

Part I	Introduction	1
1.	Introduction	2
1.1	Motivation	2
1.2	Contributions of this Thesis	4
1.3	Organization of this Thesis	5
2.	Background and Preliminaries	7
2.1	Mathematical Notations	7
2.2	Preliminaries on Cryptography	8
2.3	Preliminaries on Physical Attacks	17
Part II	Machine Learning for Side-Channel Attacks	19
3.	Basics of Side-Channel Attacks	20
3.1	Non-profiling attacks	23
3.2	Profiling attacks	28
3.2.1	Template Attacks	31
3.2.2	Stochastic Approach	33
	High Dimensional Subspace	36
3.3	Side-Channel Evaluation Metric	37

4. Basics of Machine Learning	38
4.1 Introduction to Machine Learning	38
4.2 Examples of Machine Learning Algorithms	40
4.2.1 Support Vector Machine	40
Support Vector Regression	44
One-class Support Vector Machine	48
4.2.2 Random Forest	50
4.2.3 Neural Networks	51
4.2.4 k-Nearest Neighbors	55
4.3 Application of Machine Learning for Profiling based Attacks	56
5. Machine Learning for Profiling SCA	62
5.1 Comparison of Machine Learning Methods	63
5.1.1 Experiments	65
5.1.2 Discussion	76
5.2 Leakage Modeling	79
5.2.1 Experiments	80
5.2.2 Discussion	90
6. Basics of Hardware Trojans	92
6.1 Introduction to Hardware Trojans	93
6.2 Trojan Detection Methods	95
6.3 Open Problems in Trojan Detection	102
7. Machine Learning for Side-Channel based Trojan Detections	105
7.1 Proposed Method based on One-class SVM	106
7.2 Experiments	107

7.2.1	Evaluation on Controlled Settings (Simulation)	109
7.2.2	Evaluation on Real Device (FPGA)	115
7.3	Discussion	127
 Part III Theoretical and Practical Aspects of Fault Attacks		129
8.	Basics of Fault Attacks	130
8.1	Fault Injection Methods	131
8.2	Fault Models	137
8.3	Types of Fault Attacks [35]	139
9.	Investigation on Theoretical and Practical Fault Attacks	142
9.1	DFA on LEA Block Cipher	143
9.1.1	Introduction to DFA	143
9.1.2	Introduction to LEA	145
9.1.3	Attack Methodology	147
9.1.4	Simulation Results	151
9.1.5	Observations	153
9.1.6	Discussion	155
9.2	Practical Fault Attacks using Laser Injection	155
9.2.1	Laser Station Setup	156
9.2.2	Profiling of the Laser Station	157
9.2.3	Practical Attack on AES	160
9.2.4	Discussion	163

Part IV Conclusion	166
Bibliography	172

List of Figures

2.1	The AES encryption (left) and decryption (right) process	12
2.2	AES SubBytes layer	13
2.3	AES ShiftRows layer	13
2.4	AES MixColumns layer	14
2.5	AES AddRoundKey layer	15
2.6	AES Key Schedule algorithm	16
2.7	Classification of Physical Attacks (main focus on active semi-invasive and passive non-invasive)	18
3.1	Illustration of SPA	25
3.2	Example of DPA on AES first S-box output	27
4.1	Illustration of different learning concept: supervised (left), unsuper- vised (right), semi-supervised (bottom)	40
4.2	Illustration of SVM	41
4.3	Non-linear classification using SVM	42
4.4	Illustration of SVR (the area between the dash line defines $\bar{L} \pm \varepsilon$) . .	46
4.5	SVR estimation on non linear function	47
4.6	Illustration of One-class SVM	48
4.7	Illustration of Random Forest	51
4.8	Illustration of Neural Networks	52

4.9	Illustration of Backpropagation Feed Forward method	53
4.10	Illustration of k-Nearest Neighbors	55
4.11	Relations between SCA and Machine Learning	57
5.1	Power trace of one AES encryption	66
5.2	Comparison results using different feature selection methods	67
5.3	Comparison results using different number of training data	69
5.4	Performance results with different parameters	72
5.5	Comparison results of different learning algorithms - noise 30dB	73
5.6	Performance results with different parameters on 20dB noise	74
5.7	Comparison results of different learning algorithms - noise 20dB	74
5.8	Intercalated model	76
5.9	Comparison results with intercalated leakage model	77
5.10	Evaluating quality of leakage model with CPA	81
5.11	Leakage detection and points selection	83
5.12	Guessing entropy of various training methods (5 features)	83
5.13	Guessing entropy of different noise level	86
5.14	Comparison of different subspaces	89
5.15	Guessing entropy on simulated higher order data	90
6.1	Effect of trojan on side-channel leakage	100
6.2	Example of hardware trojan that induces fault on AES [24]	103
7.1	Example of classification using one-class SVM	108
7.2	Illustration of decision boundary with linear kernel	108
7.3	Illustration of decision boundary with RBF kernel	109
7.4	Performance of one-class SVM on non-Gaussian data	110

7.5	RBF kernel on simulated univariate HT data	112
7.6	RBF kernel on simulated multivariate HT data	113
7.7	Trojan inserted into the cipher using partial reconfiguration to ensure the cipher is identical under different trojan designs	116
7.8	EM measurement on decoupling capacitors of SAKURA-G	117
7.9	Placement and routing of circuits on FPGA for AES-128.	118
7.10	EM trace (genuine and trojan infected) with zoomed version	119
7.11	The accuracy results of validation and testing phase using all sample points	122
7.12	Result on 128-bit trojan - supervised scenario	122
7.13	Result on 8-bit trojan - supervised scenario	124
7.14	Result on unsupervised scenario	125
8.1	Absorption depth for particular wavelengths in Silicon [71].	135
8.2	Fixed laser spot on different technology size [41]	136
8.3	Illustration of Fault Attacks concepts	140
9.1	Illustration of DFA on AES	144
9.2	i -th round function of LEA	147
9.3	Fault propagation in rounds 23 and 24 of LEA	148
9.4	Retrieved key bits from single bit flip in modular addition	154
9.5	Device under test during the experiment.	156
9.6	Backside de-packaging	157
9.7	Region for skip instruction	159
9.8	Timing of instruction skip for different bytes	160
9.9	Illustration of skip instruction in the last round of AES	161
9.10	Number of key bytes retrieved using different laser strength	161

9.11 Regions where faults occurred	162
9.12 Infective Countermeasure for Block Cipher [164]	164

List of Tables

2.1	State representation of AES	11
4.1	Commonly used kernels for SVM [51]	43
7.1	Accuracy results on 128-bit trojan (supervised) with single data/plaintext and majority voting	123
7.2	Accuracy results on 8-bit trojan (supervised) with single data/plaintext and majority voting	124
7.3	Accuracy results on unsupervised with single data/plaintext and ma- jority voting	126
9.1	Changes of values in modular addition after a bit flip.	150

Glossary

AES:	Advanced Encryption Standard
CPA:	Correlation Power Analysis
DES:	Data Encryption Standard
DFA:	Differential Fault Analysis
DPA:	Differential Power Analysis
EM:	Electromagnetic Emanation
FPGA:	Field Programmable Gate Array
GE:	Guessing Entropy
HD:	Hamming Distance
HT:	Hardware Trojan
HW:	Hamming Weight
kNN:	k-Nearest Neighbors
ML:	Machine Learning
NN:	Neural Networks
RBF:	Radial Basis Function
RF:	Random Forest
SA:	Stochastic Approach
SCA:	Side-Channel Attacks
SNR:	Signal-to-Noise Ratio
SVM:	Support Vector Machine
SVR:	Support Vector Regression
TA:	Template Attack

ABSTRACT

The security of a cryptosystem is often compromised, not from a theoretical point of view, but by the leakage caused by the physical implementation of the cryptographic algorithm. A new class of attacks, called physical attacks, has shown the capability to exploit the unintentional physical behaviors from the cryptographic device, which usually provide enough information to recover the secret keys. Different methods have been proposed for conducting the attacks. Two of the main focus of physical attacks are side-channel attacks and fault attacks.

For side-channel attacks, the strongest cryptanalysis can be carried when the attacker can profile the targeted device. In profiling based side-channel attacks, a model is constructed to characterize the leakage behavior from the device. Recently, machine learning algorithms have been proposed as alternatives for the classical profiling based attacks. Machine learning and side-channel analysis are two different fields of study, however they are similar, in a sense that both are mostly dealing with the same problem (i.e., classification).

For fault attacks, the aim of the attacker is to disrupt the execution of cryptographic algorithms. Based on the erroneous results, it is possible to gain some additional information regarding the secret key. Many methods can be used to force a fault to the device, however, laser fault injection is still considered as the preferred tools for injecting faults, due to its high precision and repeatability.

In this thesis, various aspects of the physical attacks are covered, with an em-

phasis on block cipher implementation. Specifically, this thesis focuses on several topics:

- **Using machine learning as a tool for profiling based side-channel attacks.** We first investigate different machine learning methods as alternative methods for classical profiling based attacks. Then, in the next work, machine learning is utilized for the construction of an accurate leakage model. Lastly, machine learning is used in combination with profiling side-channel attacks for detection of hardware trojan (malicious modification in the device), which can also be extended to the scenario where there is no golden reference.
- **Investigating theoretical fault attacks and its practicability.** We first provide a theoretical fault attacks on LEA block cipher. Then, investigations and experiments are done to highlight the practicability of fault attacks, focusing on laser fault injections on smart cards.

Part I

INTRODUCTION

1. INTRODUCTION

1.1 *Motivation*

Cryptography is an important field, on which the security of information systems highly depends on. It covers a broad area of aspects in information security, such as message authentication, non-repudiation, *etc.* However, in the earlier times, cryptography was only associated with the encryption and decryption of data. These concepts (encryption and decryption) are the basic foundations to ensure a secure communication among different parties. The message (plaintext) can be hidden by encrypting it into unreadable message (ciphertext) with a secret key (encryption key). It can only be decrypted back to the original message with the corresponding secret key (decryption key). This ensures that even if an attacker or adversary can intercept an insecure communication channel, no confidential information is leaked.

In real world applications, the confidential data are often protected by utilizing strong cryptographic algorithms. However, it was found out that the physical processes of the implementations can potentially leak some information about the data. Furthermore, given the access to the physical device, the attacker can use any method to force read the data inside the device. This allows the retrieval of the secret information, such as the secret key, without the need to find any weakness in the cryptographic algorithms. Hence, this new type of attacks, called physical attacks or physical cryptanalysis, is gaining more attention, due to its effectiveness

and practicability.

Some successful physical attacks have been reported, which exploit the physical or side-channel leakages of the devices (commonly referred as side-channel attacks). To give some examples, attacks on commercially available products that implement the KeeLoq block cipher have been provided in [5, 63], attacks on security applications employing passive RFID tag have been shown in [84, 100], attacks on MIFARE Classic contactless smartcards were shown on [56, 129], and attacks for Wireless Java-Based PDA were shown in [67]. Practical attacks have also been conducted on OpenSSL, a popular cryptography library (see [3]). Hence, physical attacks are considered as serious threats for the security of many of cryptographic embedded devices.

Among different types of side-channel attacks, the profiling based approaches have been considered as the strongest possible methods that can be mounted. By using a device similar to the targeted ones, the attacker can learn or characterize its behavior and build a mathematical model, which can in turn be used to conduct the attacks on the real target device. When attacking the real target, the effectiveness of the attacks is mainly dependent on how accurate the model is. Hence, the task of the attacker is mainly reduced to building an accurate model.

Side-channel attacks (or analysis) can also be used as a tool for testing or for verifying of the device during the design and manufacturing process. One of the rising threats for cryptographic devices during this process comes from the hardware trojans. This covers any malicious alteration done by the attacker inside the device itself and it can be set to perform any malicious action. In the context of information security, these trojans can be used to enhance the physical attacks, either by increasing the side-channel leakage or as alternative tools for injecting faults or errors. Alternatively, it can simply be used to reveal some secret information

through some covert channel. The threat of trojans can be mainly attributed to their unknown nature, mostly when the modification is hidden under the complex circuitry inside the device. One of the proposed detection method is to utilize the side-channel leakage to detect if there is any changes which might be attributed to the malicious alterations, that affects the physical behavior of the device.

In side-channel attacks, the attacker is more restricted, in the sense that only the physical leakage can be observed in order to avoid leaving any physical trail. Another category of physical attacks, called fault attacks, allows the attacker to deliberately force a disruption on the working device. By observing the errors or faults produced by device, the attacker can perform an analysis to recover some crucial information. Hence, in some sense, fault attacks allow more freedom for the attacker compared to side-channel attacks. In fault attacks, the main issue of the adversary is to produce informative faults efficiently and how to exploit these faults to retrieve meaningful information.

1.2 Contributions of this Thesis

The main contributions of this thesis are as follow:

- In the area of side-channel attacks, we have performed a comparison on different machine learning algorithms with classical profiling based attacks under standardized metric. Machine learning has recently emerged as an alternative tool for profiling attacks and has been applied for side-channel attacks. In our experiments, we have investigated different cases and highlighted the scenario in which machine learning can be a better alternative than the classical methods. We also proposed a new method for leakage modeling based on Support Vector Regression (SVR). By utilizing the method in SVR, we show that it can

be used to generalize the leakage, and hence, better approximate the leakage behavior.

- In the area of hardware trojans, we have proposed a detection method based on single class classification in machine learning, exploiting the side-channel leakage of the device. We used the one-class Support Vector Machine (SVM) method to generalize the leakage behavior of the genuine device. We then investigated the impact of different learning parameters in one-class SVM on simulation. Then, we performed a comparison with a recently proposed detection method. We also showed that the proposed method can be extended to the case where the genuine device is not available.
- In the area of fault attacks, we investigated the block cipher LEA and proposed a new attack, utilizing the fault propagation in the modular addition operation in different rounds of the encryption process. We also proposed a simple and practical fault attack on microcontroller. First, we conducted the profiling of the laser fault injection station. Then, by observing the nature of the fault inflicted on the microcontroller, we could design an attack, which requires as low as a single fault in order to retrieve the whole key.

1.3 Organization of this Thesis

The thesis will be organized as follows:

- In Part I, Chapter 2, we first describe the mathematical notations that will be used in this thesis. Then, we provide some background on cryptography and physical attacks.
- In Part II, we describe the results related to side-channel attacks.

-
- In Chapter 3, we recall the concept of side-channel attacks, specifically on profiling based attacks, and in Chapter 4, we explain the basics of machine learning and its applications in side-channel attacks. In Chapter 5, we present our results, the comparisons of machine learning based approaches with classical profiling based attacks, and the application of a machine learning based approach for leakage modeling.
 - In Chapter 6, we give an overview about the basic of hardware trojans with some detection methods based on side-channel. We then describe the Template based trojan detection method. In Chapter 7, we present the results of our investigation regarding the application of machine learning approach for hardware trojan detection and also a comparison with previously mentioned Template based method.
 - In Part III, we describe the topics related to fault attacks. In Chapter 8, we first present the basic concepts of fault attacks. In Chapter 9, we describe LEA block cipher and then present our theoretical attack. Next, we show our profiling of the laser fault station, in order to highlight practical fault models achieved, followed by a simple practical attack.
 - In Part IV, we conclude the thesis and discuss some possible problems and directions for future works.

2. BACKGROUND AND PRELIMINARIES

2.1 Mathematical Notations

In this section, we will define the common notations that will be used in this thesis. Unless specified otherwise, these notations will be used as described.

We start by introducing the basic notation for vectors and matrices. The vectors will be denoted by letters with pointing arrow above (e.g., $\vec{x}, \vec{Y}$), and matrices will be denoted by bold upper-case letters (e.g., $\mathbf{A}, \mathbf{B}$). The normal version of the letters (e.g., x, y) will be used to denote single element variables.

The concatenation of two vectors $\vec{x} \in \mathbb{R}^m, \vec{y} \in \mathbb{R}^n$ will be denoted $[\vec{x} \ \vec{y}] \in \mathbb{R}^{m+n}$. Hence, $\mathbf{A} = [\vec{A}_1 \dots \vec{A}_n]$ can be seen as the concatenation of vectors to form a matrix. Similarly, the concatenation of two matrices $\mathbf{A} \in \mathbb{R}^{k \times m}, \mathbf{B} \in \mathbb{R}^{k \times n}$ will be denoted $[\mathbf{A} \ \mathbf{B}] \in \mathbb{R}^{k \times (m+n)}$. We also let $\vec{0}^m$ denote the zero-vector in dimension m , and use $\vec{1}^m$ to denote the m -dimensional vector where all entries are set to one.

We use the following notation $\{x_i\}_{i=1}^n$ to denote the (ordered) set $\{x_1, \dots, x_n\}$. We use $\cup$ and $\cap$ to denote union and intersection set respectively. Next, $|x|$ is used to denote the absolute value of x . For logical operator, we use $\oplus$ to denote the exclusive-OR operation. We use $\boxplus$ to denote the modular addition operation.

The ℓ_p norm (for $1 \leq p < \infty$) of a vector $\vec{x} = (x_1, \dots, x_m) \in \mathbb{R}^m$, given by $(|x_1|^p + \dots + |x_m|^p)^{1/p}$, is denoted by $\|\vec{x}\|_p$. The infinity norm ℓ_∞ of $\mathbf{x}$ is defined as $\|\vec{x}\|_\infty := \max\{|x_1|, \dots, |x_m|\}$. For simplicity, we denote the ℓ_2 norm (i.e., the

Euclidean norm) by $\|\cdot\|$. We also let $\langle \vec{a}, \vec{b} \rangle = a_1b_1 + \dots + a_nb_n$ denote the inner product of two vectors.

We use $p(x)$ (can be used interchangeably with $P(X = x)$ or $P_X(x)$ with random variable X) to denote the probability density function around point x under some defined probability distribution. Then, $p(y|x)$ or $p(y|X = x)$ denotes conditional probability of y , given the information of $X = x$.

The notation ∇ is used to denote the gradient of the function f with respect to each component $\vec{x}$. It can be written as $\nabla f(\vec{x}) = \left[\frac{\partial f}{\partial x_0}, \dots, \frac{\partial f}{\partial x_n} \right]$. Each component of the vector $\nabla f(\vec{x})$ is a partial derivative of f with respect to each x_i ($i \in \{0, \dots, n\}$).

2.2 Preliminaries on Cryptography

In its very basic form, cryptography is the study of protecting the secret information from a malicious third party. It can be done by transforming the information into unreadable form. In cryptography, a cipher can be defined as an algorithm used for encryption (encode the information or plaintext into gibberish unreadable form or ciphertext) or decryption (decode the information or plaintext back from the ciphertext).

The history of cryptography can be dated back to thousands of years ago. The earliest use of cryptography was reported to be around 1900 BC in Egypt, in the form of hieroglyphs, used by the Egyptians in a non standard fashion. It is believed that this was not used for protecting the information, however, it incorporated the transformation or encoding of information. In around 400 BC, there was reported use of cryptography in Greece, where they used a wooden stick or *scytale* to wrap the tape and write the message on the wound tape. Without the *scytale*, the message would appear meaningless, and hence, to understand the message, the receiver must

also have a *scytale* of the same size. This is a form of transposition cipher, where the order of the characters in the plaintext is permuted or rearranged to obtain the ciphertext. In around 100 BC, Julius Caesar used a substitution cipher, commonly referred as Caesar cipher. In a substitution cipher, each character in the plaintext is substituted with another character to form the ciphertext. In Caesar cipher, the substitution was obtained by a 3-character shift, for example, A becomes D, *etc.*

Most of ciphers used in classical time were based on a basic substitution and transposition cipher. However, these kind of ciphers could be easily broken using simple statistical methods, such as frequency analysis. Around the 16th century, Vigenere proposed a cipher, which can be described as multiple Caesar ciphers with different shift values. A keyword is chosen to determine the value of the shift for different characters. The keyword will be repeated until it matches the length of plaintext. It highlights the use of a key for encryption rather than depending on the secrecy of the cipher. This becomes a major principle, known as Kerckhoffs's principle [92], that the security of an encryption system should only depend on the secrecy of the secret key, not on the secrecy of the system itself.

Nowadays, more advanced cryptographic algorithms have been designed. Many modern cryptographic algorithms rely on theoretical hardness in solving mathematical problems. The aim is to make the computational cost of breaking the system high enough so that it exceeds the profit that could be gained by an attacker, and thus discourage the attacker from doing so.

In the modern cryptography setting, there are two major paradigms, the symmetric key (private key) and asymmetric key (public key) cryptography. In symmetric key cryptography, the secret key used for encryption and decryption processes are the same, while in asymmetric key cryptography, different keys are used: the key for encryption is made public for anyone to send encrypted messages, but the key

to decrypt the message is kept secret. Of course, the decryption key must not be easily obtained from the public key. Most of the algorithms used in asymmetric key cryptography are relying on mathematical problems that have no efficient solution. Examples of asymmetric key cryptography algorithms are RSA (Rivest - Shamir - Adleman) [146] and ECC (Elliptic Curve Cryptography) [27], which use hardness of solving integer factorization and discrete logarithm problem respectively as a way to provide security.

On the other hand, for symmetric key cryptography, it can be classified into block and stream ciphers. In stream ciphers, the plaintext is encrypted using a pseudorandom bit stream acting as the key (or commonly referred as key stream). Usually the key stream is generated from a master key and simply exclusive-OR-ed with the plaintext. The most popular example of stream cipher is RC4 (Rivest Cipher 4) [170], though it has been shown to have multiple vulnerabilities. On the other hand, block cipher encryption means that a block of the message with a specific fixed length will be mapped to a block of ciphertext with the same length. Usually, the length used for the block is either 64 / 128 / 256 bits. The most commonly used block ciphers are DES (Data Encryption Standard) [125] and AES (Advance Encryption Standard) [54], though the former is considered insecure (block size and key size are too small for current standard) and can be brute forced [103]. The construction of block ciphers itself can be classified into Feistel structure, *i.e.*, DES, or SPN (Substitution Permutation Network), *i.e.*, AES.

The analysis of the theoretical weaknesses of the cryptographic algorithms is called cryptanalysis. Given the ciphertexts, the main goal of the attacker is to determine some hidden relation towards the plaintexts or even better to recover the key. If some relations or weaknesses are found in the cryptographic primitive, the cipher should be deemed as insecure and not to be used. The common attack tools

for modern block ciphers are linear and differential cryptanalysis [79]. Linear cryptanalysis tries to "linearize" the non-linear operations in the cipher while differential cryptanalysis tries to find a relation between the differences in the input and the differences in the output. However, even when a weakness of the algorithm is found, it might not be feasible to carry a practical attack [14], for example, due to a high computational complexity.

Case Study: Advance Encryption Standard (AES)

Here, we will describe the AES block cipher which will be the main target for our attacks. AES (or previously known as Rijndael) [54] was chosen as target because it is the current NIST (American National Institute of Science and Technology) block cipher standard, selected in 2001, as replacement of DES and also because it is the most commonly used and widely studied cipher. The construction of the AES cipher is based on SPN construction. AES has a fixed 128-bit block size, and the choices for the key size are 128, 192 and 256 bits. The internal state of the cipher can be seen as a 4x4 matrix of bytes, as shown in Table 2.1. Each byte is represented as $a_{i,j}$, the i -th row and j -th column element in the matrix.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Tab. 2.1: State representation of AES

The total number of rounds executed in the cipher depends on the key size:

- 10 rounds for 128-bit key,
- 12 rounds for 192-bit key, and

- 14 rounds for 256-bit key.

The encryption and decryption process are shown in Figure 2.1.

Fig. 2.1: The AES encryption (left) and decryption (right) process

One round of AES is composed of four operations:

- **SubBytes** or S-box

In the **SubBytes** operation, the byte $a_{i,j}$ will be substituted by $S(a_{i,j})$, illustrated in Figure 2.2. Here, S is an 8-bit to 8-bit non-linear mapping, commonly referred to as S-box. It can be implemented in different way, but the easiest is by using a look-up table (LUT). This operation is the only source of non-linearity in the cipher.

- **ShiftRows**

In **ShiftRows**, each row of the state matrix is shifted by some offset (see Figure 2.3): except the first row, the byte of each row will be shifted to left with offset 1, 2, 3 byte for row 2, 3, 4 respectively. This layer is to ensure that there is

Fig. 2.2: AES SubBytes layer

a mixture of states from different columns for the next operation. Otherwise, AES could be considered as a block cipher running on 4 independent 32-bit block ciphers.

Fig. 2.3: AES ShiftRows layer

- MixColumns

In the MixColumns layer, the four bytes of each column in the matrix will be

mixed together using a linear transformation. This operation is illustrated in Figure 2.4. More precisely, it is done thanks to a matrix multiplication with

$$\text{matrix } C_M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}.$$

The entries of the matrix are elements in $\text{GF}(2^8)$, where multiplication is done modulo irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. Each column element is then treated as a polynomial over $\text{GF}(2^8)$ which is then multiplied with $03.x^3 + 01.x^2 + 01.x + 02$ modulo polynomial $x^4 + 1$.

Fig. 2.4: AES MixColumns layer

- AddRoundKey

For each round, the 128-bit round keys are obtained from the master key through the key scheduling algorithm. In **AddRoundKey**, as shown in Figure 2.5, the 128-bit round keys obtained are combined with the round states, using

Fig. 2.5: AES AddRoundKey layer

bitwise exclusive-OR operation. The key schedule algorithm is as follow:

- Rotate. It takes as input a 32-bit word, and performs a 8-bit left rotation.
- Rcon. It is computed as $Rcon(i) = x^{i-1}$ in polynomial form in Rijndael finite field.
- S-box. It is basically the same as SubBytes described earlier.

Then the main step of the key schedule algorithm is as follow:

- Take as input a 32-bit word and the iteration number i .
- Perform rotation on the word, followed by S-box operation on each byte.
- Then, on the first byte perform xor-operation with $rcon(i)$, returning 32-bit word output.

Figure 2.6 highlights the whole key schedule for AES 128. The block x is

Fig. 2.6: AES Key Schedule algorithm

obtained by performing the main step of the key schedule algorithm mentioned before, where the input is the previous 32-bit value. Each block x_{4i} will be used as key for round i .

The key schedule algorithm of AES can be reversed, so knowing one of the round key allows the attacker to directly obtain part or the whole master key. In the last round, the **MixColumns** operation is omitted. According to the Rijndael documentation, it is to make the cipher and the inverse more similar in structure. Moreover, it is explained that the omission of the **MixColumns** operation in the last round neither improve nor reduce the security.

2.3 Preliminaries on Physical Attacks

Physical attacks target the cryptographic device directly, rather than finding a weakness in the cryptographic algorithm. There are two ways to categorize the physical attacks: the action of the attackers and the level of invasiveness. Based on the action of the attackers, the attacks can be classified as:

- Active attacks: the attacker tries to induce faults or change the behavior of the device,
- Passive attacks: the attacker tries to obtain secret key through observation only.

Based on the level of invasiveness, the attacks can be classified as:

- Invasive: the attacker can do anything on the device,
- Semi-invasive: the attacker only destroys the package (de-packaging), but the device remains intact and is still functional,
- Non-invasive: the attacker leaves no physical trails showing that the device has been attacked.

The combination of both classifications can be summarized in Figure 2.7. In general, both semi-invasive and non-invasive attacks are relatively moderate to low cost and easier to perform compared to invasive attacks. Here, we put more emphasis on passive non-invasive attacks and active semi-invasive attacks. As the name suggests, passive non-invasive attacks mean that there will be no damage on the physical layout of the device. Most of the time, the owner of the device does not even realize that the device is being attacked, since there are no observable physical trails. The example of non-invasive attacks is the side-channel attacks. It can be

Fig. 2.7: Classification of Physical Attacks (main focus on active semi-invasive and passive non-invasive)

performed by observing the physical leakage of the device such as the power, or by observing the physical behavior, such as the timing. Then, statistical analysis can be done in order to perform the attacks. For example, in [42], it is shown how to conduct a timing attack on OpenSSL. In this attack, the attacker can obtain the secret key by observing time variances in the RSA decryption process.

For semi-invasive attacks, the outer layer of the device will have to be removed or de-packaged so the chip surface will be exposed. In active semi-invasive attacks, the attacker can inject faults on the device during the operation. This can cause disruption of some functionality of the device. Some of the changes inflicted on the device could be in the form of deactivation of countermeasure or bypass of security check. It could also be faults in the computation process, such that the produced output is incorrect, which then could be exploited for attacks. Some external tools can be used to produce the faults, for example, laser, *etc.*

Part II

MACHINE LEARNING FOR SIDE-CHANNEL ATTACKS

3. BASICS OF SIDE-CHANNEL ATTACKS

In the past, the cryptography community had neglected to take into consideration that although a cryptosystem could be theoretically secure, its implementation might not be as secure. Then, in 1996, Kocher [95] showed that there exists a method to retrieve the secret key that neither deals with finding theoretical weaknesses of the cryptographic algorithm nor breaks the system using brute force/exhaustive search. Instead, by utilizing the physical effects of the implementation, in this case the timing behavior, the information about the secret key can be deduced. In his work, he explained that execution of different key values might take different time, and hence revealed some information about the key. His findings showed that although the algorithm can be proven to be secure theoretically, when implemented on ordinary digital circuits, the physical side-effect observed during the processing of the algorithm, such as the timing and power consumption [96], could potentially leak information if properly analyzed.

This publication attracted a lot of attention from the cryptography community and it represented the groundwork of side-channel attacks (SCA) on cryptographic implementations. Ever since, both academic and industrial researchers have shown great interest in SCA, and a lot of attacks and potential countermeasures have been published. One of the main motivation in studying SCA is that besides being theoretically applicable, it has been repeatedly shown that SCA is able to practically break even provably secure cryptosystems.

SCA normally exploits the physical leakage from electronic devices, and this physical leakage carries some information regarding the processed internal state of the device. The side-channel leakage can provide alternative methods to retrieve some useful information for the attacker, such as how many rounds are performed, various operations executed by the device, which can be exploited to retrieve the secret information. Hence, these information can be used to retrieve secret materials (like encryption keys) currently being processed. This information is usually not available within the cryptographic algorithm, since it will be hidden through series of round functions. Traditional cryptanalysis methods will have to make the analysis on the plaintext or ciphertext layer, or based on reduced rounds of the cipher. For SCA, using the leaked information, the attacker can then reduce the cryptanalysis to only partial process of encryption, such as a single round.

In general, SCA can be categorized into profiling and non-profiling based attacks. For non-profiling based attacks, the attacker performs the attack directly on the device by measuring the side-channel leakage followed by a statistical analysis in order to retrieve the secret. In contrast, for profiling based attacks, the attacker uses a cloned or similar device to precisely characterize or model the target and requires lesser measurements when conducting the attacks.

Both profiling and non-profiling can be considered as divide-and-conquer approaches, where smaller parts of the information are retrieved. In case of block ciphers, for example, on AES, the S-box is processed in blocks of smaller size (8-bit), and the attack can aim at retrieving the round key of each block independently, either at the beginning or at the end of the execution (otherwise the key search space will be too large). Normally, the S-box is used as target of the attacks due to its non-linearity property [140]. Based on this property, the S-box will affect on average $1/2$ of the output bits for any 1 bit difference in the hypothesis. For most cases, once

the round keys have been retrieved, the master key can be determined. However, in other scenarios, for example when the key schedule is irreversible, the attacker can retrieve the secret key for corresponding round, and use the information to attack the other rounds in similar manner.

When conducting the attacks, the attacker needs to utilize the side-channel leakage. The leakage normally depends on the processed intermediate value. The intermediate value is normally described as a function f of the plaintext (x) and key (k), or $IV_{x,k} = f(x, k)$. The function f represents an operation in the cipher. For example, in AES, the intermediate value could be the output of the first S-box in the first round, and hence, $IV_{x,k} = S(x \oplus k)$. Thus, the leakage can be formulated as follow:

$$T_{x,k}^l = L^l(IV(x, k)) + \epsilon = L^l(f(x, k)) + \epsilon, \quad (3.1)$$

where L^l is the leakage function that maps the intermediate value ($IV_{x,k}$) of any plaintext and key pair to its side-channel leakage $T_{x,k}^l$ at point in time l , and ϵ is the (assumed) mean free Gaussian noise ($\epsilon \sim N(0, \sigma^2)$).

Leakage Model

In general, a leakage model is used to approximate the leakage behavior or the leakage function. Commonly used leakage models are:

- bit model: $Bit_i(IV(x, k))$, returns 0 if the i -th bit of $IV(x, k)$ is 0, and 1 otherwise,
- Hamming weight (HW): $HW(IV(x, k)) = \sum_i Bit_i(IV(x, k))$, returns how many bits that are of value 1, for example, in the AES case, the Hamming weight of S-box output is the value between 0 (all bits are 0) and 8 (all bits are 1).

- Hamming distance (HD): $HD(IV_i(x, k), IV_j(x, k)) = HW(IV_i(x, k) \oplus IV_j(x, k))$, returns the Hamming weight of exclusive-OR of two intermediate value, for example, in the AES case, they can be the S-box input and S-box output.

In an electronic circuit, the registers consume much more power than the combinatorial logic, and the power consumption (or transition) is synchronized with the clock which increases the signal-to-noise ratio (SNR) of the leakage and makes it easier to attack. The power consumption of these registers is directly proportional to the number of transitions or in other words, the number of bits updated. This relationship is very well captured by the HD model, and the HW model is a variant of the HD model when the register is initialized to 0. HW model is usually used for modeling the leakage of microcontroller or software implementation, because the registers are usually precharged, whereas HD model is usually used for the case of hardware, where no precharge is done. One of the property of HW and HD model is that both models assume equal weighting for all the bits. However, there are also other more complex models, like linear regression model, which are based on the assumption that each bit is weighted differently when processed. In general, there are more leakage models, however, HW and HD models are among the most commonly used ones, because of their simplicity.

3.1 Non-profiling attacks

In non-profiling based attacks, the attacker measures the side-channel leakages, and then performs statistical analysis in order to obtain the hypothetical key value. There are two main techniques, usually for power analysis, though they could also be used for other side-channels such as electromagnetic emanation (EM), namely: Simple Power Analysis (SPA) and Differential Power Analysis (DPA).

SPA can be performed if the leakage from the device is distinguishable. It means that the attackers can easily distinguish different processed values or operations, simply by observing the side-channel measurements through visual inspection. To give one example, let us look at RSA implemented using Square and Multiply exponentiation algorithm [70]. For the decryption, we need to compute $c^d \bmod N$, where d is secret exponent and N can be a very large number. To calculate, d is expressed as binary notation, $d = 2^{k-1}d_{k-1} + \dots + 2^1d_1 + 2^0d_0$. Then, initialized $z = 1$, for $i = k-1$ until 0, calculate $z = z^2 \bmod N$, and if $d_i = 1$, calculate additional $z = z \times c \bmod N$.

It can be observed that when the bit value $d_i = 0$, only squaring operation is performed, whereas when $d_i = 1$, squaring and multiplication are performed. The attack can then be performed when the power consumption for square operation can be differentiated from multiplication. Typically, the multiplication operation will consume more power than squaring operation. When the attacker can distinguish between square and multiply operation, it can then be determined whether the processed value is 0 (square) or 1 (square and multiply), just by observing the trace, as can be seen in Figure 3.1. In this attack, as few as a single measurement is enough to recover the secret key.

In DPA, the idea is to perform a divide-and-conquer approach to obtain parts of the round key at a time, utilizing additional information leaked. The standard procedure for DPA attacks is as follow [114]:

- Measure the side-channel leakage (power, timing, *etc*) when performing the encryption or decryption, and collect a number of measurements (N) for the analysis. The measurements can be denoted using $\mathbf{T} \in \mathbb{R}^{N \times N_P}$, where N_P denotes the length of each measurement or the time sample in which the measurements are done.

Fig. 3.1: Illustration of SPA

- Determine the intermediate operation(s) to be attacked, which depends on non-constant data (usually plaintexts or ciphertexts) and unknown key material.
- Calculate the hypothetical intermediate value ($\tilde{IV}_{x,k}$), based on the non constant data x and all possible key hypothesis $k \in K$. In this phase, the attacker calculates $v_{i,j} \in \mathbf{IV}$, where $v_{i,j} = f(x_i, k_j), i = 1, \dots, N, j = 1, \dots, N_P$ for different plaintext value x_i and key hypothesis k_j .
- The attacker now tries to map the intermediate values to the corresponding leakage. Here, the attacker assumes a hypothetical leakage model $\mathbf{H} = \tilde{L}(\mathbf{IV}) \approx \mathbf{T}$. The leakage model is highly dependent on the device and the more accurate the leakage model, the better the attack will be. As mentioned earlier, most commonly used leakage models are HW and HD model.
- In the final step, the attacker then tries to determine the relation between the hypothetical leakage ($\mathbf{H}$) with the real leakage ($\mathbf{T}$), and distinguish the correct key hypothesis from the wrong key hypothesis. In this case, a distin-

guisher function D is used to calculate the metric of similarity between the hypothetical and the real value. The value $D(\vec{H}_k, \vec{T}_j)$ measures the similarity for hypothetical value with key candidate k and the measurements at point in time j . The key candidate with the best similarity (with respect to some metric) would most likely be the correct key.

In the first paper about DPA [96], Kocher used the Difference of Means (DoM) as the distinguisher. The leakage model used is simply the value of one bit of the intermediate value. Using AES as example, the attacker uses the first round S-box output as the target. For the leakage model, the attacker chooses 1 bit of the output, for example the Least Significant Bit (LSB). Using multiple plaintexts, the attacker calculates the hypothetical intermediate values for the S-box output for different key hypothesis. Since the S-box is a 8-bit to 8-bit mapping, there are 2^8 possible inputs and hence 2^8 possible key candidates for each plaintext. After the S-box outputs have been calculated, the traces are separated according to whether the LSB is 0 or 1 for a specific key candidate. The mean is then calculated for each group of traces, depending on whether the bit value is 0 or 1 (μ_0 and μ_1). The key hypothesis that leads to the largest difference ($\mu_0 - \mu_1$) is then considered as the correct key. This is shown in Figure 3.2a, where the correct key candidate, marked with blue has the highest difference, compared to wrong key hypothesis, marked with red. Thus, the term DPA originally refers to this method of taking the difference of means. However, in later publications the term DPA can be used interchangeably with other methods that uses different statistical concepts.

Later, Brier *et al.* [40] proposed Pearson correlation as alternative statistical tests for distinguisher. This is known as CPA (Correlation Power Analysis). The correlation measures the linear relationship between two data, in this case, the hypothetical intermediate value and the measured side-channel leakage. It can then

be formulated as follow:

$$r = \frac{\sum_{i=1}^N (h_{i,k} - \bar{h}_k)(t_{i,j} - \bar{t}_j)}{\sqrt{\sum_{i=1}^N (h_{i,k} - \bar{h}_k)^2 \sum_{i=1}^N (t_{i,j} - \bar{t}_j)^2}}, \quad (3.2)$$

where $\vec{t}_j = \{t_{1,j}, \dots, t_{N,j}\}$ and $\vec{h}_k = \{h_{1,k}, \dots, h_{N,k}\}$ are the side channel measurements at point in time j and the hypothetical value for key hypothesis k . The values $\bar{t}_j$ and $\bar{h}_k$ denote the mean value of $\vec{t}_j$ and $\vec{h}_k$ respectively. The higher $|r|$ is, the stronger the correlation coefficient and the linear relationship between the hypothesis and the measurements. Normally, under the correct model, the key candidate with the highest (absolute) correlation represents the correct key. Again, we use AES as example. Regarding the leakage model, we use the HW model from the first S-box output. For each key candidate, we calculate the correlation between the hypothetical value and the traces. In Figure 3.2b, it can be observed that the correct key candidate, marked with blue, achieves the largest negative value, which means that the traces have inverse linear relationship with the leakage model.

Fig. 3.2: Example of DPA on AES first S-box output

In both methods, it can be seen that AES S-box is commonly chosen as the

target, owing to its non-linearity property as described earlier. Intuitively, one bit difference in the S-box input will affect most of the bits in the output. Hence, the correct key will lead to better distinguishability from the wrong key hypothesis (see Fig 3.2). However, in case of a linear operation like exclusive-OR, one bit difference will affect exactly one bit in the output. Hence, for any key hypothesis that differs only in one bit from the correct key hypothesis, the difference will be harder to distinguish, and it can also be easily hidden by the noise. Another reason is that, for S-box operation, the key hypothesis size is small (for AES, it is 2^8 key candidates).

Another method that was recently proposed is the mutual information [68], based on Shannon entropy in information theory. It can be used to measure the non-linear dependency between the data. It is generic, in the sense that the attack can be performed independently of the leakage model. The main issue with this attack is that it requires the estimation of the probability distribution of the leakage, which might be hard to determine accurately.

3.2 Profiling attacks

In profiling based attacks, the attacker is assumed to have full control over the training devices, which is chosen to be similar to the real target. In this scenario, the attacker can do anything on the device, and thus, the attacker has the full freedom to build a profile out of the device. Profiling based attacks can be considered as one of the strongest attacks, assuming that the model or profile is built correctly. Hence, most of the efforts are usually spent on doing the profiling of the device. Profiling based attacks have the advantage that they can build different profiles for different intermediate values, and hence, can better approximate the leakage

behavior of the (targeted) device. Common examples of profiling based attacks are Template Attacks (TA) [46] and Stochastic Approach (SA) [152].

Profiling based side-channel attacks are usually carried out in two phases: A profiling (learning or training) phase and a key extraction (attacking) phase. In the profiling phase, the leakage caused by the processed intermediate value can be characterized either as a classification (*i.e.*, TA) or as a regression (*i.e.*, SA) problem. In classification problem, given the traces, the attacker tries to predict the classes associated with the traces. The class is usually defined by the intermediate value and the leakage model, for example, for the HW of AES S-box output, the classes are the value from 0 to 8. In regression problem, given different classes or intermediate values, and the traces value T , the attacker tries to establish the relationship between them, by defining an approximation function $\tilde{L}$. For example, in AES case, we can use linear regression to denote the linear relationship between the HW of AES S-box output and the traces, as $T = \tilde{L}(HW(S(x \oplus k))) = \beta_1 HW(S(x \oplus k)) + \beta_0 + \epsilon$. The β_1 and β_0 are linear coefficient, and ϵ is some error term or noise.

The ways in which the profiling based attacks are conducted are similar to non-profiling attacks. On the training device, the attacker measures the side-channel leakage, determines the targeted intermediate value and estimates the hypothetical intermediate value based on some leakage model. The difference is that, for this phase, the attacker can control the device as opposed to non-profiling attacks. Thus, the attacker can set the key and the plaintext/ciphertext as wanted, and therefore have a perfect knowledge of the intermediate values. Also, the attacker can better determine which features (time samples) of the measurements are informative, and can discard the rest of the features, as well as building a leakage model that better approximates the behavior of the leakage. This way, the attacker can characterize the leakage for different intermediate values more accurately. The characterization

obtained from the training device is called the profile of the device. The attack will then be carried on the real target device, which is assumed to behave almost identically to the training device. During the attack phase, the attacker measures the side-channel leakage from the target device and tries to match it with the profile to determine which key hypothesis is the most probable.

Before the traces can be used for attacks, they are usually preprocessed. In the preprocessing phase, several actions can be done, such as normalization, data transformation, feature selection, *etc.* This is usually done to minimize the effect of noise in the data or to remove redundancy in the traces. In feature selection phase, multiple features of the side-channel leakage, e.g. different number of time samples (N_P) in the side-channel measurements are chosen and used in the profiling phase. The features selected should contain the most relevant information regarding the intermediate values. Often, the feature selection method plays a crucial role, as irrelevant features can make the profiling harder and the results obtained will be worse since it would add noise to the computation.

In general, there are different feature selection methods, such as: correlation [144], or SNR based [77] feature selection. The SNR can be defined as the ratio of the variance of the signal over the variance of the noise. Both approaches only consider each feature or time sample independently of each other. Another SNR based method was proposed by [76], which takes into account the relations among different features in the leakage. Instead of calculating the SNR of each individual point in time, they calculated the joint SNR of multiple samples. They also highlighted that when multiple samples that have a high SNR individually are combined, the resulting SNR might not be necessarily optimal.

Other SNR based feature selection methods have also been published recently, using the normalized interclass variance or NICV [23], and using the t-test [60]. In

NICV, the aim is to find features in which, the inter-class variance of different classes are the largest (maximize the difference of different classes). In t-test method, a normalized difference between set of data obtained using fixed input and with random input values is calculated, where the features that maximize the data dependency (as highlighted by the peaks of the difference) are selected.

Alternatively, rather than selecting different features, the set of features can be mapped into another dimension using Principal Component Analysis (PCA). In this case, the mapped features (principal components) will be independent of each other, with the first principal component having the largest variance or information, and every subsequent principal component will have the highest variance that is orthogonal to all the previous components. One of the application has been shown for the case of TA [11]. However, up to now, there is no absolute method to determine which feature selection is the best, and the research is still ongoing.

3.2.1 Template Attacks

One of the most popular profiling based methods is Template Attacks or TA [46]. The idea of TA is that the attacker can build different profiles for different classes $M \in \{M_1, \dots, M_{N_c}\}$, where each class corresponds to different intermediate values. In the original publication, the profiles are built for all possible value for part of the key and plaintext. In the AES example, the number of profiles for one S-box output will be equal to $2^8 \times 2^8$ for all possible 8-bit key candidates and 8-bit plaintext values. However, the size of the profile will be large, and more traces are needed in order to build each profile. Later, a leakage model is introduced to reduce the number of classes used to build the profile [64]. For example, the common classes used are usually the HW or the HD of the S-box output instead, so the number of classes could be reduced to 9 classes (0 to 8).

In TA, the main assumption used by the attacker is that the side-channel leakages are drawn from a multivariate Gaussian distribution. Given the profiling measurements $\mathbf{T} = \{\vec{T}_i | 1 \leq i \leq N\}$, where $\vec{T}_i$ defines each trace with N_P features, the attacker groups the traces according to their classes to obtain the sets $\{\mathbf{T}_{M_1}, \dots, \mathbf{T}_{M_{N_c}}\}$. Each $\mathbf{T}_{M_i} = \{\vec{T}_{M_{i_1}}, \dots, \vec{T}_{M_{i_{n_i}}}\}$ contains all the traces corresponding to class M_i . For each class M_i , since the traces are assumed to be drawn from a multivariate Gaussian distribution, the attacker calculates the mean and covariance matrix:

$$\mu_{M_i} = \frac{1}{n_i} \sum_{j=1}^{n_i} T_{M_{i_j}} \quad (3.3)$$

$$\Sigma_{M_i} = \frac{1}{n_i - 1} \sum_{j=1}^{n_i} (T_{M_{i_j}} - \mu_{M_i})(T_{M_{i_j}} - \mu_{M_i})^T \quad (3.4)$$

The pair $(\mu_{M_i}, \Sigma_{M_i})$ for different class M_i is commonly referred to as template. Hence, in total, N_c templates are required for N_c distinct classes.

During the attack phase, after measuring a new set of N_A traces $\mathbf{I} = \{\vec{I}_i | 1 \leq i \leq N_A\}$ from the target device, the attacker uses the maximum likelihood (or log-likelihood) as distinguisher.

$$\begin{aligned} k &= \underset{k^*}{\operatorname{argmax}} \log p(k^* | \mathbf{I}) \\ &= \underset{k^*}{\operatorname{argmax}} \sum_{i=1}^{N_A} \log \frac{p(I_i | \mu_{M(k^*)}, \Sigma_{M(k^*)}) p(k^*)}{p(I_i)}. \end{aligned} \quad (3.5)$$

Here, log-likelihood is often used to avoid numerical problem when taking exponentiation. Due to the monotonicity of the logarithm function, the key with the highest $p(k^* | \mathbf{I})$ will also have the highest $\log(p(k^* | \mathbf{I}))$. Then, some constant term in the

log-likelihood that are not key dependent can be removed.

$$k = \operatorname{argmax}_{k^*} \sum_{i=1}^{N_A} -\frac{1}{2} \left(\log \left((2\pi)^{N_P} |\Sigma_{M(k^*)}| \right) + \left(z_i^T \Sigma_{M(k^*)}^{-1} z_i \right) \right), \quad (3.6)$$

where $z_i = (\vec{I}_i - \mu_{M(k^*)})$. Here, $(\mu_{M(k^*)}, \Sigma_{M(k^*)})$ refers to the template for class $M(k^*)$, based on the intermediate value obtained from key hypothesis k^* .

The advantage of TA is that the attack is easy to conduct since it is a simple calculation of the means and variances of the traces for different classes. In this case, the attacker can rank the key candidates based on their likelihood instead of doing direct classification. Hence, if the most probable key is incorrect, the attacker can check for the next most probable key and so on. One of the limitations of TA is that it is based on the assumption that the data are drawn from a normal distribution. Hence, TA will tend to perform poorly when the data used is not normally distributed, since it will be poorly estimated.

3.2.2 Stochastic Approach

Another example of profiling based attack is the Stochastic Approach or SA [152]. In SA, the profiling phase is further split into: (1) estimation of the deterministic part and (2) estimation of the noise covariance matrix. Thus, the training set $\mathbf{T}$ can also be separated into two sets, $\mathbf{T}_1$ and $\mathbf{T}_2$. Unlike TA, SA uses only one covariance matrix to characterize the noise and also uses linear regression instead of classification by the means of each class.

In order to characterize the deterministic part of the leakage, a subspace representation of the estimated intermediate value is used. The function φ maps the intermediate value $f(x_i, k)$ to a subspace of dimension u . Hence, for plaintext x_i and key k , $\varphi(f(x_i, k)) = \{\vec{a}_{i,j}(k)\}_{j=0}^{u-1} = [a_{i,0}(k) \dots a_{i,u-1}(k)]$. There are different

intermediate value representations, also called base function. For example, if we are using the 8-bit representation of the AES S-box output for the targeted intermediate value, then, the subspace representation with 9 dimension is $a_i = [a_{i,0}(k) \ a_{i,1}(k) \ \dots \ a_{i,8}(k)]$, with $a_{i,j}(k)$ being the j^{th} -bit of the intermediate value, $f(x_i, k)$, for $1 \leq j \leq 8$. and $a_{i,0}(k) = 1$ defines the constant term.

The deterministic part can then be determined as $\tilde{L} = \mathbf{A} \cdot \beta$, where $\mathbf{A} = (a_{i,j}(k))_{1 \leq i \leq N_1; 0 \leq j \leq u}$, with $\vec{a}_i$ as a row element of $\mathbf{A}$ and N_1 denotes the number of measurements in $\mathbf{T}_1$. We can then determine $\beta = (\beta_{j,l})_{0 \leq j \leq u; 1 \leq l \leq N_P}$, the coefficients representing the contribution of each element to the leakage, based on the first training set of traces $\mathbf{T}_1$, as follows:

$$\beta = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{T}_1, \quad (3.7)$$

where

$$\beta = (\beta_{j,l})_{0 \leq j \leq u; 1 \leq l \leq N_P}, \quad (3.8)$$

are normally called beta coefficients. The estimation of the deterministic part is basically the estimation of the beta coefficients. The solution presented in equation (3.7) minimizes the mean square error (MSE) between the estimator of the deterministic part and the observed leakage on each feature l :

$$MSE(l) = \frac{1}{N_1} \| \vec{T}_1^l - \mathbf{A} \vec{\beta}_l \| \quad (3.9)$$

Thus, the deterministic leakage model for feature l for plaintext and key pair (x, k) can be expressed as

$$\tilde{L}^l(f(x, k)) = \sum_{j=0}^{u-1} a_j(k) \cdot \beta_{j,l}. \quad (3.10)$$

Since $a_0 = 1$, $\beta_{0,l}$ represents the offset of the data at feature l . We denote $\tilde{\mathbf{L}} = [\tilde{L}^1(f(x, k)) \dots \tilde{L}^{N_P}(f(x, k))]$, the set of leakage functions mapping the intermediate value at different points in time. Hence, the deterministic part can be considered as a generalization of the basic leakage model. When the bitwise representation model is used, and the beta coefficients are almost equal, then it behaves similarly to the HW or HD models. However, if the beta coefficients are different (for example, one bit is leaking more than the rest), then this model could better approximate the leakage behavior compared to classical HW or HD model.

The next phase is to estimate the noise of the traces. This is done with the remaining training set of traces, $\mathbf{T}_2$. The noise is assumed to be multivariate normal and to have zero mean, so the sample covariance matrix of the noise can be estimated as follows with a second training set of traces $\mathbf{T}_2 = \{\vec{T}_{2,i} | 1 \leq i \leq N_2\}$:

$$\Sigma = \frac{1}{N_2 - 1} \sum_{i=1}^{N_2} (\vec{T}_{2,i} - \tilde{\mathbf{L}}(f(x_{2,i}, k)))(\vec{T}_{2,i} - \tilde{\mathbf{L}}(f(x_{2,i}, k)))^T \quad (3.11)$$

Then, in the key extraction (attack) phase, N_A traces $\mathbf{I} = \{\vec{I}_i | 1 \leq i \leq N_A\}$ are measured from the target device and maximum likelihood (or log-likelihood) is used to reveal the key, similarly to the TA case. With a further simplification (removing all non key dependent components), the equation can be written as:

$$k = \underset{k^*}{\operatorname{argmin}} \sum_{i=1}^{N_A} (z_i^T \Sigma^{-1} z_i), \quad (3.12)$$

where $z_i = \vec{I}_i - \tilde{\mathbf{L}}(f(\tilde{x}_i, k^*))$.

High Dimensional Subspace

For SA, the selection of subspace used for modeling the leakage behavior is also crucial. Generally, for 8-bit data, either 2-dimension (HW, HD or bit model) or 9-dimension (bitwise model) subspaces are used. However, a higher dimensional subspace can also be used if there is some inter-dependent leakage among different bits [77]. It could be possible that there are some interactions between two or more bits during transitions, which might contribute to the overall leakage. This behavior can be caused by the internal properties of the circuit structure.

We consider the 9-dimensional subspace from the 8-bit data, which corresponds to the bits of the intermediate value. We then consider high-dimensional subspace to capture the dependencies among different bit values. If we consider different interactions among different n bit lines, we can consider n as the order of dependencies. The resulting subspace is a $\sum_{i=0}^n \binom{8}{i}$ -dimensional subspace, where $1 \leq n \leq 8$.

For a 9-dimensional subspace, it is assumed that different bitwise transitions are independent from each other. It is spanned by the basis vector: $a_{i,0}(k) = 1$ and $a_{i,j}(k) = j^{th}$ -bit of $S(x_i, k) - \frac{1}{2}$, for $j = 1, \dots, 8$. The subtraction by $\frac{1}{2}$ is to ensure that it is mean-free.

We can denote $B_0 \cup B_1 = 1 \cup \{a_{i,1}(k), \dots, a_{i,8}(k)\}$ the set of all bases of the 9-dimensional subspace.

For higher dimensional subspaces, we denote:

$$B_n = \{a_{i,j_1}(k) \cdots a_{i,j_n}(k) - 2^{-n} | 1 \leq j_1 < \dots < j_n \leq 8\}$$

Thus, the set of the basis function can be defined as $B_0 \cup B_1 \cup \dots \cup B_n$ for a $\sum_{i=0}^n \binom{8}{i}$ -dimensional subspace.

3.3 Side-Channel Evaluation Metric

To compare the performance of different attacks, Standaert *et al.* [159] proposed two different evaluation metrics, namely the success rate and the guessing entropy metric. These metrics measure the efficiency of the attacks to utilize the side-channel leakage for key recovery. Success rate is usually denoted by the order, usually as n -th order success rate, where it defines the probability that the attacker's n best key guesses contain the correct key.

The guessing entropy, on the other hand, defines the average rank for the correct key in the key hypothesis ranking outputted by the distinguisher, or alternatively, the average number of key guesses needed to obtain the correct key. It can be formulated as $GE = \frac{1}{n} \sum_{i=1}^n \text{rank}_i(k)$, where $\text{rank}_i(k)$ denotes the rank order of the correct key k in the prediction from the results of the attacks in experiment $i \in \{1, \dots, n\}$.

Both the success rate and the guessing entropy measure the efficiency of the attacks, in terms of the number of attack traces. To compare the performance of different attacks, usually the attacker can either:

1. define them in terms of the number of how many attack traces required to achieve specific guessing entropy or success rate, or
2. define them in terms of the success rate or guessing entropy for a given fixed number of attack traces used.

For example, the attack A can be considered better if it requires less attack traces in order to reach guessing entropy 1, compared to attack B.

4. BASICS OF MACHINE LEARNING

Recently, new profiling based attacks which apply machine learning methods have been proposed. Machine learning can be considered as a joint subfield of pattern recognition and artificial intelligence, and is a well-studied topic with strong theoretical foundations as well as a wide range of possible applications. Formally speaking, machine learning can be defined as the study of how a machine improves its performance based on previous experiences or training [120].

4.1 *Introduction to Machine Learning*

Most of machine learning algorithms are designed to deal with classification of the data, or to find the (hidden) structure of the data. In general, machine learning methods are categorized as supervised [101] and unsupervised learning [55]. In some cases, semi-supervised learning [176] can also be included in the category, standing between supervised and unsupervised learning.

The difference of these concepts is highlighted in Figure 4.1. The upper left part of the figure shows the supervised learning. Intuitively, in supervised learning, the machine is given a set of training data with the label or class available, and it tries to build a model that associates the data with the label. Its performance is then tested on independent sets, by matching the prediction (label) it outputs with the actual label.

When building the model using different learning algorithms, there are usually

learning parameters that need to be tuned. Determining the parameters and then test it using the same set of training data is not recommended, since the constructed model will just "memorize" the label of the training data, and it could not generalize to unseen test data. This is referred as overfitting.

Normally, to avoid overfitting, a set of training data is hold out as validation data. The training is done on the rest of the training data, and evaluation of the performance is done on the validation data. After the best parameter has been determined and the model is built, the final evaluation of the learning algorithm can be done on the test data. However, in this case, when the training data is split, the number of data available for training is reduced, and the performance can be biased depending on the splitting of training and validation data.

To overcome the issue, cross-validation method is used [98]. The general form is k -fold cross validation. The training data is split into k sets. Then the training is done using $k - 1$ sets of the training data, and validated on the remaining set. This is done repeatedly on all different $k - 1$ sets, and the average is computed in order to calculate the overall performance. If the size of k is equal to the number of training data, this is called leave-one-out validation.

Back to the concept of learning algorithm, in the upper right part of the figure, it shows the unsupervised learning, In the unsupervised learning, the machine is given a set of unlabeled data, and it tries to determine the hidden structure of the data, for example the clustering or grouping of the data based on similarity metric, where the data that are similar will be more likely to be grouped together.

Lastly, the bottom part of the figure shows the concept of semi supervised learning. For semi-supervised learning, the scenario is quite similar to unsupervised learning. Most of the available data are unlabeled, however, there is a small subset of the data which are labeled. Thus, the learning algorithm can use these data to

guide the prediction of the other unlabeled data.

Fig. 4.1: Illustration of different learning concept: supervised (left), unsupervised (right), semi-supervised (bottom)

4.2 Examples of Machine Learning Algorithms

4.2.1 Support Vector Machine

One of the most popular learning algorithm for classification is Support Vector Machine (SVM). The current version of SVM was proposed by Cortes and Vapnik [51], which incorporates non-linear classification. In the default binary classification problem (only with 2 classes), given $\vec{t}_i \in \mathbb{R}^{N_P}$ as the set of profiling data, and the corresponding classes, $c_i \in \{\pm 1\}$, a separating hyperplane H is constructed. This hyperplane can be written as $\langle \vec{w}, \vec{t} \rangle + b = 0$, where $\vec{w}$ is the normal vector to the hyperplane, $\vec{t} \in \mathbb{R}^{N_P}$ is a point on the plane, and $\frac{b}{\|\vec{w}\|}$ defines the offset of the hyperplane to the origin in the direction of the normal vector. This concept is

Fig. 4.2: Illustration of SVM

highlighted in Figure 4.2.

In the original design, hard margin SVM is used, where the goal is to find a separating hyperplane that has maximum distance from the data in both classes. In this case, the hyperplane is constructed such that all the data in each side of the hyperplane will belong to the same class. This might cause the problem of overfitting in the presence of outliers. To tackle this issue, the soft margin SVM is used instead. In soft margin SVM, the additional slack variable ξ_i is introduced into the equation to define how much the data t_i deviates from its original class (distance as measured from the hyperplane H). This might cause some misclassification on non-separable data, but improves the overall performance.

Another problem with SVM is that it was originally designed as a linear classifier. To deal with a non-linear classification, the kernel trick was introduced. Using the kernel trick, the features can be mapped to the higher dimension, $p \rightarrow \phi(p)$. For

example, if $p = \{p_1, p_2\}$, one of the possible mapping is $\phi(p) = \{p_1^2, p_2^2, \sqrt{2}p_1p_2\}$, a mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$. This is illustrated in Figure 4.3. The kernel trick allows the non-linear mapping without requiring explicitly the exact mapping function ϕ . It only requires the computation of inner product of the vectors, which is defined by the kernel function $K(\vec{t}_i, \vec{t}_j) = \langle \phi(\vec{t}_i), \phi(\vec{t}_j) \rangle$. Some of the most commonly used kernels are depicted in Table 4.1 with some adjustable parameters, i.e., C , γ , d , etc.

Fig. 4.3: Non-linear classification using SVM

In general, SVM tries to solve the following equation:

$$\underset{w, b, \xi}{\operatorname{argmin}} \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^N \xi_i \quad (4.1)$$

$$\text{subject to: } c_i(\langle \vec{w}, \phi(\vec{t}_i) \rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i,$$

where: $\vec{w}$ defines the normal vector of the hyperplane, C defines the cost of misclassification (regularization parameter), ξ defines the slack variable for data $\vec{t}_i$ from its targeted class c_i , and b defines the distance from the origin.

By transforming the optimization problem into its dual, the equation can be rewritten as:

$$\underset{\alpha}{\operatorname{argmax}} \sum_{i=1}^N \alpha_i + \frac{1}{2} \sum_{i,j=1}^N c_i \alpha_i K(\vec{t}_i, \vec{t}_j) \alpha_j c_j \quad (4.2)$$

$$\text{subject to: } \alpha_i \geq 0 \forall i \text{ and } \sum_{i=1}^N \alpha_i c_i = 0,$$

where α_i is the Lagrange multiplier [166]. The decision boundary calculated after solving the dual can then denoted as follow:

$$H(\vec{t}) = \text{sgn}\left(\sum_{i=1}^N c_i \alpha_i K(\vec{t}_i, \vec{t}) + b\right) \quad (4.3)$$

The parameter C can be considered as a regularization parameter as it balances the error in the training data and the norm of weights ($\|\vec{w}\|$) as seen in the equation earlier. It defines the cost of misclassification, with smaller value of C tending to ignore some misclassification near the decision boundary in order to increase the margin, while larger value of C tending to fit all the data, resulting in overfitting decision boundary. For the most commonly used kernel, Radial Basis Function (RBF), γ defines the width of the kernel, *i.e.*, how flexible is the decision boundary.

In general, SVM is used to solve the binary classification problems, but it can also be extended into a multi-class classification problem [59]. The most common approaches for extension are one-against-one and one-against-all. Both approaches are shown to be equivalent, however, the former approach are usually used, for simplicity reason. The basic idea is to construct multiple binary classifications, where each classifier is build for every possible class against another class.

For determining the parameters in SVM, the simplest method is by performing a grid search (exhaustive search) on some specified parameter space [83]. The

Tab. 4.1: Commonly used kernels for SVM [51]

Kernel name	Kernel function
Linear	$K(t_i, t_j) = t_i^T t_j$
Radial basis function	$K(t_i, t_j) = \exp(\gamma \ t_i - t_j\ ^2)$
Polynomial	$K(t_i, t_j) = (t_i \cdot t_j)^d$

parameters are then evaluated by performing cross validation, and the parameters with the best cross validation results are used. However, there are also different methods that could also be used to search for the best parameters, but grid search is considered as one of the simplest to perform.

Support Vector Regression

Support Vector Regression or SVR is based on support vectors like in SVM, but is used for regression process instead of a classification like in SVM. In regression, the output will be a real value (continuous), whereas in classification, the output will be a class value (discrete). Suppose we have a set of training data $\{(\vec{a}_i, t_i)\}_{i=1}^N \subset \mathbb{R}^u \times \mathbb{R}$. In SVR, the idea is to find a function $\bar{L}$, which maps $\vec{a} \in \mathbb{R}^u$ to $\bar{L}(\vec{a})$, such that it has at most ε deviation from the actual value t . The ε is a parameter whose value has to be determined (which will be discussed later). In other words, we are interested in determining $\bar{L}(\vec{a}) = \langle \vec{w}, \phi(\vec{a}) \rangle + b$, where $|\bar{L}(\vec{a}) - t| \leq \varepsilon$.

The formulation is similar to SVM, but now we are interested in regression instead of classification. To deal with some error (outside the ε deviation), slack variables (ξ_i, ξ_i^*) are introduced for penalizing the error. The problem can then be formulated as:

$$\underset{w, b}{\operatorname{argmin}} \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \quad (4.4)$$

subject to:

$$t_i - \langle \vec{w}, \phi(\vec{a}_i) \rangle - b \leq \varepsilon + \xi_i$$

$$\langle \vec{w}, \phi(\vec{a}_i) \rangle + b - t_i \geq \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

To solve the optimization problem, a Lagrangian function is calculated including all the constraints and the Lagrangian multipliers $(\alpha_i, \alpha_i^*, \text{ etc})$ [166]. Some of the

variables will then vanish to achieve the optimality, and the resulting dual optimization problem can then be formulated as:

$$\begin{aligned} \operatorname{argmax}_{\alpha, \alpha^*} \quad & -\frac{1}{2} \sum_{i,j=1}^N (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle \phi(\vec{a}_i), \phi(\vec{a}_j) \rangle \\ & - \varepsilon \sum_{i=1}^N (\alpha_i + \alpha_i^*) + \sum_{i=1}^N t_i (\alpha_i - \alpha_i^*) \end{aligned} \quad (4.5)$$

subject to:

$$\begin{aligned} \sum_{i=1}^N (\alpha_i - \alpha_i^*) &= 0 \\ \alpha_i, \alpha_i^* &\in [0, C] \end{aligned} \quad (4.6)$$

We then get

$$\vec{w} = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \phi(\vec{a}_i), \quad (4.7)$$

and thus, by substituting it in $\bar{L}(\vec{a}) = \langle \vec{w}, \phi(\vec{a}) \rangle + b$, we get

$$\bar{L}(\vec{a}) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \langle \phi(\vec{a}_i), \phi(\vec{a}) \rangle + b. \quad (4.8)$$

To avoid calculating the function $\phi(\vec{a})$ directly, we use the kernel function $K(\vec{a}_i, \vec{a}_j) = \langle \phi(\vec{a}_i), \phi(\vec{a}_j) \rangle$. For more detailed explanation of SVR, as well as how to determine the value of b , we refer to the tutorial by Smola and Schölkopf [157].

The solution will then be of the form:

$$\bar{L}(\vec{a}) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) K(\vec{a}_i, \vec{a}) + b \quad (4.9)$$

where, $0 \leq \alpha_i, \alpha_i^* \leq C$ and $K(\cdot, \cdot)$ is the kernel function, see Table 4.1. The key idea is similar to SVM, and kernel trick can also be applied to map the input space to the higher dimension. The common problem encountered in SVR is that it requires

an additional parameter to tune (ε), as opposed to SVM (in SVM with RBF kernel, the required parameters are only C and γ).

Fig. 4.4: Illustration of SVR (the area between the dash line defines $\bar{L} \pm \varepsilon$)

In general, ε defines the tolerance region in which the penalty or loss is considered for any predicted value $|\bar{L} - L| \leq \varepsilon$. Hence, any error less than ε will be tolerated. The loss function for SVR is then defined as ε -Insensitive Loss ($l_\varepsilon = \max(0, |\bar{L} - L| - \varepsilon)$). As illustration, given a set of data $(x, f(x))$ shown in Figure 4.4, from a random function f , SVR can estimate the function $\bar{f}$ that approximates f , which is defined as the solid line. The estimated $\bar{f}$ can closely approximate f even though the function might be non-linear. As can be seen from the figure, most of the data are maintained within the ε -tube ($\bar{L} \pm \varepsilon$), shown as the region between the two dashed line. Similarly, in Figure 4.5, we show that given another random functions, SVR can estimate the functions quite well, without any need to find the correct model (for both functions, one only needs to give the pair (x, y) as training data and set the parameters accordingly).

To deal with the parameters tuning, we apply the heuristic approach from Cherassky and Ma [48]. For RBF kernel, they provide a heuristic formula to es-

Fig. 4.5: SVR estimation on non linear function

timate different parameters. In short, we will describe how to determine the value for each parameter. To calculate the regularization or the parameter C ,

$$C = \max(|\bar{t} + 3\sigma_t|, |\bar{t} - 3\sigma_t|), \quad (4.10)$$

where $\bar{t}$ and σ_t are the mean and standard deviation of the traces in specific points in time. Similarly,

$$\varepsilon = 3\sigma \sqrt{\frac{\ln(N)}{N}}, \quad (4.11)$$

where N is the number of training data, and σ is the variance of noise in the input, which can be estimated using

$$\hat{\sigma}^2 = \frac{k}{k-1} \cdot \frac{1}{N} \sum_{i=1}^N (t_i - \hat{t}_i)^2. \quad (4.12)$$

The estimation of $\hat{t}_i$ is done using k-nearest neighbor (which will be explained later)

Fig. 4.6: Illustration of One-class SVM

and the value of k are typically in range 2-6. Also,

$$\gamma = \frac{1}{2(\sqrt[d]{c_t})^2}, \quad (4.13)$$

where γ is the parameter for RBF kernel, c_t is constant from 0.2-0.5, and d is the dimension of the input data.

One-class Support Vector Machine

One-class SVM is another method based on SVM algorithm [153]. As the name implies, it constructs the classification model based on only one class, and build the separator hyperplane around the boundary of the training data. Once the model has been constructed by the algorithm, the new data will be verified using this model. Hence, if any new data is different from the training data, it will fall outside the boundary and could be considered as outliers, see Figure 4.6.

The idea of the method to separate the training data from the origin in some feature space. Intuitively, the origin acted as the representative of the other class, in order to simulate a binary classification problem, based of only one class. The separation is done by constructing a hyperplane as in SVM and the aim is to max-

imize the distance from the origin. Based on this construction, a region is formed by the hyperplane covering most of the training data. The classification is done as follow: if the data falls within the region, it returns +1 which means it might not be outlier, otherwise, it returns -1 which means that the data might be suspicious.

In general, one-class SVM is formulated by the following equation, which is similar to normal SVM:

$$\underset{w, \xi, \rho}{\operatorname{argmin}} \frac{1}{2} \|\vec{w}\|^2 + \frac{1}{\nu n} \sum_{i=1}^N \xi_i - \rho \quad (4.14)$$

$$\text{subject to: } (\langle \vec{w}, \phi(\vec{t}_i) \rangle) \geq \rho - \xi_i, \quad \xi_i \geq 0 \quad \forall i,$$

The decision function is then formulated as follow (based on the same method to solve SVM):

$$f(\vec{t}) = \operatorname{sgn} \left(\sum_{i=1}^N (\alpha_i) K(\vec{t}_i, \vec{t}) - \rho \right). \quad (4.15)$$

For one-class SVM, the parameter ν denotes the upper bound of the misclassification or margin error of the training data (rates of outliers assumed to be in the data) as well as the lower bound of number of training data that are used as support vectors. If the value of ν is set to be too small, it is assumed that the number of outliers are small and it could result is high false negative rate (outliers deemed as good or genuine data). On the other hand, if ν is set to be too large, most of the training data and thus large portion of the genuine data will be deemed as outliers, resulting in high false positive (false alarm) rate. Other parameters work the same, just like in SVM, and the kernel trick still applies (see Table 4.1).

4.2.2 Random Forest

Random forest or RF [38] is an ensemble learning algorithm that is based on the construction of multiple decision trees. In ensemble methods, the predictions from multiple learning algorithms are combined to obtain a more accurate prediction. The decision tree itself [39] is a classification algorithm which can be illustrated as a tree with if-else rules, based on the attributes or features. However, decision tree is sensitive to small changes in the training data. The decision tree that is grown large enough will usually tend to overfit the training data, *i.e.*, having low bias, but with high variance trade-off. Hence, some post-processing methods such as pruning are sometimes required after construction of the decision tree. RF is designed to address these problems of instability in the decision tree. In RF, multiple decision trees are allowed to grow large, without the need of post-processing. These decision trees are trained on different subsets of the training data using bootstrapping methods (the training data is sampled uniformly with replacement), and the decision is then made by taking the average or majority voting of the decisions given by the trees. This concept is highlighted in Figure 4.7.

So, given a training data $\mathbf{T} = \{\vec{T}_i | 1 \leq i \leq N\}$, each trace $\vec{T}_i$ with N_P features and class $\vec{m} = \{m_i | 1 \leq i \leq N\}$, $m_i \in \{M_1, \dots, M_{N_c}\}$, for each decision tree D_j , where $j = 1, \dots, B$ and B is the number of trees grown, take as sample (with replacement) a set $\mathbf{T}_b$ with features $\vec{l} = \{l_1, \dots, l_p\}$, $l_j \in \{1, \dots, N_P\}$ and $\vec{c}_b$, a subset of $\mathbf{T}$ and $\vec{c}$ respectively. The bootstrapping method ensures that the trees are trained with different sets of data, hence reducing the sensitivity to noise or outliers and the random selection of the features ensure that the trees are uncorrelated, hence it increases the accuracy, by reducing the effect of bad features. In the end, the decision will be made as follow: $c_{pred}(t) = f_{dec}(D_1(t), \dots, D_B(t))$, where f_{dec} is the

Fig. 4.7: Illustration of Random Forest

deciding function, *i.e.*, averaging or majority voting. In general cases, the RF can achieve better result compared to decision trees by the means of variance reduction (through averaging or voting).

4.2.3 Neural Networks

Artificial neural networks or NN [102, 120] are learning algorithms that are based on the idea of mimicking the action of the neurons in human brains, in the way of how it processes a signal. In the academic community, NN have extensively been used, because of their abilities to derive patterns from complex data. NN can be organized into different layers, as highlighted in Figure 4.8. The input layer receives the data, and then the hidden layer(s) processes the data, which is later passed to the output layer. Nodes from different layers are interconnected with each other, and each of them contains an activation function. Each of the input to a node will be multiplied by its weight, and processed through the function to determine if the node should be activated, depending on some thresholds (hence the name activation

function).

Fig. 4.8: Illustration of Neural Networks

The most basic NN approach is the backpropagation feed forward networks. The main idea is that the data or the signal will be sent forward, and the output and the corresponding error will be calculated. Then, the backpropagation method is used to reduce the error, by readjusting the weights, until the training data is learned. Often, the resulting networks are considered as a black box. The user will only need to provide the training data, and let the networks train themselves. When the neuron i receives the input data from the other neurons $i_1, \dots, i_n$, the propagation function is used to calculate the output value that will be processed using activation function. The propagation function, which is usually a weighted sum, takes the inputs together with the weight, and calculates:

$$n(\vec{w}, \vec{i}) = \sum_{d=0}^n w_d i_d,$$

where $i_0 = 1$. The weights are normally randomized at the beginning of the train-

ing. A non-linear activation function σ_A is used to determine output of the node ($\tilde{g} = \sigma_A(n(\vec{w}, \vec{i}))$). Commonly used activation functions are the sigmoid functions (usually defined as $S(g) = \frac{1}{1+e^{-g}}$). For a binary classification, the sigmoid functions are chosen as activation function because they have good properties: easily differentiable, bounded between zero and one in the output, and monotonically increasing.

Fig. 4.9: Illustration of Backpropagation Feed Forward method

To evaluate the weights, the error should be calculated. Let us consider the case for a single neuron. The calculated error function is denoted as follow:

$$E(\vec{w}) = \frac{1}{2} \sum_j (g_j - \tilde{g}_j)^2.$$

Basically, the error function calculates the sum of the squared difference between the actual value and the predicted value. Then, based on the derivative of the error, $\nabla E(\vec{w})$, the weights will be updated using the following equation:

$$\vec{w} = \vec{w} - \eta \nabla E(\vec{w}).$$

This is called the gradient descent method, and the variable η defines the learning rate. By denoting $\Delta\vec{w} = -\eta\nabla E(\vec{w})$, the equation can then be expressed as $\vec{w} = \vec{w} + \Delta\vec{w}$.

For multiple layers networks, the backpropagation method is used to learn the weights of the networks, given the nodes and the interconnections. This is illustrated in Figure 4.9. First, all the weights will be initialized to some random numbers. Let w_{pq} denotes the weight on the edge from node q to node p . Next, for each training data, we propagate forward from the input to the output layer through the networks. Then, we propagate the errors backward through the network. Since the sigmoid function is differentiable, the value $\Delta w_{pq} = \eta\delta_p i_{pq}$ can be determined, where the index q and p denote the index of the incoming and target node. Hence, the weight for each node can be recalculated as follow:

$$w_{pq} = w_{pq} + \eta\delta_p i_{pq}.$$

The error term (δ) for the k -th node in the output layer, δ_{o_k} , and the l -th node in the hidden layer, δ_{h_l} , can be determined as:

$$\delta_{o_k} = \tilde{g}_{o_k}(1 - \tilde{g}_{o_k})(g_{o_k} - \tilde{g}_{o_k})$$

and

$$\delta_{h_l} = \tilde{g}_{h_l}(1 - \tilde{g}_{h_l}) \sum_{o \in output} w_{oh_l} \delta_o.$$

The learning process is repeated until the termination condition is met, which is usually determined by number of training iterations or epoch, and by the performance accuracy on the validation data.

4.2.4 k -Nearest Neighbors

The k -nearest neighbors or k NN [160] is a non-parametric method which can be used for either classification or regression. The concept of k NN method is illustrated in Figure 4.10. The main idea of this algorithm is that the class of the test data is determined by taking majority vote based on the class of the nearest k training data, *i.e.*, $c_{pred}(t) = f_{maj}(c(t_1), \dots, c(t_k))$, where t_i is the i -th nearest neighbor of the training data t . The "nearest" property is determined by the distance metric. The most commonly used metric is the Euclidean distance. One of the main advantage of this algorithm is the simplicity of the method. However, one of the problems with k -nearest neighbor method is that the classification can be easily affected by the noise in the training data. Also, from the description, we could see that the algorithm makes the decision only when given the new data and hence, it has to keep all the training data, and it will require more memory resources.

Fig. 4.10: Illustration of k -Nearest Neighbors

4.3 Application of Machine Learning for Profiling based Attacks

From the observation, there might be a parallel between machine learning approaches and SCA. To be more precise, we can see the similarity between profiling based attacks and supervised learning, where both methods are trying to build a model from a set of data with known label. In Figure 4.11, it is illustrated what are the differences and similarities between profiling based attacks (represented with TA) and machine learning approaches. In both approaches, the attacker builds a leakage profile by classifying the data into different classes. The main difference is on how they build the profile. The classes for TA are constructed by assuming that each class is based on multivariate Gaussian distribution, while for machine learning, different methods can be applied.

State-of-the-Art [85]

In a general side-channel context, machine learning had been applied for acoustic SCA [154]. Some of the applications of machine learning for acoustic SCA include NN, which were used to learn information being typed on a keyboard [12, 173, 177], or the sound of a printer [15]. However, in the general context of SCA, most of the focus will be on power and EM side-channel instead.

Because of the rather vague classification of what can be considered as machine learning approach, we will only consider works that explicitly use this term. There are some older works that might have used machine learning techniques, but were not specifically addressed as such. For example, [141] performed software code reverse engineering using Kohonen networks [99] to learn different signatures based on the power consumption and electromagnetic analysis of instructions for processors, [89] performed an attack on ECC protected with randomized side-channel

Fig. 4.11: Relations between SCA and Machine Learning

countermeasure [130] by using Hidden Markov Model [142] to model the intermediate values and mask values as probabilistic finite state machines, [105] performed higher-order attack [119] on boolean masking implementation [50] using Gaussian mixture model [26], For alternative distinguisher, [21] proposed using cluster analysis to identify key candidate that leads to maximum cluster separation. Also, [158] proposed using the Principal Component Analysis (PCA) [156] as alternative distinguisher.

In SCA, the TA itself can be considered as a machine learning algorithm. TA is similar to Quadratic Discriminant Analysis (QDA) [75] and for reduced TA, where only the diagonal of the covariance matrix is preserved, it can be considered as Naive

Bayes classifier [145]. Peeters *et al.* [135] stated that they performed some evaluation using an approach based on machine learning and suggested the usage of machine learning to better estimate the statistical distributions of the power consumption in the device. However, no further explanations were provided.

The first paper [82] that used the term machine learning performed a binary classification (with only 2 classes) using the intermediate values as the class label. As an example, they used the 4-th bit of AES Sbox output (either 0 or 1). Though no real attack was presented, it provided an insight on how machine learning (in this case, Least Square Support Vector Machine, or LS-SVM [161]), could be used for classifying intermediate bit values in comparison to TA. This method is a variant of SVM in which the learner solves a set of linear equations instead of convex quadratic programming. The impact of different parameters of LS-SVM were studied to determine their influence on the performance of the learning algorithm. It was shown that the result of LS-SVM are better when the classes can only be separated non-linearly.

In the context of SCA, in the previous work from Lerman *et al* [106], they compared different methods, namely RF [38], SVM [51], and Self Organizing Maps (SOM) [99] for single bit profiling based attack against the classical TA. They also compared different feature selection approaches, and they adopted the feature selections method from machine learning, the minimum redundancy maximum relevance (mRMR) [136] which uses mutual information to determine which subsets of features that maximize the relevance between the features and the class, while minimizing the redundant features (do not include the feature into the subset if it does not add additional information). Also, they compared the feature selection with the dimensionality reduction technique, the PCA, which transformed the training data containing correlated features into new data where the features are uncorrelated

(most of the information will be contained within the first few dimensions).

Later, Heuser and Zohner [78] used the multi-class SVM for dealing with the attack on multi-bit values (HW model), as opposed to the single bit values used in the earlier works. Furthermore, instead of directly predicting the class of the intermediate value, they calculated the probability of the intermediate value belonging to certain class, and they used maximum likelihood principle to determine the key. To compare its performance against classical methods, guessing entropy [159] was used as a metric. They also performed the experiments on noisy data to compare the results of the attacks. It was shown that the SVM-based attacks are more robust to noise compared to conventional TA. This approach was later improved by Bartkewitz and Lemke-Rust [20], with a new multi-class classification strategy, based on the ordering of the classes. Moreover, they also used (linear) SVM as a preprocessing tool for feature selection, similar to Brank *et al.* [34].

Another scenario for the attacks was studied by Lerman *et al.* [109]: the attacker was assumed to have access to the similar device with only several known fixed keys (partial knowledge), and tried to attack the targeted device with unknown keys. The attacker used Partitioning Around Medoids (PAM) [91] to cluster the traces from the similar device, and the known fixed key from this device is used for determining the correct features (points in times) as well as to help in labeling the traces. In the attack phase, the attacker used the clusters to determine which class the traces from the targeted device belong to. Thus, this scenario highlighted the situations where the attacker can only partially profile the device.

Lerman *et al.* [107] then introduced a new method for profiling based attacks, which takes advantage of the time series behavior in the measured traces. The main idea was that different points in time are related or have dependencies with each other, so lazy learning [33] was introduced to capture those dependencies. By taking

into account the dependencies of different points in time, lazy learning, combined with RF or SVM, could achieve good success rate even in the presence of noise.

Another method based on NN was proposed by Martinasek and Zeman [118]. NN were used to classify the AES key, and they could determine this key based on a single trace. The profile was built based on different key candidates. It achieves 85% empirical success rate and 80% theoretical success rate. The authors [117] later proposed to average the power measurements as preprocessing method, which improved the success rate to 96%.

Lerman *et al.* [108] proposed an attack against Rotating S-boxes Masking (RSM) [124] masking countermeasure on AES using machine learning methods. Here, SVM and RF were compared against TA and SA, using the EM traces from DPA Contest V4 [2]. They showed that by using SVM, they only required 26 measurements to retrieve the secret and have faster computational time, even when using non-optimal parameters for training SVM. They claimed that SVM outperformed the classical attacks because the data were not based on Gaussian distribution, an assumption required by TA and SA.

Heyszl *et al.* [80] introduced unsupervised clustering (k-means [74]) for attacking cryptographic exponentiations. This approach allowed the attacker to retrieve the exponents within one execution of ECC implementation on the Field Programmable Gate Array (FPGA). To enhance the attack, multiple EM measurements of single execution were combined, and by using this approach, the attacker did not need to know the precise location of the leakage.

Perin *et al.* [137] proposed a new attack based on clustering techniques that attacked the exponentiation algorithm, implemented with randomization countermeasure [16]. They showed that it only required a single EM trace to recover the whole secret exponents. They used the k-means algorithm as preprocessing tool

for selecting points of interest, and fuzzy k-means [29] for the cluster classification. Statistical techniques were then used to recover the exponents from the cluster classification.

From these works, it can be seen that machine learning can be adopted for SCA. So far, machine learning has been used for both building model *i.e.* to characterize the leakage as well as a distinguisher for the key recovery. Different performance metrics have been considered for comparing these attacks, with success rate and guessing entropy are the most commonly used. Based on the previous works, in general, most of the proposed machine learning based methods are achieving better than or at least similar in terms of some performance metrics as the classical attack, both for profiling and non-profiling case. In general, the machine learning based methods usually achieve better success rate in retrieving the key, or having lower guessing entropy (lower number of key guesses required) compared to classical attacks.

5. MACHINE LEARNING FOR PROFILING SCA

In this chapter, we investigate different machine learning algorithms for profiling attacks. From the previous works, it has been shown that machine learning methods can achieve at least the same or better performance compared to classical attacks. In this work, we standardize the comparison based on the performance metric commonly used in SCA as well as the set of data used for comparison. This is done in order to verify the previous results, since in previous works, the comparisons are usually done under different settings. We investigate under which parameters and experimental settings, the machine learning can truly outperform the classical attack. We choose some of the most commonly used algorithms, and compare their performances with the classical profiling attacks, such as TA. We then highlight what are the trade-offs among different learning algorithms and the classical TA.

After the initial experiments, we then investigate different perspective of using machine learning for SCA. One of the main aspect of SCA is to have an accurate leakage model, so we investigate alternative machine learning method, which can be used to improve the quality of the leakage model. If we consider that there are different interactions among different bit lines during the processing of the device, then there is a non-linear leakage, which can be modeled using high dimensional subspace in SA. Here, we use the machine learning approach to automatize the process of determining the subspace used for the model. This is based on the kernel trick method used in the machine learning algorithm, which does not require explicit

mapping, but rather the observation on the available traces. Based on the experimental results, the proposed method could approximate the leakage better than the previous SA method. Most of this works have been reported in [87].

5.1 Comparison of Machine Learning Methods

In this section, we choose several machine learning algorithms that will be investigated. The learning algorithms will be modified to return the class probability instead of direct classification. In this case, the maximum likelihood approach can be used to rank the key candidates, and thus, provide a fair comparison for different attack methods. Here, we choose the following algorithms for comparison: Support Vector Machine (SVM), Random Forest (RF), Neural Networks (NN) and k-Nearest Neighbors (kNN). Both SVM and RF are chosen because they have been used in different applications before [20, 78, 106, 108] and they showed promising results. NN have also been used before [118], but it was using a direct classification instead. The kNN method has not been used for SCA before, however, the algorithm is chosen because of its simplicity. Then, we apply maximum likelihood on the probabilistic output of the different key hypothesis to obtain the rank of possible key candidates. For comparing the performance of different methods, we use the guessing entropy metric, as described in Section 3.3.

Probabilistic Output

Instead of predicting classes directly, the attacker is more interested in obtaining the probability that some traces belonging to certain classes. By obtaining the probability output, the attacker can then use the established maximum likelihood distinguisher to rank the key candidate.

To apply maximum likelihood for the key distinguisher, the output of SVM is modified so that instead of predicting the class, we calculate the probability of the given data for each class [172]. The probability will be computed using the distance of the data to the separating hyperplane. This method was also used in the previous work [78].

During the key recovery phase, maximum likelihood can then be used as a distinguisher to refer to the most likely correct key candidate. To compute the likelihood of a certain class, given the key hypothesis, $M(k^*)$, we use:

$$k = \underset{k^*}{\operatorname{argmax}} \prod_{i=1}^{N_A} p_{SVM}(M(k^*)|\vec{t}_i), \quad (5.1)$$

where P_{SVM} is the probability output by SVM that $\vec{t}_i$ belongs to class $M(k^*)$.

In order to obtain a probabilistic output when using the RF, the attacker can calculate the probability of each class as follow: First, the attacker calculate the probability $\hat{P}_{RF}(\tilde{C}_{k^*}|\vec{t}_i)$ for each class on each separate tree $t = 1, \dots, T$. Then, the weighted average of the class posterior probabilities over the selected trees can be calculated as follow:

$$P_{RF}(\tilde{C}_{k^*}|\vec{t}_i) = \frac{1}{\sum_{t=1}^T \alpha_t I(t \in S)} \alpha_t \hat{P}_{RF}(\tilde{C}_{k^*}|\vec{t}_i) I(t \in S), \quad (5.2)$$

where S is the set of trees that comprises the prediction and α_t denotes the weight of the tree in the prediction. The maximum likelihood can then be calculated using the same method as in TA and SA.

For NN, in order to obtain a probabilistic output of the class, the attacker use softmax transfer function at the final layer. The softmax function can be seen as a function mapping the output value to a probabilistic output. Assuming that each

of the output nodes represent each class, the probability can then be calculated as follows:

$$P_{NN}(M(k^*)|\vec{t}_i) = \frac{e^{o_{M(k^*)}}}{\sum_{i=1}^N e^{o_i}}, \quad (5.3)$$

where o_i defines the output of node i .

In order to obtain a probabilistic output for kNN, the attacker simply estimates the probability of each class based on the k nearest neighbors. The probability of the new data t_N to belong to class m_{pred} can be computed as:

$$p(m_{pred}|t_N, \mathbf{T}) = \frac{1}{k} I(\vec{m} = m_{pred}), \quad (5.4)$$

where $\vec{m} = \{m(t_1), \dots, m(t_k)\}$ defines the class of the k -nearest neighbors and I is the identity function. This method estimates the probability based on the histogram of the data in the nearest neighbors. This is highlighted in the material for [122]. Alternatively, the density estimation method to construct probability density function based on the data can be used, though it might require a more complex calculation.

5.1.1 Experiments

We have performed our initial experiments on the comparison of different learning algorithms with TA. In these experiments, we intend to see how the machine learning algorithms perform in the SCA scenarios. Here, we describe our target and the experimental setup. For our experiments, we take a straight forward AES implementation running on a standard 8-bit microcontroller implementation, and we exploit the power side-channel leakage from the first round S-box output. For our evaluations, we use a setup consisting of a modified Arduino UNO microcontroller board running with a clock frequency of 16 Mhz and a Lecroy WaveRunner with a sampling rate up to 10 GB/S. In Figure 5.1, we provide a sample of power trace of

one AES encryption obtained from this setup. Here, we use the same device for the profiling and attacking phase because it is easier and more standard for comparison of different attacks. All of the machine learning algorithms are implemented in Matlab.

Fig. 5.1: Power trace of one AES encryption

Feature Selection

For our initial experiments, we use different feature selection methods to choose the informative features: the correlation based feature selection to determine features with the highest correlation with HW model within different clock cycle, and the minimum redundancy maximum relevance (mRMR) method [136]. For this experiment, we use 10k traces for training or profiling phase. For the machine learning algorithms, we use the default parameter(s) for training. Here, we consider the following parameters:

Fig. 5.2: Comparison results using different feature selection methods

- SVM: the kernel (linear), $C(1)$, and γ (not applicable),
- RF: number of trees (100) , number of attributes (square root of number of feature),
- NN: number of hidden layers (10), depth of hidden layers (1), epoch (500), learning rate (0.01),
- kNN: number of neighbors ($k = 100$)

The preliminary experiments are conducted with different number of selected features chosen using correlation and mRMR methods. The first observation that can be drawn is that the learning algorithms are performing almost similarly, and

will eventually converge to guessing entropy 1. However, when the experiments are repeated 30 times with different sets of training data, it shows that for NN and RF with lower number of features, there are some experiments where the guessing entropy does not converge to 1, whereas for other learning algorithms, they converge with more or less similar number of attack traces. It shows that the RF and NN methods are more susceptible to misclassification with fewer number of features.

By taking the average of the performance, we highlight that, with fewer number of features, the performance are slightly worse then with more number of features, which is expected. For example, with 5 features, around 8-10 attack traces are required on average, whereas with 1-2 features, around 12-15 attack traces are required, also not to mention the issue with NN and RF earlier. With higher number of features (> 5), the performance does not improve much and same number of attack traces are required. Hence, we choose 5 features for the subsequent experiments, to minimize the computational time and effort.

In Figure 5.2a - 5.2b, we show the averaged performance using different feature selection methods with 5 features. We observe from the figure that the results from both feature selection methods are almost similar. With correlation method, different learning algorithms converge to guessing entropy 1 after 8 attack traces, whereas, with mRMR, the rate of convergence is faster, but it is not so stable (there are some noise, such as the peak for 3 attack traces),.

In Figure 5.2c - 5.2d, we show additional observations when no feature selection is used, and we simply use all features in one clock cycle (63 features), or all features in one clock cycle with some additional features outside the clock cycle, acting as additional noise (80 features). For these experiments, we can see that the result of machine learning approach is better compared to TA, especially with SVM and RF. This is more or less similar to the observations made in [110] about RF (requiring

fewer number of training data with ability to cope with some irrelevant features), and [34] regarding SVM (using linear kernel and set the weights of irrelevant features to 0). One problem with the random features is that with more features, it requires more computational efforts. Thus, for the rest of the works, we will use correlation based feature selection since it is more stable, and fix the number of features used to 5.

Fig. 5.3: Comparison results using different number of training data

Number of Training Data

For the next experiments, we vary the number of traces used for training. In Figure 5.3, we plot the guessing entropy for different algorithms, trained under different numbers of training data. To ensure the convergence of the results, we repeat each

experiment 30 times with randomly chosen training and testing data. Also, we ensure that there is no overlap between the training and testing data to ensure fair comparison of the performance. In this experiment, the machine learning methods are trained using the same default parameters.

From Figure 5.3, we can observe that the different learning algorithms, again, have almost similar performance. It is also obvious that with more training data, the performance achieved can be better, assuming the additional training data does not add additional noise. However, from the figure, we can also observe that for NN approach, the performance is worse on lower number of training data, whereas the other approaches can achieve better performance comparable to TA, even on fewer number of training data (1000 and 5000 training data).

Another observation drawn from this experiment is regarding the issue on the convergence of guessing entropy, especially on lower number of training data. The guessing entropy for most of the algorithms, judging based on the results of different experiments, will have a high variance before finally converges to 1 with higher number of attack trace. This might due to the reason that, for a lower number of training data, it might not be sufficient to build good profile. Also, since our profiling is done based on HW leakage model, for fewer number of attack traces, it might not be sufficient to reduce the possible key candidates.

Machine Learning Parameters

In this experiment, we investigate the impact on different learning parameters. For NN, we fix the depth of hidden layer to be 1, and for RF, we fix the number of attributes as the square root of number of features. This is usually used as based on rule of thumb for both approaches, as well as to reduce the computation effort. We use the same set of feature chosen in the previous experiment, and compare the

performance on the fixed set of training and validation data. For training we used a set of 10k data, and for validation, we used a separate 1k data. The idea is similar to cross validation. In this case, we test the performance on the validation data set. In this experiment, we show the performance highlighting the impact of parameters.

For SVM, we only consider the RBF and linear kernel for training, as they are the most commonly used kernels. We define the parameter search space as follow: $C \in \{0, 001, 0.01, 0.1, 15, 10, 50, 100\}$ and $\gamma \in \{0.0001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$. The search space is chosen to be not very dense, since it will require a lot of time to evaluate. The same method is then applied to other learning algorithms. For RF, we consider the number of trees grown ($n \in \{1, 5, 10, 50, 100, 500\}$). For kNN, we consider the number of neighbors ($k \in \{1, 5, 10, 50, 100, 200, 250, 300, 500\}$). For NN, we consider the number of hidden layers ($h \in \{3, 5, 10, 20, 50, 100\}$), learning rate ($\alpha \in \{0.0001, 0.0001, 0.001, 0.01, 0.1, 1\}$), and epoch number ($e \in \{100, 200, 500\}$). For activation function, we use log-sigmoid function, in order to obtain 0-1 output value for each class, and calculate the probability output with softmax function.

In Figure 5.4, we only show the result of parameters which eventually converge to guessing entropy 1. For NN, we only include the result with hidden layer 10 for simplicity, and since they are having almost the same results. For most of the learning parameters, the performances are quite similar, with some parameters are performing slightly better. This is understandable since there is not so much noise in the traces used as training data. However, it is clear that some parameters could lead to better results compared to others. Also, in relation with previous experiments, it can be observed that some of the default parameter, like in SVM and RF, can achieve almost optimum results.

Fig. 5.4: Performance results with different parameters

Noisy Environments

To further investigate, we compare the performance of the various methods under noisy environments. We simulate the noise disturbed measurement data, by adding the white Gaussian noise to the captured power traces. We use the *AWGN* (Additive White Gaussian Noise) function of Matlab to add the noise in a post processing step to the traces. Here, we generate a new set of artificial noise of standard deviation σ from the power traces from the microcontroller: 2.5σ (SNR 30 dB) and 8σ (SNR 20 dB). Both training and testing data are assumed to be affected by the same noise in the experimental setting.

First, we investigate the case for low noise scenario, 30dB. We use the parameters

which could achieve better performances in the previous experiment. We note that this is a different set of data, and the performance obtained in this experiment might not be optimal. However, we could also say that, since the noise level is low, the impact on the performance would not differ greatly.

Fig. 5.5: Comparison results of different learning algorithms - noise 30dB

In Figure 5.5, when we simulate 30 dB noise, we can observe that with the lower noise setting, it only has an effect when there is a fewer number of training data, especially with NN, and to lesser extent, RF. Eventually, when the number of training data increases, the performance improves. In the case of NN, better results can be achieved with around 10 and 20 hidden layers and enough training epoch (> 200).

Next, to deal with noisier scenario (20 dB), we perform another parameter selection to highlight the importance of parameters in machine learning algorithms. We use the same set of parameter search space as described earlier in order to look for learning parameter that leads to better performance. Note, that the choice of the search space is still based on heuristic, and hence, the performance might not reflect the most optimum result. Here, we can observe clearly the impact of different

Fig. 5.6: Performance results with different parameters on 20dB noise

parameters on performance results.

Fig. 5.7: Comparison results of different learning algorithms - noise 20dB

In Figure 5.6, we show the difference of the results under different parameters. For the SVM, we observe that in general, the linear kernel is performing better,

whereas, for the RBF kernel, it requires smaller γ values in order to get better results. For smaller values of γ , the decision surface will be smoother, and for this type of data, it behaves more like linear kernel, which justifies the results. For RF and kNN, the impact can also be more observable for different parameters. In this case, higher value of the parameters, the number of trees and neighbors, could help to average out the effect of the noise. In NN, the results suggest that there is more relation between the hidden layer and the performance, compared to the other parameters. Nevertheless, the other parameters need to be optimized as well. Here, we only display the parameters whose performances show the trends of convergence to guessing entropy 1.

To perform comparison on test data, we have conducted 10-fold cross validation in order to determine the optimal parameters, before testing in on the set of attack traces. In Figure 5.7, we show the experimental results for the case of 20 dB noise. Here, it requires 30 attack traces with 40k training data, before it shows the trend for convergence. For most of the classifiers, the algorithms are performing well, despite requiring a higher number of attack traces. However, in this case, we could also argue that the performance of different learning algorithms are more or less the same as classical methods, which does not really highlight the advantage of the machine learning approach. This might be due to the reason that we are adding the white Gaussian noise, which favors the TA.

Other Leakage Model

In this experiment, we try to perform the attacks based on different leakage model. Here, we use the intercalated model [82], where there is only a binary class, odd and even HWs (see Figure 5.8). This model might not be commonly used, but it can still be exploited. For the machine learning algorithms, we perform another

Fig. 5.8: Intercalated model

cross validation to determine the optimal parameters for the attacks, using similar procedure as before.

In Figure 5.9, we provide the result of the experiment. As expected, the TA is not performing well, since in this leakage model, the distribution for each class is no longer based on Gaussian distribution. However, we could see that the machine learning approach, especially SVM and NN can perform well to exploit the leakage model, even though, more attack traces are required in this case. Hence, this example could highlight the case when the machine learning based approaches outperform the classical TA.

5.1.2 Discussion

In this work, we have performed a standardized comparison of different machine learning algorithms, based on the standard metric used in SCA, to provide a valid

Fig. 5.9: Comparison results with intercalated leakage model

comparison and verify the results shown in the previous works. We have chosen some of the commonly used machine learning algorithms, consisting of several algorithms used previously and another prospective algorithm. We then compare their performance based on the guessing entropy metric. We observe that these learning algorithms can be used as alternative for profiling attacks, since they will eventually lead to the recovery of the key, when sufficient training and attack data are provided.

Here, for a successful profiling, the optimal parameters have to be determined. In most cases, this can be achieved using grid search, which is basically brute forcing the parameter search space, and it will require a lot of time. Alternatively, heuristic approaches can be used, but the performance might not be optimal as well and they will require more complex calculations. Hence, finding an optimum parameter is a challenge when dealing with machine learning approach. This might also due to the reason that the parameters are usually data dependent. Also, the profiling time might be a problem for machine learning based methods. For our experiments, both NN and RF will take much longer than SVM, which is in turn longer than both kNN and TA (NN and RF require around $10\times$, SVM around $2-3\times$ times required

for kNN and TA in our experiments).

This can be attributed to how the models are build, where in the case of TA and kNN, they are using the simplest models. On the other hand, in NN, the epoch parameter determines how many iteration done to adjust the weight, and in RF, the number of trees determine the number of trees grown, both of which will consume more time. In SVM, both parameters, the cost and kernel parameter, determine how precise does it need to solve the system of equations. The kNN does not have this problem, but instead, it needs to keep all the training data, which might be problem, when the training data is too large.

These results show that while comparable, the results of the learning algorithms might not always better than TA. This might due to the fact that the device we used has a leakage that resembles HW model, and the noise distribution of the leakage in different classes is close to the normal distribution. Also, even with addition of noise, most of the noise will be Gaussian in nature, which is usually the case in theory and in practice. This goes in favor of TA. Hence, TA might still be the obvious choice when performing standard profiling attacks.

From another point of view, we can also observe that their performance are quite close to TA, which might also be useful when the leakage behavior does not necessarily follow the normal distribution. From the experiment, it can be shown that when the leakage distribution is non-Gaussian, it can still be exploited for a successful key recovery, which does not work for the case of TA. Also, in the another scenario, for example when the chosen features are not necessarily the best ones, we can still observe that the machine learning algorithms, such as SVM or RF, achieve better performance, sometimes even with a fewer number of attack traces required.

5.2 Leakage Modeling

In previous experiments, it was established that machine learning approaches can benefit if the leakage behavior from the device is non-linear. We then investigate the potential application of machine learning approach for other areas in SCA. Normally, in SCA, most of the research focus are placed on finding a good distinguisher. However, as mentioned earlier, there are different areas that need to be considered when conducting SCA, which might be optimized as well. One of the area of interest is to determine a better leakage model. As described earlier, most of the researches are assuming that the leakages are in the form of HW or HD model. From the perspective of the attacker, if he can build more accurate leakage model, then any distinguisher can be used for attack [171].

In this section, we investigate Support Vector Regression (SVR), which can be used as an alternative profiling tool or as a tool for leakage modeling for side-channel analysis. The concept of SVR is based on support vectors like in SVM, but it uses them for soft margins in the regression process instead of a classification like in SVM. Hence, it inherits the good properties of SVM (like non-linear fitting with the kernel tricks) and a more robust behavior due to soft margins. Thus, the SVR method can be used to generalize the non-linear leakage model, which is similar to the the high dimensional subspace for SA.

We adopt the method of SA by replacing the linear regression with SVR during the model building process to describe the deterministic part of the leakage by $\bar{L}^l$, for specific point in time $1 \leq l \leq N_P$. Suppose we have a set of training data in a subspace representation (as it is for SA). The general procedure is quite similar to SA, with the difference in the first step of the learning phase in which the leakage model is determined. The remaining steps are similar to SA, cf. Algorithm 1. First,

Algorithm 1: SVR Analysis for profiling based side-channel analysis

1 Training phase:

2 Capture training data (x, k) with side-channel leakage $\mathbf{L}$ from training device TUE_A

3 Characterize the deterministic part of $\mathbf{L}$:

$$\mathbf{\bar{L}}^l(f(x, k)) = \bar{L}^l(a) = \sum_{i=1}^{N_1} (\alpha_i - \alpha_i^*) K(\vec{a}_i, \vec{a}) + b \quad (4.9)$$

the l^{th} -feature with $\vec{a}$ as subspace (bitwise) representation of $f(x, k)$ and $\vec{a}_i$ as subspace (bitwise) representation of $f(x_i, k)$

5 Solve Eq. (4.9), w/ parameter from Eq. (4.10) - Eq. (4.13) to obtain

$$\bar{\mathbf{L}} = [\bar{L}^1(f(x, k)) \dots \bar{L}^{N_P}(f(x, k))]$$

6 Characterize Noise:

$$\Sigma = \frac{1}{N_2-1} \sum_{i=1}^{N_2} (\vec{T}_{2,i} - \bar{\mathbf{L}}(f(x_{2,i}, k)))(\vec{T}_{2,i} - \bar{\mathbf{L}}(f(x_{2,i}, k)))^T$$

8 Attack phase:

9 Reveal secret key by using the leakage model of the training phase and Maximum Likelihood

$$\mathbf{10} \quad k = \operatorname{argmin}_{k^*} \sum_{i=1}^{N_A} (z_i^T \Sigma^{-1} z_i),$$

$$\mathbf{11} \quad \text{where } z_i = \vec{T}_i - \bar{\mathbf{L}}(f(\tilde{x}_i, k^*)).$$

the attacker measures N traces, N_1 traces for model building and N_2 traces for noise covariance estimation. The leakage model is then estimated using SVR. Then, the attacker measures N_A traces from the targeted device, and uses maximum likelihood to determine the key, using the profile (the leakage model and the noise covariance matrix).

5.2.1 Experiments

SVR has been proposed earlier as a technique to build the leakage model for SCA. The leakage model is an integral part of SCA flow and if it is wrongly estimated then it has a negative impact of efficiency of the attacks. We then use the introduced method based on SVR to estimate the model.

For the experiments, we use the same setup as in the previous experiments (refer

to Section 4.2), In order to rate the efficiency of the attacks and hence, the quality of the model (since all attacks are using the same distinguisher), we again use the guessing entropy (GE) as a comparison metric. To recall, this metric reflects the rank of the correct key candidate after performing the analysis. If the correct key candidate is positioned on rank 1, the attack is considered to be successful and the correct key can be revealed. All SVM methods are implemented in Matlab by using LibSVM [45] for the machine learning algorithms. In the first step, we analyze the leakage modeling efficiency of SVR. To do this, we compare the leakage model built using SVR, SA and the standard HW model on a common set of traces.

Fig. 5.10: Evaluating quality of leakage model with CPA

Evaluating the Quality of Leakage Modeling Using CPA

To compare the quality of the model, we use one of the most basic and widely used attack *i.e.* CPA. We collect a set of 50k traces from an AVR microcontroller running AES implementation. These traces are used to estimate model using SA (*i.e.* linear regression) with 9 (basic) as well as 256 (maximum) dimensions. Then, we repeat the same leakage modeling step on the same traces using SVR technique.

Note that the leakage models for SA9, SA256 and SVR are estimated at each time sample independently. Finally, we launch 4 CPA attacks using the following models: HW, SA with 9-dimensions (SA9), SA with 256-dimensions (SA256) and SVR. The results are shown in Figure 5.10.

It is evident that SVR-based model clearly outperforms SA9 and HW model. In fact, SVR is precise enough to approach SA256 model i.e. the leakage model with the highest complexity. As microcontrollers are known to have leakage model that approximates HW model, the difference between HW, SA9 and SVR is not drastic. Such analysis can be very useful in hardware implementations where the leakage model can be very different from HW model. The SVR model itself might be further enhanced to match SA256.

In the figure, there are some higher correlation peaks which are noticeable at other time points. To analyze, we first calculate the SNR using leakage detection method, NICV, which is highlighted in Figure 5.11a. From the figure, there is no such observable peak detected at those time points. The initial assumption is that there are non-linear leakages at those points. In the Figure 5.10a, at point 3622, there is a peak which shows similar trend for the stochastic approach, which highlight that there might be leakage that is captured by SA256, but not in SA9. This is also not captured by SNR using NICV since it only detects linear leakage. Another possibility is that there is overfitting of the data, that the model actually captures some noise as data dependent leakage.

Since we have established that SVR is a good technique for leakage modeling, in the following we analyze SVR in a profiled setting.

(a) SNR using Leakage Detection (NICV) (b) Points selected using different feature selection methods

Fig. 5.11: Leakage detection and points selection

Fig. 5.12: Guessing entropy of various training methods (5 features)

Evaluating SVR in a Profiled Setting

In this section, we perform a basic comparison of SVR with other profiling based methods. We want to show that in basic profiling setting, SVR could also achieve

comparable performance with the other profiling methods. First we collect 50k power traces, under the same experimental setting. Here, we also include the classification based profiling attacks to highlight the comparison with regression based methods. For classical profiling methods, we use TA and SA to represent classification and regression based profiling attack respectively. For machine learning based methods, we also compare SVM and SVR with linear kernel (SVM-Linear, SVR-Linear) as well as non-linear RBF kernel (SVM-RBF, SVR-RBF). The choice of linear kernel is to highlight the performance in comparison to kernel trick, in this case with the commonly used RBF kernel. For this experiment, we do not use the optimized parameters, instead we use the heuristic parameters based on previous works (Eq. (4.10) - Eq. (4.13)).

We first set the number of profiling traces to 25k, which is half the number of measurements, and we fix this as the training set. From the previous experiments, we note that this number of experiments is sufficient for building a good model. The remaining 25k are left for evaluation of the attack. Then, we vary the number of the traces for the profiling and attack phase, and we observe the performance of each method. For classification based methods, TA and SVM, we use HW leakage model ($IV = HW(S(x, k))$). For regression based methods, SA and SVR, we use bitwise model of the intermediate value with subspace of 9-dimensions ($IV = [1 \ \varphi_1(S(x, k)) \dots \varphi_8(S(x, k))]$), and 8-dimensions respectively ($IV = [\varphi_1(S(x, k)) \dots \varphi_8(S(x, k))]$). Here, the function $\varphi_u(p)$ maps p to the u^{th} -bit of p .

To perform the attack, we first select relevant features for model building. For TA and SVM, the traces are used to determine the class of the intermediate values, while for SA and SVR, the value of the traces are estimated by the subspace representation of the intermediate value. Then, in the attack phase, the model is verified using the set of attack traces. Based on our previous experiments (Section 5.1.1), we

observe that 5 features are sufficient for a successful attack. To recall, in previous experiments, we obtained different point in times selected using different feature selection methods. To illustrate, in Figure 5.11b we include the one trace obtained, and different selected points. Note in this case, we have 2 SNR based feature selection, the SNR computed using the stochastic model (SA), which is model based, or based on using NICV, which is more generic. In this experiment, we only consider the features chosen using correlation based (for classification based profiling, which is based on previous experiments) and SNR based (for regression based profiling). In the figure, we could see that the point 3545 has been selected by different methods (which is also consistent with Figure 5.10a), where the other selected points are more scattered depending on which method used.

The results of the evaluation with various amounts of attack traces are depicted in Figure 5.12. All the methods are performing equally good on collected microcontroller traces after using more than 10k profiling traces, less than 10 traces are needed for the attack phase to get a GE of 1. Due to our experimental results, it seems that the classification based methods (TA and SVM) perform less efficiently than the regression based methods (SA and SVR) for a training set below 5k traces. However, this might be due to fact that the HW model might not be an optimal model for classification based methods. Another alternative model is to use the value of S-box value output ($IV = S(x, k)$), but it will require much more profiling traces, which may not be efficient for this case.

Here, we can see that the SVR method is still comparable with the classical profiling attack, cf. Figure 5.12.

Evaluation on Noisy Traces

In the next experiment, we verify the robustness of the methods to a noise disturbed measurement data. To obtain noisy traces, we simulate the noise by adding white Gaussian noise to the captured power traces in order to investigate the performance of profiling based attacks in the presence of noise, similar as in previous experiment.

Again, we generate from our initial set of 50K power traces two additional sets with an artificial noise with standard deviation σ of the μC power traces: 2.5σ (SNR 30 dB) and 8σ (SNR 20 dB). This means that for both phases, the training and the attack phase, we are working on noisy data drawn from approximately the same distribution. From previous results, we observe that the guessing entropy will eventually converge after a fixed number of profiling traces. Hence, in this work, we fix the training set to 40k traces, and use the remaining 10k for evaluation of the attack accuracy.

Fig. 5.13: Guessing entropy of different noise level

In the early stage, when performing the attack with SVM using a HW class model, we notice a worse performance compared to the results obtained previously. We speculate that it might be due to the class imbalance of the intermediate values

in our case, since in previous work, the measurements are done to ensure that the resulting traces have a balanced number of different classes.

Since the plaintexts are uniformly distributed, the resulting HW is binomially distributed, and for 8-bits value, HW 4 ± 1 is dominating and HW 0 or 8 are only represented by a few. If the classes are not clearly separable, due to the noises, SVM will simply assign the class of each of the traces into the majority value. Hence, we adjust the data so that different classes will have approximately the same number of data. We reduce the number of classes into 3 ($IV < 4$, $IV = 4$, and $IV > 4$), and we obtain much better classification as opposed to when using a HW class.

From Figure 5.13, we can see that in general, both regression based methods are performing better than the classification based methods for different noise level. From our analysis results, the SVR (both linear and RBF kernel) can also perform well in the presence of noise. In this case, both regression based methods can perform better than the classification based methods. This could be due to the nature of the additional noise, which can be better modeled by the covariance matrix in the regression based methods. However, this observation only highlight that both SVR and SA are comparable, but it does not highlight the advantage of SVR based method.

Evaluation on Different Subspaces

In this section, we investigate if using higher dimensional subspaces could achieve better performance. As mentioned before in Section 3.2.2, a higher dimensional subspace can also be used if it is assumed that there is an inter-bit dependent leakage. In the 9-dimensional subspace, it is assumed that the leakage from each bit is independent from each other. However, the leakage of different bits can be affected by the others, for example cross-talk (phenomenon where switching activity

in a wire affects the neighboring signal [93]) during the bit transition. Hence, we would like to observe the result on different subspaces. Thus, in this section, we will only consider the regression based methods.

As mentioned earlier, the equation for linear regression in 9-dimensional subspace can be written as:

$$\bar{L}(a) = \sum_{j=1}^8 a_j \cdot \beta_j,$$

where $a_0 = 1$, $a_{j \neq 0}$ is the bitwise intermediate value (bit-wise value of S-box output), and β_j is the linear coefficient. We then consider a higher dimensional subspace $\mathcal{F}_u$, where $u \in \{9, 37, 93, 163, 219, 247, 255, 256\}$. For example, in 37-dimensional subspace, it is assumed that there is also an interaction between two bit-lines, and it can be written as:

$$\bar{L}(a) = \sum_{j=1}^8 a_j \cdot \beta_j + \sum_{1 \leq j_1 < j_2 \leq 8} a_{j_1} \cdot a_{j_2} \cdot \beta_{j_1 j_2}.$$

Thus, we perform the experiment using all the different subspaces. We denote each method according to its dimension, for example SA37 refers to SA on 37-dimensional subspace.

First, we use the original data, and we try on different subspaces for SA. Again, for this experiment (and the remaining experiment), we set 40k for training and the remaining data for evaluation. For SVR, we use the original 8-bit subspace as used before, and we check if the kernel method could determine the suitable mapping. To avoid overfitting, for our experiments, we repeat $30 \times$ with different sets of training and test traces for each experiment and take the average value. In Figure 5.14, we show results from the attack on the original data. We can see that SVR could achieve similar performance as SA with higher dimensional subspace, which is better

Fig. 5.14: Comparison of different subspaces

than the original SA. As the attack works with as low as 2 traces, some statistical glitches might be present in Figure 5.14.

Next, we simulate the traces with a higher dimensional model. First, we approximate the linear coefficient from the original data, and we generate additional higher dimensional coefficients randomly. We simulate 2 cases, where the additional coefficients are approximately $1\times$ and $5\times$ the coefficient of the original data. This is chosen to highlight the case where the additional coefficient contributed equally with the original coefficient and additional coefficient have much more impact over the original coefficient respectively. We simulate the coefficient up to 256-dimension subspace.

To generate the traces, we use the following method [65]: First, we obtain the deterministic part based on the known intermediate value, and we add the additional coefficients. Then, based on the estimation of the noise behavior [57] of the original data, we add noise to the deterministic part with the noise behavior similar to the original data. In Figure 5.15, we show the result of the experiment on the simulated data when using different subspaces.

Fig. 5.15: Guessing entropy on simulated higher order data

Here, the experimental results show that SVR can generalize the leakage behavior with a higher dimensional sub-space. Unlike in SA, for SVR even when using the 8-bit dimensional model, it can achieve the same performance as SA with a higher dimensional subspace. This may be attributed to the kernel trick, which allows the mapping of the data to a higher dimension.

5.2.2 Discussion

We have proposed SVR as an alternative method for leakage modeling, which could be automated to higher dimensional leakage. From these experiments, we can see that when decent level of noise is present, the SVR based profiling attack (with linear and RBF kernel) still performs quite well. One of the possible justifications is that the loss function used by SVR (ϵ -insensitive loss) is quite robust to noise, since it allows some deviations in the prediction, and hence, it might not be affected that much by the noise, cf. Figure 5.13b. Overall, we can see that the performance of SVR is comparable to the other classical attacks, even in the presence of noise.

One of the main advantage of SVR is that its kernel method (for example RBF)

can be used to generalize the leakage model, namely, it can be used to model a non-linear function. From the experiments with the additional simulated coefficient, it is clearly depicted in Figure 5.13a to Figure 5.13b that SVR with RBF kernel shows a good capability for generalizing the leakage behavior. Thus, in this case, this approach allows us to estimate the leakage without necessarily determining which subspace to use. So, we can see that SVR is better than the standard SA for the case where the device has a higher dimensional leakage.

However, this could also be a problem for RBF kernel, when it tries to map the data to a higher dimension. When the noise level is very high, it might try to mistakenly fit the data, and thus, it could lead into the case of overfitting. In our previous experiments, the noise level is quite decent, and thus the performance remains fine. Also, this could lead to high computational complexity for the model estimation. Hence, further investigations are needed to determine how sensitive the RBF kernel is towards the noise.

6. BASICS OF HARDWARE TROJANS

In the tale of Trojan war, the Greeks made a wooden horse and sent it as a gift to the Trojans. Unknowingly, they hid several men inside, and at night, they came out and opened the gate for the army of Greece to attack the city of Troys. Hence, the term trojan is now commonly associated with a trick that causes the target to unintentionally invite the malicious party into a secure protected place. In modern days, the term trojan is commonly used to refer to software trojan, which is, as the name might imply, a malicious program that presents itself as harmless and tricks the innocent user to run or install it, and performs malicious actions on the user's device.

In the context of hardware, with the growth of the semiconductor technologies, the chips are getting smaller and more complex, and also the resources required for the chip manufacturing are increasing. Hence, nowadays, most of the chip manufacturing tasks are being delegated or outsourced to other companies from around the world, rather than being done solely by the company that sells those chips. However, this opens up a new possibility for malicious parties to modify or tamper with the design of the chip, since it is hard to check and control each individual chip being manufactured in different foundries during the manufacturing process. Even if the device is supposed to be secure by design, a small alteration during the manufacturing process can easily compromise the integrity of the chip. This maliciously alteration introduces a new concept, which is now commonly referred as hardware

trojans.

In the side-channel community, hardware trojan could sometimes be considered as physical attacks [62], as it could be used to enhance side-channel attacks (increase SNR of the leakage) and to conduct fault attacks (inject more precise and controllable faults). Alternatively, hardware trojan detection using the side-channel leakage is usually considered as the application of the constructive side-channel analysis.

6.1 Introduction to Hardware Trojans

Hardware trojans have received a lot of attentions from the research community. Many researches have been done on both trojan designs and detections. For trojan designs and detections, there has been an arm's race between the defenders and the attackers, and consequently the research can be divided into two main aspects: how to stealthily insert a hardware trojan (attacker's or designer's perspective)? and how to successfully and efficiently detect the trojan (defender's or evaluator's perspective)? However, until now, there is no standardized metric yet for evaluating the trojan designs and detection methods in general. Also, there is a lack of relations between the design and the detection methods. For example, a new detection method is usually verified against generic trojan or a new trojan design is usually not verified against existing detection methods. As such, it might be hard to compare performance of the new detection methods, or the new trojan designs.

From the perspective of the designer, the main objective is to insert malicious circuitry or to make any modification to the chip that cannot be easily detected, while the main goal for the trojan evaluator is to ensure that no malicious changes or modifications have been made that compromise the integrity of the chip. In the taxonomy of trojan designs [162], there are three main categories for classifying hard-

ware trojans, namely physical characteristics, activation characteristics and action characteristics. Physical mainly refers to the trojans size (what are the modifications due to the trojans?), the trojan distribution in the circuit (are they concentrated on specific part or distributed across the circuit?), the layout of the circuit (does it change because of the trojans?) and the type of the trojans: adding or deleting gates (functional) or modification of existing wires or logic (parametric). Activation refers to how the trojans are activated, in particular, whether they are always on (no trigger needed), externally triggered (*i.e.*, keyboard. switch, component outputs) or internally triggered (*i.e.*, counter/timing, temperature, frequency) trojans. Action characteristics refer to what the trojans do, such as: trojans that leak information, disable the system (or *kill-switch*), or modify the functionality or specification of the circuit.

When implemented correctly, hardware trojans can be very hard to detect using conventional detection methods, due to the complexity of a typical chip layout. Hardware trojans have some similarities to their software counterparts, however, there are some main differences. First, the flexibility of hardware trojans is more restricted, in the sense that once deployed, no modification can be made by the designer. However, it is more dangerous, once the chip is infected, it is deemed useless. Unlike software trojans, these hardware trojans cannot be easily removed since it is physically incorporated into the target devices. Hardware trojans can also be used to attack different range of targets, for example, attacking cryptographic chip, where there is no software involved. Also, hardware trojans can remain dormant or unnoticed for a very long time, thus making it harder to detect, due to high cost for maintaining run-time monitoring.

Many of potential malicious actions which can be performed using the hardware trojans are only limited by the resources of the trojan designer. As described earlier,

they can be used to disrupt a system or leak confidential information, which on itself can be a threat especially to financial or defense applications. There have been a few reported attacks that might be attributed to hardware trojans: a backdoor introduced by Israel into Syrian radar, which allowed them to bomb a Syrian military facility without being detected [4], or the possibility that National Security Agency (NSA) inserted a backdoor in Crypto AG's encryption devices which allowed them to wiretap the communications among the important figures on different countries [116].

6.2 Trojan Detection Methods

A wide variety of different approaches have been proposed for hardware trojan detections. For a reference, a survey by [162] has listed down the classification of the trojan and detection methods from the earlier publications. Trojan detections can be mainly classified into destructive and non-destructive methods [44]. In destructive methods (such as reverse engineering), the process will likely destroy the tested chip (it may end up with a verified yet unusable chip). Usually this method is used only to check when the chip is used as a reference for the genuine chip or "golden chip". After profiling has been done, the chip is reverse engineered to ensure that it is indeed genuine. However, this process might cost a lot and requires a lot of expertises.

On the other hand, the non-destructive methods have a lower risk of destroying the chip. It can be further classified into invasive and non-invasive methods. Invasive methods refer to approaches that add extra logic to the chip to assist in trojan detection, such as scan-chain or sensors. For example, in [175], they proposed on-chip structure including a ring oscillator network (RON), which is distributed across

the circuit, in which any changes will affect the frequency of the ring oscillator and thus revealing the trojan. It can also be used for prevention at the design stage. For example, in [10, 17, 43], they presented a different modes of operation, in order to make it harder for the adversary to insert the trojan. In [126, 127], they applied the method from coding theory to transform the circuit in order to protect it from the trojan insertion, by encoding the internal state such that no information about the original state was revealed. Hence, without the knowledge of the original state, it is hard to insert a trojan without disrupting the structure. On the other hand, non-invasive methods do not modify the original design and trojans are detected by monitoring the behavior of the chip, such as the physical behavior or output of the chip.

The non-invasive methods used can be mainly classified into side-channel based approaches and logic (functional) testing based approaches. In side-channel based methods, the physical leakage will be observed, to see if there is any anomaly, with respect to a golden chip or other original reference. This is a problem by itself because the golden chip or original reference might not always be available. Another problem is that the impact of the trojans can be easily hidden within the process variation or random noise of the device, especially if the trojans are inserted carefully. A skillful adversary could cleverly insert the trojans such that any physical changes caused by the trojans will be hidden within those random noises.

In logic testing based methods, random test vectors are used as inputs to the suspected chip in order to trigger the activation of the trojans. One of the main concern is that the trojans are not necessarily activated with trigger pattern, so its application is quite restricted on trigger based trojans. Also, the size of the patterns for testing tends to grow exponentially with the size of the inputs, thus making it less efficient for testing all the patterns. However, it has advantage in the sense that

it is not affected by the process variation even if the trojan size is small and carefully inserted.

Side-channel based Trojan Detection Methods

The first trojan detection method was proposed by Agrawal *et al.* [7]. They proposed the generation of the fingerprint for the chip based on its power characteristic. This fingerprint, or reference, was constructed by transforming the power traces measured from the genuine chip using Karhunen-Loève transform (which is similar to Principle Component Analysis/PCA). It was verified that in the trojan infected chips, the power characteristics will behave differently from the reference, as they carry additional leakages, which are due to the contribution of the trojans, and upon transformation, the differences would be magnified and could be clearly observable. Different sizes of trojans could then be detected using this method. However, it was quite prone to noise and process variation. If there was large noise or variation in the measurement, and the trojan was small enough, it would be hidden and this method would not be able to detect trojan properly.

Later, Kutzner *et al.* [104] evaluated the fingerprint based trojan detection method on a fully functional lightweight trojan embedded on PRESENT block cipher, carefully designed and placed. They showed that in real measurements, due to noise and process variation, the trojan would remain undetectable by the method. Here, they introduced a new detection method, called Difference of Probability Distribution (DPD). It was also based on the difference in the side-channel characteristics. It exploited statistical properties of the probability distribution functions, built from side-channel measurements of an investigated circuit design. In this method, they constructed the probability distribution from multiple time samples, over the encryption process. Then, they calculated and combined the difference of cumula-

tive distribution between each pair of tested design. They showed that the difference between trojan infected and genuine design is significantly more observable as opposed to between genuine designs. Their method can also be used to identify which time samples where the trojans are active, by observing which time samples that have a suspiciously large difference in the calculated cumulative distribution. Here, it was assumed that the trojans were only inserted in some of the device under test, and the rest were genuine.

To deal with process variation, Du *et al.* [58] introduced the self-referencing method. This is commonly used with some randomly generated vector, but the emphasize is still on utilizing the side-channel leakage. The basic idea was as follow: first, the evaluator divided the chip into several regions (partition region method), and then, generated the random test vector that could maximize the chance of triggering the trojans, and compared the power characteristic of different regions. This comparison was used to reduce the effect of the process variation. It was shown that the proposed method was robust enough in detecting small trojans, even under sufficiently large process variation.

Next, Narasimhan *et al.* [123] proposed the Temporal Self-Referencing (TeSR) approach, which compared the power profile at two different time windows, in order to get rid of the process variation. Furthermore, this approach did not require the existence of golden chip, and could also be used for detecting large sequential trojans. The justification is that, for trojan free chip, if they are going through the same states, their power profiles will be similar, but in case of trojan infected chip, there will be differences due to the trojan activity or the activation of the trojans.

Later, Li *et al.* [111] proposed another detection method that did not require the presence of golden chip and only relied on the path delay. The basic idea is to insert sensor to the chip design to capture the chip characteristic, and then, generate 2

delay patterns (delay predicted by the sensor and actual delay) to be correlated. The correlation determined if trojans were present in the chip.

A non-destructive side-channel analysis based reverse engineering (SCARE) was proposed by Wang *et al.* [169], which could be used for both cases where golden design might or might not be available. For trojan detections, they used side-channel to extract structural and functional information from the chip. In this work, they chose the current based side-channel approach, and adopted the self-referencing approach.

Lately, Ngo *et al.* [128] proposed the theoretical method, to determine the false positive rate for the trojan detection, taking into account the process variation and properties of the trojan. The side-channel leakage was modeled as Gaussian distribution. It was assumed that the trojans will shift the average leakage of the chip. It allowed the estimation of the false positive (and negative) based on the difference of mean between the genuine and the trojan infected device. To reduce the false positive rate, the thresholding technique (only keeping the features with higher absolute difference) was used to avoid the misclassification caused by the process variation. The proposed approach in this work was similar to TA in profiling based attacks, in which there was only one class, the genuine class, and the device will be deemed suspicious when its probability falls outside user defined boundary.

Template based Trojan Detection

One of the main interest in trojan detection is to determine the capability of a detection method under the impact of noise or process variation. In previous work [128], the effects of trojans on the chip or device are modeled and formulated. The main assumption is that the side-channel leakage is following the (univariate or multivariate) normal distribution. It is described that the side-channel leakage of processed

Fig. 6.1: Effect of trojan on side-channel leakage

intermediate value is deterministic, depending on the processed data, based on an unknown leakage function (see Section 3). Assuming all other experimental parameters are kept constant, the side-channel leakage will still be sensitive to process variation. This variation can be modeled as a random noise following a normal distribution.

Following this assumption, consider the case when the trojans are being inserted into the circuits. Since the trojans are usually denoted as additional circuitries (it can also be deletion or substitution of circuitries, but the main principle remains the same) small enough to avoid conventional detection method, the variation caused will be negligible relative to the process variation or noise and the main effect is the change in the leakage, which can be modeled as a constant offset shift from the genuine circuit in the side-channel leakage. Thus, if the leakage of the genuine chips are modeled as a normal distribution with mean μ with variance σ^2 , the leakage impacted by the trojans can be modeled similarly with $\mu + z$ with variance σ^2 , where z denotes the offset. This concept is highlighted in Figure 6.1.

From the figure, the decision boundary is determined as the hyperplane in the middle of the two means. This decision boundary ($\tilde{\mu} = \mu + \frac{z}{2}$) can be considered

as optimal, balancing the false positive and false negative rate provided that the normality assumption holds. The false positive (FP), and similarly, the false negative (FN) can be considered as the region where the data falls outside the decision boundary and as a result, is classified under a wrong class. To be more precise, false negative refers to the case where the outliers were deemed as good or genuine data, whereas false positive refers to false alarm.

The FP can then be calculated using the following formula for the case of normal distribution:

$$P_{FN} = P_{FP} = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\tilde{\mu}} e^{\frac{(x-(\mu+z))^2}{2\sigma^2}} dx \quad (6.1)$$

This provides a theoretical estimation on how likely the misclassification will be, given the effect of the trojan and the process variation.

To detect the trojan, TA based method is used, with only two classes available, genuine and infected class. However, for most of the time, since the mean for the trojan infected chips is unknown, the profiling of the infected chips might not be possible. Hence, for classification of the device, the decision boundary is constructed based on user defined threshold. The device will be classified as genuine or trojan infected, if it is above or below certain threshold.

This method can be used to build different models for different FPGAs. To tackle the problem with the process variation, two methods are used to preprocess the traces: the sum of absolute difference and the threshold techniques. For the first approach, the sum of absolute difference, the mean of genuine traces from N FPGAs ($\vec{t}_g = \frac{1}{N} \sum_{i=1}^N \vec{t}_i$) is used for reference. Then, the sum of absolute difference will be performed for all traces ($S_i = \sum_{no_samples} |\vec{t}_i - \vec{t}_g|$). For the threshold technique, instead of directly summing up all features, a heuristic approach is first used to choose relevant features used for summation.

- First, the matrix $\mathbf{D}_g$ and matrix $\mathbf{D}_t$ are computed. The former defines the absolute difference of the genuine traces with their mean, while the latter defines the absolute difference of tested traces with the mean of genuine trace.
- For each trace $\vec{t}_i$ from FPGA i , for each j -th feature, compare $D_{t_{i,j}}$ with $\max(D_{g_j})$ (maximum value of $\mathbf{D}_g$ in column j).
- If $D_{t_{i,j}} < \max(D_{g_j})$, set $D_{t_{i,j}} = 0$, otherwise, keep the value.
- If the value of $D_{t_{i,j}}$ is 0 for all j , then FPGA i is classified as trojan free.
- Otherwise, redo the sum of all difference method on the updated $\vec{D}_{t_i}$ and updated $\mathbf{D}_g$ (set all the columns in $\mathbf{D}_g$ to 0 where $\vec{D}_{t_i}$ are 0).

In this way, the irrelevant features can be discarded and hence, removing one of the source of noise.

6.3 Open Problems in Trojan Detection

From the previous works, some problems have been addressed that can occur in the trojan detection. To detect the presence of the trojan, most methods try to observe if there is any anomaly the chip. Common detection idea is to generate random test patterns hoping to trigger the activation of the trojan or to observe the side-channel behavior hoping to see if there is any suspicious behavior which might be attributed to the trojan. There have been some problems with those approaches. Such problems can be in the form of: the large size of input test vector needed to check in order to determine the triggering condition, specific triggering pattern that is required for triggering condition, as well as the process variation of tested device hiding any anomaly that might reveal the trojan. Also, for most of the methods,

Fig. 6.2: Example of hardware trojan that induces fault on AES [24]

the prior profiling of the chip will require the presence of a "golden chip", which is a chip that is trojan free.

Another problem for the trojan detection is that most of the methods are dependent on the type of the trojan inserted, which might be unknown to the chip designer. From a trojan designs perspective, hardware trojan can be presented in a wide variety of designs. This is one of the issues when one is designing trojan detection methods, due to the unknown nature and behavior of the trojan. Since there can be many varieties of trojan, it is hard to model or cover them all and hence, there is no method that could cover the detection of all types of trojans. Most of the described previous works are only restricted to detecting the trojans under some assumptions. Also, in most scenario, one could argue that if the trojans are carefully inserted by the attacker, they can avoid being detected until it is too

late, with a high probability.

However, if the trojans are to be restricted only in the context of information security, then the types of the trojans can be restricted as well. Most of the trojan designs can be assumed to aim at the recovery of the secret information or the key material. Hence, the design of the trojans used are typically the ones that leak the secret through some output channels or through some side-channel leakages, or enhance physical attacks, such as, increasing the quality of the leakages, bypassing the side-channel countermeasures or injecting fault in the intermediate round of encryption or decryption process to enable fault attacks. Typically, most of these trojans will contribute some changes in the physical or side-channel behavior.

To give an example, let us look at a trojan that leak the information through physical leakage, the trojan Side-Channel (TSC) [90]. The idea is to hide the secret information through an "encryption" function, and leak it through side-channel or physical leakage. For evaluator who does not know this function, the leakage will appear as random noise, while the trojan designer, having the full knowledge of the function, can "decrypt" the secret information. An earliest example of such trojan is shown by Lin *et al.* [113], where they use the power side-channel to leak the secret key, which is masked using some random stream from linear feedback shift register (LFSR) to avoid detection by conventional side-channel analysis methods. To give another example, we look at the trojan that enhance fault attacks. Bhasin *et al.* [24] designed the trojan to induce fault attacks on AES. Basically, they reproduced Piret's attack [138] using hardware trojans. The schematic of this trojan is highlighted in Figure 6.2. In both examples, it can be safely assumed that these trojans will require the insertion of additional circuitry (either for leaking data or for injecting faults), and in most cases, it means that the trojan will affect the physical behavior, such as the power consumption or EM emanation.

7. MACHINE LEARNING FOR SIDE-CHANNEL BASED TROJAN DETECTIONS

In this work, we investigate the detectability of the trojan by observing the EM leakage and adopt a machine learning approach for single class classification, as an alternative tool for trojan detection. The general concept of this learning algorithm is that it requires only the representation of the genuine data to construct a classification model, which is suitable for the trojan detection, due to the unknown behavior of the trojan. We show that based on the experimental results, the method can achieve good overall accuracy for trojan detection, even in the presence of noise and for smaller trojan size.

Using the side-channel leakage of the genuine device for training, the profile can be constructed and used for determining whether the tested device is suspicious or not. Another possible scenario with this method is that the given data for training might already contain trojans. Hence in this case, the algorithm can be trained and tested using the data measured from the same set of devices, similar to the unsupervised learning concept, and return the prediction for each device, either genuine or infected. In this scenario, the method can be used for the case when there is no golden reference available.

7.1 Proposed Method based on One-class SVM

For the machine learning approach, we use the one-class SVM as described earlier in Chapter 4.2.1 to build a profile based on the training data. This method is commonly used for the anomalies or outliers detection. Thus, it can also be used for trojan detection by modeling the trojans as outliers. The argument is that in the machine learning problem, it is easy to gather the training data, however, abnormal data or outliers might be expensive or impossible to collect. The reason is that it is hard to simulate all possible cases for the abnormality of the data. Thus, this can be a problem if the outliers detection problem is to be solved using the conventional classification methods. The single class machine learning methods is then used to generalize the distribution density of the training data. So rather than separating the training data from the outliers, the learning algorithm defines the structure of the training data. The concept described is similar to the trojan detection problems. The training data used can be obtained from the verified genuine chip or golden model (supervised case), or the set of chip under test (unsupervised case). The one-class SVM can also utilize the kernel method, just like in normal SVM method.

For the RBF kernel, the shape of the decision boundary is determined using the γ variable. Smaller value of γ will tend to produce a smoother boundary but with a higher chance of underfitting, while a higher value of γ will tend to produce more complex boundary with higher chance of overfitting. Based on the description of parameter ν , we can see that the decision boundary is created such that approximately ν of training data is considered as outliers, which could be defined as false positive. Hence, assuming that the training and testing data are drawn from almost identical distribution, intuitively, the attacker could adjust the parameter $\nu \in (0, 1)$, based on expected ratio of genuine and trojan infected. If the value of ν is too small,

most of the data will be classified as genuine, whereas if ν is too large, most of the data will be classified as trojan infected.

Related Works

Previously, one-class SVM method had been used by Gustin *et al.* [73] to detect modification in the IP core. The power signature was constructed from the genuine IP, and the validation was done by comparing the generated data with the reference signature. It was used to distinguish the implementation of PRESENT lightweight cipher from other lightweight ciphers.

Later, Bao *et al.* [18] proposed a reverse engineering approach to identify trojan-free IC using one-class SVM approach. They extracted several features from the high resolution image obtained using scanning electron microscope (SEM) on the exploited layer of the IC and used them to train the classifier. The constructed model was then used to classify whether the IC is trojan infected or not.

7.2 Experiments

In this section, we conduct the experiments to show the construction of the decision boundary based on single class machine learning approach. In Figure 7.1, we give as example the learning results from the simulated bivariate normal distribution. Here, the distribution of the training data are shown, the green color denotes the data considered as genuine, whereas the red color denotes the suspicious data. It can be observed that the RBF kernel can approximate the distribution of the data better compared to linear kernel.

For the next experiment, we consider two scenarios. The first experiment is done on the simulated data. This experiment is conducted to investigate how the

Fig. 7.1: Example of classification using one-class SVM

Fig. 7.2: Illustration of decision boundary with linear kernel

learning parameters affect the classification results using one-class SVM. The next experiment is conducted on the FPGA platform. Here, we test the proposed method on the real device, where two trojan examples with different sizes are inserted into the design.

Fig. 7.3: Illustration of decision boundary with RBF kernel

7.2.1 Evaluation on Controlled Settings (Simulation)

We conduct the experiment using simulation on standard normal distribution data to investigate different parameters of one-class SVM. We simulate the data, and divide it into training and testing data. In the Figure 7.2, we show the decision boundary constructed from one-class SVM with linear kernel. In linear kernel, based on the formulation, the algorithm considers the point of origin as the second class when constructing the decision boundary. Hence, the distribution where the values are closer to origin could make the learning more difficult.

Fig. 7.4: Performance of one-class SVM on non-Gaussian data

In the Figure 7.3, we show the decision boundary constructed from one-class SVM with RBF kernel. We can see that both tails of the distribution are considered as suspicious, since it will be the region that most likely overlap with the trojan area. Also, we can see that the higher the γ , the more complex the boundary, and it can potentially lead to overfitting. We also note that, from Figure 7.1, the RBF kernel doesn't have problem with the origin as the case for linear kernel.

Since the generated random data is standard normal distribution, one could easily check that the decision boundary constructed by one-class SVM is consistent with the error function used in the previous work (eq. 6.1). This is under the assumption that the test data is drawn from the same distribution from the training data (since the error is measured based on training data), and the training data is closely following the behavior of its distribution.

For example, one can easily verify that the result in this simulation is consistent with the decision boundary based on normal distribution. In Figure 7.2a, if we calculate the decision boundary based on the error function, in order to obtain 10% false negative, the decision boundary should be at $\mu - 1.28$. The theoretical boundary

agrees with the experimental result, since this is based on perfect simulation. For Figure 7.5a, based on two sided hypothesis testing, to obtain 10% false negative, the decision boundary is on $\mu \pm 1.64$. which is almost consistent with the constructed decision boundary. Hence, in normal distribution case, both SVM and TA method will more likely achieve similar results.

We also highlight the case when the training data is non-Gaussian. We use the example of training data drawn from the bimodal Gaussian mixture and the chi-square distribution. We simulate the training data and then, the training result can be observed in Figure 7.4. We illustrate how the decision boundaries are constructed in this scenario. Here, we illustrate the advantage of the RBF kernel to generalize the result, even if the data is not based on normal distribution.

For the next experiment, we simulate the experiments on simulated data generated using normal distribution under different parameters. The aim is to better understand the effect of different parameters on the prediction of the data. Hence, we consider the learning parameter of one-class SVM, the ν value and the kernel parameter. For the training data, we use a fixed difference of means ($\Delta\mu$) and variance (σ^2). Since we assume the condition that the noise behavior can be modeled as Gaussian distribution, the impact of the trojan can be modeled as the offset from the genuine leakage ($\Delta\mu = \mu' - \mu$), where μ' is the leakage from trojan infected and μ is the leakage from trojan free. For each of the experiment, we repeat it 50 times and take the average of the result. We denote the results in terms of false positive rate and false negative rate. However, we could also use accuracy as a performance metric. Accuracy can be defined as: $ACC = 1 - \frac{FP+FN}{FP+FN+TP+TN}$, where ACC defines accuracy, FP defines false positive rate, FN defines false negative rate, TP defines true positive rate and TN defines true negative rate.

To conduct the experiment, we begin by collecting the data from univariate

Fig. 7.5: RBF kernel on simulated univariate HT data

Fig. 7.6: RBF kernel on simulated multivariate HT data

normal distribution. We generate in total 80k data, in which 50k will be used for training and the remaining for testing. The training data will only contain the genuine class, while the testing data will have mixture of both genuine and affected data. Then, we compare the result based on different parameters of the learning algorithm, ν and γ . The results is highlighted in the Figure 7.5 for RBF kernel. For the genuine data, we set the mean of the data to be random value, bar the origin (since it will affect the linear kernel), and adjust the variance accordingly. Both genuine and trojan are assumed to have same noise variance (σ^2), and the shift

caused by trojan is modeled as shift in the mean ($\Delta\mu$).

From the figure, it can be clearly seen that the false positive is controlled by parameter ν , since the data is drawn from the same distribution, and hence the results are quite similar. On the other hand, the false negative is highly dependent on the trojan impact and the noise, shown by higher false negative when the trojan impact is closer to the genuine. From the results, the trade off between the false positive and false negative has to be balanced. For the case of $\Delta\mu : 5, \sigma : 2$, RBF kernel, the choice of $\nu = 2^{-4}$ or 2^{-5} can lead to stable false positive and negative, as well as minimizing them (around 5%) for the choice of $\gamma = 2^{-8}$ to 2^{-5} . In the case of $\Delta\mu : 2, \sigma : 1$, the large noise affects the accuracy of the classification. Here, the choice of ν between 2^{-3} to 2^{-2} can lead to stable false positive and negative trade-off, as well as minimizing them (around 20-23%) for the choice of $\gamma = 2^{-9}$ to 2^{-2} . In this case, we use the detection accuracy (ACC) as the metric to balance the trade-off, since in Figure 7.5e - 7.5f, we can see that the maximum rate of the accuracy also minimizes both false positive and negative rate.

Next, we include additional parameter, which is the dimension of the data. We simulate the training data based on multivariate Gaussian with 5 features. We fix the mean value around the origin for simulation on RBF kernel. For the covariance matrix, we generate a random positive semidefinite matrix. To simulate the trojan contribution, we insert random offset (on the range of 1-5 for each feature from the mean point of the data) on the data. The sample size is still the same, 50k for profiling and 30k for validation. The result is highlighted in the Figure 7.6. In the case of multivariate normal with dimension 5, we could clearly observe the impact of γ parameter (larger value) on the performance (Figure 7.6a - 7.6b), which is not observable on univariate case. In general, the trend observed is similar to the univariate case.

In our simulation experiments, most of the data are generated from normal distribution. This is done to generalize the data, since for most of the leakage can be assumed to be normally distributed. However, the method can also be generalized to other distribution as well. In the next section, we highlight our experiment on the FPGA platform.

7.2.2 Evaluation on Real Device (FPGA)

Hardware trojans can be inserted into a security-critical circuit at any design stage. From an adversary's perspective, hardware trojans are preferably to be inserted into the circuit with no or extremely minor modification of the original circuit.

Trojan Insertion on Partial Reconfiguration

To emulate the real device scenario, we rely on the **Partial Reconfiguration** technique from Xilinx FPGAs to conduct the trojan insertion into a post P&R AES cipher. In partial reconfiguration of FPGA, the circuit is separated into **static** and dynamic **Partial Reconfigurable (PR)** parts, where the dynamic part can be refreshed in real-time using a different bitstream, without modifying the rest of the circuit. The advantage here is significant for our trojan detection experiments, since we can ensure the absolutely identical cipher (static) but changing the size of the trojans (dynamic). So the variances only come from the trojan circuits and hence permit fair experiments. The region constraint for dynamic and static part of the FPGA are applied at the early design stage and is shown in Figure 7.7.

The cipher algorithm is implemented on a Xilinx Spartan-6 FPGA, soldered on Sakura-G Board, similar to the setup described in [128], running at a frequency of 25 MHz.

Fig. 7.7: Trojan inserted into the cipher using partial reconfiguration to ensure the cipher is identical under different trojan designs

Platform Setup and Cipher Implementation

The measurement platform is shown in Figure 7.8, where the EM radiation of the FPGA internal core from the decoupling capacitor (C40) is measured by a high-precision EM probe. The environmental noise from the surrounding on-board components and IO banks are shielded by an EM-proof cover, which helps to increase the SNR for our measurement.

The insertion of the trojan is similar to the one mentioned earlier which was highlighted in Figure 6.2. The trojan [24] is used to inject a fault on the specific round to enable fault attacks on AES, though in our experiment, we do not trigger the trojan (it remains dormant). There are 2 triggering condition for the trojan: when the eight round of AES is computed and when N least significant bits (LSB) of 128 bits at the `AddRoundKey` at at value 1. The payload of this trojan is an exclusive-OR (XOR) gate. If the trojan is triggered, it will inject fault at the 8-th

Fig. 7.8: EM measurement on decoupling capacitors of SAKURA-G FPGA board.

round, in order to produce Piret's DFA [138]. We use 2 different versions of the trojans, varying the value of N (8 and 128). The trojan itself is placed within the boundary of AES crypto-processor. Here, the placement and routing in the original circuit are kept the same for both genuine circuit and trojan circuit, cf. Figure 7.9. We observe the EM activity of triggering circuit or payload which is enough to detect trojan.

EM Measurement

For the experiments, the traces are collected as follows: we collect traces for 100 different plaintexts which are randomly chosen from a uniform distribution. Each trace is averaged 256 times on scope before saving it for the experiments, to reduce measurement noise. The signal from the EM probe is recorded with WaveRunner 610Zi oscilloscope from LeCroy. Here, only parts of EM traces are recorded (from

(a) genuine design (b) 128-bit trojan (c) 8-bit trojan

Fig. 7.9: Placement and routing of circuits on FPGA for AES-128.

(a) the EM trace (genuine and trojan infected)

(b) the EM traces (zoomed)

Fig. 7.10: EM trace (genuine and trojan infected) with zoomed version

sixth rounds onwards), however, we could argue that since the trojan is not triggered, recording the traces over longer timing would not affect the overall performance that much. We repeat the trace measurement 200 times from the genuine device, where 150 are used for training and remaining 50 for testing. For the circuit under test, 50 traces (averaged 256 times on scope) each for same 100 plaintexts are collected. One sample of the EM trace is shown in Figure 7.10, where it can be seen that the trojan impact is almost negligible (which might also be attributed to the noise).

Preprocess for Trojan Detection

In the supervised setting, we first split the genuine traces for training and testing. For training phase, we train the one-class SVM using different parameters on RBF kernel. We use the following sets of features when representing the data for training:

1. the sum of absolute difference [128],
2. the thresholding technique [128],
3. all the sample points in the traces, and

Algorithm 2: One-class SVM for trojan detection

- 1 **Training phase:**
 - 2 Record power or EM traces $\mathbf{T}_g$ from the genuine devices (supervised),
 or $\mathbf{T}$ from the tested devices (unsupervised) under the same plaintext.
 - 3 Perform feature selection method to obtain $\mathbf{T}'_g$ or $\mathbf{T}'$.
 - 4 Determine user defined parameter ν and γ .
 - 5 Construct the decision boundary f_g or f (Eq. 4.15) based on $\mathbf{T}'_g$ or $\mathbf{T}'$.
 - 6 **Testing phase:**
 - 7 Record power or EM traces $\mathbf{T}_A$ from the tested devices
 with same plaintext for training. Perform feature selection to get $\mathbf{T}'_A$.
 - 8 Predict the temporary class labels using f_g or f on $\mathbf{T}'_A$.
 - 9 Repeat the previous steps with different plaintexts to obtain predictions.
 - 10 Perform majority voting from the collected prediction
 under different plaintexts to determine the final class labels.
-

4. the local peaks in the traces.

The third feature selection method is basically keeping all the time sample points in the traces for training. The last feature selection method is done by selecting the sample points where the traces $\geq$ [the mean of the trace + $v \times$ standard deviation of the trace]. It is similar to the idea of thresholding technique. The differences are that it selects multiple local peaks in the traces as features, only with respect to the training data, without the influence of testing data, and it does not sum all the selected points.

For comparison purpose, we adopt the Template based approach, and same setting are used for classification. Different thresholds are used for determining the decision boundary for Template based attacks, and similarly, different learning parameters are used for construction of the boundary for one-class SVM. In Algorithm 2, we describe the procedure for conducting the experiments.

Normally, the learning parameters need to be tuned beforehand, by further splitting the training data into training and validation data. Here, the performance of different parameters will be checked by performing cross validation, and the best

performing parameter will be chosen. This is to ensure that there is no overfit, in the sense that the training error decreases, but the testing error increases. However, for the single class classification, this process might be difficult, since there is only one class, the genuine class. Normally, for single class classification, after the training phase, during the validation test, we evaluate the performance of different learning parameters using the validation data. Here, the validation data consist of genuine data and a small number of anomalous (trojan) data. The performance is evaluated using the accuracy metric (ratio in which the predicted label match the actual label). The presence of the anomalous data in the validation set is usually done by generating the data artificially. However, sometimes, this might mean that the algorithm has prior knowledge of the outliers. Hence, the parameter selection might be an issue, since the real trojan data might be unknown.

Majority Voting

To improve the performance, we adopt the majority voting to determine the class, based on the classification results on different plaintexts. In majority voting, for every plaintext, we first obtain the predicted class for each trace corresponding to that plaintext. The process is repeated with the traces obtained using different plaintexts. Then, based on the predictions obtained from multiple plaintexts, the final prediction for the class is determined by assigning from the class that is mostly predicted by the different plaintexts.

Experimental Results

We start the experiment with the largest trojan ($N = 128$). In general, we check the performance accuracy from different learning parameters in the validation phase, and based on the observation, we choose the better performing parameters for the

Fig. 7.11: The accuracy results of validation and testing phase using all sample points

testing phase. In Figure 7.11, we plot the results using the third feature selections (all points) as an example. From the figure, we can see that there are several learning parameters that could lead to higher accuracy performance for both validation and testing phase. Thus, the classification results do not overfit.

Fig. 7.12: Result on 128-bit trojan - supervised scenario

During the testing phase, new sets of traces are collected from the suspected device. Using the parameters obtained from the previous experiments, we conduct

Tab. 7.1: Accuracy results on 128-bit trojan (supervised) with single data/plaintext and majority voting

One-class SVM			Template based		
Method	1 data	majority voting	Method	1 data	majority voting
1	68.1%	83.6%	1	66.4%	70.2%
2	85.3%	93.4%	2	83.8%	92.2%
3	95.3%	99.4%	3	74.7%	59.2%
4	80.6%	95.0%	4	76.2%	79.8%

the experiments on this new set of testing data. Then, we show the comparison of the results from the one-class SVM and the Template based attacks in Figure 7.12. In this experiment, for the last feature selection method, we choose the value of v to be 1.5, which allows us to keep enough sample points for training. For the Template based detection method, to classify each trace, a user define boundary needs to be defined when classifying each trace. We then perform similar training on the data (training and validation phase) to determine the optimum boundary value. For each experiment, we repeat it 50 times and average the results.

From the experiment, we can see that the one-class SVM approach is performing better than the Template based method. For both cases, the performance results for the threshold technique are quite similar for both methods. For one-class SVM approach, using all points can help increasing the accuracy, whereas in Template based method, adding more points or features will just add more noise. Also, for one-class SVM, the trend is that by increasing the number of plaintexts and adopting the majority voting approach, the accuracy results can be increased. The detailed results are presented at Table. 7.1. From Figure 7.12, we could see that for most cases, the accuracy rate starts to converge with around 20 plaintexts for the majority voting.

To further verify the results, in the next experiments, we investigate a smaller

Fig. 7.13: Result on 8-bit trojan - supervised scenario

Tab. 7.2: Accuracy results on 8-bit trojan (supervised) with single data/plaintext and majority voting

One-class SVM			Template based		
Method	1 data	majority voting	Method	1 data	majority voting
1	68.6%	86.4%	1	67.3%	66.5%
2	75.9%	91.3%	2	75.8%	89.2%
3	82.6%	93.3%	3	74.6%	64.0%
4	67.0%	74.4%	4	63.9%	80.6%

trojan size with $N = 8$. In Figure 7.13, we show the experimental results and similar trend can be observed, however, with lower accuracy. In this experiment, we show that the proposed heuristic feature selection method, fail to increase the performance accuracy, performing the worst, among different feature selection methods. Again, the threshold technique method performs similarly for both cases, and can achieve the best results for Template based attacks. For detailed comparison, the results are presented at Table. 7.2. From Figure 7.13, the accuracy rate increases with more plaintexts, but with slower rate compared to previous.

In general, from different experiments with different trojan sizes, we can see that one-class SVM is performing better than the Template based approach. Also,

different feature selection methods are more suitable for different methods, namely using all points for one-class SVM and thresholding for Template based. The main problem with the former is obviously when the number of samples grow and hence, more training data are required. We also show that the majority voting based approach can improve the overall classification accuracy. By observing the results, for one-class SVM case, the performance with more plaintexts are better than using only a single plaintext. This is not true for the Template based case, as the results are getting worse when adding the information from more plaintexts.

Fig. 7.14: Result on unsupervised scenario

For the unsupervised scenario, we also show that the one-class SVM can achieve good accuracy performance. In this case, we first collect the traces from the set of suspected devices and use all the traces for training. It is assumed that in the set of suspected devices, a few of the devices are infected with the trojans, and hence, the training data will contain some portions of the outliers. Then, for validation and testing, we collect a new set of data, from the same suspected devices, and use the data for testing. For the Template based methods, it could also be used in similar approach, using the user defined confidence interval or threshold, under

Tab. 7.3: Accuracy results on unsupervised with single data/plaintext and majority voting

One-class SVM (128-bit)			Template based (128-bit)		
Method	1 data	majority voting	Method	1 data	majority voting
3	77.6%	93.7%	3	74.6%	85.7%
4	70.1%	92.4%	4	79.4%	79.8%
One-class SVM (8-bit)			Template based (8-bit)		
Method	1 data	majority voting	Method	1 data	majority voting
3	70.3%	81.6%	3	68.1%	77.2%
4	57.6%	66.7%	4	60.8%	65.0%

which the dataset is trained, and for any data outside this interval or threshold, it can be deemed as trojan infected. In Figure 7.14, we show the results from the experiments. The solid lines denote the one-class SVM approach and the dashed lines denote the Template based methods. The detailed results are presented at Table. 7.3.

For this experiments, since the mean of the genuine traces is unknown, we only consider two feature selection approaches, either using all points, or using only the local peaks based on heuristic methods. The experimental results show that using all points could lead to higher performance result. However, the performance of the heuristic method is getting worse with smaller size of trojan as compared to larger trojan. This might mean that for the smaller trojan, the distinguishing points are not located in the peak of the traces, but rather during the period where the device is less active.

In this case, we can see that when the trojan size is large, the accuracy of the Template based method is not as good as the one-class SVM method. This could be due to the reason that the distinguishability of the trojan is better and by incorporating it on the training data, it disturbs the construction of the template. In one-class SVM based methods however, by adjusting the learning parameters,

the additional noise from trojan can be minimized. When the trojan is smaller, the leakage is behaving more like genuine (smaller offset between the genuine and trojan infected) and hence, Template based method could approach the performance of the one-class SVM.

7.3 Discussion

In [128], the authors proposed a metric for detecting hardware trojans using side-channel leakage under process variation, using Template-like method for trojan detection. Here, we propose another alternative method for trojan detection. We adopt the single class learning algorithm, one-class SVM, which commonly used for outliers detection. In this method, we model the trojans detection as outliers detection problem. We discuss some of the impact of the learning parameters on the performance of the algorithm, especially the ν parameter and the kernel parameter (γ). The choice of parameter ν can be determined by using cross validation methods. However, normally, during the validation phase, it is assumed that the data contain both Trojan and genuine data, whereas, in practice, the Trojan data might not be available. Hence, a more sophisticated parameter selection method could be used in the future.

Regarding the choice of features, we compare 4 different methods. For the two methods from the previous works [128], as they sum all the points, it adds more noise to the computation. However, the threshold technique still allows good classification for the Template based approach. For the next method, we use all the sample points, and the one-class SVM can extract the necessary information which allows good classification on the data. In the last method, we choose the local peaks in the traces, however, some information might not be in the peaks, which limits the

accuracy of the prediction.

As mentioned earlier, there are some open problems when performing trojan detection. The most commonly faced issues are regarding the presence of golden chip and process variation. For golden chip, we show that the approach can be adopted as an unsupervised approach. We show the results when adopting the unsupervised approach for one-class SVM. We also need to highlight that this is the first result with unsupervised learning. Though the performance is not as good as the supervised scenario, it can still be an alternative, especially when dealing with the case when golden chip is not available. Thus, in the future, this method can be investigated and optimized later.

To improve the overall accuracy of the predictions, it is shown that by combining with majority voting method, the performance of the algorithm can be improved. However, one open question is that how to determine what is the minimum number of plaintexts required for majority voting, as we could observe from the experiments, for different trojans, the prediction accuracies are converging at different rates. Also, to further improve the success of trojan detection, for the future work, this method could also be integrated together with other state-of-the-art methods, such as using logic testing based approach (classical trojan detection approach), or using some methods in SCA for increasing SNR in the traces.

Regarding the issues with noise and process variation, we have shown that this method can still achieve the theoretical boundary shown in previous work. However, here we only investigate the performance result when the classification performance is disturbed by the noise, rather than process variation, as in the previous works. Hence, further investigation of the impact of noise and process variation will be a future work.

Part III

THEORETICAL AND PRACTICAL ASPECTS OF FAULT ATTACKS

8. BASICS OF FAULT ATTACKS

In previous part, we have investigated SCA, which can be considered as passive attacks. Here, we consider another category of attacks, the fault attacks, which are considered as active physical attacks, where the attacker tries to force a faulty behavior into the device. For normal applications, the fault incurred will either output undesirable results or damage the chip, however, in context of cryptography, this can lead to the potential recovery of important informations. The difference with SCA is that some changes are required at physical level of the chip in order to conduct the attacks. The main idea in fault attacks is to inject the faults to disturb the internal state of the device when executing the cryptographic algorithm, in order to produce an erroneous output. This can be done, either by changing the operating condition of the device, so it performs wrong computation, or by directly changing the value of the processed data. Hence, by observing the faulty behavior or faulty output, the attacker can derive some information or relations regarding the secret information. In other cases, it can be used to attack broader range, not just for key recovery, for example, it allows the attacker to bypass the key or security check. In this scenario, the attacker does not really need to recover the key.

Fault attacks can be categorized as non-invasive or semi-invasive attacks. In some of the attacks, the device or the chip needs to be de-packaged before the attack, whereas for the others, the attacks can be done without the need of chip de-packaging. For example, EM fault injections are the type of fault attacks that

do not require the de-packaging, however, the localization of the fault might not as precise as say, laser fault injection. The process of de-packaging the chip itself is usually costly and needs to be done by the experts, because the chip could be damaged in the process. Also, fault attacks are typically considered as profiling based attacks, because a lot of time and efforts will be spent on characterizing the device before performing the attacks on the actual target device. Hence, most of publications usually only present the idea of the attack theoretically and verify the results based on some simulations.

8.1 Fault Injection Methods

In general, there are different methods for injecting faults into the targeted device. Fault injection techniques include some methods such as:

- global fault:
 - clock glitch: the attacker can shorten the length of a clock cycle. It can cause a temporary error in the computation. For this method, no de-packaging of the device is required. However, it cannot choose which location in the device to be faulted.
 - voltage glitch: the attacker changes the operating condition of the device, by causing a variation or spike in the power supply for a short period of time. This might force the device to skip the execution of the instructions. Again, this method does not require de-packaging of the device. Similarly, it also cannot choose which location in the device to be faulted.
 - heat: the attacker increases the temperature near the device. This is shown to be capable of producing multiple errors. However, it is hard to

control the errors, which can easily spread to other data, and for some device with low tolerance to the change in temperature, it could actually destroy the device.

- localized fault:

- optical fault injection: for example, the attacker can use laser beam at a very specific spot in the device to change the value of the internal state. It can be used with high precision, but it also requires expertise in operating the laser, especially since the laser can permanently damage the device if not properly calibrated. Other alternative tools could be: x-ray, flash, gamma ray.
- EM pulse: the attacker uses a strong electromagnetic (EM) disturbance near the device to cause temporary change of the signals, which in turn causes error in the computation. However, in order to change value on the specific part of the device, other parts need to be shielded from the EM, and in some cases, the device needs to be de-packaged. Also, to perform localized EM pulse with a small coil, the distance between the coil and the device must be close enough to achieve a high precision.

Among those methods, supply voltage glitching and laser attacks are the most popular types, but, the electromagnetic fault injection is also getting more attention as it has the advantage of keeping the chip in its original package. Alternatively, hardware trojans have also been used as a tool for fault injection [24]. The problem is that it requires lot of expertises and resources to insert working, undetectable trojan, but the advantage is that it can inject more controlled and precise faults.

Laser Fault Injection

In order to perform a practical attack, fault injection techniques need to be performed. The fault injection can be done using different tools, limited only to: price, repeatability, precision, and user-friendliness.

Laser fault injection is normally considered as one of the most powerful and precise techniques, with a high degree of repeatability. The problem is that it requires high price as well as trained and experienced personnel to operate the laser device. Also, when performing laser fault injections into an integrated circuit, the target area has to be directly accessible. Therefore, it is necessary to de-package the chip. This can be done either by using specific types of acids (necessary for the front-side de-packaging) or by mechanical grinding and milling the epoxy layers (possible only for the back-side of the chip). This itself possess another problem, because the chip might be destroyed during the de-packaging process.

Both sides of the chip (the front side and the back side) have different properties that require different laser wavelengths. Normally, for the front side, it is possible to use the red (808 nm) or the green (532 nm) laser to make the fault injection, and for the back side, it requires at least near-infrared (1064 nm) laser. The two ways to inject faults into an integrated circuit using the laser beam can be summarized as follow:

- **Front side attacks** - green (532 nm) and red (808 nm) lasers can be used for these attacks. Visibility of components makes these attacks easier. Reflective effect of the metallic components can lower the accuracy. Also, bonding wires could be cut in process of fault de-packaging, which would make the chip useless.
- **Back side attacks** - near infrared (1064 nm) laser is more suitable for these

attacks, because laser needs to go through the silicon layer. Positioning is harder because the components are not visible but there is no problem with the reflection.

From the summary, there are some others parameters which should be taken into consideration in order to increase the chance of success of injecting faults, such as visibility of the components due to thickness of a silicon substrate, smoothness of a chip surface, and also other parameters, such as the beam spot size, and precise positioning table.

When considering the back-side laser fault injection, there are two essential parameters – the *reflection coefficient* and the *absorption coefficient*. The *absorption coefficient* determines how far can a light with a certain wavelength penetrate into a particular material before it is completely absorbed. The *reflection coefficient* is a ratio of the amplitude of the reflected wave and the amplitude of the incident wave.

In Figure 8.1, we can see the absorption depth, computed as an inverse of the *absorption coefficient*. This plot shows us, why we cannot use green (532 nm) or red (808 nm) laser beams for back side fault injection attacks, since the green laser can only penetrate $\sim 1 \mu\text{m}$ thick silicon substrate and the red laser can only penetrate $\sim 10 \mu\text{m}$ thick silicon substrate, which might not be enough.

The advantages of using laser to perform fault attacks are claimed to be its precision and reproducibility. Spatial coherence allows a laser beam to be focused to a tight spot, therefore with a small diameter it is possible to aim at very small components of an integrated circuit. With focused beam, it is possible to disturb the transistors and change bit values in the registers. Also, given the same parameters, it should be possible to repeat the experiment with high probability of the same results.

However, there are also some disadvantages connected with a usage of a laser

Fig. 8.1: Absorption depth for particular wavelengths in Silicon [71].

equipment for the fault injection. First, as mentioned earlier, the chip surface has to be accessible by the laser, which means that we need to de-package the chip before. Another disadvantage is a possibility of destroying the chip either by a large number of repetitions or by a high laser energy.

The construction of the laser setup itself is important. In Figure 8.2, the comparison is shown between a fixed laser spot size ($1\mu\text{m}$), and different technology sizes [41]. From Moore's Law [121], when technology becomes more advanced, the transistor size decreases as well. Hence, it is becoming harder to achieve more precise faults, such as single bit flips, on the newer technology. As such, more precise laser equipments will be needed to keep up with the current state-of-the-art.

So far, only a few practical evaluations of laser fault injections have been reported. However, most of these attacks are based on random fault models, which could also be easily achieved without the need of a laser injection. Here, we will give some examples of previous works concerning more advanced laser fault injection experiments.

In [53], Courbon *et al.* reported a practical application of a back-side laser fault

Fig. 8.2: Fixed laser spot on different technology size [41]

injection on a 90 nm microcontroller. They managed to set/reset a byte stored in the register. They concluded that bit resets can be done with lower energy hitting the back side of the chip than the energy needed for bit sets.

In later publication [52], they successfully conducted an attack on AES implementation running on a 130 nm microcontroller. In their experiment, they have opened another chip from the front-side in order to identify the flip flops using an EM. After determining these potential points of interest, they were able to narrow down the area which had to be scanned by the laser to perform a successful attack on registers. However, there is no detailed report on the experimental setup, and it can be safely assumed that producing such results needs very precise and expensive equipment.

Dutertre *et al.* [61] were doing experiments on a 350 nm microcontroller. They

performed single-byte fault injections in an SRAM and implemented a Piret and Quisquater's fault attack on AES. More detailed results on experiments on SRAM cells are provided in [148]. The authors showed that the bit-flip fault model is not feasible with the laser fault injection technique, only the bit-set/reset fault model can be achieved.

Agoyan *et al.* [6] presented a Differential Fault Attacks or DFA (which will be described later) on AES implemented on a 350 nm microcontroller by performing multiple byte faults. They aimed at the SRAM, attacking the surface of the chip from the front side. They used a green laser beam (~ 532 nm) with $5.5\ \mu\text{m}$ diameter, 20x magnifying objective lens and a positioning table with $0.1\ \mu\text{m}$ precision.

Roscian *et al.* [147] had also investigated possibilities of a laser fault injection, using AES implementation running on ASIC. They used relatively large laser beam spot (square spot $125 \times 125\ \mu\text{m}^2$) and they could successfully perform bit flips and bit sets/resets. Despite the size of the beam, a large part of induced faults were single-bit faults. With these results they were able to perform a successful DFA on AES.

8.2 Fault Models

For conducting fault attacks, other than fault injection methods, the attacker also needs assume certain fault models. The choice of the fault models and the timing of the fault injections are usually dependent on structure of the cipher, as well as the implementation. Some of the attacks do not require sophisticated model, such as the fault attacks on RSA implementation with Chinese Remainder Theorem (CRT) which only need to inject random faults [32]. Generally, the choice of fault models is crucial, which is comparable to the importance of leakage model in SCA.

There are few parameters that characterize the fault models. First is the location of faults, whether the injected faults affect the global region or they only affect a small region (local faults). The other parameter is the nature of the faults, whether the fault made on the device are transient fault, which is temporary and affects the variable once (after reset, the calculation will be fine) or a permanent fault.

Here, we list down several most commonly used fault models. [19]:

- Single/multiple bit flip: different number of bits value(s) in the destination register are flipped (0 to 1 or 1 to 0). This is the most commonly used model in theoretical attacks.
- Instruction skip: one or two instructions in the program are skipped.
- Random byte fault: several values of bytes are randomly changed.
- Stuck-at fault: the value at the destination register is stuck at certain value, in general case, either to 1 (set) or 0 (reset).

Quite often, the fault model can be unrealistic to achieve, or requires a very high cost to achieve. In practice, for the fault models, there are also some considerations [131]. First, it depends on whether the choice of bit to flip can be controlled freely or not. For most of the time, this might not be the case. Another thing is the timing of the faults, whether it affect huge blocks or can be precisely controlled to attack specific operations. Lastly, it depends on the probability of injecting the faults, how often can the faults be repeated under the same experimental settings.

In general, the effect of the faults mainly depends on which platforms are used for the cryptographic implementation. Most of the time, random fault model is the easiest model to obtain, whereas more precise faults require much more resources. Hence, this contrasts with the theoretical attack model that requires more precise

fault models, on specific timing and location, in order to perform successful attacks. Thus, in general, often there are two kinds of papers about fault attacks in publications [97]: the one that shows how to induce specific faults practically on the device, and the one that exploits specific fault model on cryptographic cipher, assuming the fault model can be realized in practice.

8.3 Types of Fault Attacks [35]

The first fault attack was shown in 1996 by Boneh, DeMillo and Lipton [32]. The main idea was to disturb the computational process and exploit the error in order to obtain the key. The attack was first shown to be applicable for public key cryptosystem. The attack exploits RSA implementation using Chinese Remainder Theorem (CRT). The attack enabled an efficient factorization of the modulus N with just one pair of faulty and correct ciphertext.

Many types of fault attacks have then been proposed for different cryptographic algorithms. These fault attack methods include for example: Differential Fault Attacks (DFA), Collision Fault Attacks (CFA), Ineffective Fault Attacks (IFA), and Safe-Error Attacks (SEA). So far, DFA has been considered as the most popular method to attack block ciphers. In general, most of these attacks are only verified theoretically through simulation. In most cases, these attacks would require a strong assumption on the fault model.

The most commonly used method is the DFA. The main principle is to inject faults in chosen round of the algorithm in order to get the desired fault propagation by the end of the encryption/decryption process. The secret key can then be determined by examining the differential propagation between the correct and faulty ciphertext. This attack requires at least one correct ciphertext and one or few in-

Fig. 8.3: Illustration of Fault Attacks concepts

correct ciphertexts. In general, the attacks use a similar concept with differential cryptanalysis. The first attack using this technique was the work by Biham and Shamir [25]. To perform the attacks, they combined the fault model concept and differential cryptanalysis on DES algorithm, and they provided a successful attack. By inserting a fault into the last round during the execution, which results into faulty ciphertext, the attacker can then compare this ciphertext with the correct one to gain additional information about the secret key.

The next method is CFA [28]. For this attack, the attacker invokes fault in the beginning of the algorithm to obtain faulty ciphertext and then he tries to find a plaintext, which encrypts into the same ciphertext as the faulty ciphertext in the previous case, by using the same key. The idea of this attack is similar to collision

attack. The typical fault model for this attack is stuck-at fault. By forcing the data to always be of specific value, the attacker can know which bits that have to be changed in order to force the collision.

The other method is IFA [49]. The goal of the IFA is to find such fault that does not result into incorrect ciphertext. Here, the attacker tries to find a plaintext which could lead to the same ciphertext with and without injecting specific faults. Hence, the attack is considered successful when the faulty and correct ciphertext are the same. The main problem of this method is to determine if the fault is actually invoked or not.

Similarly, in SEA [174], the attacker is more interested in determining if the fault will affect the output. It was first shown to attack RSA with CRT implementation, by identifying dummy operation or unused intermediate value in the algorithm. Hence, by injecting faults at different time in the execution, and then checks if the output is affected or not, it can distinguish which operation is useful and which operation is dummy. Though similar, there is difference between IFA and SEA. The difference is that IFA is more dependent on the effect of the fault on the processed data, while for SEA, it analyze if the manipulated data actually affect the results, and thus it has a more relaxed fault model.

In 2010, Fault Sensitivity Analysis [112] was proposed. This method is effective even for some DFA resistant implementations and does not restrict the fault model to a few bits or bytes. It exploits side-channel information, such as sensitivity of a device to faults and uses this information to retrieve the secret.

9. INVESTIGATION ON THEORETICAL AND PRACTICAL FAULT ATTACKS

In this section, we investigate the theoretical and practical aspects of fault attacks on block ciphers. First, we investigate DFA on block cipher LEA. For the current fault attacks, most of the investigations are conducted on SPN and Feistel structures. However, the ARX structure has not received so much attention, with current work done only on SPECK cipher. Another cipher with such structure is LEA which has not been investigated so far, and hence, might be interested to investigate. Another motivation to investigate this cipher is that LEA-128 is a Korean standard [13]. Most of the works done on this part have been reported in [86].

We exploit the properties of the non-linearity of the modular addition operation used in the round function. To recover the key, our attack requires two different positions of fault injections - in the last round and in the penultimate round. For our attack, by using a random bit-flip model, we are able to recover a 128-bit secret key by using around 258 faulty ciphertexts on average. If the precise fault position is known, our attack requires only 62 faulty ciphertexts in average.

Then, we explore practical fault attacks using laser injection to achieve high fault precision and repeatability. We investigate the possibility to inject different fault models and types on the back-side of a microcontroller. We perform experiments on the low end Atmel microcontroller, which is one of the most common microcontrollers available on the market due to the popular Arduino UNO platform. The

profiling experiments have been reported in [36], where we observe that some theoretically fault models, such as bit and byte flips, are harder to obtain and might require more resources to achieve.

Based on the observation of the fault behavior, we are able to disturb the instruction execution process and to skip different operation instruction for each byte separately. This attack could then be applied to any block cipher with post key whitening and reversible key schedule. Thus, we design and perform a simple yet very effective attack on AES block cipher, and are then able to retrieve the AES-128 secret key with just one faulty and one correct ciphertext pair. The attack itself has been reported in [37].

9.1 DFA on LEA Block Cipher

9.1.1 Introduction to DFA

When it was first introduced, DFA was used to attack the block cipher DES [25]. Since AES is now the most popular symmetric block cipher, majority of attacks were then aimed on this algorithm. The idea of DFA is highlighted in Figure 9.1, with AES as the target of the attack. This example is based on Piret and Quisquater attack [138]. As mentioned earlier, the aim is to inject fault during execution of cryptographic algorithm and to collect the faulty results. For example, on AES, the fault can be injected at a byte state before the last or the penultimate `MixColumns`. The fault then propagates to other parts of the data, and by the end of the execution, most of the other parts of the data will be affected by the fault. The difference between the correct and the faulty output is then analyzed in order to reduce the key search space. So, in AES, when the fault is injected at the last state before `MixColumns`, the fault will be propagated to 4 different bytes which can be analyzed

Fig. 9.1: Illustration of DFA on AES

to recover 4 bytes of the key. If the fault is injected in the penultimate **MixColumns**, the fault will be propagated to 4 different bytes in the second last round of AES, and after the last **MixColumns**, the faults could be further propagated to all bytes in the ciphertext.

The first DFA on AES was proposed by Giraud [69]. It was shown that the full key can be revealed by either using 50 faulty ciphertexts by inducing bit faults or 250 faulty ciphertexts by using the byte fault model. Piret and Quisquater [138] then showed that it is possible to perform DFA on AES using a byte fault model while needing only two faulty ciphertexts, and only need to brute force the remaining key of size 48 and 40 bits. Later, Tunstall *et al.* [163] showed that it is possible to retrieve the key with single fault, with only 2^8 number of key search space. A multi byte DFA was proposed by Saha *et al.* [150]. It showed that if up to 3 bytes

of the diagonal in the AES state could be faulted, the secret key could be retrieved with only 2-4 faulty ciphertexts. In [8], Ali and Mudkhopadhyay showed that with multiple byte fault, it only required single faulty ciphertext in order to get the key.

There are also results on other ciphers beside AES. For SPN structures, the first DFA on lightweight block cipher PRESENT [30] was published in 2010 [168]. Using the faults on single nibble, their attack could recover the key using 64 pairs of correct and faulty ciphertexts on average. The most efficient attack on PRESENT so far was proposed by Gu *et al.* [72]. It was using a 2-byte random fault model, attacking the intermediate value after round 28. For PRESENT-80, they needed two 2-byte faults.

Another DFA attack is shown for lightweight block cipher CLEFIA [155], which is based on Feistel structure. The first DFA on CLEFIA was proposed by Chen *et al.* [47]. The idea was to use a random one byte fault. They required only 18 faulty ciphertexts to recover 128-bit secret key and 54 faulty ciphertexts for 192/256 keys. In [9], Ali and Mudkhopadhyay showed improved DFA that allowed the attack on CLEFIA protected with countermeasures against normal DFA and 8 faults to retrieve 192 and 256 bit keys.

For the attack on ARX structure, Tupsamudre *et al.* [165] showed an attack on SPECK cipher [22]. The authors aimed at the only non-linear operation, at the modular addition. They were able to recover the n -bit secret key by using $n/3$ bit faults on average.

9.1.2 Introduction to LEA

LEA [81] is a symmetric block cipher, based on the ARX design. It offers fast software encryption, is comparable to lightweight ciphers, and comes in the same key size variants as AES. According to security evaluation report [31], LEA has been

shown to be secure against state-of-the-art cryptographic attacks.

The block size of LEA cipher is 128 bits, with the size of the key can be 128, 192, or 256 bits. The number of rounds for each key size is 24, 28, and 32, respectively. It is a pure ARX cipher, consisting of modular Addition, bitwise Rotation, and bitwise exclusive-OR (XOR) operations on 32-bit words. We will further describe the design of the 128-bit key size variants of the cipher.

First, the 128-bit intermediate value $\vec{X}_0$ is set to the plaintext $\vec{P}$. Then, a key schedule process creates r round keys. The 128-bit output $\vec{X}_{i+1} = (X_{i+1}[0], \dots, X_{i+1}[3])$ of i -th round is computed as:

$$\begin{aligned} X_{i+1}[0] &\leftarrow ROL_9((X_i[0] \oplus RK_i[0]) \boxplus (X_i[1] \oplus RK_i[1])) \\ X_{i+1}[1] &\leftarrow ROR_5((X_i[1] \oplus RK_i[2]) \boxplus (X_i[2] \oplus RK_i[3])) \\ X_{i+1}[2] &\leftarrow ROR_3((X_i[2] \oplus RK_i[4]) \boxplus (X_i[3] \oplus RK_i[5])) \\ X_{i+1}[3] &\leftarrow X_i[0] \end{aligned}$$

The resulting ciphertext is then obtained after r rounds in the following way:

$$C[0] \leftarrow X_r[0], C[1] \leftarrow X_r[1], C[2] \leftarrow X_r[2], C[3] \leftarrow X_r[3].$$

The whole process is depicted in Figure 9.2.

Here, we use $X_i[j]$ to denote 32-bit vector of round i and block j . Also, we denote ROL_i as i bit left rotation, and ROR_i as i bit right rotation.

Let $\vec{K} = (K[0], K[1], K[2], K[3])$ be a 128-bit key. We set $T[i] = K[i]$ for $0 \leq i < 4$. Round keys $RK_i = (RK_i[0], \dots, RK_i[5])$ for $0 \leq i < 24$ are then computed as follows:

$$T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4])),$$

Fig. 9.2: i -th round function of LEA

$$\begin{aligned}
T[1] &\leftarrow \text{ROL}_3(T[1] \boxplus \text{ROL}_{i+1}(\delta[i \bmod 4])), \\
T[2] &\leftarrow \text{ROL}_6(T[2] \boxplus \text{ROL}_{i+2}(\delta[i \bmod 4])), \\
T[3] &\leftarrow \text{ROL}_{11}(T[3] \boxplus \text{ROL}_{i+3}(\delta[i \bmod 4])), \\
RK_i &\leftarrow (T[0], T[1], T[2], T[1], T[3], T[1]),
\end{aligned}$$

where $\delta[i]$ for $0 \leq i < 8$ are key generating constants, obtained from a hexadecimal expression of $\sqrt{766995}$, where 76, 69, and 95 are ASCII codes of 'L,' 'E,' and 'A.'

9.1.3 Attack Methodology

From the attack methodology point of view, the closest attack is the attack proposed by Tupsamudre *et al.* [165], aiming at SPECK cipher. Since SPECK uses the ARX structure as well, the authors aimed at the only non-linear operation, at the modular

addition.

To perform a DFA on LEA, we propose using two single bit flip faults, at $X_{22}[0]$ and at $X_{23}[2]$. The propagation of the faults can be observed in Figure 9.3. We then retrieve the key by exploiting the modular addition operation.

Fig. 9.3: Fault propagation in rounds 23 and 24 of LEA

To describe the proposed method, we first show how it works on a normal modular addition. First, let us assume that the operation is done on 32-bit values, $\vec{A}$

and $\vec{B}$. The modular addition can then be expressed as:

$$\begin{aligned}\vec{D} &= (\vec{A} \boxplus \vec{B}), \\ D_j &= (A_j \oplus B_j \oplus c_j),\end{aligned}$$

where $j \in \{0, \dots, 31\}$, and c_j is a carry bit from the previous addition with $c_0 = 0$. The idea is, that the fault can be injected at $\vec{A}$, and by observing $(\vec{A} \oplus \vec{A}^*)$ and the pair of correct-faulty outputs $(\vec{D}, \vec{D}^*)$, the attacker can retrieve the value of $\vec{A}$ and $\vec{B}$. Here, $\vec{A}^*$ denotes the faulty value of $\vec{A}$, where $A_k^* \neq A_k$ for $k \in \{j, j+1, \dots, j+n\}$ and $n \geq 0$. The value k denotes the position of fault(s). If $n = 0$, then it is a single bit fault, otherwise, it is a multiple consecutive bit fault.

From the output value $\vec{D}$ and $\vec{D}^*$, the attacker can also observe $\vec{D} \oplus \vec{D}^*$. The attacker then checks how many N bit difference(s) are in $\vec{D} \oplus \vec{D}^*$. First, we consider the case with only 1 bit difference in $(\vec{A} \oplus \vec{A}^*)$, more specifically, at bit j . Starting from the location of fault j , the attacker calculates N . If the value of $N = 1$, it can be concluded that the carry bit at bit $j+1$ is not flipped and hence, from the left part of the Table 9.1, we can conclude that the value of $B_j = c_j$ (highlighted with red color). However, if $N_1 > 1$, it can be concluded that the carry bit at $j+1$ is flipped, and thus, $B_j \neq c_j$. Note that this attack requires that the carry bit c_j is known and thus, it relies on the assumption that the 0-th bit could be faulted in the process (since $c_0 = 0$ is the only known carry from the beginning). Once the values of B_j , c_j and D_j are known, the value A_j can be determined ($A_j = D_j \oplus B_j \oplus c_j$), as well as the value c_{j+1} (by observing the carry bit of $A_j + B_j + c_j$).

Next, we consider the case when there are multiple consecutive bit differences at $(\vec{A} \oplus \vec{A}^*)$, in bits $j, \dots, j+n$. First, the attacker needs to determine if the carry bit at the bit $j+i$ ($i \in \{0, \dots, n\}$) is flipped or not. The attacker observes $(D \oplus D^*)_{j+i}$.

Tab. 9.1: Changes of values in modular addition after a bit flip.

First faulty bit						Consecutive faulty bit						
A_j	A_j^*	B_j	c_j	c_{j+1}	c_{j+1}^*	A_j	A_j^*	B_j	c_j	c_j^*	c_{j+1}	c_{j+1}^*
0	1	0	0	0	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	1	0	0	0
0	1	1	0	1	0	0	1	1	0	1	0	1
0	1	1	1	1	1	0	1	1	1	0	1	1
1	0	0	0	0	0	1	0	0	0	1	0	0
1	0	0	1	1	0	1	0	0	1	0	1	0
1	0	1	0	0	1	1	0	1	0	1	1	1
1	0	1	1	1	1	1	0	1	1	0	1	0

$$(D \oplus D^*)_{j+i} = (A_{j+i} \oplus B_{j+i} \oplus c_{j+i}) \oplus (A_{j+i}^* \oplus B_{j+i} \oplus c_{j+i}^*)$$

$$(D \oplus D^*)_{j+i} = 1 \oplus c_{j+i} \oplus c_{j+i}^*$$

So, if the value of $(D \oplus D^*)_{j+i} = 1$, it means that $c_{j+i} = c_{j+i}^*$, and similarly, if $((D \oplus D^*)_{j+i} \neq 1) \Rightarrow (c_{j+i} \neq c_{j+i}^*)$. Then, the attacker determines the value of the carry bit in the next bit ($j + i + 1$) using similar approach. The Table 9.1 summarizes the scenario when the carry values for the current bit and the consecutive bit are flipped or not. To highlight, if the value of the current bit is not flipped and if the value in the next carry bit is not flipped as well, then $B_{j+i} = c_{j+i}$ (left part, highlighted with red color). If the value of the current bit is not flipped and if the value in the next carry bit is flipped, then $B_{j+i} \neq c_{j+i}$ (left part). On the contrary, if the value of the current carry bit is flipped, and also the value in the next carry bit is flipped, then $A_{j+i} = c_{j+i}$ (right part). If the value of the current bit is flipped and if the value in the next carry bit is not flipped then $A_{j+i} \neq c_{j+i}$ (right part, highlighted with yellow color). Once the value of either A_{j+i} , or B_{j+i} is known, the other value and the next carry bit c_{j+i+1} can be determined.

9.1.4 Simulation Results

We test the proposed method by conducting the experiments using simulated faults for verification. Here, the attacker chooses one fixed plaintext, measured multiple times, and injects fault at different bit locations. Then, he collects all the faulty ciphertexts. The attacker then has to inject faults at two different positions of different rounds. The main target for this attack is LEA-128, due to its key structure. Each of the round keys can be segmented into 6 parts, each part consists of 32 bits. In LEA-128, the second, the fourth and the last part ($RK_i[1]$, $RK_i[3]$, and $RK_i[5]$) are identical, based on the key scheduling. In previous section, it is shown that the attack exploits the modular addition $\vec{D} = (\vec{A} \boxplus \vec{B})$.

Faults at $X_{22}[0]$

First, the attacker injects single bit flip fault at $X_{22}[0]$, and the fault will propagate to $X_{24}[0]$, $X_{24}[2]$ and $X_{24}[3]$. The fault propagation will not be affected by the XOR operation with the round keys.

1. The attacker first exploits the difference $(X_{24}[3] \oplus X_{24}^*[3]) = (X_{23}[0] \oplus X_{23}^*[0])$. Then, he can pinpoint the location of initial fault injected by taking the last bit difference in $ROR_9(X_{23}[0]) \oplus ROR_9(X_{23}^*[0])$, since the attacker knows that the fault is a single bit flip and the locations of faults have been changed due to a rotation.
2. Then, the attacker can construct $(X_{22}[0] \oplus X_{22}^*[0])$. Here, the difference $(X_{24}[3] \oplus X_{24}^*[3])$ can be considered as caused by multiple consecutive bit flips injected at $X_{23}[0]$.

Input: $(X_{23}[0] \oplus X_{23}^*[0])$, $X_{24}[0]$, $X_{24}^*[0]$,

Output: estimated $(X_{23}[0] \oplus RK_{23}[0]), (X_{23}[1] \oplus RK_{23}[1])$.

However, since $X_{23}[0] = X_{24}[3]$, the round key $RK_{23}[0]$ can be calculated, and using the key schedule algorithm, $RK_{22}[0]$ can be determined as well from $RK_{23}[0]$.

3. Using the knowledge from the previous step, attacker can continue with revealing partial information.

Input: $(X_{22}[0] \oplus X_{22}^*[0]), X_{23}[0], X_{23}^*[0]$,

Output: estimated $(X_{22}[0] \oplus RK_{22}[0]), (X_{22}[1] \oplus RK_{22}[1])$.

Since the attacker already knows $RK_{22}[0]$, he can reveal $X_{22}[0]$ based on the estimated output, which in turn reveals $X_{23}[3]$.

4. Here, $(X_{22}[0] \oplus X_{22}^*[0])$ is the same as $(X_{23}[3] \oplus X_{23}^*[3])$.

Input: $(X_{23}[3] \oplus X_{23}^*[3]), X_{24}[2], X_{24}^*[2]$,

Output: estimated $(X_{23}[3] \oplus RK_{23}[5]), (X_{23}[2] \oplus RK_{23}[4])$.

However, as the $X_{23}[3]$ is known by previous step, the attacker obtains $RK_{23}[5]$, which, by key scheduling, is the same as obtaining $RK_{23}[1]$ and $RK_{23}[3]$.

By the end of phase 1, the attacker has obtained the values of $X_{22}[0](= X_{23}[3])$, $RK_{22}[0]$, $X_{23}[0]$, $RK_{23}[0]$, $X_{23}[1] \oplus RK_{23}[1]$, $RK_{23}[1]$, $X_{23}[2] \oplus RK_{23}[4]$, $RK_{23}[3]$ and $RK_{23}[5]$.

Faults at $X_{23}[2]$

With the partial knowledge of the key and the intermediate states from phase 1, the attacker injects another single bit flip fault at $X_{23}[2]$.

1. By observing $(X_{24}[2] \oplus X_{24}^*[2])$, he can determine $(X_{23}[2] \oplus X_{23}^*[2])$, since it is a single bit flip fault.

Input: $(X_{23}[2] \oplus X_{23}^*[2])$, $X_{24}[1]$, $X_{24}^*[1]$,

Output: estimated $(X_{23}[1] \oplus RK_{23}[2])$, $(X_{23}[2] \oplus RK_{23}[3])$.

From the previous phase, $X_{23}[1]$ can be determined since $X_{23}[1] \oplus RK_{23}[1]$ and $RK_{23}[1]$ are known. Then, $RK_{23}[2]$ can be determined from $(X_{23}[1] \oplus RK_{23}[2])$ and $X_{23}[1]$.

2. Similarly, since $RK_{23}[3]$ is known from the previous phase, $X_{23}[2]$ can be calculated with $(X_{23}[2] \oplus RK_{23}[3])$, obtained in previous step.
3. Since $(X_{23}[3] \oplus RK_{23}[5])$ has been determined in previous phase, together with $X_{23}[2]$ from the previous step and $X_{24}[2]$, which is the output, the attacker can determine $RK_{23}[4]$.

Thus, the remaining of last round key, $RK_{23}[2]$ and $RK_{23}[4]$, can be retrieved in this phase.

9.1.5 Observations

One observation regarding the attack is that the last bit (MSB) of the modular addition output cannot be obtained because the next carry bit cannot be determined. This condition holds for each part of the key and hence, in total, 4 bits of the key could not be determined. Thus, these remaining key bits have to be brute-forced (with complexity 2^4).

Another problem could also occur due to the rotation of the bits. Due to rotation ROL_9 , the resulting bit difference might not be consecutive, for example, $ROL_9([0 \dots 0 \ 1 \dots 1]) = [1 \ 1 \ 0 \dots 0 \ 1 \dots 1]$. To mitigate this problem, it can be con-

Fig. 9.4: Retrieved key bits from single bit flip in modular addition

sidered as separate multiple consecutive faults. From the example, the fault can be considered as $[1\ 1\ 0\ \dots\ 0]$ and $[0\ \dots\ 0\ 1\ \dots\ 1]$. Since the attack only considers bits of D and D^* which coincide with the faulty bits in A^* , the previous method still works.

In Figure 9.4, we show the number of faults required to retrieve the value of the input of the modular addition. In the case where each bit could be flipped precisely, around 30 faults are required. However, if the bit could be flipped only randomly, more faults are required.

Based on the experiment, if the attacker cannot choose the location of a single bit flip fault, and hence, can only inject faults randomly at single bit, it requires ≈ 130 and ≈ 128 for the first and the second fault respectively in order to determine the last round key, minus the 4 bits mentioned earlier.

However, if the attacker can determine the precise location of a single bit flip, it requires only ≈ 31 and ≈ 31 faults for the first and the second fault respectively. This is because each fault reveals one bit of the key and in each attack, 2 parts of the last round key could be determined. For the previous attack model, more faults are required in order to obtain the value of each bit, similar to the coupon collec-

tor problem ($\Theta(n \log n)$). These results are verified by performing 500 independent experiments on simulations.

Another observation is that the attack requires quite strict fault model, single bit flip or to lesser extent, multiple consecutive bit flips. To achieve such fault model, it requires a high precision disturbance on the device, in this case, for laser injection case, a higher precision in the laser spot size and precision in the coordinate of the device.

9.1.6 Discussion

We present a successful attack on LEA which enables the key recovery with remaining 2^4 brute force complexity. Here, we require quite a number of faults. This is because on LEA, the only non-linear operation, modular addition, does not propagate much into different bits and hence, for our current fault model, we will need more bit flip faults. Our aim for the future work is then to investigate other fault models to reduce the fault complexity. Another alternative is to investigate different rounds in which the fault will propagate more and can still be easily exploitable.

9.2 Practical Fault Attacks using Laser Injection

In this part, we investigate the possibility of using laser fault injection for conducting practical fault attacks, which could be used to verify previous theoretical attacks. Our experiment are conducted using laser fault injection station. As described earlier, in order to perform the attack, we need to do a prior profiling on the laser station to determine the possible fault models. Then, using the available fault model, we determine the attack model to carry out the fault attacks.

9.2.1 Laser Station Setup

For our station setup, it is composed of a near infrared diode pulse laser with a pulse power of 20 W (reduced to 8 W with 20x objective and further to 7W with 50x objective). The pulse repetition rate is 10MHz and spot size of $30 \times 12 \mu\text{m}$ ($15 \times 3.5 \mu\text{m}$ with 20x objective and $6 \times 1.4 \mu\text{m}$ with 50x objective). Intentional NOP operations (do nothing) are inserted at beginning of each node to overcome the delay 100 ns between trigger and laser injection.

The target device is Atmel ATmega328P microcontroller running at 16 MHz, with 1 KB EEPROM, 32 KB Flash, and 2 KB SRAM. This is chosen as the target device because it is commonly used device and for simplicity reason. The chip is $3 \times 3 \text{ mm}^2$, manufactured in 350 nm CMOS technology. The microcontroller needs to be first de-packaged and mounted on a Arduino UNO development board. Laser injection is performed using a X-Y positioning table with a step precision $0.05 \mu\text{m}$ as shown in Figure 9.5.

Fig. 9.5: Device under test during the experiment.

Communication with the Device Under Target (DUT) is done via RS232 interface. We use an oscilloscope for measuring the power consumption of the DUT, for capturing the trigger signal and the laser diode current, so we could determine the

delay between sending the trigger signal and activating the laser beam.

Fig. 9.6: Backside de-packaging

9.2.2 Profiling of the Laser Station

In order to determine the types of fault models that can be obtained using the laser fault injection, and which fault models are actually usable for the attack, the laser properties have to be profiled, as well as the targeted device. For our experiments, we first scan the whole area of the chip using random input, and record the location where faulty outputs are observed. Then, more detailed explorations are performed on that location.

The codes for our experiments are written in assembly language, using the Arduino programming platform. Experiment steps are described in Algorithm 3.

The addition of NOP instructions is crucial in order to avoid fault injection in the bus or in the clock signal. It is to ensure that the laser only affects the registers where the data are stored, or only the load operation when loading the data to the

Algorithm 3: Procedure for laser profiling of the device

-
- 1 Send 10 bytes to the device via RS232 interface from the PC.
 - 2 Store the data in a variable.
 - 3 Set the trigger signal on Arduino pin 13 to HIGH (5V).
 - 4 Load each byte to a different target register (LD $rN, X+$).
 - 5 Perform 10 NOP instructions (10x62.5 ns).
 - 6 Activate the laser beam.
 - 7 Read the data from registers and send them back to the PC (ST $Y+, rN$).
 - 8 Compare the data with the sent data.
-

registers.

We vary the parameters of the laser setup, such as laser strength, glitch length and glitch offset. One type of fault that we obtain from the profiling is the skip instruction fault, which could skip the instruction performing a load to the register, and such, the value obtained is shifted. We observe that the skip instruction fault has high repeatability, and is more consistent, as compared to other types of faults (some faults, like instruction change, rarely happen, even if the parameters of laser setup are fixed). Hence, it shows that for most of the theoretical fault attacks, it might not so easy to inject such fault in practice, and a lot more resources are required in order to inject such faults.

The first important parameter is the position at which it was able to perform the instruction execution disturbance. In Figure 9.7 we can see this position with respect to the whole area of the chip (back side).

To investigate further on the instruction skip attack, we adjust the laser setting, as follow:

- Glitch length – 150 ns
- Step size – 15 μm (200 steps each direction).
- Laser power – 1.8%

Fig. 9.7: Region for skip instruction

This profiling step requires at least 2 hours to complete. The following code snippet is then repeated 25 times in the program:

```
LD    r0, -Y    (2 clock cycles)
EOR   r0, r25   (1 clock cycle)
ST    Y, r0     (2 clock cycles)
```

The experimental results show that it is possible to set the timing very precisely and to skip the desired instructions¹. Figure 9.8 shows different timings for successful attacks on particular bytes. One idea to exploit this fault behavior is to skip some operations during the encryption/decryption process, and the simplest idea is to skip the post key whitening. The XOR instruction (EOR) used earlier could be considered as a simulation for the post key whitening. In this case, we consider AES, since it the most widely used block cipher.

¹ The CPU does not have sophisticated error handling for "illegal" opcodes. If you start executing e.g. 0xFFFF the program counter will simply increase by 1 or 2 until you wrap around to address 0x0000 or hit some useful code. So an illegal opcode will be the same as a NOP. (http://atmel.force.com/support/articles/en_US/FAQ/Illegal-Opcode)

Fig. 9.8: Timing of instruction skip for different bytes

9.2.3 Practical Attack on AES

After the successful profiling phase, we are able to perform a simple attack on the AES implementation. The idea of the attack is to skip the **XOR** instructions in the last **AddRoundKey**, so that the resulting output of the encryption process is the output of the last **ShiftRows**. Therefore, if we take the **XOR** of this output and the correct output, we will be able to recover the last round key. With the inverse key schedule it is then easy to recover the master key. The main idea of the attack is captured in Figure 9.9.

Our results show that with a long-enough laser glitch, it is possible to skip the whole **AddRoundKey** operation in the last round. Figure 9.10 shows the number of faulty bytes corresponding to different laser power together with faults that lead to successful key bytes retrieval. We can see that with the laser power around 2% and

Fig. 9.9: Illustration of skip instruction in the last round of AES

above it we were able to retrieve all the key bytes with just one fault injected into the encryption process. The figure also shows us that all the faults injected in this area are instruction skip attacks -all faulty bytes lead to key retrieval. Since the **AddRoundKey** lasts 48 clock cycles (16 load and 16 xor instructions), the laser glitch length in this case was $3\ \mu\text{s}$.

Fig. 9.10: Number of key bytes retrieved using different laser strength

In Figure 9.11, we can see the dependency on the position of the laser. In this

case, all the other parameters were fixed. The area that produces faults in all of 16 bytes is approximately $20 \times 55 \mu m^2$. large (around 0.012% of the whole chip area). Thus, it requires the precise location in order to determine the location where this fault occurs.

Fig. 9.11: Regions where faults occurred

It is worth mentioning that by using this attack model it is easy to break implementations with countermeasures which perform comparison check. For example, the countermeasure which performs encryption, decryption and then compare plaintexts in order to check for the errors. In this case, it is necessary to perform a second instruction skip attack during the first `AddRoundKey` in decryption phase, therefore the resulting plaintext will stay the same as the original input.

Also, some hardware countermeasures can be overcome by this technique, e.g. a light detection sensor on the front side of the chip or an energy sensor which detects a higher energy on the chip surface. Since the laser energy required for this type of

attack is low, the sensor would not be triggered.

9.2.4 Discussion

Laser-based instruction skip attack

For our practical work, we have evaluated a laser-based instruction skip fault attack technique on a microcontroller. The attack could be performed on any block ciphers, ideally with post key whitening and reversible key schedule. However, for other block ciphers, the attack could still be performed, by targeting different operations or different rounds, but it will require more skip instruction faults. The instruction skip technique has several advantages over data errors, usually induced in an SRAM or in the registers. The power of a laser required for such technique is much lower than the power needed to set/reset bits in memory. Hence, it can avoid detection by sensor, and lower the chance of damaging the device. Our experiments show a very high repeatability of such attack, together with a high precision of skipping particular instructions in microcontroller code. This fault can also be repeated or performed several times during the execution of the code and thus, it can bypass common countermeasures which perform comparison check or implement instruction redundancy.

In [164], an infective countermeasure for block cipher was proposed. The idea, as highlighted in Figure 9.12, is as follow: the cipher is processed with redundancy and randomized data, and under normal condition, the countermeasure will output correct ciphertext, whereas if the fault is injected, the random value will be returned, which will prevent successful cryptanalysis. The proposed method is further modified in [133], to include more formal proof, and to consider instruction skip attack. In the modified version, the order of execution is randomized, and they

show that it will further reduce the number of successful instruction skip fault, with the increasing number of dummy rounds. Here, the attacker model assumed for the countermeasure is quite simple, however, for practical attack, where the attacker could skip multiple instructions at different timing, *i.e.*, skip the random number generation as well, the attack could bypass the countermeasure, and such, it might be quite hard to protect in practice.

Fig. 9.12: Infective Countermeasure for Block Cipher [164]

In general, the countermeasure for instruction skip could usually be summarized as follow:

- redundancy countermeasure, which can be further categorized into:
 - information redundancy, for example, the data is encoded or transformed, and thus, without knowing the encoding scheme, the retrieved information might not be useful, and
 - instruction redundancy, for example, the instruction could be repeated several times or by using idempotent function, such that skipping any of them will not affect the output. However, the instruction redundancy might be prone to multiple skip instructions.

- randomization and shuffling of the order of operations, in such a way the attacker does not know when the targeted operations are executed.

These ideas for countermeasure are also highlighted in [134]. In order to prevent instruction skip attack, they add information redundancy in the form of appending the processed data with additional user known data (predefined plaintext and key), such that if any instruction is skipped, it can be detected by comparing with intermediate value of the known data.

Other issue regarding laser fault injection

Here, further investigation might be necessary if we want to realize other common attacks, as we have yet to achieve success rate in obtaining common fault model, such as bit flips or byte flips. One of the future aim is then to obtain such faults in order to replicate the theoretical result. Another future work is to check different platforms, such as on FPGA, which is quite commonly used for cryptographic implementation.

However, there could also be some issues that can be associated with the laser fault injection in general, which make laser fault attack harder to perform than other techniques of fault injections. It might be impossible to make a perfect profiling, since each chip has a different layout, different manufacturing process and even if we are aiming at the specific chip, de-packaging can cause small but significant differences on the surface which can result in a different fault sensitivity. The second problem is a chip survivability. The microcontroller used in our experiments was durable enough to withstand several weeks of experiments in a row without any observable damage. This could be due to the older manufacturing process, making the connections and transistors large. However, with more advanced technologies, the size is much smaller and therefore chips are more vulnerable to optical fault attacks.

Part IV

CONCLUSION

CONCLUSION

New types of attacks on cryptosystem have emerged, which target the physical properties of the cryptographic device. What differentiates these attacks from the classical cryptanalysis methods is that they do not necessarily rely on the theoretical weakness of the cryptographic algorithms. There have been various reports about these attacks carried out on real devices, which highlight the potential threats of these attacks. As such, these attacks, commonly referred as the physical attacks, have attracted a lot of attentions from the cryptographic community, and have been investigated thoroughly. In this thesis, the focus is emphasized on physical attacks, specifically on the cryptographic block ciphers. The research is mainly aimed at the non-invasive side-channel attacks, specifically on the profiling attacks, and the semi-invasive fault attacks. Different methods are investigated as alternative tools for conducting the attacks.

In the first part of the thesis, the focus is on the side-channel attacks. We investigated new methods for profiling which are based on the machine learning field. Machine learning and side-channel analysis are two different fields of study, yet they have common properties, in a sense that both are mostly dealing with the same problem (i.e., classification). We then conducted the experiments to compare different machine learning algorithms against classical profiling attacks. We showed that in the basic scenarios, the performance of machine learning methods are similar and are not necessarily better than the classical profiling methods such as TA, in

terms of the performance. However, under certain circumstances, such as the cases with irrelevant features or with non-Gaussian model, the machine learning methods can achieve a better performance. Hence, it remains a question, what is the most appropriate scenario to use the machine learning methods.

Next, we used the machine learning method during the phase of profiling, namely for the leakage modeling. The approach is based on support vector regression (SVR), which inherits the benefits of support vector machine such as kernel tricks to handle non-linear behavior and error tolerating soft margin in the learning process. Our first findings demonstrated that SVR can be used to construct a good leakage model. This was shown from the CPA result, that SVR can perform better than classical Hamming weight model and bitwise model. We also verified the performance, even under the noisy conditions. We then showed the effectiveness of the SVR based method on inter-bit dependencies of non-linear leakage properties by using kernel tricks.

In the next chapter, the main focus is on the hardware trojans. When employed on cryptographic device, they can be used to enhance physical attacks, such as increasing the SNR of the leakage, bypassing the side-channel countermeasure or being used as alternative tools to inject more precise faults. Different hardware trojan detection methods have been proposed in the literature. One of the possible way is to use the side-channel based methods, since in general, the modification of the chip can impact the physical leakage. However, there are common issues that need to be considered, such as the noise in the device or the requirement of a golden reference. In this work, we have adopted the machine learning based approach, one-class SVM, to construct a classifier for trojan detections based on the side-channel leakage. Based on the simulation results, we have investigated the effect of the learning parameters for the classification results. We considered different cases,

which was done to simulate the trojan behaviors on the chip. Also, we showed that the one-class SVM can be used even in more general case (not necessarily from normally distributed data). For the experiments on the FPGA platform, we adopted the majority voting method based on different sets of data. This method was shown to be able to improve the overall accuracy performances, based on the experiments conducted with the trojans of different sizes. We also showed that the method can be conducted in both supervised and unsupervised settings, which for this case, can help to mitigate the problem with golden chip.

In the second part of the thesis, the main focus is on the fault attacks. First, we presented an attack on a new block cipher, the LEA cipher. We proposed DFA exploiting the structure of the cipher, to reduce the number of faults required for the analysis. Based on a random bit flip model, aiming at the last two rounds of the cipher, the proposed attack was able to retrieve the secret key with around 258 faulty encryptions in average, and 62 faulty encryptions with more precise fault locations. The attack also highlighted the fault propagation behavior of the non-linear operation in the cipher, the modular addition operation. In the next work, we investigated the feasibility of laser fault injections for the practical attacks. The aim is to be able to conduct a simple yet efficient practical fault attacks using laser. For this purpose, we first performed a profiling using laser fault injection on the microcontroller, in order to determine the practical fault models. From the profiling, one of the fault behavior that can be observed is the skip instruction faults. After the profiling and determining correct parameters for the instruction skip attacks, we evaluated this technique by performing a DFA on AES. We were able to skip all the instructions associated with the last `AddRoundKey` operation, resulting to a wrong ciphertext. By simply performing exclusive-OR on this output with the correct ciphertext, we were able to retrieve the last round key and to use the inverse

key schedule in order to get the original secret key. This fault attack method was very easy to perform and extremely powerful since it needed only one correct and one faulty ciphertext in order to reveal the full AES key. It can also bypass some of the proposed countermeasures for the fault attacks.

Future Works

In this part, we would like to highlight what are the potential future works or directions. For side-channel attacks, the inclusion of machine learning methods provide an interesting alternative for classical profiling attacks. However, it might be argued that the capability of machine learning has not been fully utilized. First, this refers to the choice of the learning algorithms used. The algorithms used so far, including this work, are mostly based on commonly used methods. A wider exploration on different learning algorithms or a development of dedicated learning algorithm for side-channel analysis might increase the chance of having much better performance. One can argue that under current settings, the maximum likelihood distinguisher, the machine learning approach is on disadvantage. Take SVM as an example, it is optimized for binary classification. Though it is possible to extend to multi-class classification and force the output to return probabilistic prediction, it might not be optimal. Another argument is that, currently machine learning is used in side-channel attack mostly as a replacement for classification. However, based on our results that uses machine learning for leakage modeling, it might open a possibility to further investigate different aspects of side-channel which can be optimized using machine learning.

For hardware trojans, we would like to further investigate the combination of the machine learning approach with some of the existing preprocessing methods

to improve the classification results. Alternatively, we could investigate a better and more efficient parameter selection methods, since we can see that the performance can be directly tied to the choice of parameters used. Also, different outliers detection methods could be investigated.

For fault attacks, the future direction could be in the form of a more complete profiling of the microcontroller. In our work, we have observed the skip instruction behavior, which can be realized into attacks. However, more precise fault models such as single bit flip can be useful, especially since it can be used to verify theoretical attacks. Though we have shown practical results on microcontroller, investigation on different platforms, such as FPGA, could be done in the future. Another direction is that, for our recent works, the perturbation tool used is laser injection. However, other alternatives, such as EM probe, which are also popular because it mitigates the need for de-packaging the chip, could be investigated and compared. Alternatively, more advance methods can be used. This include multiple fault injections and combined attacks, such as combining the side-channel and fault attacks.

BIBLIOGRAPHY

- [1] *HOST 2011, Proceedings of the 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 5-6 June 2011, San Diego, California, USA*. IEEE Computer Society, 2011.
- [2] DPAContest V4. <http://www.dpacontest.org/v4/>, July 2013.
- [3] Onur Aciğmez and Werner Schindler. A Vulnerability in RSA Implementations Due to Instruction Cache Analysis and Its Demonstration on OpenSSL. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 256–273. Springer, 2008.
- [4] Sally Adee. The Hunt For The Kill Switch. *IEEE Spectr.*, 45(5):34–39, May 2008.
- [5] Wim Aerts, Eli Biham, Dieter De Moitie, Elke De Mulder, Orr Dunkelman, Sebastiaan Indesteege, Nathan Keller, Bart Preneel, Guy A. E. Vandenbosch, and Ingrid Verbauwhede. A Practical Attack on KeeLoq. *J. Cryptology*, 25(1):136–157, 2012.
- [6] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. Single-bit DFA using Multiple-byte Laser Fault Injection. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pages 113–119, Nov 2010.

-
- [7] Dakshi Agrawal, Selçuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan Detection using IC Fingerprinting. In *IEEE Symposium on Security and Privacy*, pages 296–310. IEEE Computer Society, 2007.
 - [8] Subidh Ali and Debdeep Mukhopadhyay. A Differential Fault Analysis on AES Key Schedule Using Single Fault. In Luca Breveglieri, Sylvain Guilley, Israel Koren, David Naccache, and Junko Takahashi, editors, *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, pages 35–42. IEEE Computer Society, 2011.
 - [9] Subidh Ali and Debdeep Mukhopadhyay. Improved Differential Fault Analysis of CLEFIA. In Fischer and Schmidt [66], pages 60–70.
 - [10] Yousra Alkabani. Trojan Immune Circuits Using Duality. In *DSD*, pages 177–184. IEEE, 2012.
 - [11] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.
 - [12] Dmitri Asonov and Rakesh Agrawal. Keyboard Acoustic Emanations. In *IEEE Symposium on Security and Privacy*, pages 3–11. IEEE Computer Society, 2004.
 - [13] Telecommunications Technology Association. 128-Bit Block Cipher LEA. *TTA Standard.TTAK.KO-12.0223*, 2013.

-
- [14] Jean-Philippe Aumasson. Cryptanalysis vs. Reality. https://media.blackhat.com/bh-ad-11/Aumasson/bh-ad-11-Aumasson-CryptanalysisVSReality_WP.pdf, 2011.
 - [15] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic Side-Channel Attacks on Printers. In *USENIX Security Symposium*, pages 307–322. USENIX Association, 2010.
 - [16] Jean-Claude Bajard, Laurent Imbert, Pierre-Yvan Liardet, and Yannick Teglia. Leak Resistant Arithmetic. In Joye and Quisquater [88], pages 62–75.
 - [17] Mainak Banga and Michael S. Hsiao. ODETTE: A Non-Scan Design-for-Test Methodology for Trojan Detection in ICs. In *HOST* [1], pages 18–23.
 - [18] Chongxi Bao, Domenic Forte, and Ankur Srivastava. On application of one-class SVM to reverse engineering-based hardware trojan detection. In *Fifteenth International Symposium on Quality Electronic Design, ISQED 2014, Santa Clara, CA, USA, March 3-5, 2014*, pages 47–54. IEEE, 2014.
 - [19] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.
 - [20] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient Template Attacks Based on Probabilistic Multi-Class Support Vector Machines. In Stefan Mangard, editor, *CARDIS*, volume 7771 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2012.
 - [21] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential Cluster

-
- Analysis. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2009.
- [22] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.
- [23] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Side-channel leakage and trace compression using normalized inter-class variance. In Ruby B. Lee and Weidong Shi, editors, *HASP 2014, Hardware and Architectural Support for Security and Privacy, Minneapolis, MN, USA, June 15, 2014*, pages 7:1–7:9. ACM, 2014.
- [24] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, Xuan Thuy Ngo, and Laurent Sauvage. Hardware Trojan Horses in Cryptographic IP Cores. In Fischer and Schmidt [66], pages 15–29.
- [25] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Jr. Kaliski, Burton S., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer Berlin Heidelberg, 1997.
- [26] Jeff Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical report, 1998.
- [27] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, New York, NY, USA, 1999.
- [28] Johannes Blömer and Jean-Pierre Seifert. Fault Based Cryptanalysis of the

- Advanced Encryption Standard (AES). In Rebecca N. Wright, editor, *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.
- [29] Hans-Hermann Bock. Origins and Extensions of the k-means Algorithm in Cluster Analysis. *Journal Électronique d'Histoire des Probabilités et de la Statistique [electronic only]*, 4(2):Article 14, 18 p., electronic only–Article 14, 18 p., electronic only, 2008.
- [30] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsøe. PRESENT: An Ultra-Lightweight Block Cipher. In Paillier and Verbaauwhede [132], pages 450–466.
- [31] Andrey Bogdanov, Kerem Varici, Nicky Mouha, Vesselin Velichkov, Elmar Tischhauser, Meiqin Wang, Deniz Toz, Qingju Wang, and Vincent Rijmen. Security Evaluation of the Block Cipher LEA. Technical report, July 2011.
- [32] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 37–51, Berlin, Heidelberg, 1997. Springer-Verlag.
- [33] Gianluca Bontempi, Mauro Birattari, and Hugues Bersini. Lazy Learning: A Local Method for Supervised Learning. In L. C. Jain and J. Kacprzyk, editors, *New Learning Paradigms in Soft Computing*, pages 97–137. Springer-Verlag, Heidelberg, Germany, 2001.

-
- [34] Janez Brank, Marko Grobelnik, Nataša Milic-Frayling, and Dunja Mladenic. Feature Selection Using Support Vector Machines. Technical Report MSR-TR-2002-63, Microsoft Research, 2002.
 - [35] Jakub Breier and Dirmanto Jap. A Survey of State-of-the-Art Fault Attacks. In *International Symposium on Integrated Circuits (ISIC)*, pages 152 – 155, Singapore, Dec 2014.
 - [36] Jakub Breier and Dirmanto Jap. Testing Feasibility of Back-Side Laser Fault Injection on a Microcontroller. In Stavros Koubias and Thilo Sauter, editors, *Proceedings of the 10th Workshop on Embedded Systems Security, WESS 2015, Amsterdam, The Netherlands, October 8, 2015*, page 5. ACM, 2015.
 - [37] Jakub Breier, Dirmanto Jap, and Chien-Ning Chen. Laser Profiling for the Back-Side Fault Attacks: With a Practical Laser Skip Instruction Attack on AES. In Jianying Zhou and Douglas Jones, editors, *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, CPSS 2015, Singapore, Republic of Singapore, April 14 - March 14, 2015*, pages 99–103. ACM, 2015.
 - [38] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.
 - [39] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
 - [40] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Joye and Quisquater [88], pages 16–29.
 - [41] Eric Brier, David Naccache, Phong Q. Nguyen, and Mehdi Tibouchi. Modulus Fault Attacks against RSA-CRT Signatures. In Preneel and Takagi [139], pages 192–206.

-
- [42] David Brumley and Dan Boneh. Remote Timing Attacks Are Practical. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.
- [43] Rajat Subhra Chakraborty and Swarup Bhunia. Security Against Hardware Trojan Attacks Using Key-Based Design Obfuscation. *J. Electronic Testing*, 27(6):767–785, 2011.
- [44] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware Trojan: Threats and Emerging Solutions. In *HLDVT*, pages 166–171. IEEE, 2009.
- [45] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [46] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
- [47] Hua Chen, Wenling Wu, and Dengguo Feng. Differential Fault Analysis on CLEFIA. In Sihan Qing, Hideki Imai, and Guilin Wang, editors, *Information and Communications Security, 9th International Conference, ICICS 2007, Zhengzhou, China, December 12-15, 2007, Proceedings*, volume 4861 of *Lecture Notes in Computer Science*, pages 284–295. Springer, 2007.
- [48] Vladimir Cherkassky and Yunqian Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural Networks*, 17(1):113–126, 2004.

-
- [49] Christophe Clavier and Antoine Wurcker. Reverse Engineering of a Secret AES-like Cipher by Ineffective Fault Analysis. In Fischer and Schmidt [66], pages 119–128.
- [50] Jean-Sébastien Coron and Louis Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In Koç and Paar [94], pages 231–237.
- [51] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.
- [52] Franck Courbon, Philippe Loubet-Moundi, Jacques J. A. Fournier, and Assia Tria. Increasing the Efficiency of Laser Fault Injections using Fast Gate Level Reverse Engineering. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 60–63, May 2014.
- [53] Franck Courbon, Philippe Loubet-Moundi, Jacques J.A. Fournier, and Assia Tria. Adjusting Laser Injections for Fully Controlled Faults. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design*, Lecture Notes in Computer Science, pages 229–242. Springer International Publishing, 2014.
- [54] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
- [55] Peter Dayan, Maneesh Sahani, and Grégoire Deback. Unsupervised Learning. In *The MIT Encyclopedia of the Cognitive Sciences*. The MIT Press, 1999.
- [56] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A Practical Attack on the MIFARE Classic. In Gilles Grimaud and François-Xavier

-
- Standaert, editors, *CARDIS*, volume 5189 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2008.
- [57] Dariusz Dereniowski and Marek Kubale. Cholesky Factorization of Matrices in Parallel and Ranking of Graphs. In Roman Wyrzykowski, Jack Dongarra, Marcin Paprzycki, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics, 5th International Conference, PPAM 2003, Czestochowa, Poland, September 7-10, 2003. Revised Papers*, volume 3019 of *Lecture Notes in Computer Science*, pages 985–992. Springer, 2003.
- [58] Dongdong Du, Seetharam Narasimhan, Rajat Subhra Chakraborty, and Swarup Bhunia. Self-Referencing: A Scalable Side-Channel Approach for Hardware Trojan Detection. In Mangard and Standaert [115], pages 173–187.
- [59] Kaibo Duan and S. Sathya Keerthi. Which Is the Best Multiclass SVM Method? An Empirical Study. In Nikunj C. Oza, Robi Polikar, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, volume 3541 of *Lecture Notes in Computer Science*, pages 278–285. Springer, 2005.
- [60] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016.
- [61] Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, and Assia Tria. Reproducible Single-byte Laser Fault Injection. In *Ph.D. Research in Micro-*

-
- electronics and Electronics (PRIME), 2010 Conference on*, pages 1–4, July 2010.
- [62] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. Private Circuits III: Hardware Trojan-Resilience via Testing Amplification. <http://perso.uclouvain.be/fstandae/PUBLIS/177.pdf>, 2016.
- [63] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2008.
- [64] M. Abdelaziz Elaabid, Sylvain Guilley, and Philippe Hoogvorst. Template Attacks with a Power Model. *IACR Cryptology ePrint Archive*, 2007:443, 2007.
- [65] Xitao Fan, Stephen Sivo, and Sean Keenan. *SAS for Monte Carlo Studies: A Guide for Quantitative Researchers*. SAS Institute, 2002.
- [66] Wieland Fischer and Jörn-Marc Schmidt, editors. *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*. IEEE, 2013.
- [67] Catherine H. Gebotys, Simon Ho, and C. C. Tiu. EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA. In Rao and Sunar [143], pages 250–264.
- [68] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International*

-
- Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.
- [69] Christophe Giraud. DFA on AES. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.
- [70] Daniel M. Gordon. A Survey of Fast Exponentiation Methods. *J. Algorithms*, 27(1):129–146, 1998.
- [71] Martin A. Green and Mark J. Keevers. Optical Properties of Intrinsic Silicon at 300 K. *Progress in Photovoltaics: Research and Applications*, 3(3):189–192, 1995.
- [72] Dawu Gu, Juanru Li, Sheng Li, Zhouqian Ma, Zheng Guo, and Junrong Liu. Differential Fault Analysis on Lightweight Blockciphers with Statistical Cryptanalysis Techniques. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 27–33. IEEE Computer Society, 2012.
- [73] Ludovic Gustin, François Durvaux, Stéphanie Kerckhof, François-Xavier Standaert, and Michel Verleysen. Support Vector Machines for Improved IP Detection with Soft Physical Hash Functions. In *COSADE*, 2014.
- [74] John A. Hartigan and M.A. Wong. Algorithm AS 136: A K-means Clustering Algorithm. *Applied Statistics*, pages 100–108, 1979.

-
- [75] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2 edition, 2009.
- [76] Alexander Herrmann and Marc Stöttinger. Evaluation Tools for Multivariate Side-Channel Analysis. In Andy D. Pimentel, Stephan Wong, Gerardo Pelosi, Israel Koren, Giovanni Agosta, and Alessandro Barenghi, editors, *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC 2015, Amsterdam, Netherlands, January 19-21, 2015*, pages 1–6. ACM, 2015.
- [77] Annelie Heuser, Werner Schindler, and Marc Stöttinger. Revealing Side-Channel Issues of Complex Circuits by Enhanced Leakage Models. In *DATE*, pages 1179–1184, March 2012.
- [78] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Schindler and Huss [151], pages 249–264.
- [79] Howard M. Heys. A Tutorial on Linear and Differential Cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.
- [80] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations. In *CARDIS*, 2013.
- [81] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, KwonHo Ryu, and Dong-Geon Lee. LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig,

-
- editors, *Information Security Applications*, volume 8267 of *Lecture Notes in Computer Science*, pages 3–27. Springer International Publishing, 2014.
- [82] Gabriel Hospodar, Elke de Mulder, Benedikt Gierlichs, Ingrid Verbauwhede, and Joos Vandewalle. Least Squares Support Vector Machines for Side-Channel Analysis. In *COSADE*, 2011.
- [83] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A Practical Guide to Support Vector Classification. Technical report, Department of Computer Science, National Taiwan University, 2003.
- [84] Michael Hutter, Stefan Mangard, and Martin Feldhofer. Power and EM Attacks on Passive 13.56 MHz RFID Devices. In Paillier and Verbauwhede [132], pages 320–333.
- [85] Dirmanto Jap and Jakub Breier. Overview of Machine Learning based Side-Channel Analysis Methods. In *International Symposium on Integrated Circuits (ISIC)*, pages 38–41, Singapore, Dec 2014.
- [86] Dirmanto Jap and Jakub Breier. Differential Fault Attack on LEA. In Ismail Khalil, Erich Neuhold, A Min Tjoa, Li Da Xu, and Ilsun You, editors, *Information and Communication Technology - Third IFIP TC 5/8 International Conference, ICT-EurAsia 2015, and 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, Held as Part of WCC 2015, Daejeon, Korea, October 4-7, 2015, Proceedings*, volume 9357 of *Lecture Notes in Computer Science*, pages 265–274. Springer, 2015.
- [87] Dirmanto Jap, Marc Stöttinger, and Shivam Bhasin. Support Vector Regression: Exploiting Machine Learning Techniques for Leakage Modeling. In Ruby B. Lee, Weidong Shi, and Jakub Szefer, editors, *Proceedings of the Fourth*

-
- Workshop on Hardware and Architectural Support for Security and Privacy, Portland, OR, USA, June 14, 2015*, pages 2:1–2:8. ACM, 2015.
- [88] Marc Joye and Jean-Jacques Quisquater, editors. *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*. Springer, 2004.
- [89] Chris Karlof and David Wagner. Hidden Markov Model Cryptanalysis. In Walter et al. [167], pages 17–34.
- [90] Markus Kasper, Amir Moradi, Georg T. Becker, Oliver Mischke, Tim Güneysu, Christof Paar, and Wayne Burleson. Side Channels as Building Blocks. *J. Cryptographic Engineering*, 2(3):143–159, 2012.
- [91] Leonard Kaufman and Peter Rousseeuw. *Clustering by Means of Medoids*. Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. Fac., Univ., 1987.
- [92] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, January 1883.
- [93] Desmond A. Kirkpatrick and Alberto L. Sangiovanni-Vincentelli. Techniques for Crosstalk Avoidance in the Physical Design of High-performance Digital Systems. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design, ICCAD '94*, pages 616–619, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
- [94] Çetin Kaya Koç and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester,*

-
- MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*. Springer, 2000.
- [95] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
- [96] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [97] François Koeune and François-Xavier Standaert. A Tutorial on Physical Security and Side-Channel Attacks. In Alessandro Aldini, Roberto Gorrieri, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design III, FOSAD 2004/2005 Tutorial Lectures*, volume 3655 of *Lecture Notes in Computer Science*, pages 78–108. Springer, 2004.
- [98] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1137–1145. Morgan Kaufmann, 1995.
- [99] Teuvo Kohonen, Manfred R. Schroeder, and Thomas S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.
- [100] Thomas Korak, Thomas Plos, and Michael Hutter. Attacking an AES-Enabled NFC Tag: Implications from Design to a Real-World Scenario. In Schindler and Huss [151], pages 17–32.

-
- [101] Sotiris B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. *Informatica (Slovenia)*, 31(3):249–268, 2007.
- [102] David Kriesel. *A Brief Introduction to Neural Networks*. Zeta2 edition, 2007.
- [103] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, Andy Rupp, and Manfred Schimmler. How to Break DES for EUR 8,980. In *International Workshop on Special-Purpose Hardware for Attacking Cryptographic Systems — SHARCS’06, Cologne, Germany*, <http://www.sharcs.org>, April 2006.
- [104] Sebastian Kutzner, Axel Y. Poschmann, and Marc Stöttinger. Hardware Trojan Design and Detection: A Practical Evaluation. In *Proceedings of the Workshop on Embedded Systems Security, WESS ’13*, pages 1:1–1:9, New York, NY, USA, 2013. ACM.
- [105] Kerstin Lemke-Rust and Christof Paar. Gaussian Mixture Models for Higher-Order Side Channel Analysis. In Paillier and Verbauwhede [132], pages 14–27.
- [106] Liran Lerman, Gianluca Bontempi, and Oliver Markowitsch. Side Channel Attack: an Approach based on Machine Learning. In *COSADE*, 2011.
- [107] Liran Lerman, Gianluca Bontempi, Souhaib Ben Taieb, and Olivier Markowitch. A Time Series Approach for Profiling Attack. In Benedikt Gierlichs, Sylvain Guilley, and Debdeep Mukhopadhyay, editors, *SPACE*, volume 8204 of *Lecture Notes in Computer Science*, pages 75–94. Springer, 2013.
- [108] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS*, 2013.

-
- [109] Liran Lerman, Stephane Fernandes Medeiros, Nikita Veshchikov, Cédric Meuter, Gianluca Bontempi, and Olivier Markowitch. Semi-Supervised Template Attack. In *COSADE*, 2013.
 - [110] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.
 - [111] Min Li, Azadeh Davoodi, and Mohammad Tehranipoor. A Sensor-Assisted Self-Authentication Framework for Hardware Trojan Detection. In Rosenstiel and Thiele [149], pages 1331–1336.
 - [112] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault Sensitivity Analysis. In Mangard and Standaert [115], pages 320–334.
 - [113] Lang Lin, Wayne Burleson, and Christof Paar. MOLES: Malicious Off-Chip Leakage Enabled by Side-Channels. In *ICCAD*, pages 117–122. IEEE, 2009.
 - [114] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
 - [115] Stefan Mangard and François-Xavier Standaert, editors. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*. Springer, 2010.

-
- [116] John Markoff. Old Trick Threatens the Newest Weapons. http://www.nytimes.com/2009/10/27/science/27trojan.html?pagewanted=all&_r=0, 2009.
- [117] Zdenek Martinasek, Jan Hajny, and Lukas Malina. Optimization of Power Analysis Using Neural Network. In *CARDIS*, 2013.
- [118] Zdenek Martinasek and Vaclav Zeman. Innovative Method of Power Analysis. *Radioengineering*, 22(2):586–594, 2013.
- [119] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Koç and Paar [94], pages 238–251.
- [120] Tom M. Mitchell. *Machine Learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.
- [121] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8), April 1965.
- [122] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
- [123] Seetharam Narasimhan, Xinmu Wang, Dongdong Du, Rajat Subhra Chakraborty, and Swarup Bhunia. TeSR: A Robust Temporal Self-Referencing Approach for Hardware Trojan Detection. In *HOST* [1], pages 71–74.
- [124] Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. RSM: A Small and Fast Countermeasure for AES, Secure Against 1st and 2nd-Order Zero-Offset SCAs. In Rosenstiel and Thiele [149], pages 1173–1178.
- [125] National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. October 1999. supersedes FIPS 46-2.

-
- [126] Xuan Thuy Ngo, Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Linear complementary dual code improvement to strengthen encoded circuit against hardware trojan horses. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015*, pages 82–87. IEEE, 2015.
- [127] Xuan Thuy Ngo, Sylvain Guilley, Shivam Bhasin, Jean-Luc Danger, and Zakaria Najm. Encoding the state of integrated circuits: a proactive and reactive protection against hardware trojans horses. In Ting Yu and Shengqi Yang, editors, *Proceedings of the 9th Workshop on Embedded Systems Security, WESS '14, New Delhi, India, October 17, 2014*, pages 7:1–7:10. ACM, 2014.
- [128] Xuan Thuy Ngo, Zakaria Najm, Shivam Bhasin, Sylvain Guilley, and Jean-Luc Danger. Method taking into account process dispersion to detect hardware trojan horse by side-channel analysis. *J. Cryptographic Engineering*, 6(3):239–247, 2016.
- [129] David Oswald and Christof Paar. Breaking MIFARE DESFire MF3ICD40: Power Analysis and Templates in the Real World. In Preneel and Takagi [139], pages 207–222.
- [130] Elisabeth Oswald and Manfred Josef Aigner. Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2001.

-
- [131] Martin Otto. *Fault Attacks And Countermeasures*. PhD thesis, University of Paderborn, 2005.
- [132] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*. Springer, 2007.
- [133] Sikhar Patranabis, Abhishek Chakraborty, and Debdeep Mukhopadhyay. Fault tolerant infective countermeasure for AES. *IACR Cryptology ePrint Archive*, 2015:493, 2015.
- [134] Conor Patrick, Bilgiday Yuce, Nahid Farhady Ghalaty, and Patrick Schaulmont. Lightweight Fault Attack Resistance in Software Using Intra-Instruction Redundancy. *Selected Areas in Cryptography (SAC 2016)*, to appear.
- [135] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved Higher-Order Side-Channel Attacks with FPGA experiments. In Rao and Sunar [143], pages 309–323.
- [136] Hanchuan Peng, Fuhui Long, and Chris H. Q. Ding. Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(8):1226–1238, 2005.
- [137] Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine. Attacking Randomized Exponentiations Using Unsupervised Learning. In *COSADE*, 2014.

-
- [138] Gilles Piret and Jean-Jacques Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In Walter et al. [167], pages 77–88.
- [139] Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011.
- [140] Emmanuel Prouff. DPA Attacks and S-Boxes. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 2005.
- [141] Jean-Jacques Quisquater and David Samyde. Automatic Code Recognition for Smartcards Using a Kohonen Neural Network. In *CARDIS*. USENIX, 2002.
- [142] Lawrence R. Rabiner. Readings in Speech Recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
- [143] Josyula R. Rao and Berk Sunar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.
- [144] Christian Rechberger and Elisabeth Oswald. Practical Template Attacks. In

-
- Chae Hoon Lim and Moti Yung, editors, *WISA*, volume 3325 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2004.
- [145] Irina Rish. An Empirical Study of the Naive Bayes Classifier. Technical report, 2001.
- [146] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
- [147] Cyril Roscian, Jean-Max Dutertre, and Assia Tria. Frontside Laser Fault Injection on Cryptosystems - Application to the AES' Last Round. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 119–124, June 2013.
- [148] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells. In Fischer and Schmidt [66], pages 89–98.
- [149] Wolfgang Rosenstiel and Lothar Thiele, editors. *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*. IEEE, 2012.
- [150] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A Diagonal Fault Attack on the Advanced Encryption Standard. *IACR Cryptology ePrint Archive*, 2009:581, 2009.
- [151] Werner Schindler and Sorin A. Huss, editors. *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012*,

-
- Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*. Springer, 2012.
- [152] Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In Rao and Sunar [143], pages 30–46.
- [153] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, and John C. Platt. Support Vector Method for Novelty Detection. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 582–588. The MIT Press, 1999.
- [154] Adi Shamir and Eran Tromer. Acoustic cryptanalysis: On nosy people and noisy machines. <http://tau.ac.il/~tromer/acoustic/>.
- [155] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.
- [156] Lindsay I Smith. A Tutorial on Principal Components Analysis. Technical report, Cornell University, USA, February 26 2002.
- [157] Alexander J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.
- [158] Youssef Souissi, Maxime Nassar, Sylvain Guilley, Jean-Luc Danger, and Florent Flament. First Principal Components Analysis: A New Side Channel

-
- Distinguisher. In Kyung Hyune Rhee and DaeHun Nyang, editors, *ICISC*, volume 6829 of *Lecture Notes in Computer Science*, pages 407–419. Springer, 2010.
- [159] François-Xavier Standaert, Tal Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.
- [160] Oliver Sutton. Introduction to k Nearest Neighbour Classification and Condensed Nearest Neighbour Data Reduction, February 2012.
- [161] Johan A. K. Suykens and Joos Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.
- [162] Mohammad Tehranipoor and Farinaz Koushanfar. A Survey of Hardware Trojan Taxonomy and Detection. *IEEE Design & Test of Computers*, 27(1):10–25, 2010.
- [163] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In Claudio Agostino Ardagna and Jianying Zhou, editors, *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings*, volume 6633 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2011.
- [164] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying Fault Invariant with Randomization - A Countermeasure for AES Against Differential Fault Attacks. In Lejla Batina and Matthew Robshaw, editors,

-
- Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2014.
- [165] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Differential Fault Analysis on the Families of SIMON and SPECK Ciphers. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 40–48, Sept 2014.
- [166] IB Vapnyarskii. Lagrange Multipliers. *Hazewinkel, Michiel, Encyclopedia of Mathematics, Springer, ISBN, 978:1–55608*, 2001.
- [167] Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*. Springer, 2003.
- [168] Gaoli Wang and Shaohui Wang. Differential Fault Analysis on PRESENT Key Schedule. In Muren Liu, Yuping Wang, and Ping Guo, editors, *2010 International Conference on Computational Intelligence and Security, CIS 2010, Nanning, Guangxi Zhuang Autonomous Region, China, December 11-14, 2010*, pages 362–366. IEEE, 2010.
- [169] Xinmu Wang, Seetharam Narasimhan, Aswin Raghav Krishna, and Swarup Bhunia. SCARE: Side-Channel Analysis Based Reverse Engineering for Post-Silicon Validation. In Vishwani D. Agrawal and Srimat T. Chakradhar, editors, *VLSI Design*, pages 304–309. IEEE, 2012.
- [170] Rick Wash. Lecture Notes on Stream Ciphers and RC4.

-
- [171] Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The Myth of Generic DPA...and the Magic of Learning. *IACR Cryptology ePrint Archive*, 2012:256, 2012.
- [172] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability Estimates for Multi-Class Classification by Pairwise Coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.
- [173] Sarah Yang. Researchers recover typed text using audio recording of keystrokes. http://www.berkeley.edu/news/media/releases/2005/09/14_key.shtml, 2005.
- [174] Sung-Ming Yen and Marc Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.
- [175] Xuehui Zhang, Andrew Ferraiuolo, and Mohammad Tehranipoor. Detection of Trojans Using a Combined Ring Oscillator Network and Off-Chip Transient Power Analysis. *JETC*, 9(3):25, 2013.
- [176] Xiaojin Zhu. Semi-Supervised Learning. In *Encyclopedia of Machine Learning*, pages 892–897. 2010.
- [177] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard Acoustic Emanations Revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1):3:1–3:26, November 2009.