Strive to Fail

Deep Learning-based Side-channel Analysis for Evaluators

Azade Rezaeezade

Strive to Fail

Deep Learning-based Side-channel Analysis for Evaluators

Strive to Fail

Deep Learning-based Side-channel Analysis for Evaluators

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus,
prof. dr. ir. H. Bijl,
chair of the Board for Doctorates
to be defended publicly on
Monday, 15 June 2026 at 12:30

by

Azade REZAEEZADE

This dissertation has been approved by the (co)promotors. Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof.dr.ir. R.L. Lagendijk,	Delft University of Technology, *promotor*
Prof.dr. L. Batina,	Radboud University, *promotor*
Dr. S. Picek,	Radboud University, *copromotor*
Independent members:	
Prof.dr. G. Smaragdakis,	Delft University of Technology
Prof.dr.ir. N. Mentens,	Leiden University, The Netherlands
Prof.dr. F.X. Standaert,	UCLouvain, Belgium
Prof.dr.-Ing. S. Mangard,	Graz University of Technology, Austria
Prof.dr. F.A. Oliehoek,	Delft University of Technology, *reserve member*

Keywords: Side-channel Analysis, Deep Learning, Deep Learning-based Side-channel Analysis, Physical Security Evaluation, Hardware Security

Printed by: Ipskamp Printing

Cover by: Didi Grootjen

ISBN 978-94-6518-330-5

An electronic copy of this dissertation is available at
https://repository.tudelft.nl/.

Contents

Summary xi

Samenvatting xiii

1 Introduction 1
- 1.1 Cryptography . . . 2
- 1.2 Implementation Curse . . . 4
- 1.3 Side-channel Analysis . . . 6
  - 1.3.1 Security Evaluation . . . 8
- 1.4 Problem Statement . . . 9
  - 1.4.1 Security Evaluation and Deep Learning Usage Implications . . . 9
  - 1.4.2 Research Questions . . . 10
- 1.5 Contributions . . . 11

2 Preliminary 15
- 2.1 Covered Cryptographic Algorithms . . . 15
  - 2.1.1 AES . . . 15
  - 2.1.2 Ascon . . . 16
  - 2.1.3 Kyber . . . 17
- 2.2 Attack Points . . . 18
  - 2.2.1 AES Sbox Output . . . 19
  - 2.2.2 Ascon-128a Sbox Output . . . 20
  - 2.2.3 Pair-pointwise Multiplication in Kyber . . . 21
- 2.3 Leakage Model . . . 22
- 2.4 Countermeasures . . . 24
- 2.5 Machine and Deep Learning . . . 26
  - 2.5.1 Learning Algorithms . . . 26
  - 2.5.2 Deep Neural Networks . . . 28
- 2.6 Deep Learning-based Side-channel Analysis . . . 31
  - 2.6.1 DL-SCA as a Classification Task . . . 31
  - 2.6.2 Evaluating DL-SCA Performance . . . 32
  - 2.6.3 DL-SCA Experience Datasets . . . 34

3 DL-SCA Performance and the Influence of More Training Examples 39
- 3.1 Introduction . . . 40
- 3.2 Deep Double Descent Phenomenon . . . 41
- 3.3 Overfitting and Generalization in Side-channel Analysis . . . 44
- 3.4 Analysis Methodology . . . 45

3.5 Experimental Results . . . 49
3.5.1 ASCAD-R Dataset . . . 49
3.5.2 AES-HD Dataset . . . 58
3.5.3 Influence of Regularization Techniques . . . 60
3.5.4 Reducing Assumption Error by Changing Hyperparameters . . . 60
3.6 General Observations . . . 61
3.7 Conclusions and Future Work . . . 62

4 Fighting Overfitting in DL-SCA Using Regularization 63
4.1 Introduction . . . 64
4.2 Regularization Techniques . . . 65
4.3 Analysis Methodology . . . 67
4.4 Experimental Results . . . 71
4.4.1 Performance Comparison . . . 71
4.4.2 Deterioration Rate . . . 78
4.4.3 Profiling Time Changes . . . 79
4.5 Discussion . . . 79
4.5.1 Different Techniques Effectiveness in General . . . 81
4.5.2 Implicit and Explicit Regularization Comparison . . . 82
4.5.3 Regularizer Benefit When *GE* Is One . . . 84
4.6 Conclusions and Future Work . . . 85

5 JumpReLU Activation Function in DL-SCA 87
5.1 Introduction . . . 87
5.1.1 Activation Functions . . . 88
5.2 Analysis Methodology . . . 90
5.3 Experimental Results . . . 93
5.3.1 First Scenario: Effect on Well-known Models . . . 93
5.3.2 First Scenario: Effect on Random Models . . . 94
5.4 Conclusions and Future Work . . . 97

6 Ensemble-based DL-SCA 99
6.1 Introduction . . . 99
6.2 Ensembles . . . 101
6.3 Methodology . . . 102
6.4 Experimental Results . . . 104
6.4.1 Ascon-Unprotected . . . 104
6.4.2 Ascon-Protected . . . 106
6.4.3 Comparing with attacking AES: . . . 109
6.5 Conclusions and Future Work . . . 109

7 Leakage Model-flexible DL-SCA 111
7.1 Introduction . . . 112
7.2 Multi-Task Learning . . . 114
7.3 Leakage Model-flexible . . . 115
7.3.1 Physical Leakages Estimation . . . 115

7.3.2 Multi-bit Model . . . 116
7.3.3 Multi-byte Multi-bit DL-SCA . . . 119
7.4 Experimental Results . . . 121
7.4.1 Performance Evaluation . . . 122
7.4.2 Leakage Assessment with the Multi-bit Model . . . 124
7.4.3 Case Study of the SMAesH Challenge . . . 125
7.4.4 Hyperparameter Study . . . 127
7.5 Discussion . . . 130
7.6 Conclusions and Future Work . . . 131

8 Deep Learning-based Blind Side-channel Analysis 133
8.1 Introduction . . . 133
8.2 Blind Side-channel Attacks . . . 136
8.2.1 Gaussian Mixture Model . . . 140
8.3 Methodology . . . 141
8.3.1 Threat Model . . . 141
8.3.2 Deep Learning-based Blind Side-channel Attack . . . 141
8.3.3 Multi-point Cluster-based Labeling . . . 143
8.4 Experimental Results . . . 146
8.4.1 ChipWhisperer . . . 146
8.4.2 Kyber . . . 147
8.4.3 Ascon . . . 151
8.4.4 Targeting Countermeasures: Desynchronized CW Dataset. . . 156
8.5 Discussion . . . 158
8.5.1 Accuracy . . . 158
8.5.2 Tuning the number of PoIs . . . 161
8.5.3 Limitations . . . 161
8.6 Conclusions and Future Work . . . 161

9 Discussion 163
9.1 Distinct Characteristics of Deep Learning Usage in the SCA Domain 166
9.2 Future Directions . . . 169

Acknowledgements 193

List of Publications 195

Summary

Digital devices are now deeply embedded in modern life. These devices process sensitive information, including personal data, financial records, and data related to critical infrastructure. Cryptography is therefore a fundamental component of digital security, providing confidentiality, integrity, authentication, key exchange, and digital signatures.

Although cryptographic algorithms are designed to be mathematically secure, their physical implementations can introduce vulnerabilities. When cryptographic algorithms run on hardware, devices unintentionally leak information through side channels such as power consumption, electromagnetic radiation, and timing behavior. These leakages can be exploited through side-channel analysis to recover secret information, including cryptographic keys.

Security evaluation laboratories assess the resistance of cryptographic implementations against such attacks. However, this process is costly and must strike a balance between thoroughness and practical limitations on time, budget, data, and computational resources. Deep learning-based side-channel analysis (DL-SCA) is attractive in this context because neural networks can learn leakage characteristics directly from traces, reducing the need for manual preprocessing and explicit statistical assumptions. At the same time, deep learning introduces new costs, caused by sensitivity to neural network hyperparameter selection, instability, and overfitting in its training process.

The central problem addressed in this thesis is the tension between the benefits and costs of deep learning in side-channel evaluation. On the one hand, deep learning can reduce evaluation effort by relaxing assumptions about leakage models and reducing dependence on known data. On the other hand, it can make evaluation more expensive due to model selection, hyperparameter tuning, and the risk of overfitting. This thesis investigates how DL-SCA can be made more practical, reliable, and cost-effective for security evaluation workflows.

To this end, the thesis studies several strategies for improving DL-SCA without relying on excessive hyperparameter tuning. It examines the impact of increasing the amount of training data, regularization techniques, and ensemble learning. These approaches aim to improve generalization, robustness, and attack stability under realistic evaluation constraints. The thesis also investigates two deep learning approaches that relax major assumptions in classical SCA: leakage model-flexible DL-SCA, which avoids relying on fixed leakage models such as Hamming weight or identity, and deep learning-based blind SCA, which reduces dependence on plaintext or ciphertext by learning from noisy labels.

Samenvatting

Digitale apparaten zijn tegenwoordig diep verweven met het moderne leven. Deze apparaten verwerken gevoelige informatie, waaronder persoonsgegevens, financiële gegevens en data die verband houden met kritieke infrastructuur. Cryptografie is daarom een fundamenteel onderdeel van digitale beveiliging en biedt vertrouwelijkheid, integriteit, authenticatie, sleuteluitwisseling en digitale handtekeningen.

Hoewel cryptografische algoritmen zijn ontworpen om wiskundig veilig te zijn, kunnen hun fysieke implementaties kwetsbaarheden introduceren. Wanneer cryptografische algoritmen op hardware worden uitgevoerd, lekken apparaten onbedoeld informatie via zijkanalen, zoals stroomverbruik, elektromagnetische straling en timinggedrag. Deze lekken kunnen via zijkanaalanalyse worden misbruikt om geheime informatie, waaronder cryptografische sleutels, te achterhalen.

Laboratoria die de beveiliging van producten evalueren beoordelen de weerstand van cryptografische implementaties tegen dergelijke aanvallen. Dit proces is echter kostbaar en vereist een balans tussen grondigheid en praktische beperkingen op het gebied van tijd, budget, data en rekenmiddelen. Deep learning-gebaseerde zijkanaalanalyse (DL-SCA) is in deze context aantrekkelijk, omdat neurale netwerken lekkagekarakteristieken rechtstreeks uit signaalmetingen kunnen leren, waardoor er minder behoefte is aan handmatige preprocessing en expliciete statistische aannames. Tegelijkertijd introduceert deep learning nieuwe problemen zoals gevoeligheid voor de keuze van hyperparameters van neurale netwerken, en door instabiliteit en overfitting in hun trainingsproces.

Het centrale probleem dat in dit proefschrift wordt behandeld, is de spanning tussen de voordelen en problemen van deep learning in zijkanaalevaluatie. Enerzijds kan deep learning de evaluatie-inspanning verminderen door aannames over lekkagemodellen te versoepelen en de afhankelijkheid van bekende data te verkleinen. Anderzijds kan het de evaluatie duurder maken door modelselectie, hyperparameterafstemming en overfitting. Dit proefschrift onderzoekt hoe DL-SCA praktischer, betrouwbaarder en kosteneffectiever kan worden gemaakt in beveiligingsevaluaties.

Daartoe bestudeert dit proefschrift verschillende strategieën om DL-SCA te verbeteren zonder te vertrouwen op overmatige hyperparameterafstemming. Het onderzoekt de invloed van het vergroten van de hoeveelheid trainingsdata, regularisatietechnieken en ensemble learning. Deze benaderingen zijn gericht op het verbeteren van generalisatie, robuustheid en aanvalsstabiliteit onder realistische evaluatiebeperkingen. Daarnaast onderzoekt dit proefschrift twee deep learning-benaderingen die belangrijke aannames in klassieke SCA versoepelen: lekkagemodel-flexibele DL-SCA, die voorkomt dat men afhankelijk is van vaste lekkagemodellen zoals Hamming weight of identity, en deep learning-gebaseerde blinde SCA, die de afhankelijkheid van plaintext of ciphertext vermindert door te leren van ruisachtige labels.

1 Introduction

Azade Rezaeezade

The deep integration of digital devices into our daily lives is no longer a topic that requires explanation. From routine household tasks to the complex operations in industries and research institutions, a vast range of activities now depends on an ever-growing ecosystem of digital systems. These systems span from tiny Internet of Things (IoT) and edge devices to large-scale high-performance computing clusters and supercomputers. The number of digital devices surrounding us is overwhelming. A simple look around reveals many objects, such as phones, watches, appliances, and vehicles, which are equipped with embedded processors performing specialized functions, often without our direct awareness.

What makes this integration concerning is the importance of the information processed, stored, or transmitted using these devices. It includes highly sensitive data such as personal identifiers, financial records, and information critical to the infrastructure's security. To protect such data, manufacturers and service providers employ multiple layers of defense, among which cryptographic algorithms play a fundamental role. Cryptography provides essential security properties such as confidentiality, integrity, and authentication, making it a cornerstone of secure digital communication.

However, while widely used cryptographic algorithms are mathematically sound and resistant to known theoretical attacks using current technology, their implementations introduce new classes of vulnerabilities. These vulnerabilities arise from the physical characteristics and behavior of the devices executing cryptographic algorithms, which results in a phenomenon known as side-channel leakage. The threat is that an attacker can infer secret information, including cryptographic keys, through side channels such as power consumption, electromagnetic emissions, execution timing, or memory access patterns.

Embedded devices, which are resource-constrained specialized computing systems designed to perform specific functions within larger mechanical or electronic

systems, are among the popular targets of side-channel attacks. That is because they typically run minimal software stacks (therefore easier to isolate the target functionality), expose fewer defenses (as they are resource-constrained), are more accessible for physical probing, and are mostly part of a larger critical system.

The vulnerable nature of embedded devices, coupled with the sensitivity of the data they process, highlights the importance of rigorous evaluation of their resistance to side-channel attacks. This, in practice, translates to analyzing the resistance of the cryptographic implementations operating on embedded devices to side-channel attacks. Security evaluation labs are the source points for this evaluation. They indicate the compliance of cryptographic implementations with established security levels specified in standard frameworks and issue certifications [1, 2] for the embedded devices. However, this is a cost-intensive operation, and maintaining a balance between thoroughness and budget constraints is a central challenge during evaluation [3].

Deep learning has recently been shown to be highly effective in side-channel analysis [4–7]. Its ability to extract complex patterns from noisy data often surpasses traditional side-channel analysis methods [8], particularly in scenarios involving protections (countermeasures like masking or desynchronization, see Section 2.4). This strength, combined with fewer preprocessing requirements, makes deep learning an attractive tool for attackers and evaluators alike. However, the need for problem-specific architecture tuning, hyperparameter optimization, and large datasets makes deep learning more expensive and less reliable for evaluation, as evaluation is a systematic and structured process based on predefined criteria and budget. The main purpose of this thesis is to improve the performance of deep learning-based side-channel analysis by using strategies and techniques that are less expensive than intense hyperparameter tuning, making it scalable for evaluation workflows.

As with most advances in side-channel analysis, deep learning-based side-channel analysis improvements are inherently dual-use. The title of this thesis, Strive to Fail, reflects the dual role of failure in side-channel evaluation. On the one hand, the evaluator deliberately strives to make a cryptographic implementation fail by recovering secret information from physical leakage. In this sense, failure is a diagnostic tool: it reveals weaknesses that may otherwise remain hidden. On the other hand, the desired outcome for a secure implementation is precisely that such evaluation attempts fail, even when advanced techniques such as improved deep learning-based side-channel analysis are used. This thesis adopts this evaluation's perspective: to push implementations and evaluation methods toward their limits, so that failure, whether of the target or of the attack, becomes informative for assessing and improving security.

1.1. Cryptography

Information security results from the contribution of many components, ranging from technical mechanisms such as encryption and authentication to human-centered practices like awareness and regulatory compliance. Cryptography is a fundamental pillar for many services that contribute to information security. In reference textbooks,

cryptography is stated as the study of mathematical techniques related to aspects of information security, such as confidentiality[1], data integrity[2], and entity authentication[3] [9]. This definition highlights the connection between the mathematical tools underlying cryptography and the security goals they enable. In everyday contexts, however, cryptography is often more simply described as the science of securing information by transforming it into an unreadable format, accessible only to authorized parties. Regardless of the definition, cryptography remains essential to maintaining trust and security across digital systems. Currently, cryptography is deeply embedded in our daily lives. It secures communications via messaging apps and emails, protects financial transactions through online banking and payment systems, and safeguards personal data on smartphones, cloud storage, and government portals. It also plays a crucial role in securing embedded devices, medical systems, and cryptocurrencies.

Modern cryptography[4] is typically divided into three main branches: symmetric cryptography, asymmetric cryptography, and cryptographic hash functions.[5] Symmetric encryption uses a single secret key for purposes like encryption/decryption, origin authentication, or integrity verification. In encryption/decryption usage, with the help of an algorithm and the secret key, the original data, known as plaintext, is transformed into an obfuscated form known as ciphertext. The authorized individuals or systems can transform the ciphertext into its original form using the same secret key. In the origin authentication and integrity verification usage, a cryptographic technique uses a secret key to generate a small code called a tag for a given message. The receiver can verify if the message has not been changed during transfer and if it comes from the true origin using the secret key and the tag regeneration. The most widely used symmetric algorithm is Advanced Encryption Standard (AES) [10], which was standardized by the National Institute of Standards and Technology (NIST)[6] in 2001 [12].

Asymmetric cryptography uses a key pair: a public key that can be distributed openly and used for encryption or signature verification, and a private key that only the owner knows and uses for decryption or generating the signature, depending on the purpose of use, which are key exchange and digital signing, respectively. Key exchange is a cryptographic process that allows two or more parties to establish a shared secret key over an insecure channel securely. Digital signing is the process of using a private key to create a unique cryptographic signature that verifies the authenticity and integrity of digital data. RSA [13] and Elliptic Curve Cryptography (ECC) [14, 15] are foundational asymmetric algorithms used for digital signatures

[1] Confidentiality ensures that sensitive data is accessed only by authorized individuals or systems.

[2] Integrity ensures that data remains accurate, consistent, and unaltered during transmission or storage.

[3] Authentication is the process of verifying a user's or system's identity. It ensures that an individual or entity is who they claim to be before granting access to a system or data.

[4] We needed to summarize a vast and beautiful joint branch of computer science and mathematics into a tiny subsection. Therefore, some sentences or definitions are not precise or well-defined enough. We believe that the cryptography community will understand that.

[5] Sometimes, hash functions are considered a part of symmetric cryptography.

[6] NIST is a U.S. government agency that develops technical standards and measurements to support innovation and industry [11].

and key transport.

Finally, cryptographic hash functions are one-way transformations that generate fixed-length digests from arbitrary-length inputs. They are essential for digital signatures through random number generation, integrity verification, and authentication through digest generation, among others. A widely used modern example is SHA-3 [16], based on the Keccak sponge construction [17]. With the rise of resource-constrained embedded systems, optimized cryptographic techniques, which minimize computational complexity, memory footprint, and power, have gained attention. Algorithms like Ascon [18], referred to in the community as lightweight algorithms, are examples of cryptography algorithms designed for resource-constrained usages.

In the last two decades, the threat of quantum computers to classical cryptography has been taken more seriously. In 1994, Shor introduced an algorithm [19] that enables polynomial-time quantum computation for integer factorization and discrete logarithms. This means the current widely used asymmetric cryptographic algorithms (RSA and ECC), which are based on these mathematical problems, are broken in the presence of sufficiently large quantum computers. Post-quantum cryptography (PQC) has emerged to develop cryptographic algorithms that are resistant to attack based on Shor's algorithm running on quantum computers. The proposed schemes (e.g., Kyber [20], Dilithium [21], and SPHINCS$^+$ [22] algorithms) are based on mathematical problems such as lattice-based and hash-based, which are believed to be hard even for quantum computers. While Grover's algorithm [23] allows for a quadratic speedup in brute-force search against symmetric ciphers, current symmetric cryptography is not fundamentally broken by quantum computing, and only doubling the key size (e.g., using AES-256 instead of AES-128) is recommended.

1.2. Implementation Curse

The security of a cryptographic primitive can be considered from two complementary perspectives. On one hand, it is crucial to ensure that the algorithm's mathematical foundation and the selected parameter set used to obfuscate data remain computationally infeasible to solve. This aspect is typically evaluated using classical cryptanalysis [9]. On the other hand, the security of the algorithm implementation depends on the interplay between the underlying hardware, the software implementation, and the operating environment. This second aspect is known as physical security [24].

A cryptographic algorithm can be implemented in software or hardware, or it can be implemented using a hardware-software co-design approach. The implementation choice depends on the system's performance constraints, resource availability, and security requirements. Software implementations are flexible and portable, making them suitable for general-purpose processors and embedded microcontrollers. Hardware implementations (e.g., on FPGAs or ASICs) typically offer higher performance, lower latency, and improved physical security, making them ideal for resource-constrained or security-critical applications. A hardware–software co-design balances flexibility and efficiency by orchestrating cryptographic algorithms in software while offloading computationally intensive tasks to dedicated hardware

accelerators.

When a device performs computation, it consumes power, takes time, emits electromagnetic radiation, dissipates heat, and may even generate acoustic noise. These physical phenomena form side-channels that unintentionally leak information about the device's internal operations and the data it processes. These leakages seriously threaten the physical security of cryptographic algorithms, as an attacker can measure and analyze them to recover secret information. Although these physical attacks are often tailored to specific hardware or software implementations, they are typically more powerful than traditional cryptanalysis. This work focuses on power consumption and electromagnetic emanation side-channels.

The existence of side-channel leakages is independent of implementation style, as all computations execute on physical hardware, which consists of microelectronic devices that inherently leak information. One of the well-known origins of power and electromagnetic side-channels is the physical behavior of transistors and complementary metal-oxide-semiconductor (CMOS)[7], which are the basic building blocks of registers, processing gates, and other microelectronic elements. The origin of power side-channel leakage in CMOS circuits lies in the intrinsic nature of their power dissipation during *switching activity*. CMOS gates consume energy from three primary sources: 1) leakage currents in transistors, 2) short-circuit currents during transitions when both transistors in a CMOS (i.e., NMOS and PMOS) conduct simultaneously, and 3) dynamic power caused by charging and discharging load capacitance [24]. Among these three, dynamic power is most relevant and the most dominant from a side-channel perspective, as it introduces a direct correlation between the data being processed and the power consumed. A significant power spike occurs during 0-to-1 (and 1-to-0) transitions, where energy is drawn to charge the output node. The same transition in registers is the most relevant source of their leakage. Other circuit-level effects, such as glitches [25], static power dissipation [26], coupling [27], and signal integrity on the power delivery [28] were identified more recently as contributing to power-based side-channel leakage.

Similarly, electromagnetic radiation in microelectronic devices originates from the internal currents flowing through conducting paths during operation. As the magnitude and direction of these currents change depending on the data being processed, the resulting EM fields vary accordingly. This data-dependent radiation is a powerful source of side-channel leakage that can expose sensitive operations to external observers.

While numerous mitigation strategies such as architectural optimizations, balanced circuit designs, electromagnetic shielding, and algorithmic countermeasures (see Section 2.4) are used, eliminating leakage and the correlation between the leakage and the important data being processed remains practically impossible. This limitation stems from unavoidable manufacturing imperfections and inherent physical variations in transistor behavior, making achieving perfectly symmetrical power or electromagnetic profiles impossible. Even minimal asymmetries result in measurable differences in power consumption or electromagnetic radiation correl-

[7] CMOS technology is the dominant fabrication process for almost all modern digital integrated circuits, including microprocessors, memory chips, FPGAs, and ASICs.

ated with processed data, leaving exploitable side-channels. Consequently, while countermeasures and other techniques can reduce information leakage, the laws of physics imply that side-channel vulnerabilities can be mitigated but never eliminated.

1.3. Side-channel Analysis

Side-channel analysis (SCA) is an attack by an analyzer to evaluate the physical security of an implemented algorithm. In most cases, SCA is translated to the effort needed to retrieve the cryptographic algorithm key (or the message in key exchange mechanisms, or signature forging in digital signing) using leakage in device measurements and publicly known data, like plaintext. When the analyzer only uses measurements[8] and data from the device under attack (DUA), the side-channel analysis technique is called non-profiled analysis [9]. This kind of attack appeared first in the late 1990s, and rapidly after publication, protection mechanisms (countermeasures, see Section 2.4) started to be proposed [29, 30] and implemented to mitigate these attacks [31, 32].

Non-profiled analysis relies on a large number of measurements from the DUA. The analyst selects a smaller window of these measurements that contains the most informative leakage, such as leakage related to a sensitive intermediate value or directly to the secret key, and applies statistical analysis to infer the secret data. Some examples of non-profiled analysis are:

- **Simple Power Analysis (SPA)**, where the leakage is directly and mostly visually inspected to get information about executed operations [33]. SPA is typically aided by prior knowledge of the cryptographic algorithm and adjusting parameters (like key size or plaintext length). It is often used as a preparation step before launching more advanced attacks. The insights SPA provides can notably shorten the required attack time window.

- **Differential Power Analysis (DPA)**, which can be highly efficient for unprotected implementations. DPA first partitions the measurements based on hypothetical intermediate values (the values that are calculated using the data that the attacker knows, like the plaintext, and all the hypothetical keys), then calculates the average of the partitions, and finally subtracts the averages looking for the hypothetical key that results in the highest difference among all the averages [33]. **Correlation Power Analysis (CPA)** is a sort of DPA that uses correlation analysis as a distinguisher to identify the relationships between the leakage and the secret data [31].

When the analyzer uses a clone device in addition to the DUA to build a profile, it is called profiled analysis. The analyzer is supposed to have complete control over the clone device. Profiled analysis requires two stages: template-building and

[8] Measurements of physical leakage are also referred to as traces.

[9] In many literature works, side-channel analysis and side-channel attack are used interchangeably. Here, we adhere to side-channel analysis as much as possible to emphasize that we are examining it from an evaluation perspective.

template-matching. During the template-building stage, the analyzer gathers many measurements with different plaintexts and keys from the clone device and builds the templates (profiles). During the template-matching stage, the attacker matches the measurements from the DUA with the templates. Examples of profiled analysis come in the following:

- **Template Attack (TA)** derives multivariate models (since the leakage over consecutive time samples is not independent) of the leakage present in a big set of measurements from the clone device. These models or templates are random variables with certain probability density functions (Gaussian PDFs are commonly used because the aggregation of many independent noise sources in side-channel measurements leads, by the central limit theorem, to approximately Gaussian leakage distributions). The attacker determines the distribution parameters using the measurements to specify templates. Each template (profile) characterizes the probability density function of the collected measurements with the same intermediate values. The number of profiles is determined by all the possible intermediate values and the selected leakage model (more details can be found in Section 2.3). The attacker estimates the distribution parameters (mean and covariance in the Gaussian case) using the measurements in each class to specify templates. In the template-matching step, the most probable key is specified using the maximum likelihood method. A critical step in TA is selecting a set of *points of interest* (PoIs), the time samples from the measurements exhibiting the strongest dependency on the targeted sensitive value (the intermediate value in the process of an algorithm that includes the key, see Section 2.2 for more information). This attack is introduced with more details in Section 2.6.1.
  Template attack faces two main challenges: collecting enough measurements from the clone device to build accurate templates, and assuming an underlying distribution (e.g., Gaussian), which may not match the actual data distribution [34].

- **Deep Learning-based Attacks (DL-SCA)**. Deep learning consists of training and testing stages, which comply entirely with template-building and template-matching stages of profiled analysis. In DL-SCA [5], the templates are not built separately. Instead, a deep neural network is used to classify the measurements based on the values of the targeted sensitive value. In DL-SCA, the statistics of the unknown leakage distribution are automatically learned from the profiling set, meaning that the adversary does not necessarily need to assume the statistical distribution that describes the leakages well.

  One can see that while the template attack requires the assumption of a particular distribution of the traces, DL-SCA does not require such assumptions and is supposed to learn the characteristics of the leakage automatically with the same degree of dependency on the leakage model assumption (more details can be found in Section 2.3 and Chapter 7).

SCA is usually performed using the divide-and-conquer strategy. The divide-and-conquer strategy makes it possible to recover a long key by retrieving its smaller parts

separately. The smaller parts make the exhaustive search possible (see Section 2.3).

As deep learning has recently been shown to be highly effective in side-channel analysis [4–7], the focus of this thesis is on DL-SCA. More details about this kind of analysis are provided in Section 2.6.1.

1.3.1. Security Evaluation

Evaluating implementations against side-channel attacks is critical from a regulatory perspective because side-channel attacks pose a significant threat to devices handling sensitive data, such as keys, private certificates, or intellectual property. Regulatory requirements emphasize such evaluations to ensure reliable security judgments and to guarantee that certified IT products can be confidently procured and used without further assessment.

Two well-known frameworks that explicitly mandate the evaluation of cryptographic implementations against side-channel attacks are the Common Criteria (CC) framework [1] and FIPS 140 [2]. The CC framework focuses on protecting against side-channel attacks by involving specialist evaluations. Within CC, stakeholder groups maintain a confidential list of attack vectors and use a rating system to assess the difficulty of mounting attacks, aiming to balance strong security with evaluation costs. In contrast, the FIPS 140 framework follows a more cost-effective approach by relying on conformance-style testing based on ISO 17825:2016[10]. The latest version, FIPS 140-3, introduces the Test Vector Leakage Assessment (TVLA) method to evaluate the risk of side-channel attacks. Azouaoui et al. recently introduced a third strategy, the worst-case adversary model, in [3]. The alternative evaluation strategy, described as “working backwards”, has emerged in academic literature. This regime starts by demonstrating an attack assuming maximum capability for the adversary and then examines the effects of gradually relaxing these assumptions.

Regardless of the chosen evaluation strategy, there is a challenging trade-off that must be balanced. On one hand, there is the concern of whether the evaluation captures a product’s true security level [3]. Addressing this concern requires evaluators to conduct assessments that are as thorough as possible for the targeted security level, systematically examining all possible vulnerabilities. On the other hand, identifying and testing these vulnerabilities often demands substantial effort and resources, making the evaluation process inherently costly. In practice, security evaluations are commissioned and funded by the product manufacturer or a sponsoring organization, and the evaluation laboratory operates under a fixed budget and timeline. Consequently, evaluators must continuously balance the completeness of their assessment against practical cost constraints.

[10] ISO/IEC 17825 is an important standard in the cybersecurity field, which specifies non-invasive attack mitigation test metrics to determine compliance with the requirements specified in ISO/IEC 19790 (an upstream ISO/IEC standard).

1.4. Problem Statement

1.4.1. Security Evaluation and Deep Learning Usage Implications

Deep learning is interesting for evaluators as it promises to generalize and automate commonly used techniques such as template attacks. While other techniques rely heavily on precise PoI selection and trace alignment, deep learning can automatically learn optimal PoIs, even from misaligned traces, thereby eliminating the need (and costs) for extensive manual feature engineering or pre-processing [6, 35]. Besides that, DL-SCA does not rely on strong assumptions about leakage distribution. While template attacks require the assumption of a particular distribution (e.g., Gaussian) for the leakage, deep learning offers infinite opportunities to learn a classifier without any assumption, giving it more flexibility to handle complex leakage. Therefore, while the Gaussian template attack theoretically represents the strongest profiling model [24], DL-SCA, in practice, is more efficient in terms of modeling effort and robustness to noise and desynchronization, and it can scale better to complex leakage scenarios and higher-order attacks.

Deep learning can also facilitate SCA evaluation (and reduce some costs consequently) by mitigating common practical SCA difficulties like reliance on assuming the leakage model. Conventional SCA techniques usually rely on some assumptions about the relationship between the targeted intermediate values and the measurements obtained from the device

Another practical challenge that deep learning can facilitate in SCA is to remove the reliance on access to known parts of data (plaintext, ciphertext, nonce, etc) in both profiled and non-profiled analyses. Acquiring the actual intermediate values during the profiling phase is assumed in profile attack scenarios. Similarly, access to a known part of the data is required to build the hypothetical sensitive values in non-profiled attacks. In practice, however, such access may be limited or constrained by implementation-specific factors and use cases, thereby imposing additional costs on the evaluation process.

Considering the success and facilitation of deep neural networks in side-channel domains, evaluators are interested in using them as one of the tools helping in their evaluation path. The Federal Office for Information Security (BSI) in Germany recently published a document [36] describing the requirements for machine learning-based SCA (including deep learning), showing that evaluators have started to use machine/deep learning tools more seriously. Simultaneously, deep learning empowered attackers by lowering the barrier to mounting effective attacks. By training on large datasets of captured leakage, attackers can successfully recover secret keys even from protected implementations that deploy countermeasures such as masking [5] and hiding [37], making deep learning a growing concern for security evaluators [36]. These factors further underscore the necessity of fitting the DL-SCA into the evaluation frameworks.

However, using deep learning in the evaluation process can also introduce new sources of costs. DL-SCA generally requires more computational resources and larger profiling datasets. Besides that, deep neural networks involve highly para-

meterized architectures. While these architectures enable models to learn complex, non-linear representations of leakage data, they also make them prone to training instability and sensitivity to hyperparameter choices. As a result, hurdles such as **overfitting**[11] and **hyperparameter tuning**[12] arise in DL-SCA.

Overfitting is a critical concern within DL-SCA, as side-channel datasets are often small, and variations in implementations or countermeasures frequently require retraining or redesigning a suitable model for each new target. Hyperparameter configurations are also highly sensitive to leakage characteristics, countermeasures, and device-specific noise, and therefore do not generalize well across different targets or even similar implementations. Moreover, many hyperparameters implicitly determine how leakage is modeled, making their selection directly influence the success or failure of the attack. This sensitivity is further compounded by training instability and reproducibility issues, where small changes in hyperparameters or initialization can lead to significantly different attack outcomes.

In practical evaluation settings, these challenges are amplified by strict constraints on time, computational resources, and available datasets. As a result, hyperparameter tuning and overfitting in DL-SCA are not merely performance optimization-related considerations, but cost-critical and risk-sensitive processes that directly impact the reliability and credibility of the evaluation. These observations motivate the research questions addressed in this thesis, which aim to investigate how the benefits of deep learning can be leveraged to **reduce evaluation costs** and practical constraints, while mitigating the new cost and complexities that DL-SCA introduces in the evaluation process.

1.4.2. Research Questions

Overcoming overfitting and hyperparameter tuning **challenges** make DL-SCA costly for evaluation. At the same time, using DL-SCA can reduce the evaluation costs with the **benefits** it brings along.

Considering the **hurdles** and **benefits** that using deep learning introduces to side-channel evaluation, the mission of this thesis is twofold. First, we **investigate methods and strategies that reduce the costs of model selection and overfitting in DL-SCA**. Methods and approaches that improve the generalization of trained models are particularly valuable, as they reduce the need for intense and expensive hyperparameter tuning while enhancing the robustness of DL-SCA in real-world security evaluations. Second, we **investigate deep learning use cases for SCA that reduce evaluation costs.** The two use cases that we study are **1) whether deep learning can remove the need to assume a specific leakage model** and **2) whether deep learning can eliminate the reliance on plain/ciphertext for side-channel analysis**, which are two practical sources of challenge and cost during evaluation. Deep neural networks can bypass such practical limitations, enabling more accurate and reliable security evaluations.

[11] Overfitting occurs when a model fits the training data too precisely, including noise and irrelevant patterns.

[12] Hyperparameter tuning is the process of selecting architectural and training configurations, so that the model can learn what we desire from the data.

The research questions can then be formulated as:

- How can we reduce the costs of DL-SCA regarding model selection and overfitting, while preserving or improving generalization in realistic evaluation settings?
- To what extent can deep learning mitigate the reliance on (i) an explicit leakage model and (ii) access to plaintext/ciphertext for SCA, without degrading attack or evaluation effectiveness under realistic threat models?

1.5. Contributions

The contributions of this work are defined in terms of the research questions. To answer the first research question defined in Section 1.4.2, we investigate strategies to improve the performance of the deep neural models in SCA with less computational effort and cost.

Many failures of DL-SCA in evaluation settings stem not from insufficient model capacity, but rather limited data availability, poor generalization across targets, and instability caused by overfitting and hyperparameter sensitivity. In realistic evaluation settings, where datasets are costly to acquire and computational resources are constrained, aggressively increasing model complexity or performing extensive hyperparameter searches is impractical and inefficient. Therefore, this work focuses on enhancement techniques that improve robustness and generalization while keeping modeling and tuning effort manageable. This thesis adopts the perspective that improving how deep learning models exploit available data by stabilizing training, reducing variance, and encouraging the learning of generalizable leakage representations is a more effective strategy for reducing evaluation costs while preserving attack effectiveness. This insight motivates a shift away from architecture-centric optimization toward data-centric and learning-centric enhancement techniques that are better aligned with the constraints and objectives of practical side-channel evaluation.

Building on this insight, Chapter 3 to 7 investigate four complementary enhancement techniques. First, the impact of increasing the effective amount of training data is studied, demonstrating how larger and more diverse training sets can reduce overfitting and stabilize model performance when the working regime of the model allows that. Second, regularization techniques are examined as a principled way to constrain model complexity and improve generalization without extensive hyperparameter tuning. Third, ensemble learning is explored to mitigate model instability and sensitivity to initialization by aggregating multiple learners, thereby improving robustness across targets. Finally, multitask learning is investigated as a means to exploit shared representations across related tasks, enabling models to generalize better while reducing the need for task-specific tuning. Together, these chapters provide a structured and practical answer to the first research question by identifying strategies that lower evaluation costs while maintaining or improving the effectiveness of DL-SCA in realistic settings. The contributions by chapter are as follows:

- In Chapter 3, we investigate in detail the effect of overfitting in DL-SCA. This

chapter is based on [38].[13] We validate that the overfitting effect can often be reduced by adding more profiling traces during training, provided the chosen hyperparameter combination does not result in large assumption errors. Still, we experimentally verified that adding more profiling traces does not always increase the generalization of a model. While the interplay between the amount of training data and the deep neural network model's capacity is complex and difficult to predict in general, this instability is particularly pronounced in DL-SCA due to the high noise levels and variability present in side-channel measurements.

- In Chapter 4, we investigate the intertwined effect of a model's hyperparameters and added regularization techniques. This chapter is based on [39] and shows that the improvements offered by regularization techniques heavily depend on the neural network's hyperparameters. [14] However, adding regularization techniques can improve the average performance of DL-SCA by reducing the assumption errors, i.e., errors caused by mismatches between the neural network's implicit leakage assumptions encoded through its architecture and hyperparameters and the way it combines samples over time, and the actual leakage behavior (distribution) of the target implementation.

- In Chapter 5, which is based on [40], we investigate whether using a specific activation function, JumpReLU, can improve the robustness of DL-SCA with respect to noise, trace variability, and hyperparameter sensitivity. [15] Although JumpReLU was originally proposed to increase robustness against adversarial perturbations, we examine its impact in a side-channel context, where variability arises from measurement noise, desynchronization, and device-specific leakage characteristics. We provide a benchmark between random models using different activation functions, including JumpReLU, to see if using this function can improve the reliability of DL-SCA by improving the average performance.

- In Chapter 6, we investigate how applying ensemble learning to DL-SCA by aggregating multiple sub-optimal models trained with different hyperparameter configurations can improve attack performance and stability based on [41]. [16] This ensemble can easily outperform the best model found in a hyperparameter tuning procedure, proving that it is a cheaper solution when evaluating using deep learning. This approach also addresses the high sensitivity of a single deep neural network to hyperparameter choices, as the ensemble effectively averages out model-specific assumption errors and training instabilities, reducing the reliance on identifying a single well-tuned model. We particularly

[13] The paper title is: *To overfit, or not to overfit: improving the performance of deep learning-based SCA*.

[14] The paper title is: *Regularizers to the rescue: fighting overfitting in deep learning-based Side-channel Analysis*.

[15] The paper title is: *Jump, It Is Easy: JumpReLU Activation Function in Deep Learning-based Side-channel Analysis*. This project was led by Abraham Basurto-Becerra. My contribution consisted of helping to design the experimental methodology for evaluating the proposed idea and contributing to part of the experimental work.

[16] The paper title is: *One for All, All for Ascon: Ensemble-Based Deep Learning Side-Channel Analysis*.

highlight successful DL-SCA on protected Ascon implementations, while other evaluation techniques were not successful till the time of that work, proving the necessity of taking deep learning evaluation into account during certification.

While the earlier chapters focus on mitigating the additional costs introduced by deep learning in SCA evaluation, the second part of this thesis is motivated by the complementary insight that deep learning can also remove long-standing structural assumptions that themselves constitute sources of evaluation cost and fragility. Conventional SCA methodologies rely heavily on (i) the explicit selection of a leakage model and (ii) access to known data such as plaintext or ciphertext. These assumptions simplify analysis but are often violated in practice, forcing evaluators to invest significant effort in reverse engineering implementations, validating leakage hypotheses, or acquiring restricted data interfaces. The core insight guiding this part of the thesis is that deep neural networks can directly learn discriminative leakage representations from measurements themselves, even when classical labels or simplified leakage abstractions are unavailable or inaccurate. This ability enables new attack and evaluation paradigms that reduce dependency on explicit leakage modeling and known data access, while maintaining —or even improving— the attack effectiveness under realistic threat models.

Building on this insight, Chapters 7 and 8 investigate concrete deep learning-based approaches that relax these assumptions in practice. Rather than replacing classical SCA techniques, these chapters explore how deep learning can shift the effort away from manual modeling and data accessibility constraints toward data-driven inference, thereby reducing evaluation cost and uncertainty. The contributions by chapter are as follows:

- In Chapter 7, we propose a new attack method based on [42] [17], multi-byte multi-bit DL-SCA, which allows more flexibility regarding the leakage model assessment and can simultaneously profile the behavior of the leakage for all sub-keys with bit-level precision. This proposition helps remove the effect of over-simplified or inappropriate leakage model selection, which is a common issue in side-channel resilience evaluation.
- Lastly, in Chapter 8, which is based on [43] [18], we address key limitations in existing blind SCA techniques by employing a deep learning-based approach, significantly advancing the practical applicability of blind SCA. More precisely, blind SCA's main challenge lies in inferring labels for each measurement, as it does not rely on known plaintext or ciphertext. We model this as a deep learning problem with noisy labels and show that deep neural networks effectively identify the underlying distribution of these measurements, outperforming traditional techniques in blind SCA performance.

[17] The paper title is: *Leakage Model-flexible Deep Learning-based Side-channel Analysis*. This work was led by Dr. Lichao Wu. My contribution was mainly in conducting part of the experiments reported in the study.

[18] The paper title is: *Breaking the Blindfold: Deep Learning-based Blind Side-channel Analysis*. This work was carried out in close collaboration with Dr. Trevor Yap. We shared the main responsibilities of the project, including developing and testing the idea, and Trevor contributed substantially to the coding and experimental implementation.

To emphasize the real-world applicability, we apply DL-SCA across various platforms, cryptographic algorithms, and implementations. This thesis covers DL-SCA of three cryptographic algorithms implementations: AES, a widely used encryption standard; ASCON, the CAESAR competition winner and NIST's choice for lightweight cryptography; and Kyber, the post-quantum key encapsulation mechanism selected by NIST. We cover datasets for both software and hardware implementation. Moreover, we deploy the implementations on various platforms, including the ARM Cortex-M4 chip, which represents the highest market share among all ARM products [44], reflecting its dominance in embedded applications and low-power devices. We also cover both protected and unprotected implementations for the AES and Ascon schemes to show the generalization capability of the proposed contributions. This is all to make our experiments more complete and thorough, and the outcomes not dataset dependent. For more information about the covered datasets, look at Section 2.6.3.

2 Preliminary

This thesis focuses on the side-channel analysis of three representatives of critical trends in cryptography. We see DL-SCA of AES, a representative of classical symmetric cryptography; Ascon, a representative of lightweight cryptography; and CRYSTALS-Kyber, a well-known scheme from the post-quantum cryptography realm, all standardized. In this section, we introduce these three schemes and give more details about the attack points from each scheme targeted in the rest of this thesis.

2.1. Covered Cryptographic Algorithms

2.1.1. AES

The Advanced Encryption Standard (AES) [10] is a widely used symmetric-key block cipher standardized by NIST in 2001 [45]. It is designed to provide strong data confidentiality and is used in various security protocols, including TLS, VPNs, and disk encryption systems. AES operates on fixed-size blocks of 128 bits and supports key sizes of 128, 192, or 256 bits, resulting in $Nr = 10/12/14$ encryption rounds, respectively. The algorithm is based on a substitution-permutation network (SPN) principles, where input data undergoes multiple rounds of transformations to ensure both confusion and diffusion, two essential cryptographic properties for security.

AES encryption and decryption rely on a sequence of core transformations performed on a 4×4 byte matrix called the *state*. Each round of AES (except the final one) involves four primary operations:

1. **SubBytes:** A non-linear substitution step where each byte in the state is replaced using a fixed substitution box (Sbox) derived from the multiplicative inverse in the finite field $GF(2^8)$.

2. **ShiftRows:** A row-wise permutation where the first row remains unchanged, the second row is cyclically shifted by one byte, the third by two bytes, and the fourth by three bytes.

3. **MixColumns:** A linear mixing step where each column of the state matrix is

transformed using polynomial multiplication in $GF(2^8)$, improving data diffusion across the block.

4. **AddRoundKey:** A bitwise XOR operation between the current state and a round key derived from the main cipher key using the AES key schedule.

Decryption involves the inverse of these operations applied in reverse order.

AES follows a substitution-permutation network structure, where plaintext is transformed into ciphertext over multiple rounds of well-defined transformations. The algorithm begins with an initial AddRoundKey step, followed by $Nr - 1$ identical rounds, each consisting of SubBytes, ShiftRows, MixColumns, and AddRoundKey operations. The final round omits the MixColumns step. The security of AES relies on the combination of non-linear substitution (for confusion) and linear mixing (for diffusion), which makes it resistant to linear and differential cryptanalysis. Due to its efficiency, parallelizability, and robust security guarantees, AES has become the standard for symmetric encryption worldwide.

2.1.2. Ascon

Ascon is a family of cryptographic algorithms designed to provide secure encryption and authentication in resource-constrained environments. This family of cryptographic primitives is based on duplex sponge construction [17] and was selected by NIST in February 2023 to be standardized [46] for lightweight applications. Ascon-128a is an authenticated encryption algorithm that includes associated data, meaning it not only encrypts a message to maintain its confidentiality but also attaches a tag to the encrypted message and associated data to ensure integrity. The algorithm can take four inputs: plaintext P, associated data A, nonce N, and a key k. It outputs the authenticated ciphertext C and an authentication tag T. The algorithm includes four operational phases: *initialization, associated data process, plaintext process (ciphertext process in decryption), and finalization*. Figure 2.1 shows these four phases of Ascon. In Ascon-128a, the input of the initialization phase is a 320-bit initial state ($IV||k||N$ in Figure 2.1, consisting of the 64-bit constant IV, the 128-bit key k, and the 128-bit fresh nonce N) in the form of five 64-bit words. This five-word state updates through the algorithm phases and is used as the state (or the sponge state) for encryption (decryption) and tag generation.

In Ascon-128a, the initialization phase includes twelve rounds of the Ascon-p permutation function (shown as p^a in Figure 2.1) that process the 320-bit state. The permutation function has three parts: 1) the addition of the round constants, 2) the non-linear 5-bit Sbox (substitution layer), and 3) the linear diffusion layer. The next phase is optional, which is associated data process. Data like headers and metadata must remain in plaintext during a data exchange, but maintaining integrity is crucial for this data. The optional associated data process phase maintains this integrity. In this phase, when an associated data block (A_i) is received, its first $r = 64$ bits are XORed to the first $r = 64$ bits of the sponge state, then the whole sponge state is permuted $b = 6$ rounds (p^b in Figure 2.1). The associated data process phase updates the sponge state using the associated data blocks. Then, the updated sponge state

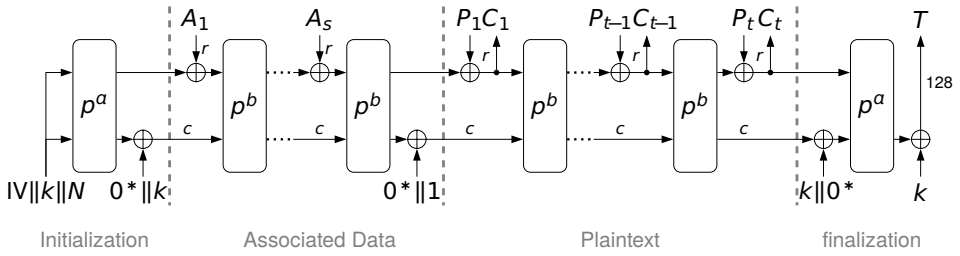


Figure 2.1: The phases of Ascon-128a encryption. The image is from [47].

proceeds the plaintext process phase. The plaintext process phase XORs the 64-bit plaintext block P_i with the first $r = 64$ bits of the sponge state to produce ciphertext block C_i. Then, the whole state is transformed by the permutations p^b with $b = 6$ to update the sponge state for the next plaintext block. The finalization phase XORs the key with the sponge state and transforms the results with p^a, $a = 12$, to provide the 128-bit authentication tag T. For more details about different parts of the Ascon primitive, one can see [48].

Compared to AES, Ascon is considered a lightweight cryptographic algorithm primarily due to its simplified internal structure and implementation-friendly design. While AES relies on byte-oriented operations such as large 8-bit Sbox lookups, finite-field arithmetic in $GF(2^8)$ (e.g., MixColumns), and a separate key schedule, Ascon replaces these components with a compact permutation based solely on bitwise logical operations, rotations, and a small 5-bit Sbox. One significant benefit of the Ascon Sbox operation is its ability to be executed through XOR and AND operations on state words [48]. This design eliminates the need for lookup tables, complex field multiplications, and external modes of operation, thereby significantly reducing code size, memory footprint, and computational overhead. As a result, Ascon achieves strong security guarantees while being well-suited for low-power and resource-constrained platforms, such as embedded and IoT devices, where the implementation cost of AES can be prohibitive.

2.1.3. Kyber

CRYSTALS-Kyber is the standard for the Key Encapsulation Mechanism (KEM), which NIST selected for Post-Quantum Cryptographic (PQC) applications. Here, we briefly introduce the key parameters and components of the Kyber algorithm, which are essential for understanding the attack in this thesis. For more information, please refer to [49].

Kyber is based on the Learning With Errors (LWE), which assumed to be a hard mathematical problem resistant to both classical and quantum attacks. In its simplest form, LWE involves distinguishing between random samples and samples of the form $b = As + e$ over a finite field, where A is a random matrix, s is a secret vector,

and e is a small error vector. Kyber builds upon a variant called Module-LWE, which generalizes LWE by working with polynomials over the $\mathbb{Z}_q[X]/(X^n + 1)$ ring. Based on the dimension of the matrix and vector of polynomials that are used (parameter k), one can control Kyber's security and efficiency.

The Kyber KEM includes three procedures for a full key exchange. First, the secret key and public key pair, (pk, sk), are generated using the KeyGen() procedure. A random seed ρ is used to derive a public matrix A with a uniform distribution. The secret key consists of a small polynomial vector s and an error vector e, both sampled from a centered binomial distribution β_η. Parameters like k (dimension of the matrix and vectors), n (the degree of polynomial in $\mathbb{Z}_q[X]/(X^n + 1)$), and q (the prime modulus) specify the security level of Kyber. In the attack introduced in this thesis, we consider security level three, which is called Kyber768. Kyber768 parameters are set as $k = 3$, $n = 256$, and $q = 3329$.

The second party in the key exchange uses the public key to encapsulate a randomly chosen message. The key encapsulation procedure is performed by the function Encapsulation(pk), which takes the public key as input and outputs a ciphertext $c = (u, v)$ and a shared secret K. In this process, a uniformly random message $m \in \{0, 1\}^{256}$ is first sampled and then encrypted using the public key to form the ciphertext $c = (u, v)$. The final shared key is subsequently derived from both m and c, binding the session key to the encapsulated message and the resulting ciphertext.

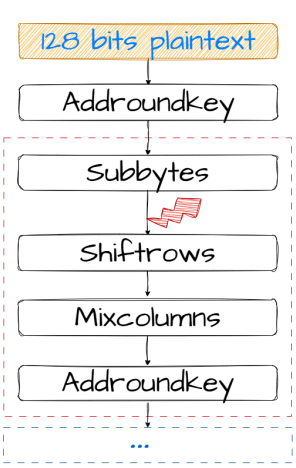
The last procedure is decapsulation, where the first party uses its secret key to decrypt the received ciphertext and extract the message. The decapsulation process is performed using the function Decapsulation(c, sk). The decapsulation procedure first decrypts the ciphertext using `Kyber.CPA.Dec` to recover the message m'. It then recomputes the encryption of m' using the public key, resulting in a reconstructed ciphertext (u', v'). If (u', v') matches the received (u, v), the shared key is derived using m' and c. This mechanism is known as the Fujisaki–Okamoto transform. If (u, v) and (u', v') do not match, a fallback key is used. This mechanism prevents information leakage during decryption failures and ensures that Kyber remains secure against chosen-ciphertext attacks, thus achieving *CCA-security*.[1] For more details about these three procedures of Kyber, see [49].

2.2. Attack Points

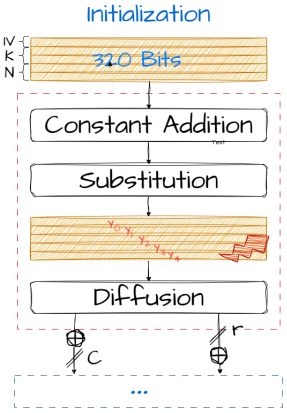
In the SCA domain, an attack point refers to the specific step, sensitive value, or operation within a cryptographic algorithm where the physical leakage is analyzed to extract secret information, such as keys. These points are chosen based on the mathematical structure of the algorithm and the strength of side-channel leakage.

[1] AES is CCA-secure if used in a secure mode. Ascon-128a is CCA-secure in its original design as it authenticates both the ciphertext and associated data. However, CCA security is more important for public-key encryptions like Kyber as they operate in an active-attacker setting i.e, attackers can submit chosen ciphertexts to a decryption oracle and learn information from acceptance or rejection behavior. Symmetric primitives are used with a shared secret and are typically protected by authentication. Therefore, CCA security is a design necessity for Kyber, while it is a usage-dependent property for AES and Ascon, arising from higher-level protocol requirements.

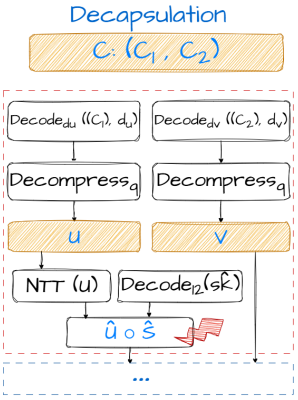
A practical attack point should depend directly on secret data, have predictable intermediate values using known inputs (plaintext, ciphertext, nonce, etc.), and different key hypotheses should be statistically distinguishable using it. Therefore, a practical attack point can vary depending on the implementation specification (if it is software or hardware, or if few or many traces are available) and the final goal of the attacker (if the attacker wants to retrieve the secret key or the session key exchanged between two parties). In the following, we explain the attack points we used for attacking the implementation of cryptographic schemes introduced in Section 2.1. Figure 2.2 visualizes the attack points used in this thesis for the three different schemes. The details about these attack points can be found in Section 2.2.1 to 2.2.3.



(a) First round of AES Sbox output as attack point.



(b) First permutation function Sbox output as attack point in Ascon-128a initialization phase.



(c) Pointwise multiplication of ciphertext and key as attack point in Kyber decapsulation procedure.

Figure 2.2: Attack points of covered schemes in this thesis.

2.2.1. AES Sbox Output

The first-round Sbox output of AES is very popular for SCA. This popularity has multiple reasons. First, as can be seen in Eq. (2.1), the first-round Sbox output directly depends on the key (the secret value that the attacker wants to extract), and the plaintext (known in many attack scenarios). Besides, this attack point is also known to exhibit strong side-channel leakage, as the Sbox computation involves key-dependent non-linear transformations. The non-linear Sbox improves the statistical distinguishability between correct and incorrect key hypotheses by increasing the data-dependent variation in the leakage, which causes significant switching activity in hardware. This activity is well captured by common leakage models and can be exploited by SCA.

Knowing the values of the plaintexts as the inputs, alongside the divide-and-conquer strategy, allows us to make a small set of hypotheses based on possible keys and find the correct key by exploiting the leakage of the attack point in many non-profiled attack scenarios. In the profiling phase of a profiled attack, this sensitive value for each trace, combined with the assumed leakage model, is used to label the traces. Figure 2.2a shows the Sbox output as the attack point in the first round of AES. Eq. (2.1) shows the relation between the key and plaintext in the attack point, focusing only on $b = 8$ bits at a time. As a consequence of using the divide-and-conquer strategy, k and p are partial (only eight bits here); therefore, an attacker can extract the whole key of AES-128 within sixteen attacks using this attack point.

$$f(k, p) = Sbox(k \oplus p). \tag{2.1}$$

It is worth mentioning that the Sbox output is not the only possible attack point in AES. Some attacks consider other building blocks of AES, like the output of ShiftRows, MixColumns, or a combination of multiple attack points [50].

2.2.2. Ascon-128a Sbox Output

Since the key is directly processed in Ascon's initialization and finalization phases, these two phases are candidates for side-channel attacks. During the Ascon-128a encryption's initialization phase, as shown in Figure 2.1, the secret key is directly involved. The secret key, along with the nonce (a user input that is changing for every encryption) are manipulated using the first round of the permutation function. Since this function processes something we know (the nonce) and the secret information we aim to obtain (the key), it can be the target of side-channel analysis. For the same reasons listed for AES Sbox output in Section 2.2.1, the best attack point in Ascon's initialization is the non-linear Sbox output of the first-round permutation function (Figure 2.2b). Therefore, in this thesis, we focus on the Sbox output of the first-round permutation in the initialization phase as the attack point.

The Ascon-128a Sbox is a column-wise operation on the sponge state. The Sbox operation takes a 5-bit input (x_0 to x_4) that includes only one bit from each input word, and gives a 5-bit output (y_0 to y_4) to build the output words (Figure 2.3).

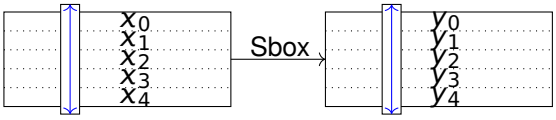


Figure 2.3: Ascon-128a column-wise Sbox and the y_i outputs. The Sbox operation takes a 5-bit input,x_i, that includes only one bit from each input word and gives a 5-bit output, y_i, that contains one bit from each output word.

The Ascon-128a Sbox can be implemented easily with XOR and AND. The five y_i output bits of the Sbox can be expressed as the XOR- and AND-combinations of five x_i input bits according to Eq. (2.2):

$$
\begin{aligned}
y_0 &= x_0 + x_1 + x_2 + x_3 + x_1x_2 + x_0x_1 + x_1x_4,\\
y_1 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_1x_2 + x_1x_3 + x_2x_3,\\
y_2 &= x_1 + x_2 + x_4 + x_3x_4 + 1,\\
y_3 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_0x_3 + x_0x_4,\\
y_4 &= x_1 + x_3 + x_4 + x_0x_1 + x_1x_4.
\end{aligned}
\tag{2.2}
$$

The inputs to the first round of the permutation are the key split into two words, k_1 and k_2, the nonce split into n_1 and n_2, and the public constant (IV). Thus, we can replace the x_i with their original values and rewrite Eq. (2.2).[2] From Eq. (2.2), it is clear that y_4 depends only on the high part of the key, k_1, as is shown in Eq. (2.3). This characteristic makes y_4 a practical intermediate value for SCA. It is possible to recover k_1 with the divide-and-conquer strategy and retrieve k_1 in eight attacks using 8-bit chunks.

$$
f(k_1, n_1, n_2) = k_1 \& (255 \oplus IV \oplus n_1) \oplus n_1 \oplus n_2. \tag{2.3}
$$

To obtain the remaining key bits (k_2), we use y_0 or y_1 (since they have non-linear terms including x_2) as the intermediate value. The other half of the key, k_1, can be taken as a known, and its recovered value from Eq. (2.3) can be replaced in the new selected intermediate value to recover k_2.

2.2.3. Pair-pointwise Multiplication in Kyber

When attacking Kyber implementation using SCA, extracting two different critical values is possible. The first value we can extract is the secret key. An attacker can extract the secret key from the key generation procedure, where it is generated, or from the decapsulation procedure, where it is used for decrypting the ciphertext. If we attack the key generation procedure, we only have one trace to extract the secret key as the key is generated only once. In contrast, if the usage setting is not ephemeral, we can use multiple traces to extract the secret key through decapsulation. Second, we can extract the message, which can be used to retrieve the long-term session key from encapsulation or decapsulation. Since the message changes for every key exchange, the message should always be recovered using a single trace. This thesis only focuses on extracting the secret key using decapsulation. For that purpose, we need to consider the decryption module in the decapsulation procedure.

In the core part of the decryption module, the secret key in the Number Theoretic Transform (NTT) domain, $\hat{s}^T$, is multiplied by the u part of the ciphertext in the NTT domain (looking into Figure 2.2c, one can see that the u part of the ciphertext is acquired form decompressing the C_1 part of the C using d_u parameter) as is shown in Eq. (2.4).

$$
m := \mathrm{Encode}_1\left(\mathrm{Compress}_q\left(\mathbf{v} - \mathrm{NTT}^{-1}\left(\hat{\mathbf{s}}^T \circ \mathrm{NTT}(\mathbf{u})\right), 1\right)\right). \tag{2.4}
$$

[2] We neglect the addition of the constant to the last 8 bits of the lower part of the key (k_2), which is the only operation that occurs before the Sbox in the first round of the permutation function.

The NTT is a specialized variant of the Discrete Fourier Transform that operates over finite fields. Transferring the polynomials into the NTT domain provides an efficient way of multiplying them ($\mathcal{O}(n \log n)$ instead of quadratic time complexity).

The efficiency that transforming to the NTT domain offers stems from the fact that one can calculate the coefficients of the deciphered message using pair-pointwise multiplication of the coefficients of u and secret key, s, in the NTT domain. Using the roots of unity, one can convert a polynomial of degree 255 into 128 polynomials of degree one. This is called an incomplete NTT[3]. Eq. (2.5) shows the expression of the polynomial $a(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1}$, with $n = 256$, in the NTT domain. As one can see, each degree one polynomial in the NTT domain has two coefficients.

$$\text{NTT}(a) = (a_0 + a_1 x,\ a_2 + a_3 x,\ \ldots,\ a_{254} + a_{255} x)\,. \tag{2.5}$$

The forms of the u part of the ciphertext and the secret key s are the same as Eq. (2.5) in the NTT domain. In a "complete" NTT, one can use pointwise multiplication to compute the coefficients of the multiplication results, i.e., one can simply calculate $NTT(a \cdot b) = (c_0 = a_0 \cdot b_0, c_1 = a_1 \cdot b_1 =, \ldots, c_n = a_{n-1} \cdot b_{n-1})$. In the case of Kyber, with the incomplete transform of the polynomials to the NTT domain, we need to use pair-pointwise multiplication instead of pointwise multiplication to compute the result of $(a \cdot b)$. With pair-pointwise multiplication, the coefficients of the deciphered message in the NTT domain can be obtained using the coefficients of u and s in the NTT domain as follows:

$$\begin{aligned} m_{2i} &= s_{2i} \cdot u_{2i+1} + s_{2i+1} \cdot u_{2i}, \\ m_{2i+1} &= s_{2i} \cdot u_{2i} + s_{2i+1} \cdot u_{2i+1} \cdot \zeta_i, \end{aligned} \tag{2.6}$$

where ζ_i is the root of unity corresponding to the considered polynomial, and the summation is modulo q. All the multiplications in Eq. (2.6) involve multiplying a coefficient from the secret key we would like to retrieve through SCA, and a coefficient of the ciphertext that is usually a public variable. We choose to attack this point because it is the only place where the secret key coefficients are being processed with the ciphertext coefficients.[4] Using the divide-and-conquer strategy, it is possible to extract the secret key (one coefficient at a time). We consider the attack point for Kyber as $f(s_{2i}, u_{2i}) = s_{2i} \cdot u_{2i}$. The attack point for Kyber considered in this thesis is also shown in Figure 2.2c.

2.3. Leakage Model

In SCA, a leakage model is a mathematical model that predicts how secret-dependent sensitive values targeted for the analysis relate to the leakage that exists in the

[3] This is called incomplete NTT because the implementation of Kyber skips the last layer of NTT as there are only n roots of unity, and the modulus polynomial $(X^n + 1)$ can at most be factorized into polynomials of degree one.

[4] There are more effective attack points for SCA of Kyber depending on the attack scenario and implementation. However, for the attack scenario that we follow in Chapter 8, this is the most useful attack point.

measurements. Let's focus on the AES Sbox output as the attack point to describe the leakage model more clearly. As discussed earlier, the divide-and-conquer strategy allows us to focus on extracting only 8 bits of the key during each analysis (attack). The sensitive value is then the 8-bit Sbox output, which can be calculated as shown in Eq. (2.1). The real and unknown leakage function L of the targeted sensitive value $f(k, p)$ and some additive noise Z, modeled as a normal random variable $Z \sim \mathcal{N}(0, \sigma^2)$, shapes the measurement at time t. Eq. (2.7) gives the measurement at time I_t, as a realization of the random variable.

$$I_t = L_t(f(k, p)) + Z_t. \tag{2.7}$$

The index t is used to show that the leakage function and the noise distribution can be different for different time samples of the measurements.

The goal of an analyst is to find the function L, which maps the finite set of possible sensitive values (in the case of AES Sbox output, $\mathbb{F}_2^8 = \{0, 1\}^8$) to the set of real numbers $\mathbb{R}$. However, the function L is usually unknown to everyone, including the manufacturer and the attacker. Therefore, the function should be estimated for SCA. There are two ways to estimate the function L: **numerical approximation** or **making hypotheses**.

The **numerical approximation** of the leakage model is closely aligned with the principles of stochastic attacks [51]. In essence, stochastic attacks aim to determine an approximate function, $\hat{L}$, that best estimates the unknown true leakage function, L. If we consider L as a pseudo-boolean function, it can be expressed as a linear combination of monomial basis vectors $u \in \mathbb{F}_2^n$. Accordingly, there exists a set of real-valued coefficients a_u such that, for any sensitive intermediate value $y \in \mathbb{F}_2^n$, the leakage model can be reformulated as:

$$\hat{L}(y) = \sum_{u \in \mathbb{F}2^n} a_u \cdot g_u(y),\ a_u \in \mathbb{R}, \tag{2.8}$$

where g_u denotes the base function of the intermediate data. The common assumption is that the leakage of each byte depends on its 8 constituent bits. Consequently, the base functions are defined as $[1, y[1], y[2], \cdots, y[8]]$, where $y[j]$ represents the j^{th} bit of y. Hence, the leakage function L can be approximated by a multivariate polynomial in the bit-coordinates $y[j]$, with coefficients in $\mathbb{R}$. The standard approach to estimating the coefficients a_u is to apply the *ordinary least squares* (OLS) method [52, 53].

However, Eq. (2.8) also unveils the core limitations of the stochastic attack. This model approximates the linear portion of L using base functions but fails to encompass non-linear parts. At the circuit level, CMOS power consumption is not strictly linear with respect to the number of switching bits due to effects such as glitches, short-circuit currents, and signal contention. Furthermore, leakage may exhibit inter-bit and inter-register dependencies, where the leakage of a given bit depends on the values of other bits, violating the assumption of independent contributions captured by linear base functions. Nonlinearities can also be introduced by memory accesses, microarchitectural effects, and the analog measurement chain

itself. These factors can significantly reduce the accuracy of linear leakage approximations and motivate the use of higher-order models or nonlinear learning techniques. Furthermore, Eq. (2.8) neglects potential multivariate key-dependent noise terms. These constraints limit the discriminative power when identifying different leakages, leading to mediocre performance when, for instance, dealing with low numbers of side-channel traces [54].

Making leakage model hypotheses is widely used in academia and industry. The commonly used leakage model hypotheses can be interpreted as specific instantiations of Eq. (2.8), obtained by restricting the choice of base functions and coefficients. For instance, the following leakage models are commonly used to construct $\hat{L}$:

- **Hamming weight (HW) and Hamming distance (HD) models:** These models assume that the leakage function is proportional to the sum of the bit values (HW) or bit transitions (HD). In the HW case, the leakage function can be expressed as Eq. (2.9)

$$\hat{L}(y) \approx \sum_{j=1}^{8} y[j], \tag{2.9}$$

  which corresponds to Eq. (2.8) with identical coefficients a_u for all single-bit base functions, assuming equal contribution of each bit to the leakage.

- **Identity (ID) model:** The ID model assumes that different bit patterns of the sensitive value produce distinguishable leakage levels. In this case, the leakage function is approximated as Eq. (2.10):

$$\hat{L}(y) \approx y, \tag{2.10}$$

  implicitly allowing different weights for different bit combinations and thus relaxing the equal-contribution assumption of the HW/HD models.

There is also a Most/Least Significant Bit–based distinguisher, which does not model the full leakage but instead focuses on a specific part of it. MSB/LSB focus on a single bit of the sensitive value. Accordingly, the leakage function is approximated as Eq. (2.11):

$$\hat{L}(y) \approx y[j], \tag{2.11}$$

where j corresponds to the MSB or LSB, representing a highly constrained instance of Eq. (2.8).

2.4. Countermeasures

Countermeasures are hardware, software, or algorithmic techniques designed to reduce or ideally eliminate the exploitable leakage of sensitive information and make it significantly harder for an attacker to extract secret data. Countermeasures typically aim to break or weaken the dependency between the processed sensitive values and the observable side-channel signal.

Masking is a widely used countermeasure against side-channel attacks. At its heart, it involves splitting every sensitive intermediate variable that depends on secret parameters into several shares. The goal is to randomize the processed data, making it difficult for an attacker to predict the leakage. For instance, a variable x can be split into $d + 1$ shares x_i such that:

$$x = \bigoplus_{i=0}^{d} x_i, \tag{2.12}$$

where $\bigoplus$ denotes bitwise XOR. The computations are then performed on these shares instead of the original sensitive value. The security level of a masking scheme is often defined by its masking order d. This is commonly referred to as d^{th} order SCA security, where no information about the secret can be recovered from observing less than $d + 1$ shares or intermediate values. Theoretical studies show that the complexity of a successful side-channel attack increases exponentially with the masking order d [55]. Intuitively, each masking share hides a part of the secret behind independent randomness. To recover the secret, an attacker must combine more and more noisy leakage signals at once; every extra share adds another layer of randomness, so the useful signal shrinks much faster than linearly, which is why the required attack effort grows exponentially with the masking order. Different methods exist for splitting and operating on shares, often tailored to the mathematical operations involved in the cryptographic algorithm. Examples include:

- **Boolean Masking**, where a value x is split into n shares $(x_1, x_2, \ldots, x_n)$ such that their bitwise XOR equals x as shown in Eq. (2.12).
- **Arithmetic Masking**, where, x is viewed as an element of $\mathbb{Z}/q\mathbb{Z}$ and split into n shares whose sum modulo q equals $x (x = \sum_{i=1}^{n} x_i)$.
- **Multiplicative Masking**, where x is an element of a finite field K and is split into shares whose product equals x ($x = x_1 \times x_2 \times \cdots \times x_n$), and more.

Masking linear operations is generally straightforward, as the operation can be applied independently to each share, and the shares' combined result remains correctly masked. Non-linear functions like Sbox are much more challenging, as they require "crossing domain borders" or combining shares from different domains. To maintain security, dedicated measures are needed, often involving fresh random shares and registers to prevent the propagation of information-revealing glitches.

There are other kinds of countermeasures. These include, but are not limited to:

- **Shuffling**, which randomizes the order of operations that are not dependent on each other. For example, it can be used during polynomial multiplication in lattice-based schemes. An example can be seen in the work of Zijlstra et al. [56], or Chen et al. [57].
- **Randomization**, which includes inserting random delays between operations, making the attacks more difficult, as they require trace synchronization. The source of this randomization can be a dummy operation or the clock. It is a very lightweight countermeasure and was implemented, for example, in [58].

- **Duplication** (a sort of **hiding** countermeasure), which simply uses two cores that run on the same input data but with different keys, while each core is driven by different randomized clocks. This countermeasure comes at a high cost, doubling resource usage.

2.5. Machine and Deep Learning

2.5.1. Learning Algorithms

Let us start by considering a well-known definition of learning in the context of computer programs from Mitchell [59]: "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*." and define the important elements in this definition with more emphasis on the application in the side-channel domain.

In this definition, the **Task** *T* refers to the specific activity or goal the program aims to accomplish, for example, recognizing objects, predicting values, or translating text. Learning itself is not the task but the means of achieving the ability to perform it. Below are some examples of tasks that can be accomplished through machine learning. These are only a few of the many types of tasks possible.[5]

- Classification: The program assigns inputs to one of several predefined output categories, for instance, the task can be identifying whether an image contains a cat or a dog.
- Density or Probability Function Estimation: The program learns the probability distribution underlying observed data, describing how likely different outcomes are. An example is modeling the distribution of customer purchasing behavior to detect unusual transactions.
- Regression: The program predicts a numerical value based on input data, for instance, the task can be estimating the future price of a house in a neighborhood based on the prices of the sold houses in the past year.
- Translation: The program converts a sequence of symbols in one language into another. A well-known example is when you use a translation tool to translate a sentence from English to another language and vice versa.
- Denoising: The program reconstructs a clean version of data that has been corrupted or distorted. An example can be removing background noise from an audio recording.

The **performance measure** *P* is a quantitative way to evaluate how well a machine learning algorithm performs a specific task. It defines what "success" means for the system and provides a numerical basis for comparison. The choice of *P* depends on

[5] This section is based on the presentation of deep learning concepts in the book Deep Learning by Goodfellow, Bengio, and Courville [60]. In particular, the definitions, terminology, and illustrative examples in this section follow that source, with required adaptation to the context of this thesis.

the task: for classification, it is often accuracy (the proportion of correct outputs) or its complement, the error rate. For regression, measures like mean squared error may be used, while for density estimation, the likelihood assigned to observed data can indicate performance. Since real-world use depends on how the model generalizes, P is typically computed on unseen test data rather than the training set. Selecting an appropriate performance measure can be challenging as it must reflect the system's intended behavior and be practically computable, even when the ideal measure is difficult or impossible to calculate directly.

The **experience** E in Mitchell's definition refers to the information or data the algorithm is exposed to during the learning process. Most machine learning algorithms are exposed to a dataset, a collection of examples, where each example consists of several *features* that describe measurable properties of an object or event. Through this experience, the algorithm learns patterns or relationships that allow it to perform the task more effectively. Considering the experience and dataset available, there are different learning approaches. Two widely used paradigms are **supervised and unsupervised** learning.

In **supervised learning**, the algorithm experiences a dataset consisting of multiple examples, where each example consists of a set of input features and an associated label. Each example can be represented as a pair (x, y), where $x \in \mathbb{R}^m$ is a vector of observed features and y is a corresponding value or vector that the algorithm should learn to predict. The objective of supervised learning is to learn a mapping function $f : x \to y$ that minimizes the difference between the predicted output $f(x)$ and the true label y. In other words, supervised learning can be described as the process of estimating the *conditional distribution* $p(y \mid x)$, which captures how likely different outputs y are, given an input x. The algorithm achieves this by observing several examples of (x_i, y_i) pairs and adjusting its parameters, θ, to improve predictions according to a performance measure such as accuracy or mean squared error by computing $\hat{y} = \arg\max_\theta p(y \mid x)$ or, equivalently, by minimizing an expected loss function[6] over the training examples. Once trained, the model can infer the most probable output for new inputs using the learned function. Supervised learning covers tasks such as classification and regression, where the model learns from labeled examples provided. These tasks rely on learning the relationship between inputs and outputs so that the model can generalize to new, unseen data.

In **unsupervised learning**, the algorithm is not provided with labeled outputs. It only experiences input data x containing many features without any associated labels. The algorithm must learn to identify useful patterns or structures on its own; for example, it should learn the underlying *probability distribution* $p(x)$. Clustering, where similar examples are grouped together, and density estimation, where the system estimates how data are distributed in the input space, are two examples of unsupervised learning. Unsupervised learning helps the system make sense of data without explicit guidance from a teacher, often serving as the foundation for more complex tasks like data compression, feature extraction, or anomaly detection.

The central goal of machine learning is to achieve good **generalization**, the ability

[6] A loss function (or cost function) is a mathematical function that quantifies the difference between a model's predicted output and the actual target value.

of a model to perform well on new, unseen data rather than merely on the data used for training. While minimizing the training error ensures good performance on the training set, what truly matters is minimizing the **generalization error**, which is the expected error on new inputs drawn from the same data-generating distribution. The gap between training and test performance reflects how well a model generalizes. Under standard assumptions (such as independent and identically distributed (i.i.d.) sampling of training and test examples), this generalization behavior can be mathematically analyzed and related to the learning process.

A model's capacity determines how complex a relationship it can represent between inputs and outputs. Models with low capacity are unable to capture the underlying patterns in the data and thus suffer from **underfitting**, characterized by high training and test errors. In contrast, models with high capacity can memorize noise or irrelevant details from the training data, leading to **overfitting**, where the training error is very low. Still, the test error increases due to poor generalization. The ideal model has sufficient capacity to fit the true underlying function but not so much that it captures random fluctuations in the training set.

A **hyperparameter** is a setting that controls the behavior or structure of a machine learning algorithm but is not learned directly from the training data. Instead, it must be chosen externally, often through experimentation or systematic search, a process known as **hyperparameter tuning**. Typical examples of hyperparameters include the number of layers and neurons, the learning rate, the activation function and initialization method in a neural network, the regularization strength, the kernel type and kernel parameters in support vector machines, or the depth of decision trees. Hyperparameters strongly influence the capacity of a model and, if optimized solely on the training set, may lead to overfitting by favoring overly complex models that fit the training data too well. To mitigate this risk, a separate validation set—comprising samples not used during parameter learning—is employed to evaluate model performance and guide hyperparameter selection. This ensures that hyperparameters are chosen to promote good generalization rather than merely minimizing training error.

2.5.2. Deep Neural Networks

Deep Neural Networks (DNNs) are powerful tools for learning complex mappings between inputs and outputs. They can be applied to various tasks, including classification, regression, translation, and density estimation. A deep neural network is composed of multiple layers of interconnected processing units, called **neurons**, each of which transforms its inputs through a weighted sum followed by a nonlinear activation function σ.[7] Formally, the output of a layer can be written as Eq. (2.13):

$$h^{(l)} = \sigma(W^{(l)}h^{(l-1)} + b^{(l)}), \tag{2.13}$$

where $h^{(l-1)}$ is the input to the l-th layer, $W^{(l)}$ and $b^{(l)}$ are the weights and biases, and $\sigma(\cdot)$ is a nonlinear activation function such as ReLU or sigmoid. By composing many

[7] An activation function is a mathematical operation applied to a neuron's output that introduces nonlinearity, enabling a neural network to learn complex relationships between inputs and outputs.

such layers, the network can learn hierarchical representations of data, gradually transforming raw input features into output.

The goal of training a neural network is to find the set of parameters $\theta = \{W^{(l)}, b^{(l)}\}$ that minimize a cost function (also called a **loss function**) $J(\theta)$, which measures how far the model's predictions deviate from the true outputs. For a dataset of n examples $(x_{(i)}, y_{(i)})$, the cost function is often expressed as the average loss over all examples, calculated using Eq. (2.14):

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} j(f(x_{(i)}; \theta), y_{(i)}), \tag{2.14}$$

where $f(x_{(i)}; \theta)$ is the model's prediction for input $x_{(i)}$, and $j(\cdot)$ quantifies the difference between the predicted and true values, such as mean squared error for regression or cross-entropy loss for classification. Minimizing this cost function enables the network to learn the underlying mapping from inputs to outputs.

To minimize $J(\theta)$, neural networks rely on iterative optimization algorithms such as stochastic gradient descent (SGD) and its variants. During training, gradients of the cost function with respect to the parameters are computed using backpropagation, an efficient application of the chain rule. The parameters are then updated as Eq. (2.15).

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta), \tag{2.15}$$

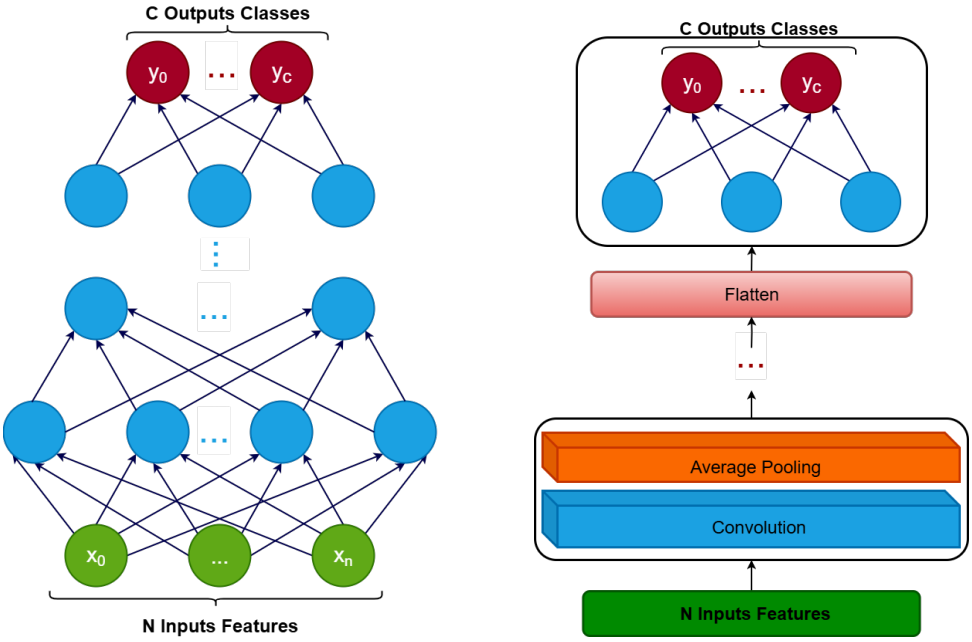
where η is the learning rate controlling the step size of each update. The network gradually reduces loss and improves its ability to generalize to unseen data through repeated updates across many training examples. The structure of deep networks (a stack of multiple nonlinear layers with many neurons) gives them exceptional capacity to approximate complex functions, though managing this capacity is essential to avoid overfitting.

Neural Networks Hyperparameters and Parameters. In neural networks, parameters and hyperparameters play different roles, and understanding their distinction is essential for effective model design and training. Parameters are the internal values that the model learns automatically during training through optimization techniques, for example, weights and biases in MLPs and the fully connected part of CNNs, and the filter (kernel) coefficients within convolutional layers. Parameters are directly responsible for the model's predictions in each iteration. In contrast, hyperparameters are values set before training and control the model's structure and learning process rather than being learned from data.

Two Common Neural Network Topologies

Neural network architectures come in various forms, each designed for specific types of data and learning objectives. Feedforward Neural Networks (FNNs) are used for general function approximation and form the foundation of many other architectures. Convolutional Neural Networks (CNNs) are specialized for processing spatial or grid-like data such as images, while Recurrent Neural Networks (RNNs) and their extensions like Long Short-Term Memory (LSTM) networks are well-suited

for modeling temporal or sequential data. More recently, Transformers have become dominant in large-scale language modeling and multimodal learning due to their ability to capture long-range dependencies through attention mechanisms. These architectures differ in structure, connectivity, and learning dynamics, allowing deep learning to address diverse domains and increasingly complex problems. Here, we introduce two widely used neural networks for SCA.



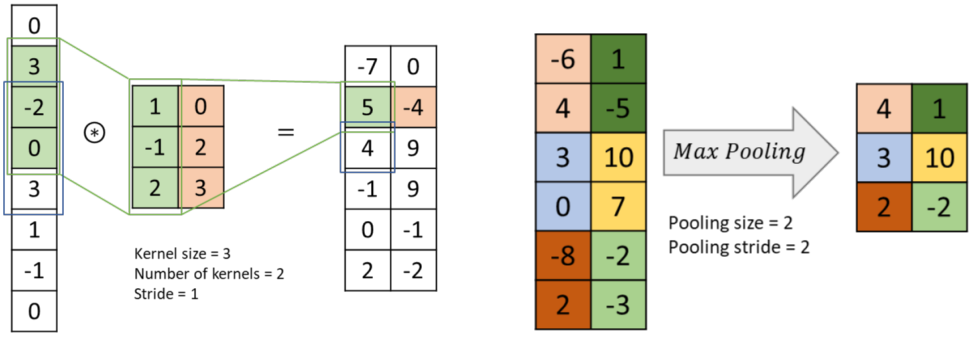
(a) MLP architecture (b) CNN architecture

Figure 2.4: Considered neural network topologies.

Multilayer Perceptron (MLP) is a type of feedforward neural network composed of an input layer, one or more hidden layers, and an output layer. Each layer contains multiple neurons, which are computational units that calculate a weighted sum of their inputs and add a bias term. A nonlinear activation function is then applied, enabling the network to learn complex, nonlinear relationships between inputs and outputs. The input layer receives the training data, which is passed through fully connected hidden layers to the output layer, where predictions are made (e.g., class labels in classification tasks). MLPs learn by iteratively adjusting the weights of the connections using optimization techniques such as gradient descent and backpropagation to minimize a chosen loss function. They are widely used for tasks like classification and regression. Figure 2.4a illustrates the general architecture of an MLP.

Convolutional Neural Network (CNN) is a specialized type of feedforward neural network designed to automatically learn spatial features from data, particularly

images. A CNN typically consists of one or more convolutional layers, where learnable filters (also called kernels) slide over the input to detect local patterns. These filters are vectors (in 1D) or matrices (in 2D) whose coefficients are updated during training using optimization techniques similar to MLPs. After each convolution, a nonlinear activation function is applied, and optionally, a pooling layer (e.g., max or average pooling) reduces the spatial dimensions while retaining the most important features. Stacking multiple convolutional and pooling layers enables the network to learn hierarchical feature representations. Finally, one or more fully connected layers are used to produce the final output. While CNNs are widely used for image classification, they can also be adapted for regression tasks by modifying the output layer and loss function. Figure 2.4b shows the general architecture of a CNN, and Figure 2.5 illustrates convolution (Figure 2.5a) and pooling (Figure 2.5b) operations.



(a) Convolution operation using two kernels

(b) Pooling operation on expanded vector using two kernels

Figure 2.5: Main operation in 1-dimensional CNN.

2.6. Deep Learning-based Side-channel Analysis

2.6.1. DL-SCA as a Classification Task

A profiling-based SCA can be formulated as a classification problem for deep neural networks. In the profiling phase, a set $\mathcal{S}$ containing N_p traces along with their labels collected from the clone device is used to train the deep neural network. Each trace is a vector of values, $\mathbf{x} = [x_1, x_2, \ldots, x_m]^\top$, sampled from the considered side-channel (for instance, power consumption or electromagnetic emanation) during the measurement. The label y for each trace is specified using the target sensitive value and the hypothetical leakage model. Let us recall our AES example from Section 2.3. With full access to the clone device in the profiling phase, we know the actual values of the key and the plaintext used during each measurement. With the Sbox output as the attack point and HW as the leakage model, the label for each trace can be calculated using Eq. (2.16):

$$y = HW(Sbox(k \oplus p)). \tag{2.16}$$

The *HW* function simply counts the number of ones in the targeted sensitive value, $Sbox(k \oplus p)$. The possible values for y in Eq. (2.16) specify the output classes for the neural network. One can guess a possible shortcoming here: the training set $\mathcal{S}$ should include examples from all the possible classes. Besides, the classes should be balanced, i.e., there should not be a class with very few examples and a class with too many examples. This shortcoming was studied in [61].

Based on the data and the training process, a model $f_{\boldsymbol{\theta}}$ is trained to predict labels on previously unseen data. Most of the supervised learning methods follow the Empirical Risk Minimization (ERM) framework, where the model parameters $\boldsymbol{\theta}$ are obtained by solving the optimization problem:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i}^{N} loss(f_{\boldsymbol{\theta}}(\mathbf{x_i}), y_i), \tag{2.17}$$

where $loss$ is the loss function.

In the attack phase, the trained model $f_{\boldsymbol{\theta}}$ is then used to classify N_a traces from the DUA. The output of this step is a matrix of probabilities showing with what probability each trace belongs to each possible output class. The output probabilities of the neural network for each class are then used to rank the most probable key, as is described in the next section.

2.6.2. Evaluating DL-SCA Performance

Two common performance metrics for deep neural networks are accuracy and loss value. Using leakage models like HW or HD, SCA suffers from severe class imbalance, where dominant leakage classes can be predicted correctly without learning meaningful key-dependent information. Moreover, accuracy evaluates each trace independently, whereas SCA usually needs to aggregate evidence into a single key across multiple traces. On the other hand, loss functions (e.g., cross-entropy) optimize how well the model predicts class labels, not how well it ranks key hypotheses. On top of that, both accuracy and loss focus only on the correct class, whereas information aggregation should be performed across all classes to detect the most probable key. As a result, minimizing loss or maximizing accuracy does not necessarily improve guessing entropy or success rate, and can even lead to conflicting conclusions about attack effectiveness [61]. Therefore, we need a metric that can accumulate the small biases of the model prediction toward the correct key. Guessing Entropy and Number of Attack Traces are two DL-SCA performance metrics.

Guessing Entropy Simply stated, guessing entropy [62, 63] is the average number of guesses that must be made before finding the correct key. The output of model inference in the attack phase of DL-SCA with N_a traces is a two-dimensional matrix of probabilities with dimensions equivalent to $N_a \times c$, where c is the number of output classes. A common practice is to use the maximum log-likelihood distinguisher to

create a cumulative sum $S(k)$ for each key candidate k in the key space as follows:

$$S(k) = \sum_{i=1}^{N_a} \log(p_{i,y}), \tag{2.18}$$

where, $p_{i,y}$ is the probability that the trace x_i belongs to class y. The class y is derived from the hypothetical key k and input d_i (the input can be the plaintext, ciphertext, nonce, etc.), which results in $y \in \mathcal{Y}$ through a cryptographic function and a leakage mode. To make it clear, let us again recall the AES example from Section 2.3 and Section 2.6.1. To link the class probabilities to each key hypothesis k, the known plaintext p_i used during measuring the attack trace is used to compute the hypothetical intermediate value $v_i(k) = Sbox(k \oplus p_i)$ and its class label $y_i(k) = HW(v_i(k))$. The probability $p_{i,y_i(k)}$ predicted by the network for this class is then attributed to key k, enabling the cumulative sum in Eq. (2.18) to evaluate how likely each key candidate is.

Using the $S(k)$ scores for all the possible keys in the key space, a key-guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ can be built. This vector arranges the key candidates in descending order of probability, with g_1 being the most likely candidate and $g_{|\mathcal{K}|}$ being the least likely.

In a typical attack, the average key ranks over multiple experiments are computed, and the top o key candidates are enumerated to obtain the correct key. Thus, guessing entropy is the average position of the correct key, k^*, in $\mathbf{g}$. Eq. (2.19) provides the formal definition of guessing entropy:

$$GE = E(rank_{k^*}(\mathbf{g})), \tag{2.19}$$

where $rank_{k^*}(\mathbf{g})$ denotes the position of the correct key k^* in the probability vector $\mathbf{g}$, and E is the expectation operator.

An attack is considered successful if $GE = 1$. Guessing entropy is usually employed to estimate the effort required to uncover the secret key k^* [64] and is a standard metric used in various SCA standardizations like ISO/IEC17825:2024. Throughout this thesis, guessing entropy, GE, is reported as the average rank of the correct key over 100 experiments. The key is fixed over the experiments, while the plaintexts (and the measurements accordingly) are changing for each experiment.

Number of Attack Traces The number of attack traces, NT, is the minimum number of traces needed to always place the correct key in the first position of $\mathbf{g}$. The number of traces is a natural and straightforward metric to indicate how costly it is to recover the device's secret key, given the number of encryption (or decryption) operations for which the attacker must measure its leakage. This metric is the most common approach to measure side-channel security. Several certification bodies and government agencies have integrated this metric into their guidelines; for example, the current version of standard ISO/IEC 17825 proposes the usage of specific side-channel analysis techniques and requires that no security flaws are found after measuring 10,000 or 100,000 traces with fixed key and variable plaintexts, depending on the desired security level [65].

2.6.3. DL-SCA Experience Datasets

We use multiple datasets throughout this thesis to evaluate whether the experimental results can be generalized. Since some datasets are reused across different chapters, this section provides an overview of all datasets used, while the individual chapters refer only to their names without repeating the details. Readers can return to this section whenever information about a dataset's characteristics is needed. As introduced earlier in Section 2.1, this thesis analyzes implementations of three cryptographic schemes, and the datasets are therefore organized according to their corresponding algorithms. All datasets used in this work are publicly available, which supports reproducibility and enables direct comparison with related work. The focus of this thesis is therefore not on acquiring new side-channel measurements, but on evaluating deep learning-based attack methods on established datasets.

AES Implementations

ChipWhisperer dataset provides measurements of an unprotected AES software implementation running on an 8-bit XMEGA mounted on a ChipWhisperer CW308 UFO board [66]. This dataset has been used in previous works [67, 68]. In total, $10,000$ traces were captured when the target encrypted $10,000$ randomly generated plaintexts with a fixed key. Each trace includes $5,000$ sample points. The usual attack point is the common first round $Sbox$ output.

AES-HD dataset[8] was introduced in [61]. This dataset is collected using an FPGA implementation of AES-128 on Xilinx Virtex-5. The implementation is unprotected. The side-channel measurements are the target's electromagnetic emanations (EM), which are represented by $1,250$ time samples. In total, $500,000$ traces were captured when the target encrypted $500,000$ randomly generated plaintexts with a fixed key. A common attack point for this dataset is the last round's $Sbox^{-1}$ output overwriting in a register that contains the previous inverse ShiftRows operation value. The leakage is modeled as $Y = Sbox^{-1}[C_j \oplus k_j] \oplus C_{j'}$ or its Hamming weight (as the implementation is hardware-based), where C_j and $C_{j'}$ are two ciphertext bytes related according to the inverse ShiftRows operation, and k_j is the corresponding round key byte. In our experiments, $j = 10$ and $j' = 6$.

ASCAD-F dataset[9] was introduced in [7]. This dataset was provided using an assembly (software) implementation of AES-128 published by ANSSI. The implementation is protected with Boolean masking. Since the first and second bytes are masked with zero masks, and the sensitive variable leaks in the first order, experiments are done targeting the third byte. The platform is an 8-bit AVR microcontroller (ATmega8515), and the measurements are the electromagnetic emanations from the target [7]. In ASCAD-F, there are $50,000$ traces for training, and $10,000$ traces for attack, all with the same key (this is why it is called ASCAD fixed). The traces

[8] https://github.com/AISyLab/AES_HD_Ext
[9] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/

have 700 features, an interval that includes most leaky time samples, considering the Sbox output as a sensitive variable.

ASCAD-R dataset[10] contains measurements from the same software implementation and measurement setup as ASCAD-F [7]. The difference is that ASCAD-R also provides traces with random keys (for the profiling phase), providing more sample points per trace. Each measurement has $250,000$ time samples, which were finally reduced to a window containing $1,400$ most-leaky time samples. The cryptographic operation was repeated $200,000$ times with varying plaintext and keys to collect the profiling set examples, then repeated another $100,000$ times with varying plaintext and a fixed key to collect the attack set examples. Details about the cryptographic design are provided in [7]. Similarly to ASCAD-F, since the first and second bytes are masked with zero (so they can be broken with first-order SCA), the experiments are done considering the third byte targeting $Y = Sbox[P_3 \oplus k_3]$ for the ID leakage model and $Y = HW(Sbox[P_3 \oplus k_3])$ for the HW leakage model.

eShard dataset[11] contains electromagnetic emanation leakage of a software implementation of AES-128 encryption. The implementation makes use of Boolean masking and a shuffling of the Sbox operation order to protect against SCA to a certain extent. The targeted chip is an STM32F446 32-bit microcontroller based on Cortex-M4, running at a clock speed of 30 MHz. Near-field electromagnetic measurements were performed using a Langer probe RF-B 0.3-3 through the epoxy package [69]. The raw traces have $35,000$ time samples, which have been trimmed to $1,400$. The dataset includes $100,000$ traces with a fixed key.

CHES-CTF dataset was released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). The target implementation is masked AES-128 encryption executing on a 32-bit STM microcontroller. CHES-CTF dataset includes $50,000$ traces with fixed keys. However, the fixed key in the training set differs from the key in the attack set. Each trace consists of $2,200$ sample points. The dataset is no longer available online.

Ascon Implementations

Ascon-Unprotected dataset[12] collected using multiple operations of the reference software implementation of Ascon-128 [70]. The traces contain power samples during the first round permutation of the Ascon-128 initialization. The training set contains $50,000$ traces captured during the first round permutation in the initialization phase of Ascon using randomly generated nonces and randomly generated keys. The test set contains $10,000$ traces captured during the first round permutation in the

[10] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key
[11] https://github.com/eshard/scared
[12] https://zenodo.org/records/10229484

initialization phase of Ascon using randomly generated nonces and a fixed key. Each trace contains 772 time samples covering the targeted permutation. The traces are collected using the ChipWhisperer Lite board and its internal oscilloscope, and the target is the STM32F4 microcontroller, a 32-bit platform.

Ascon-Protected dataset[13] introduced in [71]. The dataset is provided using a first-order protected implementation of Ascon. This implementation uses bit-interleaved and a specific masking countermeasure designed to be efficient with the Ascon Sbox.[14] The *C* implementations by the Ascon team are available in their GitHub repository [70]. Traces are collected using a ChipWhisperer Lite board and an 8-bit precision oscilloscope, coupled with the STM32F4 target running at a frequency of 7.37MHz. The dataset contains traces of Ascon's first-round permutation in the initialization phase. There are 500, 000 profiling traces with variable keys and 500, 000 attack traces with a fixed key in the dataset. Each trace has 1, 408 time samples.

Kyber Implementation

Reference-PPM dataset[15] is based on the reference implementation of Kyber from the PQClean repository[16] and uses the `PQCLEAN_MLKEM768_basemul` function as the target of observation. The Reference-PPM dataset was collected by executing the Kyber decapsulation procedure on an STM32F3 microcontroller, which features a 32-bit ARM Cortex-M4 core running at 7.372 MHz. Power measurements were acquired using the ChipWhisperer CW308 [73] platform in combination with a Lecroy 610Zi oscilloscope.

Each measurement corresponds to one execution of the decapsulation procedure using a fixed secret key and a varying ciphertext. A total of 100, 000 traces were collected. Each trace consists of 50, 000 time samples, and the measurements are focused specifically on the pairwise coefficient multiplication in the NTT domain.

The dataset contains two main components:

- Power traces: in total, Reference-PPM provides 100, 000 traces, each trace represented with 50, 000 time samples.
- Intermediate values: for each trace, there is the corresponding variable set that includes six values (2 of them are the same). The values include two coefficients of the secret key, two coefficients of the u part of the ciphertext, and the pointwise multiplication of these coefficients. All the values are shown

[13] https://zenodo.org/records/10229484

[14] The masking technique used in the implementation is called Domain-Oriented Masking (DOM). It is a specialized technique used in cryptographic hardware, but it can also be applied in software implementation. It involves dividing a circuit into separate domains, each handling only one part of the data. This separation ensures that each domain only accesses a specific portion of the data, reducing the risk of data leakage through side-channel attacks. For more reading, refer to the implementation of Ascon and [72].

[15] https://doi.org/10.5281/zenodo.15352482

[16] https://github.com/pq-crystals/kyber

in 2 bytes in little-endian format. Traces, sensitive inputs, and intermediate values are stored in MATLAB data files. More information can be found in [74].

General Notes

It is important to note that the target platforms and trace-acquisition setups differ across the datasets introduced in this section. Moreover, even when the same cryptographic algorithm is used, the underlying implementations are not identical. For instance, the AES implementations used in the ASCAD and eShard datasets differ in their code structure, countermeasures, and SCA-oriented design decisions.

The datasets also differ along several additional dimensions. Each dataset contains a different number of features, which is influenced by factors such as the objective of the dataset provider, the available acquisition resources, the characteristics of the implementation, and the constraints of the target platform. For instance, the AES-HD dataset captures the full encryption process, as the complete FPGA-based implementation executes within 11 clock cycles. In contrast, Reference-PPM focuses only on a specific function within the decapsulation procedure, since the full *C* code implementation is considerably longer. In this case, even capturing the complete decapsulation procedure is not feasible due to the limited buffer size of the acquisition setup. Besides that, the selected scope reflects the evaluation objective of the dataset provider. For example, in the Ascon datasets, only the permutation during the initialization phase is recorded because this operation was the intended target of analysis.

The datasets also vary significantly in the number of profiling and attack traces. This variation is influenced by factors such as available acquisition resources, the intended use of the dataset, and the noise levels of the traces across different implementations. In Table 2.1, one can see which datasets were used in which Chapter and why.

Table 2.1: Overview of the datasets used in each chapter and the rationale behind their selection.

Chapter	**Datasets**	Why we chose these datasets
Chapter 3	ASCAD-R AES-HD	We evaluated the effect of adding more traces in two scenarios: a high-SNR setting represented by ASCAD-R, and a low-SNR setting represented by the AES-HD dataset from an FPGA-based AES implementation.
Chapter 4	ASCAD-R AES-HD Ascon-Unprotected	Beyond the two scenarios above, this is our first attempt to test whether the DL-SCA improvements generalize beyond AES.
Chapter 6	Ascon-Unprotected Ascon-Protected	While ensemble techniques have been studied for AES, DL-SCA on Ascon has not yet been explored. This chapter evaluates DL-SCA and ensemble methods simultaneously to assess their effectiveness for Ascon.
Chapter 7	ASCAD-F ASCAD-R CHES-CTF eShard	We used a combination of different AES implementations, which vary in their leakage characteristics, to ensure that the multi-bit multi-byte approach can handle diverse leakage distributions in practice.
Chapter 8	ChipWhisperer Ascon-Unprotected Reference-PPM	Since our goal was to evaluate whether DL-BSCA is a generally applicable approach, we focused on three cryptographic algorithms that differ significantly from one another.

3

To Overfit, Or Not to Overfit: Improving the Performance of Deep Learning-based SCA

Profiling side-channel analysis allows evaluators to estimate the worst-case security of a target. When security evaluations relax the assumptions about the adversary's knowledge, profiling models may easily be sub-optimal due to the inability to extract the most informative points of interest from the side-channel measurements. When used for profiling attacks, deep neural networks can learn strong models without feature selection with the drawback of expensive hyperparameter tuning. Unfortunately, due to very large search spaces, one usually finds very different model behaviors, and a widespread situation is to face overfitting with typically poor generalization capacity.

Usually, overfitting or poor generalization would be mitigated by adding more measurements to the profiling phase to reduce estimation errors. This paper provides a detailed analysis of different deep learning model behaviors and shows that adding more profiling traces as a single solution does not necessarily help improve generalization. We recognize the main problem to be the sub-optimal selection of hyperparameters, which is then difficult to resolve by simply adding more measurements. Instead, we propose to use small hyperparameter tweaks or regularization as techniques to resolve the problem.

3.1. Introduction

As mentioned earlier, side-channel analysis explores unintentional information leakage from cryptographic devices. Consequently, it is in the interest of manufacturers to evaluate the robustness of their products against threats posed by both non-profiled and profiled analysis. Profiling analysis allow a more formal security evaluation of a device by implementing a profiling model from side-channel measurements obtained from a device identical to the device under evaluation. Since an evaluator can learn a statistical model from the existing leakage, several countermeasures against non-profiled analysis may become ineffective against profiled analysis, such as Boolean masking [5, 75]. This depends on how much knowledge an adversary has about the target implementation (e.g., secret random shares and source code). This addition of knowledge in profiled analysis allows the estimation of worst-case security scenarios.

Theoretically, the Gaussian Template Attack [34] is assumed to be the strongest profiled method in SCA. This is true if 1) an adversary can extract the most informative (leaking) points of interest (features) from side-channel measurements, 2) the true leakage distribution (which is unknown) follows a normal distribution, and 3) the adversary has an unlimited number of measurements to build the model. In recent years, deep learning has appeared as a powerful alternative to implementing profiled models [5]. In practice, a deep neural network may skip feature selection and still provide strong models that defeat Boolean masking countermeasures. Additionally, the ability of Convolutional Neural Networks to bypass trace desynchronization effects [6] also became very attractive to the SCA community. The main drawback in deep learning (across all domains and not only SCA) is the expensive hyperparameter tuning process as mentioned in Section 1.4.1. Different solutions have been investigated to do this process in profiled analysis including grid search [76], random search [77], Bayesian optimization [78], genetic algorithms [5], and reinforcement learning [79]. Such solutions converge to different model behaviors, and usually, the best model is selected from a limited number of search attempts.

As the search spaces are usually very large, even for a few hyperparameters, the search mechanism can easily find multiple sub-optimal models for the problem. In other words, a large number of deep learning models will show overfitting or underfitting effects, leading to poor generalization ability. Finding an optimal model during the search is theoretically possible by assuming an unbounded adversary in terms of training resources, which is never the case in practice [8]. For profiled analysis in general, it is common to assume that a model is sub-optimal or wrong if the assumptions about the leakage model contain errors or the number of side-channel measurements is not sufficient [80]. This is because we aim to find the actual statistical distribution of the leakage in the profiling phase. Since this distribution is unknown, we must approximate it using density estimation techniques. The estimation process is aligned with assumption and estimation errors. The former is the consequence of incorrect assumptions about the leakage model, and the latter is the consequence of insufficient side-channel measurements [81]. Reducing any of those errors will result in the improvement of the model performance.

Intuitively, a deep learning model showing sub-optimal generalization would start

to converge to a stronger model by increasing the number of profiling traces and re-training. The expectation is that fewer estimation errors (i.e., more profiling traces) would suppress eventual assumption errors made by the model (due to hyperparameters combination and leakage model).

This chapter examines this common profiled analysis assumption that more profiling traces always improve model generalization. Motivated by the deep double descent phenomenon [82], we experimentally show that poor generalization (typically justified as overfitting and underfitting) is much more a matter of specific hyperparameters choice and not the lack of profiling traces. We generate models ranging from a few thousand to millions of trainable parameters, and we verify that in several cases, independent of leakage model and deep learning model size, adding more profiling traces does not change (or even reduce) the model's generalization ability. We investigate simple techniques based on limited hyperparameter changes and regularization to override this unexpected effect.

Additionally, understanding what causes unexpected model behaviors in deep learning (such as overfitting) allows evaluators to draw more consistent conclusions about the target's security, reducing the chances of overestimating the security of a device due to assumption errors of the model. The main contributions of this chapter are:

- We experimentally verify that the assumption that "adding more profiling traces increases the generalization of a model" is not always true for deep learning-based SCA. We investigate in detail the overfitting effect in profiled analysis. We validate that in many cases, the overfitting effect can be reduced by adding more profiling traces during training if the chosen hyperparameters combination does not result in large assumption errors.
- We run experiments for software and hardware AES datasets and demonstrate that this behavior is not dataset dependent.
- Considering the first contribution, we show that when increasing the profiling set size does not improve generalization, using regularization techniques and applying small changes in hyperparameters can improve the performance by reducing the assumption errors.

This chapter does not aim to find any specific model that will break a target with fewer measurements than state-of-the-art. Rather, it provides insights into a commonly observed problem in deep learning-based SCA: overfitting. Instead of simply concluding that a model does not work due to overfitting, as done in most related works, this chapter tries to find the underlying reasons for such behavior and shows how to mitigate it.

3.2. Deep Double Descent Phenomenon

The term double descent was coined first by Belkin et al. [83]. The challenge started with the common claim in modern machine learning that "larger models are always better" [84–86], while standard statistical machine learning theory predicts that larger

models are candidates for overfitting. Belkin et al. unified the classical bias-variance trade-off and the modern practice of larger models' advantage. They showed how increasing models' capacity beyond the size of the smallest model that can fit the training data with zero empirical risk could lead to performance improvement.

Bias-variance trade-off is the concept traditionally used to describe underfitting as a result of high bias and overfitting as a result of high variance. The conventional bias-variance trade-off is shown on the left of Figure 3.1. Choosing the function class $\mathcal{F}$, like neural networks, and minimizing their training risk using an objective function, the training and test risk varies based on the capacity of $\mathcal{F}$. In this trade-off, there is a "sweet spot" where a model with a specific capacity has the best performance for the given problem. Before the "sweet spot", performance increases with increasing the capacity of the models. After the "sweet spot", the performance of the models in the training set keeps increasing, but performance in the test set decreases.

More recently, deep double descent has challenged the conventional bias-variance trade-off with the neural model performance below the "interpolation point", where the interpolation point is the capacity of the smallest model from function class $\mathcal{F}$ that can fit a training set with n traces and zero training loss. Belkin et al. showed that although models with a capacity near the interpolation point have a high test risk (average of the loss in the test set), if we keep increasing the model's capacity, the risk will decrease again, and we can even reach models with better performance than the models in the "sweet spot". One can see this double descent behavior in the right of Figure 3.1. All the models beyond the interpolation point fit the training data perfectly and have zero training risk.

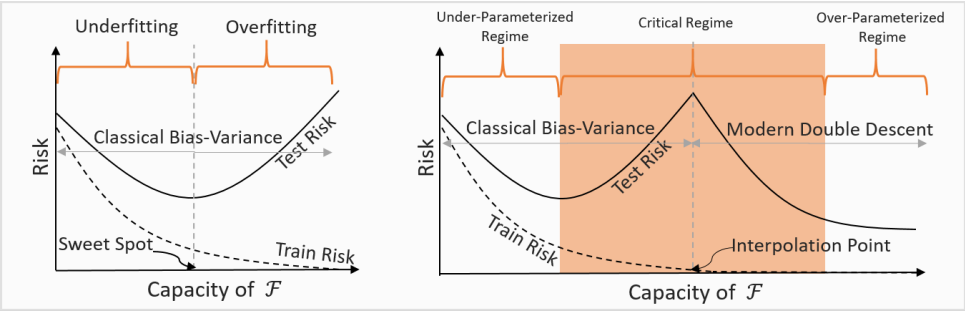


Figure 3.1: Curves for training risk (dashed line) and test risk (solid line). Left: The classical U-shaped risk curve. Right: The Double Descent risk curve.

Considering this deep double descent phenomenon, a model f from function class $\mathcal{F}$ works in one of the following regimes, depending on the proportion between the capacity of the model and the number of measurements in the training set (n):

- Under-parameterized regime: where the capacity of f is sufficiently smaller than n. The models working in this regime show an increase in performance by increasing the training set size, i.e., increasing the number of measurements in the training set until the saturation point (where increasing the training set

size does not improve the model's performance anymore).

- Over-parameterized regime: where the capacity of f is sufficiently larger than n. The models working in this regime show an increase in performance by increasing training set size unless the increase can change the regime that the model is in.
- Critically-parameterized regime: where the capacity of f and the size of the training set n are comparable. Models working in this regime may show an increase or decrease in performance by increasing the training set size. In the critical regime, the chosen model barely fits the training data, so it is fragile[1]. In the right of Figure 3.1, by increasing the model's capacity, the test risk initially increases in the critical regime, and just as the model reaches the interpolation point and can fit with zero training risk, it undergoes the second descent. In many research works, this effect cannot be observed as it is avoided through early stopping or other regularization techniques[2].

In the right of Figure 3.1, the critical regime is denoted in orange color. As one can see, the test risk reaches its peak at the interpolation point. Meanwhile, the training risk reaches zero for the first time. After that point, the training risk remains zero, and the test risk starts to decrease again. Increasing the training set size pushes the double descent curve downward by decreasing test risk. However, since a larger training set requires larger models to fit, increasing the training set also shifts the interpolation point (and the peak of test risk) to the right [82]. When we increase the training set size, these two effects combine, and we may observe that training a model with a fixed capacity with less training data shows better performance than training it with a larger training set. Consequently, increasing the training set size can hurt the performance of the model. To visually inspect this effect, one can take a look at Figure 3.2.

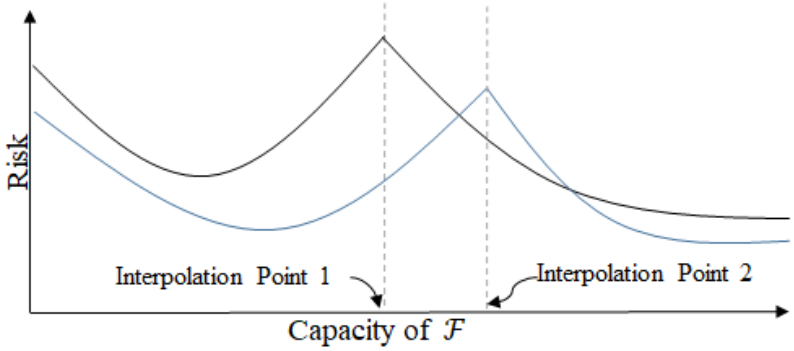


Figure 3.2: Shift of the interpolation point when increasing the training set size.

[1] Small changes in the model's hyperparameter can invalidate it.
[2] Regularization are techniques used to mitigate overfitting by reducing the complexity of the models.

3.3. Overfitting and Generalization in Side-channel Analysis

To decide about the generalization capability of neural network models in the side-channel domain, we cannot rely on accuracy and loss for the training and validation set. As discussed in [61], such machine learning metrics are not suitable in SCA (look Section 2.6.2). To overcome machine learning metrics shortcoming, besides the accuracy and loss of the model, we inspect the evolution of guessing entropy for an attack set.

Detecting overfitting and underfitting in SCA. The inability of a deep neural network to generalize in profiled SCA is usually attributed to model overfitting or underfitting. In practice, overfitting is characterized by a model that memorizes all training data but cannot generalize to different (unseen) attack traces. The main outcome is a model that mostly fits noise instead of existing leakage. This scenario becomes even more critical in realistic profiled analysis where the attack traces are collected from a different device. Underfitting happens when the model is trained for an insufficient number of iterations (i.e., epochs), and training metrics indicate a model that cannot even memorize training data.

A profiling model is considered sub-optimal if there are assumption or estimation errors [80]. In the case of deep learning, assumption errors also include hyperparameter combinations that lead to insufficient generalization. However, there are cases when the trained model still eventually recovers the correct key at the price of the increased number of attack traces. Thus, it is common to observe the following scenarios:

- Model overfits but still recovers the correct key information: in this case, training accuracy reaches 100% (resp. training loss approaches zero) at the end of the training process. The overfitting is identified by checking the validation accuracy, which is typically close to random guessing, and the validation loss, which tends to grow as training continues. However, when processing a sufficient number of attack traces during the *GE* calculation, it is still possible to rank the correct key as first (or among the first ones). Of course, as the model overfits and the *GE* is usually verified for side-channel measurements obtained from the same device (ignoring portability [87]), one cannot assume the model provides sufficient generalization.

- Model *tends* to overfit but still recovers the correct key byte: the only difference from the previous situation is that the model shows a continuous improvement in training accuracy (resp., decreasing in loss), but the training stops before it reaches 100%. As this stopping decision happens before the complete overfitting, the tendency is a slightly better (but still far from optimal) generalization.

- Model overfits or tends to overfit and does not recover the correct key byte: this is a case when the evaluator selects a combination of hyperparameters that creates a model that cannot fit existing leakage and predominantly fits noise from side-channel measurements.

The final profiling models achieved through any of these scenarios are defined as sub-optimal models. Even the scenarios that recover the secret information are far from ideal for security evaluations. The problem is that one assumes that a sub-optimal model represents the existing leakage distribution of the target. The direct consequence is a false sense of security, usually overestimating the security of the implementation. *Additionally, it remains unclear if the increase in the number of profiling traces, which reduces estimation errors, always ensures the reduction of negative effects due to the model's sub-optimality.*

Finding the best possible models. A typical procedure in deep learning-based profiled analysis is to run hyperparameter search methods to find the best possible models using available resources (processing power, time, and memory). To speed up the hyperparameter search process, an alternative is to reduce the number of profiling traces during the search and select the best possible model. Once the best model is found, one can increase the number of profiling traces again and re-train the model to improve its generalization.

In essence, for some models, adding more profiling traces requires small changes in hyperparameters to accommodate the larger profiling set and the expected improvement in generalization. As it is more likely that we are dealing with sub-optimal models (we do not follow the worst-case security settings), generalization will be limited and will also happen in models showing overfitting.

3.4. Analysis Methodology

In profiled SCA, an analysis that makes correct assumptions about the leakage model and its distributions can significantly improve deep neural network's performance by leveraging more profiling traces. As discussed in Section 3.2, recent deep learning results highlight the existence of a critical regime in which increasing training data can unexpectedly reduce model generalization. Not surprisingly, this also happens in the side-channel domain.

This chapter explores the effects mentioned above in the for DL-SCA. The analysis methodology aims to examine whether increasing profiling traces can mitigate overfitting effect in profiling models and to test the assumption that more training data always leads to a model with better generalization. To insure that the results can be generalized, six different combinations of neural network topologies (MLP and CNN), datasets (ASCAD-R and AES-HD) and leakage models (ID, HW and HD) are considered. These combinations are summarized in Table 3.1.

The methodology proceeds in the following steps:

1. For each combination listed in Table 3.1, 500 hyperparameter configurations are generated (resulting in 3000 models in total). These are referred to as the **baseline models**. The search space $\mathcal{S}$ is defined for hyperparameter selection using random search, with ranges reported in Table 3.2. For both MLP and CNN dense layers, we use the ranges listed under the *Dense layers* section of Table 3.2. The number of epochs is fixed at 200 across all settings. Hyperparameter ranges are based on previous work [76, 88, 89].

Table 3.1: Six different combinations considered in the experimental setup.

Dataset	NN topology	Leakage model	Combinations
ASCAD-R	MLP	HW	ASCAD-HW-MLP ASCAD-HW-CNN ASCAD-ID-MLP ASCAD-ID-CNN
AES-HD	CNN	HD	AES-HD-MLP AES-HD-CNN

Hyperparameters	Range
Dense layers	
Number of neurons	[100, 900], step = 100
Number of layers	[1, 8], step = 1
Convolution layers	
Number of layers	[1, 4], step = 1
Number of kernels	[4, 20], step = 1
First layer's filter size	[2, 4, 8, 12, 16]
$i^{(th)}$ layer filter size	$((i-1)filter-size)^2$
Pooling	"Average", "Max"
Pooling size	[2, 10], step = 2
Pooling stride	[2, 10], step = 2
Learning hyperparameters	
Optimizer	"Adam", "RMSprop"
Weight initialization	"random-uniform","glorot-uniform", and "he-uniform"
Activation function	"relu", "selu", and "elu"
Batch size	$[100, 900], step = 100$
Learning rate	$[0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001]$

Table 3.2: *MLP* and *CNN* models hyperparameters.

2. Each randomly generated neural network is first trained on a subset of the profiling set containing 50 000 traces. Performance of each model is estimated by averaging results over ten different validation sets of 10 000 traces each. Guessing entropy and the number of attack traces, are reported as the average over repeating the attack 100 times in each validation set for 5000 randomly selected traces in the set.

3. If a model reaches $GE = 1$, it is known as a *good model* and is kept for subsequent steps; otherwise, the model is discarded. To avoid subjective

interpretation of the term *good model*, we clarify that this is not an optimal profiling model but a trained neural network that can recover the target key with the considered number of profiling traces (i.e., 5000).

4. For all good models, training metrics are analyzed to determine the regime the model is working in and whether the deep neural network shows overfitting or a tendency to overfit. The schematics of the overfitting or tendency to overfit behaviors of the deep neural network models are illustrated in Figure 3.3.

5. All good models are re-trained with three other training set that their number of profiling traces are increased gradually. For ASCAD-R, we use 100 000, 150 000, and 200 000 traces. For AES-HD, we increase the number of profiling traces from 100 000 up to 400 000 in steps of 50 000. Models are always re-initialized with the same trainable parameters from step 1. This will keep all the other hyperparameters and settings the same and provide us with the possibility to focus on the effect of profiling size increments.

6. After re-training the good models with more profiling traces, we investigate the effect of profiling trace increase on the number of attack traces (the required number of traces to reach $GE = 1$). The assumption is that reduction in the number of attack traces reflects generalization improvement. From a deep learning perspective, the improvement in the validation accuracy results in assigning the largest probability to the correct key more frequently than in cases when the model shows less accuracy in the validation set. From a side-channel perspective, we are interested in models that can rank the correct key first with fewer attack traces, and this can be a measure to compare the generalization capability of different models.

7. When adding more profiling traces improves generalization (illustrated in the top of Figure 3.4), one of the following situations might happen:
   - The baseline model was working in an under-parameterized regime, and the hyperparameter combination error (assumption error) was small compared to the neural network model's estimation error. As a result, more profiling traces reduced estimation error and improved generalization.
   - The baseline model was working in an over-parameterized regime, and the assumption error was small compared to the estimation error. As a result, more profiling traces decreased estimation error and improved generalization.
   - The baseline model was working in a critical regime. More profiling traces did not invalidate the neural network model, and the hyperparameter combination error did not increase with an increasing number of profiling traces. Instead, estimation error decreased by adding more profiling traces, leading to decreased estimation error and better generalization.

8. When additional profiling traces do not improve the generalization (this situation is illustrated in bottom part of Figure 3.4), one of the following situations might happen:

- Independent of the regime the baseline model was working in, the chosen hyperparameters introduce assumption errors after increasing the profiling set size. The model needs to be adjusted to reduce the assumption error effects.

- The baseline model was working in the critical regime, and adding more traces to the profiling set skewed the model and decreased its performance.

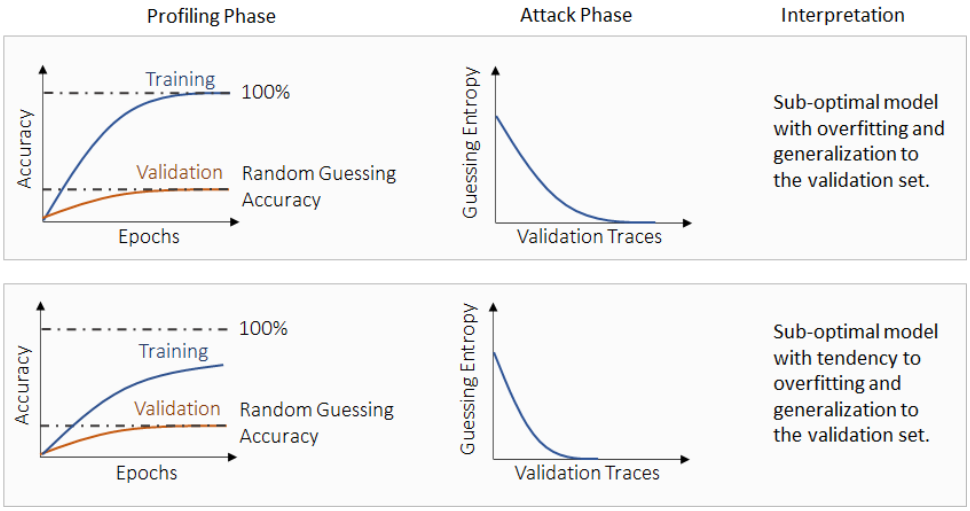


Figure 3.3: Common model behaviors.

Figure 3.3 illustrates a typical scenario identified in [61] for sub-optimal deep learning models when machine learning metrics do not reflect the performance of a model concerning guessing entropy. The fact that we illustrate accuracy as a reference machine learning metric does not imply that we are ignoring training and validation loss to decide if a model shows the potential behaviors from Figure 3.3. As proposed in [75], minimizing the cross-entropy loss should be the equivalent of solving side-channel analysis optimization. However, this principle usually does not apply to sub-optimal models with too many assumption errors, as is the case of models showing overfitting. It is important to note that for most of the models considered good models, the situation in the top part of Figure 3.4 happens more often. This means that for those models, overfitting is also caused by the lack of profiling traces, leading to insufficient profiling. On the other hand, if a model shows the behavior from the bottom part of Figure 3.4, we can modify some hyperparameters that are unrelated to the model size (e.g., learning rate, batch size, training epochs, dropout, and activation functions), which can cause the model to reduce its assumption errors.

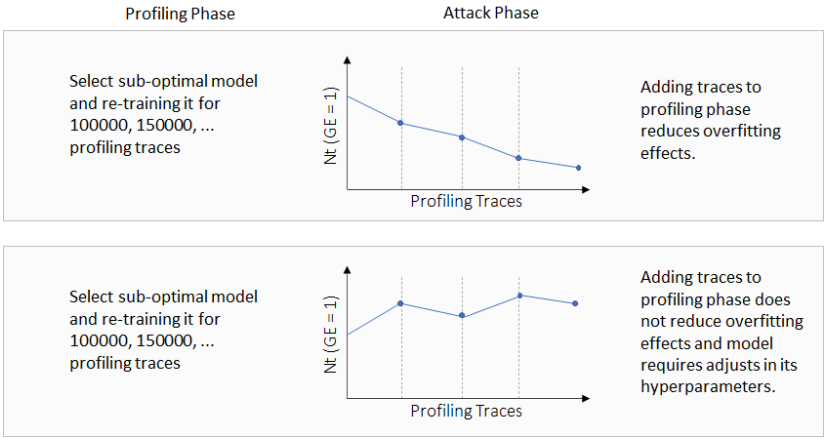


Figure 3.4: Interpreting the model behaviors.

3.5. Experimental Results

Considering the neural network capacity and the size of the profiling set, we should specify the regime that the neural network model is working on. Unfortunately, we cannot clearly specify the borders between these regimes. This is because the interpolation point definition and “sufficiently larger” and “sufficiently smaller” terms in the definition of over-parameterized and under-parameterized regimes cannot give a clear separation. Additionally, there is no widely accepted definition for the capacity of the models in deep learning, which can justify the different performance of neural network models with a similar number of trainable parameters but different topology and output layer size. To tackle these ambiguities, we selected “good” neural network models with different numbers of trainable parameters ranging from a few thousand to a few million. Then, we consider the training and validation accuracy and loss evolution to roughly specify the interpolation point for different neural network and leakage model combinations. Finally, we can roughly specify the working regime for every selected neural network model. Still, considering that the specified interpolation points are valid for the specific profiling size, by increasing the profiling size, the working regime of the model may change [82]. In the following tables, every row represents a randomly selected neural network model, denoted with a number. It can be observe that there are some models that have the same number of trainable parameters (denoted as “Params”). This simply means that different architectures have the same capacity. For various settings (datasets and leakage models), we give various numbers of models based on the number of models that converge and have different capacities (to provide good coverage from small to large model sizes).

3.5.1. ASCAD-R Dataset

Four different combinations of MLP and CNN topologies with the randomly selected hyperparameters declared in Section 3.4, and the HW and ID leakage models are

investigated for the ASCAD-R dataset. The results of experimenting with the first combination (ASCAD-MLP-HW) are shown in Table 3.3. To compare the performance of a neural network model trained with four profiling sets with different size (50 000, 100 000, 150 000, and 200 000), we consider the number of attack traces for each profiling set as the performance metric. Then refer to it as NT_1 (the average of the minimum number of attack traces that the model needs to place the actual key first when it is trained with 50 000) to NT_4 (the average of the minimum number of attack traces that the model needs to place the actual key first when it is trained with 200 000) for each neural network model. As mentioned in Section 3.3, the performance of the models that need fewer traces to reach $GE = 1$ is considered to be better.

Observe how all the neural network models ranked the correct key in the first place when trained with 50 000 traces. In many cases, by increasing the profiling size for each MLP model, the number of traces to reach $GE = 1$ follows the classical machine learning belief that more data is better (for example, #6 in Table 3.3). Still, in some cases, the performance of the neural network model (the number of attack traces) decreased when increasing the profiling set size (for example, #4 in Table 3.3). **A part of this behavior has been predicted by deep double descent**, and the proportion between the model's size and profiling set size, especially when increasing profiling size, changes the regime the neural network works in (e.g., #13 in Table 3.3). This model works near over-parameterized regime boundaries with 50 000 profiling traces since its performance in the profiling set is $\approx 97\%$. Then, by increasing the profiling set size to 200 000, its performance decreases to $\approx 60\%$. Considering the proportion between the model's size and the profiling set size, this model is working in the critical regime for 100 000, 150 000, and 200 000 profiling traces. **Still, the irregular increases and decreases in the number of attack traces considering the model's complexity are not fully justified by the double descent phenomenon.** For example one can consider #4 in Table 3.3. The accuracy of the baseline model is $\approx 45\%$, and for the rest of the profiling set sizes, its profiling accuracy is less than 30%. Considering the proportion between this model size and the profiling set sizes, and the profiling accuracy, the model works in an under-parameterized regime, and increasing number of profiling traces does not change the working regime. However, when we train the model with 100 000 and 200 000 profiling traces, it cannot reduce the key entropy at all as the model does not fit the leakage anymore.

For better visual clarity, the results of Table 3.3 and Table 3.4 are illustrated in Figure 3.5 and Figure 3.6, respectively.

One can observe the changes in the performance for two MLP architectures in Figures 3.7 and 3.8. The lighter colors in the left plot show accuracy for profiling (blue color) and validation (orange color) sets for MLPs trained with smaller profiling sets, and sequentially darker colors show these metrics after increasing the profiling set sizes to 100 000, 150 000, and 200 000 traces. Considering the profiling and validation accuracy and the proportion between the model's size and the profiling set size for #8 in Figure 3.7, we can see that the baseline model works in the critical regime, and increasing the profiling set size pushes this model toward the

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	14 109	4856	4721	1932	2352
2	14 549	2419	5000	520	631
3	28 209	3141	1577	1225	1035
4	31 149	3615	5000	622	5000
5	42 309	4045	2215	1029	972
6	44 169	4336	893	654	561
7	58 049	4034	2259	1662	1324
8	70 509	4435	1712	899	745
9	78 159	3385	1601	1306	1044
10	85 809	4742	2575	1906	1286
11	141 009	3859	1671	1020	801
12	161 209	4070	1792	1063	939
13	181 409	3357	1895	1917	1830
14	282 009	3088	1420	1078	912
15	322 209	4391	2032	1148	907
16	402 609	2559	1601	891	1139
17	523 209	4419	2515	1322	1241
18	693 909	3698	1426	1140	1046
19	724 409	4276	2238	1511	1368
20	884 809	3217	2508	1848	1606
21	964 809	1279	870	1016	1355
22	1 206 009	3117	1551	1158	970
23	1 526 409	3535	1780	1298	1305
24	1 707 009	3205	2619	5000	1111
25	1 957 509	4751	4504	2143	1911
26	2 208 009	4842	2620	3038	2278
27	2 458 509	3450	1744	1072	893

Table 3.3: *MLP* trained for the *HW* leakage model.

under-parameterized regime. To eliminate the influence of a biased attack set on the number of attack traces, we evaluated 10 different validation sets and provided mean and standard deviation values. In the right and top of Figure 3.7, one can see the mean and variance of the number of attack traces for MLP #8. In the right and bottom of Figure 3.7, one can see the average of *GE* that the model managed to reach with 5000 attack traces[3]. In Figure 3.7, increasing profiling size results in better performance concerning the accuracy, the number of attack traces, and the final *GE* that the model managed to reach with 5000 attack traces. We can even detect small changes in the number of attack traces from 150 000 to 200 000 profiling traces, which can be a sign of saturation of the model.

[3]We took incremental steps of 10 000 in case of MLP #8 and MLP #2 to make the behavior tracking easier in Figures 3.7 and 3.8

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	68 176	4840	810	225	123
2	186 156	4999	1185	337	168
3	196 256	4865	1373	514	241
4	226 556	4769	1814	469	248
5	331 656	4896	4450	1742	808
6	371 856	4991	4775	3609	1615
7	371 856	4786	3208	1370	757
8	412 056	4937	1837	543	325
9	412 056	4650	1877	464	282
10	497 356	4912	3114	1300	1263
11	497 356	4864	3161	1104	433
12	587 656	4717	3093	1625	770
13	663 056	4821	3406	1343	713
14	663 056	4937	3048	954	498
15	823 456	4989	2549	1121	576
16	828 756	4858	2675	1129	615
17	828 756	4900	3543	1620	732
18	1 039 156	4719	1047	458	281
19	1 079 256	4798	4005	2132	1105
20	1 129 456	3544	1244	561	426
21	1 129 456	2571	1026	1108	402
22	1 129 456	4829	3413	2601	1333
23	1 329 756	4986	4672	3673	2720
24	1 329 756	4836	3759	2440	1357
25	1 580 256	5000	3351	2581	1355
26	1 580 256	4651	2928	905	4818
27	1 785 856	2271	5000	5000	5000

Table 3.4: *MLP* trained for the *ID* leakage model.

MLP #2 is another MLP example that works in the under-parameterized regime. In Figure 3.8, the accuracy of the MLP model shows normal (more profiling traces increase accuracy) behavior when increasing profiling set size. The training accuracy of the baseline model is $\approx 38\%$ and decreases to 28% for the profiling set with 200 000 traces. This model's test accuracy varies from 24% for the baseline model to $\approx 28\%$ when we train it with 200 000 traces. These accuracy percentages and the proportion between the model's size and the profiling set sizes indicate that this model works in the under-parameterized regime for all profiling set sizes. Still, this model cannot reduce the guessing entropy when trained with 100 000 traces.

Considering MLP #2 in Table 3.3, while this MLP model converges to $GE = 1$ with on average 2419 attack traces for a profiling set of 50 000 traces, it cannot converge at all, i.e., cannot rank the correct key better than random guessing after increasing profiling size from 50 000 to 100 000 traces. The behavior is even more

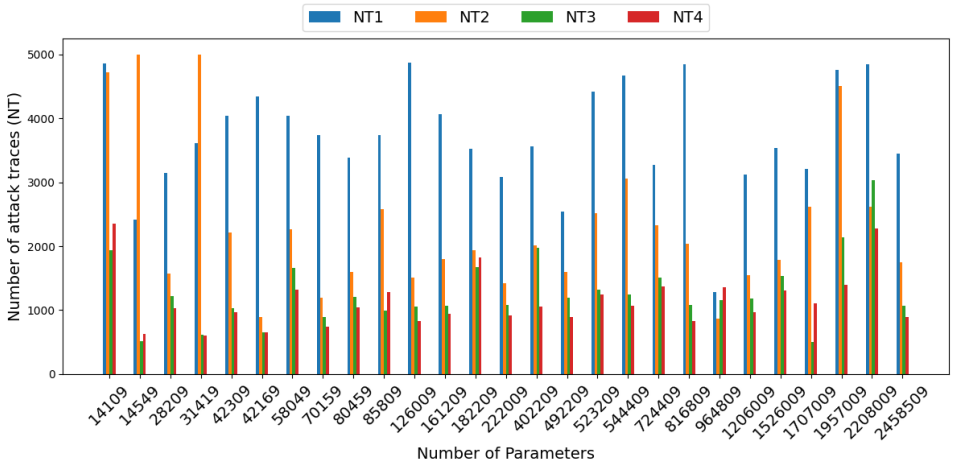


Figure 3.5: Visualization of the data in Table 3.3. See how the number of attack traces changes according to the number of neural network parameters.

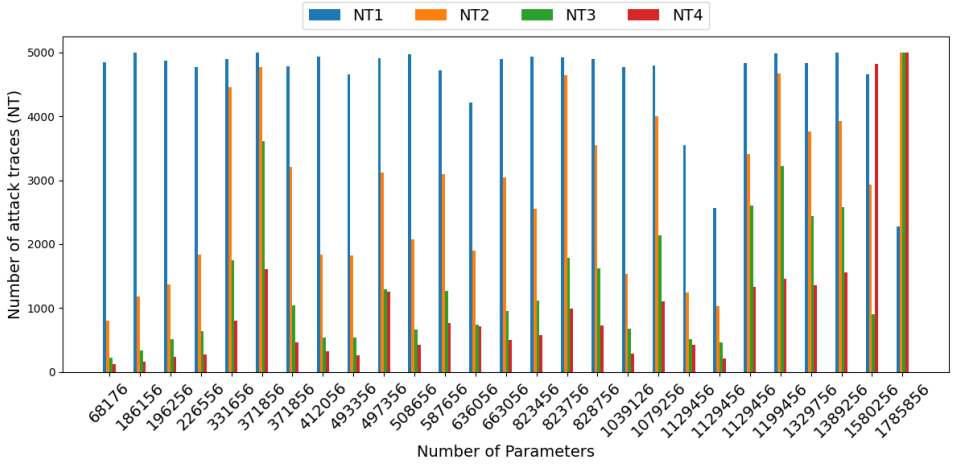


Figure 3.6: Visualization of the data in Table 3.4. See how the number of attack traces changes according to the number of neural network parameters.

surprising when the model again converged to $GE = 1$ with on average 520 attack traces when it is trained with 150 000 profiling traces. This is an example of a scenario when the hyperparameter combination contains an assumption error and increasing the profiling size invalidates the model, i.e., changes the weights of the neural network model in a way that it cannot capture the underlying leakage model distribution and find the correct key. However, there are cases like #1 (works in the under-parameterized regime) and #26 (works in the over-parameterized regime),

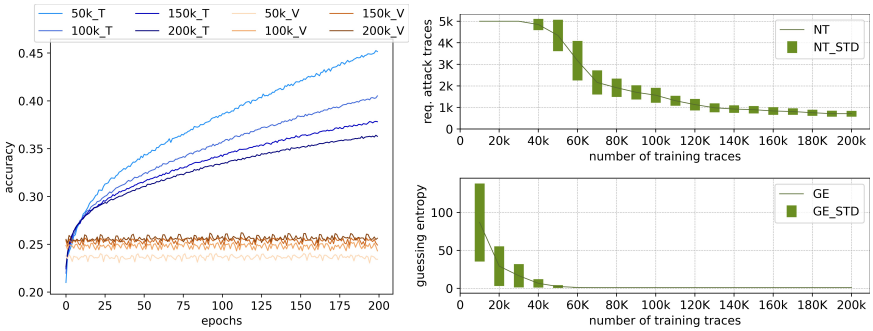


Figure 3.7: Increasing the profiling set size and accuracy in the training (T) and validation (V) set and the number of attack traces to reach $GE = 1$, #8 in Table 3.3.

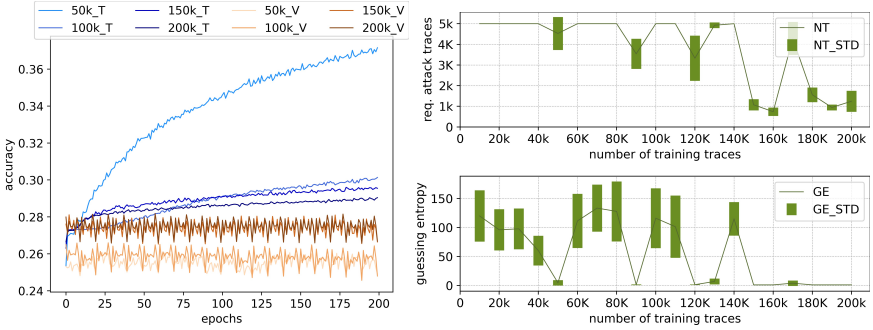


Figure 3.8: Increasing the profiling set size and accuracy in the training (T) and validation (V) set and the number of attack traces to reach $GE = 1$, #2 in Table 3.3.

where increasing the profiling set size does not change the performance according to the number of attack traces or sometimes makes it a bit worse compared to a smaller profiling set. In these cases, the assumption has an error, but it has been suppressed by increasing the profiling set size. MLP #23, #25, and #27 in Table 3.3 work in over-parameterized regime. Their profiling accuracy is more than 95% for all the profiling set sizes. Besides, the sizes of these models are at least six times larger than the largest profiling set size for ASCAD-R. In the case of MLP #24 and MLP#26, while the models have large sizes, the baseline model cannot reach more than 90% accuracy and, after increasing profiling size to 200 000 traces, the accuracy decreases under 70%. Thus, these models work in the critical regime for all the profiling sizes because the hyperparameter combination does not allow the models to fit the leakage perfectly.

Similar behavior is observed for multilayer perceptrons trained for the ID leakage model. In Table 3.4, one can see the number of attack traces for MLP models trained with 50 000, 100 000, 150 000, and 200 000 traces. There are many cases that follow the more data better generalization principle, but we can see cases that do not show a decrease in the number of attack traces when we increase the profiling

set size. Considering the size of the input layer, which is 1400 (number of features for the ASCAD-R dataset), and the size of the output layer, which is 256 (number of classes for the ID leakage model), the smallest model that we managed to find that converges to $GE = 1$ has 68 176 trainable parameters and the baseline model has $\approx$ 10% accuracy. Then the profiling accuracy decreases to $\approx$ 3%, which is still larger than the validation accuracy ($\approx$ 1% but still larger than random guessing in the case of the Identity leakage model). MLP #23, #24, and #26 work in an over-parameterized regime for the same reasons as models in Table 3.3.

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	2959	629	320	286	248
2	5689	2670	2718	801	5000
3	6579	2885	1616	883	934
4	9781	1319	654	481	521
5	15 327	4821	5000	5000	5000
6	15 439	1862	671	460	502
7	19 209	3958	1303	897	766
8	24 433	813	474	4141	477
9	31 113	3196	1300	5000	5000
10	44 905	3149	1855	1283	982
11	57 943	3660	1450	1411	803
12	63 865	2712	1270	1253	756
13	75 813	1587	1663	1497	745
14	94 661	2625	2586	2068	2604
15	100 113	3673	2930	2337	1501
16	124 585	4905	3393	1458	1596
17	189 233	4730	5000	2783	2588
18	243 129	3983	4814	2929	2487
19	391 521	2233	1721	1470	1295
20	434 121	3605	2296	2677	1964
21	541 243	4833	3467	1162	441
22	570 753	2441	1830	1165	894
23	984 649	3525	2068	1672	1661
24	1 211 881	4175	3419	1554	1600
25	1 409 249	3540	1867	1165	905

Table 3.5: *CNNs* trained for the *HW* leakage model.

We investigated CNNs and the Hamming weight and Identity leakage models to check if this behavior is still observable. The trained models and the number of attack traces in this experiment can be analyzed in three regimes. For example, CNNs #1 to #7 in Table 3.5 work in under-parameterized regime. This can be concluded by considering the profiling accuracy of these models, which is less than 35%, and the proportion between the model sizes and the profiling set sizes. These models' accuracy and loss evolution for the profiling and validation sets do not

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	6124	1676	843	938	329
2	6350	1007	5000	5000	157
3	8390	4310	5000	5000	106
4	10 646	2473	598	5000	5000
5	13 274	379	156	60	52
6	27 140	1211	111	47	29
7	32 978	5000	3320	319	57
8	54 254	5000	735	126	55
9	66 584	5000	1469	558	293
10	71 272	4003	199	5000	5000
11	86 304	5000	2139	351	157
12	91 080	5000	4964	5000	933
13	114 564	4998	2636	706	292
14	134 146	3058	622	1546	439
15	154 696	4045	1194	115	58
16	192 120	5000	581	270	5000
17	202 792	3239	1095	422	308
18	290 536	2943	5000	5000	3073
19	356 572	4767	943	577	195
20	466 312	5000	168	5000	5000
21	598 838	1946	693	466	249
22	662 432	5000	1017	516	287
23	707 656	5000	3988	1914	4482
24	718 128	5000	837	196	82
25	828 774	5000	3379	2764	967

Table 3.6: *CNNs* trained for the *ID* leakage model.

show significant overfitting, meaning that these models do not have the capacity to memorize the noise in the profiling set. This can be considered as an indication that the number of trainable parameters does not represent the capacity of the models in a way that we can compare models from two different families (cf. results for the MLP and the ID leakage model). To confirm this, we draw attention to the fact that the smallest successful models, in the case of CNNs, have less than 10 000 trainable parameters.

Neural network models like #3 in Table 3.3, and #1 and #7 in Table 3.5 are working in under-parameterized regime and follow the "more data better generalization" principle, but MLP models like #1, #2, and #4 in Table 3.3 and CNN models like #2 and #5 in Table 3.5 show a decrease in their generalization capability after increasing the profiling set size. Their hyperparameter combination imposes an assumption error on the final model, and the neural network model cannot capture the actual leakage model's distribution. To resolve this, we need to change the hyperparameters so that the final model's parameters can capture the actual leakage

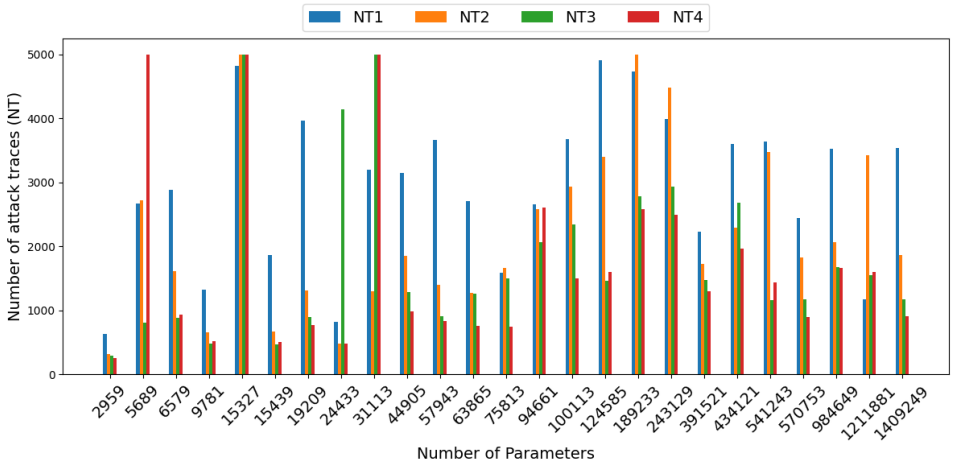


Figure 3.9: Visualization of the data in Table 3.5. See how the number of attack traces changes according to the number of neural network parameters.

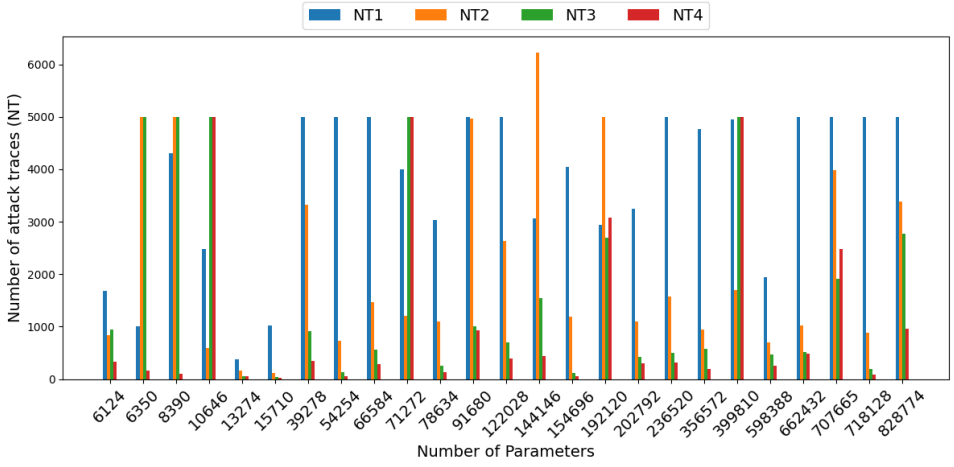


Figure 3.10: Visualization of the data in Table 3.6. See how the number of attack traces changes according to the number of neural network parameters.

model's distribution. The results in Section 3.5.4 show that in a considerable number of cases, a small change in hyperparameters, like changing the activation function or the number of epochs, can result in a neural network model that can capture the underlying leakage model distribution better.

The observations for the over-parameterized regime for MLP also hold for CNN (#23 and #24 in Table 3.5). In the case of CNN models in Table 3.6, many baseline models reach 100% accuracy, but then this accuracy decreases significantly by

increasing the profiling set sizes. Consequently, none of these models work in the over-parameterized regime for all profiling sets. The models are being pushed into the critical regime by increasing the profiling set size. For better visual clarity, the results of Table 3.5 and Table 3.6 are illustrated in Figure 3.9 and Figure 3.10, respectively.

3.5.2. AES-HD Dataset

In the case of the AES-HD dataset, we trained 500 random MLP and 500 random CNN models with the Hamming distance (HD) leakage model and again selected some of the models that converged to $GE = 1$. The AES-HD dataset has twice the number of traces in the profiling set compared to the ASCAD-R dataset, and this allows us to investigate the effect of adding profiling traces on model's generalization capability for a longer interval. As observed in Table 3.7, we trained MLP models with profiling sets including 50 000, 100 000, 150 000, 200 000, 250 000, 300 000, 350 000, and 400 000 traces. The required number of attack traces to reach $GE = 1$ for each neural network model shows that the model's generalization did not increase for many cases when we increased the number of profiling traces. The experiment has been repeated for the CNN models, and the same behavior was observed. Therefore, those results are not represented.

In most cases in Table 3.7, the models converge to $GE = 1$ for a similar number of attack traces. This means that the increase in the number of profiling traces can change the parameters of the models, and since the output probabilities of the models will change as a result of this, we can see the changes in the number of attack traces for different profiling set sizes. However, these changes are not improvements in the attack performance as they do not indicate a successful attack performance. On the other hand, the choice of attack traces can have the same effect (i.e., changes in the number of attack traces). However, there are models like #5 that, for some increases (from 150 000 to 200 000 and from 250 000 to 400 000), cannot even converge. While the training curve of these models shows normal behavior, they cannot rank the correct key better than the random guessing (thus, accuracy cannot serve as an indication of the SCA performance).

In Table 3.7, MLP #1, #2, #3, and #4, are in the under-parameterized regime. Looking at the number of attack traces for different profiling set sizes, we cannot see an absolute decrease in this metric. Since the number of trainable parameters in these models in comparison to even 50 000 profiling traces is small, they have learned the leakage model's distribution with a smaller number of profiling traces and gone to the saturation part of the learning curve where adding more traces to the profiling set does not improve the performance. Models like #19, #20, and #21 in Table 3.7 are working in an over-parametrized regime because they reach $\approx 100\%$ accuracy for all profiling set sizes. In many cases, these neural network models need more than 5000 attack traces to place the correct key to the first rank. However, by observing the GE for evolution set, we see that these models could reduce the key entropy, and by adding more attack traces, they can rank the correct key in the first place. This observation shows that models working in an over-parameterized regime cannot perform as well as models working in the under-parameterized regime for

the AES-HD dataset. Consequently, there is overfitting that damages the model's generalization from a side-channel perspective. For the models working in a critical regime, the deep double descent effect and assumption error combine. Thus, we can observe (middle of Table 3.7) that for many cases, the models cannot rank the correct key in the first place with less than 5000 attack traces.

#	*Params*	NT_1	NT_2	NT_3	NT_4	NT_5	NT_6	NT_7	NT_8	NT_9
1	12829	3294	2610	2549	2239	2041	2317	2220	2358	2011
2	26049	2795	2144	2128	2424	1904	2200	2526	2024	1938
3	26049	3268	2753	2060	2204	2470	2679	2816	2732	2608
4	27309	4700	3353	5000	3454	4384	3543	4014	3885	3965
5	60249	3681	3551	2366	5000	3224	5000	5000	2253	5000
6	136109	3445	3798	3321	3778	3391	3757	4478	3122	3118
7	196709	4918	3980	5000	4030	4293	4223	3495	2800	4222
8	292209	4527	5000	5000	5000	5000	5000	5000	5000	5000
9	378009	4867	5000	5000	4285	5000	4213	5000	4367	4288
10	378009	3706	5000	5000	5000	5000	5000	5000	5000	5000
11	412809	3735	3127	3459	2910	2854	2606	3066	2829	3297
12	468309	3880	4542	5000	5000	5000	3418	4001	3664	4732
13	824809	3456	4864	5000	5000	5000	5000	5000	5000	5000
14	824809	3533	3617	3978	5000	4235	4394	3674	4933	5000
15	880509	4339	5000	5000	5000	5000	5000	5000	5000	5000
16	985209	3771	4076	4502	4156	4601	3204	2614	3053	3219
17	1131009	2838	4824	5000	5000	5000	5000	5000	5000	5000
18	1381509	3401	5000	5000	5000	5000	5000	5000	4380	5000
19	1466409	2746	5000	5000	5000	5000	5000	4604	5000	5000
20	2383509	4134	3779	4496	5000	5000	5000	4137	5000	5000

Table 3.7: *MLP* trained for the *HD* leakage model.

For better visual clarity, the results of Table 3.7 is illustrated in Figure 3.11.

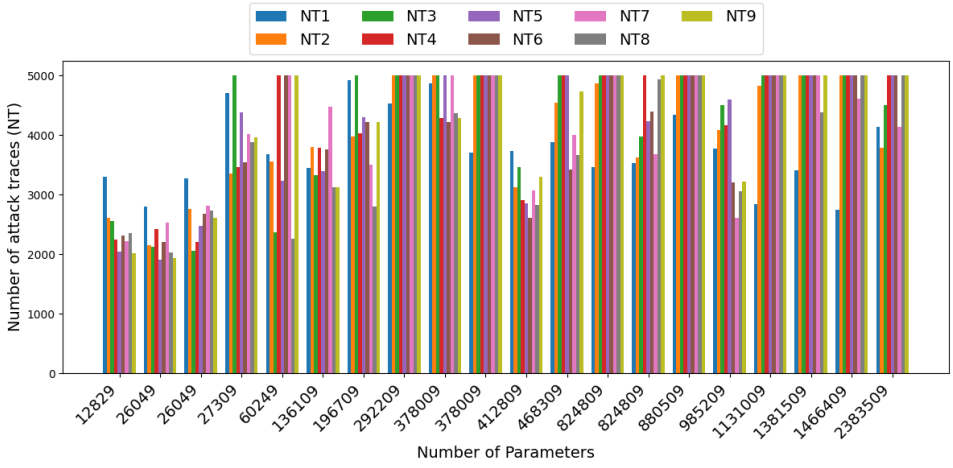


Figure 3.11: Visualization of the data in Table 3.7. See how the number of attack traces changes according to the number of neural network parameters.

3.5.3. Influence of Regularization Techniques

To check the influence of the increasing number of profiling traces in the presence of regularization on the number of attack traces, we added dropout regularization to neural network models. More precisely, we added a dropout layer with a rate of 0.5 after every dense layer in MLP and CNN models that were selected in Sections 3.5.1 and 3.5.2. The number of attack traces in this experiment shows that using this technique can improve the performance of the neural models on average, especially for models with medium to large capacity, but still cannot negate the assumption error. In Table 3.8, one can see the influence of the increasing number of profiling traces on the MLP performance and the HD leakage model for the AES-HD dataset. The same behavior was captured for both CNN and MLP models for the ASCAD-R and AES-HD datasets.

Using dropout regularization influences the working regime of the models significantly. For both MLP and CNN models, almost all the re-trained models worked in the under-parameterized regime, even for the large models, like #9 in Table 3.8. This model is the counterpart of model #17 in Table 3.7, and its performance increased considerably. However, dropout regularization does not always improve the model's performance. For example, model #10 in Table 3.8 cannot converge at all with dropout. Its counterpart is #20 in Table 3.7 that was able to rank the actual key under 10 for many profiling sizes. The influence of using regularization techniques on the performance of neural networks in DL-SCA was not studied deeply. In Chapter 4, we will see a more in depth analysis of regularization techniques influence in DL-SCA.

#	*Params*	NT_1	NT_2	NT_3	NT_4	NT_5	NT_6	NT_7	NT_8	NT_9
1	12829	5000	5000	4604	3496	3880	3578	3653	4133	3659
2	40599	4633	3283	3491	2758	3046	2854	2573	3430	2978
3	73209	3161	2574	2873	2600	2426	2548	2812	2411	2615
4	126009	3108	2677	2494	2254	2307	2765	2550	2608	2207
5	252009	5000	5000	5000	5000	5000	5000	5000	5000	5000
6	468309	2212	1740	1970	1942	1978	1784	2269	2291	2065
7	504009	3414	3258	2278	2733	2686	2496	2627	2753	2711
8	648909	1533	1462	1393	1402	1285	1455	1487	1405	1369
9	985209	957	920	902	1249	955	1084	1170	1232	1462
10	1466409	5000	5000	5000	5000	5000	5000	5000	5000	5000

Table 3.8: *MLP* trained for the *HD* leakage model and the dropout regularization.

3.5.4. Reducing Assumption Error by Changing Hyperparameters

Looking into Table 3.9, one can see the effect of changing a simple hyperparameter on the performance of a neural network model for a specific profiling size. For the sake of brevity, we provide only a few examples (one example from each combination of neural network topology and leakage models for the ASCAD-R dataset). Model #1 in Table 3.9, is the counterpart of model #4 in Table 3.3. We trained this model for different numbers of epochs (shown in the "EPC" column). While the model that has been trained with 100 000 traces for 200 epochs cannot converge at all, if we

train it for 100 epochs, it will be able to find the correct key with 647 attack traces. If we train it for 400 epochs, it will recover the key with 1196 traces. Model #2 is the counterpart of model #13 in Table 3.4. We changed this model's learning rate (shown in the "LR" column). As one can see in Table 3.9, this change improved its performance. Finally, models #3 and #4 are the counterpart of models #9 in Table 3.5 and model#2 in Table 3.6. Changing the activation function (shown in the "Act. Func." column) in the case of these two models increased the performance considerably. In many other cases, changing hyperparameters led to the improvement of the models that could not converge to $GE = 1$ for a specific profiling set size.

Table 3.9: Results with small changes in hyperparameters.

#	*Params*	NT_1	NT_2	NT_3	NT_4	EPC	LR	Act. func.
1	31149	3615	5000	622	5000	200	0.005	Adam
1	31149	3688	1196	5000	2434	400	0.005	Adam
1	31149	3856	647	5000	5000	100	0.005	Adam
2	663056	4821	3406	1343	713	200	0.0005	Adam
2	663056	4427	2053	886	496	200	0.0001	Adam
3	31113	3196	1300	5000	5000	200	0.005	Adam
3	31113	2415	5000	5000	621	200	0.0001	Adam
3	31113	3237	887	603	838	200	0.005	RMSprop
4	6350	1007	5000	5000	157	200	0.001	RMSprop
4	6350	675	133	2096	116	400	0.001	RMSprop
4	6350	1114	111	47	31	200	0.001	Adam

3.6. General Observations

- Increasing the profiling set size is not a guaranteed solution to increase the generalization capability of a neural network model in DL-SCA. The effects of the working regime on the performance of a model in the side-channel domain and the assumption error imposed by the hyperparameters combination cause the overall irregular behavior of the models regarding the increasing profiling set size.

- The under-parameterized models perform better than the critical and over-parameterized models in the SCA domain. While the theory indicates that over-parameterized models can reach better performance compared to under-parameterized models in a number of settings [90], this was not the case for SCA so far, especially for a noisy dataset like AES-HD.

- Compared to the neural network models working in the under-parameterized regime, the models working in the over-parameterized regime have a large capacity, and their number of trainable parameters is much larger than the number of traces in the profiling set. On average, such models need more traces to rank the correct key in the first place compared to the neural network models working in the under-parameterized regime. Still, it is possible to converge

to $GE = 1$ with a small number of attack traces using large models and regularization.

- In many deep learning classification applications, we are interested in the final decision of the model, i.e., the class that the model assigns to measurement and not the probability that the model calculates to assign that measurement to a specific class. Contrary, in SCA, we are interested in the probabilities that a model assigns to each class for each measurement. Thus, small deviations in the estimated distribution function based on the profiling set will lead to significant changes in the model's performance from an SCA perspective. Since over-parameterized models have a large capacity and many parameters shape the underlying leakage model distribution, they have more potential to contain assumption errors.
- A model working in a critical regime is very fragile, and in some cases, even a small change can decrease its performance noticeably. Thus, a cautionary solution could be to avoid using such models.
- Overfitting is a complex phenomenon, and we cannot trace its causes to only one source. Still, the results in this chapter indicates that hyperparameter combinations play a more significant role than the profiling set size.

3.7. Conclusions and Future Work

This chapter investigates overfitting in DL-SCA. It challenges the common assumption that more profiling traces is better, and shows that, while this may be the case, it cannot be taken for granted. Indeed, the experiments showed a number of settings where adding more profiling traces makes the attack less powerful or even unsuccessful. A simple yet powerful option to fight against overfitting is to use regularization or tweak the neural network architecture. Unfortunately, using such techniques does not also provide a guarantee to avoid overfitting. Thus, it is necessary to carefully design the model and assess its performance with different settings to understand in what regime the model works and then use the most appropriate one (which seems to be the under-parameterized regime). Finally, this chapter provides the setup to be more precise when discussing the failure of deep learning models in SCA. Instead of "simply" saying there is overfitting, one should strive to give insights what are the causes of that behavior. In future work, it would be interesting to provide a systematic approach on how to tweak hyperparameter changes to improve the attack performance. Besides, we again saw the beneficial effects of regularization (see, e.g., [35]), which can be a motivation for diving into Chapter 4.

4

Regularizers to the Rescue: Fighting Overfitting in Deep Learning-Based Side-channel Analysis

Despite considerable achievements of deep learning-based side-channel analysis, overfitting represents a significant obstacle in finding optimized neural network models. This issue is not unique to the side-channel domain. Regularization techniques are popular solutions to overfitting and have long been used in various domains.

At the same time, the works in the side-channel domain show sporadic utilization of regularization techniques. What is more, no systematic study investigates these techniques' effectiveness. In this chapter, we see the regularization effectiveness on a randomly selected model, by applying four powerful and easy-to-use regularization techniques: L_1, L_2, dropout, and early stopping. The results show that while all these techniques can improve performance in many cases, L_1 and L_2 are the most effective. Dropout worsens 30% of models for the same hyperparameter set, which points to the necessity more adjustment when using this regularization technique. Finally, if training time matters, early stopping is the best technique.

4.1. Introduction

As mentioned in Section 1.4.1, while DL-SCA is a powerful approach to profiled SCA, overfitting is one of the hurdles that an evaluator should overcome. Overfitting happens when a deep model fits the training measurements perfectly but cannot generalize to previously unseen measurements. Hence, an overfitted model cannot evaluate the product reliably. This evaluation weakness can lead to overestimating a product's robustness against SCA.

Regularization techniques are used during the training to decrease the complexity of the model on the fly and prevent overfitting. So far, many regularization techniques have been introduced. However, while there is a strong recommendation for using regularization techniques in DL-SCA from previous research [4], no comprehensive study shows a practical consequence of using them in DL-SCA. Indeed, while most of the previous works that consider regularization techniques do advocate the usage of those, they do not rely on systematic evaluation nor meaningful comparisons.

This work aims to fill this gap and show how the most commonly used regularization techniques, i.e., L_1, L_2, dropout, and early stopping, can improve DL-SCA. The main contributions of this chapter can be summarized as follows:

- We compare the influence of four popular regularization techniques (L_1, L_2, dropout, and early stopping) on randomly selected models in DL-SCA. For the reported analysis, a hundreds of deep learning models with and without regularization techniques were tested. We consider software- and hardware-based datasets and two different cryptographic algorithms to make the comparison more complete and thorough. The considered datasets are 1) ASCAD-R, 2) AES-HD, and 3) Ascon-Unprotected. Besides, depending on the considered dataset, we utilize two widely used deep learning models (MLP and CNN) and three different leakage models (Hamming weight, Hamming distance, and Identity leakage model).
- We show that the improvements that many techniques in deep learning offer heavily depend on the model's characteristics (its architectural and learning hyperparameters). In many cases, the combination of the hyperparameters and the used technique decreases the performance (while one would expect improvement).
- We introduce the deterioration rate to show how reliable a specific regularization technique is. This metric shows the probability that a model worsens after applying a regularization technique. We also consider profiling time as another metric providing insights into the performance of various techniques.
- We consider the baseline model's performance and its relation to the improvements that the regularized model can offer. We show that when the implicit regularization of the baseline model is high, adding a regularization technique worsens it. In contrast, applying regularization techniques improves the performance when the baseline model's implicit regularization is low.

Related Work: The competitive performance of the DL-SCA methods compared to more traditional SCA methods has attracted much attention in recent years. Many

researchers utilized different notions and techniques used in deep learning like reinforcement learning [79], weight visualization [76], and information bottleneck [91], to improve the performance of DL-SCA even further. Regularizers have been used from the early days of machine learning emergence, and their practicality was noticed in various domains. DL-SCA is not an exception. For example, in [91], Perin et al. proposed a technique based on the information bottleneck to monitor the training evolution. Using this technique, they could find the best epoch to stop the training. In [92], Robissout et al. introduced a metric to evaluate the performance of a deep neural network in the side-channel domain during the profiling phase. They used this metric as a monitoring metric for early stopping. In [35], Kim et al. used several regularization techniques, including dropout, L_2, and data augmentation. In [38], Rezaeezade et al. inspected the influence of increasing the training set size on the generalization power of CNN and MLP neural networks for SCA in the presence of dropout. In [77], Perin et al. used the ensemble method to improve the model's generalization and reduce overfitting. The ensemble method is a regularization technique combining multiple classifiers to form a better hypothesis. This technique reduces the final model's dependencies on the structural hyperparameters. Perin et al. showed that ensembles of many non-optimal models could even perform better than the best-obtained model. Batch normalization is another regularization technique [93] used in many DL-SCA works, e.g., [7, 75]. We note that while a number of related works used some form of regularization, they do not compare the results with and without the regularization. Hence, it is difficult to assess how well those methods performed or whether they were even necessary.

4.2. Regularization Techniques

The primary goal of training a deep learning model is to prepare it to predict unseen data accurately. This goal is interpreted as improving the generalization power of a deep neural network. The challenge is that the model is learning from the training data, so it tries to increase the accuracy (or decrease the loss) in the training set by fitting the training examples and reducing the training error. Nevertheless, the final goal is to use the model for a test set and decrease the generalization error (test error). Thus, an appropriate learning algorithm: 1) should make the training error small, and 2) should make the gap between training and test error minimal [94]. Underfitting occurs when a deep learning model does not reach a sufficiently small error value in the training set. Overfitting happens when a model cannot reduce the gap between training and test error. While both underfitting and overfitting should be avoided, the latter is more challenging to control. Especially in deep learning, where much input training data and many input features are available, the best strategy is to fix underfitting by choosing more complex models [95].

Regularization is a well-studied and widely used solution to improve the generalization power of machine learning and deep neural models. In general, **regularization** is defined as "any modification we make to a learning algorithm tending to reduce the test error but not the training error [94]." This definition covers many techniques, from adding penalties to the objective function to multi-task learning (look at Chapter 7)

and ensemble methods (look at Chapter 6). The shared part of all these techniques is reducing overfitting and improving generalization. This chapter inspects the influence of applying four regularization techniques on DL-SCA. Those techniques are L_1 and L_2 norm penalties, dropout, and early stopping, which have all been used for a long time in machine learning and now in deep learning.

L_2 and L_1 Norm Penalties

L_2 parameter norm penalty or weight decay adds a penalty term in the form of all the model's squared weights to the objective function. The simplest form of formulating L_2 regularization is shown in Eq. (4.1):

$$\tilde{\mathrm{E}}(\mathbf{W}; \mathbf{X}, y) = \mathrm{E}(\mathbf{W}; \mathbf{X}, y) + \frac{1}{2}\lambda \sum_{ij} \mathbf{w}_{ij}^2, \tag{4.1}$$

where E is an arbitrary objective function measuring the training error. $\mathbf{X}$ are the training examples and y are their corresponding labels. $\mathbf{W}$ is the current weights matrix, and λ is a parameter governing how strongly large weights are penalized. When using L_2 regularization, λ is a hyperparameter that should be tuned. $\tilde{\mathrm{E}}$ is the modified objective function. Considering Eq. (4.1), when updating weights using gradient descent, a constant term in the form of $\lambda\mathbf{W}$ subtracts from updated weights in each step. This term controls the growth of the weights and suppresses irrelevant components of the weight vector [96].

L_1 regularization is another way to penalize the size of the models (number of parameters). The difference between L_1 and L_2 regularization stems from the penalty term added to the objective function. One can see the modified objective function after using L_1 in Eq. (4.2):

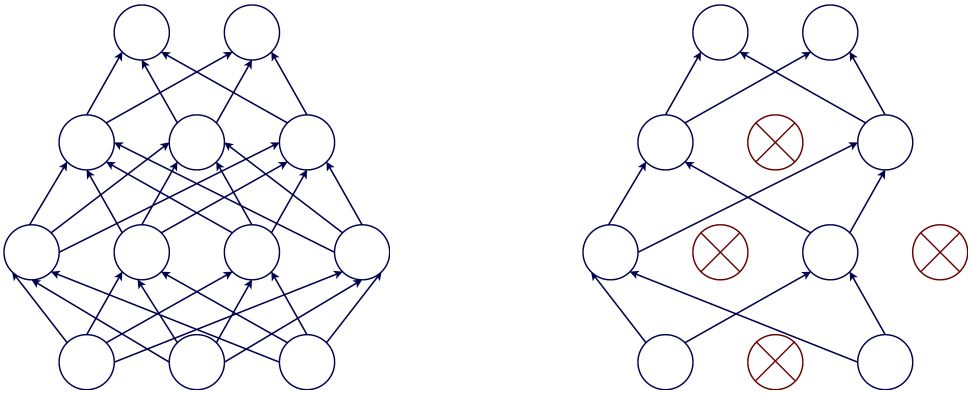
$$\tilde{E}(\mathbf{W}; \mathbf{X}, y) = E(\mathbf{W}; \mathbf{X}, y) + \frac{1}{2}\lambda \sum_{ij} |\mathbf{w}_{ij}|, \tag{4.2}$$

where $|\mathbf{w}_{ij}|$ is the sum of all the model's absolute values of weights. Adding this term to the objective function shows itself as $\lambda sign(\mathbf{W})$ term in the weight update operation using gradient descent. In practice, L_1 can be seen as a built-in feature selection mechanism because it tends to shrink some weights toward zero. Using L_1 adds λ to the hyperparameters that should be tuned.

Dropout

The idea of this technique is to temporarily remove random neural units along with their connections from the primary network during the training. Each unit may be removed with a q probability called the dropout rate. The dropout rate is a hyperparameter and should be tuned. Dropout, in essence, is training multiple smaller networks selected randomly from the bigger primary neural network by removing some neural units in each training step. Since these smaller networks share the primary neural network's weights, the averaging at test time is as simple as using the primary network without any dropout [97].

Since regular dropout can not prevent overfitting in CNN, we used spatial dropout [98] for combinations with CNN topology. In CNN models, when a filter is applied to an input vector (or matrix in the case of 2D convolution), it extracts a vector (or matrix) called **feature map**[1] and its elements are highly correlated. Dropping just one of these correlated elements may not prevent overfitting; instead, we can drop the whole feature map. Applying each filter to the convolutional layer input creates a feature map. If a convolution layer has m filters, the outcome of the layer is a set with m feature maps. Spatial dropout drops one or more entire extracted feature maps selected randomly among these m feature maps.



(a) A fully connected MLP. (b) An MLP with dropout.

Figure 4.1: An MLP structure with and without droupout.

Early Stopping

In the general definition of early stopping, when training a neural network with sufficient capacity for and enough epochs, the training error keeps decreasing, while the validation error starts to rise again after a while. This rise is a sign that overfitting starts. With early stopping, we can stop training as soon as validation error starts to rise. However, the drop and rise of the validation error are not very smooth in reality. The validation error curve shows many ups and downs, and finding the exact point where the validation error starts to increase is not possible. One solution is to stop training if the generalization does not improve after a specific number of epochs. This specific number of epochs is called “patience”.

4.3. Analysis Methodology

This chapter aims to inspect the impact of applying L_1, L_2, dropout, and early stopping on the performance of DL-SCA. To interpret the results, we compare the average performance of a large number of neural networks that use a regularization technique

[1] In a CNN, a feature map is the output of a filter applied to the previous layer.

with their average performance without any regularization technique.We considered three different datasets, including protected and unprotected implementations on hardware and software of two different primitives (AES and Ascon), with the intention of covering a more comprehensive range of side-channel evaluation use cases and showing that the conclusions hold under different circumstances. Also, we considered two widely used neural network topologies in SCA (CNN and MLP) and three common leakage models in the SCA community (ID, HW, and HD) to show how the results are affected using different regularizations. The experimented combinations are listed in Table 4.1.

Table 4.1: Eight different combinations considered in the experimental setup.

Dataset	NN topology	Leakage model	Combinations
ASCAD-R	MLP	HW	ASCAD-HW-MLP
			ASCAD-HW-CNN
			ASCAD-ID-MLP
AES-HD	CNN	HD	ASCAD-ID-CNN
			AES-HD-MLP
			AES-HD-CNN
Ascon		ID	Ascon-ID-MLP
			Ascon-ID-CNN

The following steps outline the methodology used to compare the performance with and without regularization techniques in each combination:

- **Acquiring baseline models:** In this step, we start by generating 500 neural networks with the random search. The ranges used for searching hyperparameters of MLP and CNN are listed in Table 4.2. These ranges are chosen based on the ranges reported in the previous works [7, 76, 89]. Since those 500 neural networks are generated randomly, many of them cannot decrease GE. Then, we select the 200 best as the "baseline models" and make the primary pool of models to start from.

  It is worth mentioning that in a fixed number of randomly generated neural networks, the number of MLP models that reached $GE = 1$ was significantly larger than CNN. For example, among 500 randomly generated MLPs with the HW leakage model in the ASCAD-R dataset, 172 models reached $GE = 1$, and the rest were able to decrease the GE to small numbers (less than 10). In the case of CNNs with the HW leakage model, only 70 models reached $GE = 1$, and many models could not even decrease the GE lower than a random guess. However, at the same time, the best CNN models could converge to $GE = 1$ with far fewer attack traces than MLP models. In the previous example, the five best MLP models ranked the key in first place with 550 attack traces on average, while this metric was around 50 for the five best CNN models. We refer to this observation as the "general ability of MLP models to find the key" and the "potential ability of CNN models to find the key". These names are selected

according to this practical experience that MLP models are less dependent on their hyperparameters to find the correct key. In contrast, CNN models are sensitive to slight hyperparameter changes.

Table 4.2: Searched range of MLP and CNN hyperparameters. For both MLP and CNN dense layers, we used the ranges shown in *Dense layers* part of Table 4.2.

Hyperparameters	Range
	Dense layers
Number of neurons	[10, 90], step = 10 + [100, 500], step = 100
Number of layers	[1, 8], step = 1
	Convolution layers
Number of layers	[1, 4], step = 1
Number of kernels	[4, 20], step = 1
First layer's filter size	[2, 4, 8, 12, 16]
i^{th} layer filter size	$((i-1)^{th}filter-size)^2$
Pooling	"Average", "Max"
Pooling size	[2, 10], step = 2
Pooling stride	[2, 10], step = 2
	Learning hyperparameters
Optimizer	"Adam", "RMSprop"
Weight initialization	"random-uniform", "he-uniform", "glorot-uniform",
Activation function	"relu", "selu", "elu"
Batch size	$[100, 900], step = 100$
Learning rate	$[0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001]$
Epochs	200 in ASCAD-R and AES-HD, 30 in Ascon-Unprotected

- **The average performance of baseline models:** We use two metrics to represent the average performance of baseline models. The average *GE* that 200 baseline models can reach in an attack set with 5000 attack traces is called "AVERAGE-GE". The average required number of attack traces the baseline models need to reach $GE = 1$ is referred to as "AVERAGE-NT". For those neural networks that cannot reach $GE = 1$ with 5000 attack traces, we report $NT = 5000$.
- **Re-training regularized models:** We re-train the baseline neural networks in the presence of different regularization techniques (L_1, L_2, dropout, and

early stopping). These regularization techniques have their own hyperparameters, which should be tuned for each neural network. To do this tuning, we examine a range for each regularization hyperparameter. Then we consider the best-acquired performance as the performance of that baseline model in the presence of a specific regularization technique. Again, the considered metrics are GE and NT. The tuned baseline model with the best performance is called the "regularized model."

An example can clarify the process: we want to re-train each baseline model in the presence of L_1 regularization. To do so, while all the other hyperparameters[2] of the baseline models stay the same, we add L_1 regularization to neural network layers. L_1 has a regularization constant λ that can take 12 different values listed in Table 4.3[3]. We build 12 models similar to the baseline model but having L_1 regularization with different λ values. Among the 12 re-trained models, we consider the model with the smallest GE as the best-tuned model and report its performance (the value of GE and its corresponding NT) as the regularized model's performance. If a neural network can reach $GE = 1$ with two or more λ values, we consider the smallest NT and $GE = 1$ as the performance of the regularized model.

Table 4.3: Hyperparameters of L_1, L_2, dropout, and early stopping regularization techniques and the experimented ranges.

Technique	Hyperparameter	Range
$Weightdecay(L_1)$	λ	$[5\times10^{-2}, 10^{-2}, 5\times10^{-3}, 10^{-3}, 5\times10^{-4}, 10^{-4}, 5\times10^{-5}, 10^{-5}, 5\times10^{-6}, 10^{-6}, 5\times10^{-7}, 10^{-7}]$
$Weightdecay(L_2)$	λ	$[5\times10^{-2}, 10^{-2}, 5\times10^{-3}, 10^{-3}, 5\times10^{-4}, 10^{-4}, 5\times10^{-5}, 10^{-5}, 5\times10^{-6}, 10^{-6}, 5\times10^{-7}, 10^{-7}]$
$Dropout$	$Dropout-rate$	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$
$Earlystopping$	$Patience$	$[10, 15, 20, 25, 30]$ in the ASCAD-R and the AES-HD datasets $[3, 5, 7]$ in the Ascon datasets

- **The average performance of regularized models:** The AVERAGE-GE and the AVERAGE-NT are calculated as performance metrics for the regularized models. We compare the AVERAGE-GE and AVERAGE-NT of the baseline and the regularized models for each regularization technique. This way, the influence of each regularization technique on the performance of DL-SCA can be observed. The examined ranges for hyperparameters of different regularization techniques are listed in Table 4.3.

Note: Two significant differences exist between the used intermediate value for attacking AES primitive (attacking ASCAD-R and AES-HD dataset) and Ascon

[2] Including the architectural and learning specifications.
[3] These values have been specified based on the grid search strategy.

primitive (Ascon-Unprotected dataset). Firstly, the known and variable parts used in the former are plaintext (ASCAD-R) and ciphertext (AES-HD), while it is nonce for the latter. Secondly, for recovering 128 bits of the key for the former, we use the same intermediate value and leakage model for all 16 Sbox outputs. In contrast, we can recover half of the key with the selected intermediate value for Ascon. To obtain the remaining key bits, we use y_0 or y_1 (since they have non-linear terms including x_2). The other recovered half of the key, is plugged into the next selected intermediate value, and we can recover the rest of the key. For more information about the attack points please refer to Section 2.2.

4.4. Experimental Results

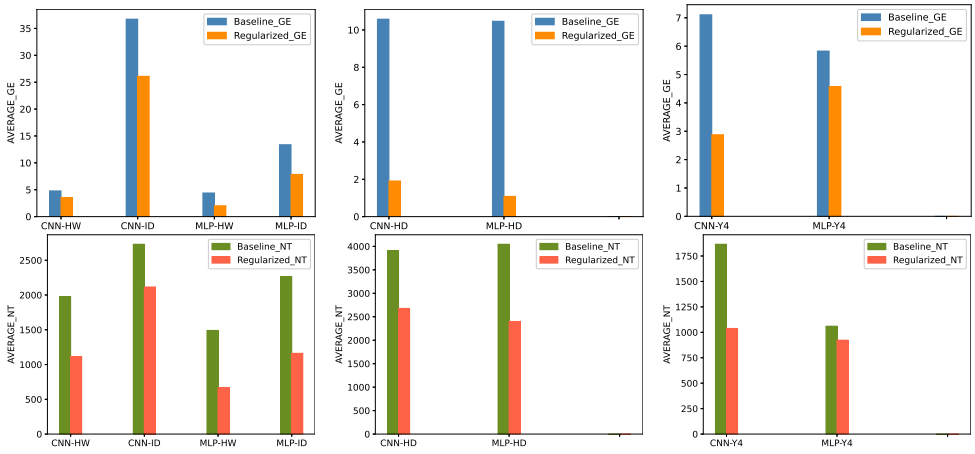
The experiments in this section reveal the influence of applying L_1, L_2, dropout, and early stopping on DL-SCA performance. We look into four metrics to explain this influence. First, we explore the modifications of the AVERAGE-NT and AVERAGE-GE for baseline and regularized models in Section 4.4.1. Then, we evaluate the models' deterioration rate in Section 4.4.2. Finally, in Section 4.4.3, we monitor the effect of using different regularization techniques on profiling time.

4.4.1. Performance Comparison

L_1 regularization penalizes the size of the model parameters in a way that causes a subset of them to become zero. The effect of this is an implicit feature selection. As a result, L_1 regularization will be most effective when the input is noisy, and the traces include samples that do not carry information. In other words, L_1 regularization helps the neural networks to extract the point of interest more efficiently. In our experiments, we use L_1 regularization in every dense and convolution layer in both MLP and CNN topologies. Figure 4.2a shows the AVERAGE-GE and the AVERAGE-NT for baseline and regularized models for the ASCAD-R dataset. Notice how regularized models always reach lower AVERAGE-GE than the baseline ones. The difference is especially pronounced for the ID leakage model and CNN. The differences become even more clear once considering the average number of attack traces, AVERAGE-NT, to break the target. In three out of four settings, regularization allows for reducing the number of attack traces in half. The results for the ASCAD-R dataset show that L_1 regularization is effective for all the combinations, but it tends to perform better with MLP models. That is because the CNN models naturally provide more effective feature selection than MLP models, and the influence of feature selection is more tangible in the case of MLP.

L_1 Regularization

Next, Figure 4.2b demonstrates the AVERAGE-GE and the AVERAGE-NT for baseline and regularized models for the AES-HD dataset. For this dataset, the baseline models perform well on average but require many traces to break the target. Adding L_1 reduces the AVERAGE-GE up to five times on average and the AVERAGE-NT for $\approx 40\%$. We do not see an even larger influence on the number of



(a) ASCAD-R (b) AES-HD (c) Ascon-Unprotected

Figure 4.2: The average performance with and without L_1 regularization, for ASCAD-R, AES-HD and Ascon-Unprotected datasets. The AVERAGE-GE (left) is calculated for baseline (blue) and regularized (orange) models. The AVERAGE-NT is calculated for baseline (green) and regularized (red) models.

attack traces probably due to a limited attack set size. Indeed, when a model fails to find the correct key, we do not have any estimation of the number of traces it would need to find the key, so we report the maximum number of traces in used for attack to show that the model did not succeed in finding the key.

Finally, Figure 4.2c shows the AVERAGE-GE and the AVERAGE-NT for baseline and regularized models for Ascon-Unprotected dataset. Since this dataset contains measurements from unprotected software implementation of a primitive, and the traces are collected using ChipWhisperer, it can be considered an easy dataset to attack. As a consequence, starting from the first step of our methodology to acquire baseline models, more than 200 of the random models were converging to $GE = 1$ in each scenario, which means AVERAGE-GE would be equal to one, and no more improvement would be possible in this regard. So we slightly modified the methodology. The modification is replacing 20% of the primary baseline models with models that do not converge ($GE > 10$) (these models are randomly selected too). This will give the regularization techniques a chance to improve some models that are not useful in the first place. Here, we see the effect of adding regularization after this modification to the baseline models' pool here. To see what improvement regularization techniques can offer when $GE = 1$, we discuss the results before modification in Section 4.5. As shown in Figure 4.2c, the AVERAGE-GE and AVERAGE-NT are already low for baseline models. However, adding L_1 regularization decreased AVERAGE-GE to less than half in the CNN with ID leakage model. While the effect of adding L_1 is smaller for the MLP with ID leakage model, it still improves models that do not

converge without any regularization. The effect of adding L_1 on the AVERAGE-NT of CNN models is considerable as well. In Figure 4.2c, one can see that using L_1 can decrease the required number of attack traces up to two times for CNN models.

Comparing CNN-HW and MLP-HW combinations in Figure 4.2a, CNN-ID and MLP-ID combinations again in Figure 4.2a, and the CNN-ID and MLP-ID combinations in Figure 4.2c we can see the "general ability of MLP models to find the key" mentioned in Section 4.3 more clearly. While the AVERAGE-GE and AVERAGE-NT are similar for CNN-HD and MLP-HD combinations in Figure 4.2b, this ability is recognizable after applying L_1 regularization.

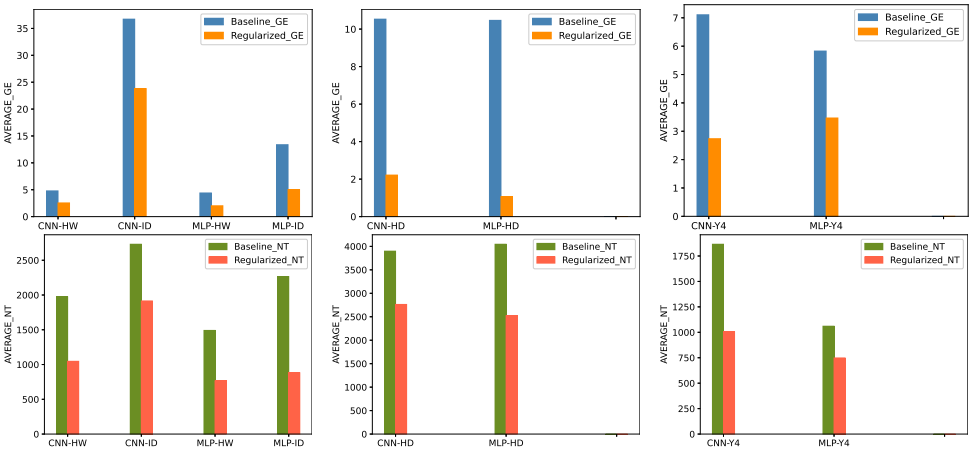
L_2 Regularization

L_2 regularization shrinks the weights to values close to zero but rarely counts irrelevant features out. In our experiments, we use L_2 regularization in every dense and convolution layer in both MLP and CNN topologies. Figure 4.3a shows the AVERAGE-GE and the AVERAGE-NT for the baseline and regularized models for the ASCAD-R dataset. As one can see, the regularized models can always reach lower AVERAGE-GE and AVERAGE-NT. The AVERAGE-GE sharp decrease in all four settings is noticeable. This decrease shows that the L_2 regularization improves models that cannot converge to $GE = 1$. The difference between baseline and regularized AVERAGE-GE is the most pronounced for the CNN with the ID leakage model. The effectiveness of L_2 regularization is also apparent when considering the average number of attack traces. Applying this regularization reduces AVERAGE-NT by half or less in three out of four settings. The improvement for the MLP with the ID leakage model is the most significant one. The observed improvements are the consequence of reducing overfitting by making the weights smaller and close to zero.

Figure 4.3b demonstrates the AVERAGE-GE and the AVERAGE-NT for the baseline and regularized models for the AES-HD dataset. As the AVERAGE-GE and the AVERAGE-NT reflect, the average performance of baseline models is almost the same for both combinations. However, the L_2 regularization improves the results more for MLP models. Adding L_2 reduces the average GE five times in the CNN-ID and seven times in the MLP-HD settings. The influence on the required number of attack traces is not as significant, but it is still more in the MLP-HD combination. Besides the reasons mentioned in Section 4.4.1, this limited effect on AVERAGE-NT results from the noise level and type in this dataset.

Lastly, Figure 4.3c represents the AVERAGE-GE and the AVERAGE-NT for the baseline and regularized models for the Ascon-Unprotected dataset. The AVERAGE-GE plot shows almost three times improvement in CNN networks and almost two times improvement in MLP networks after using L_2 regularization. The impact on the required number of attack traces does not strictly follow those numbers. However, the effect is considerable, especially for the CNN networks where regularized models need twice fewer traces than baseline models.

L_1 and L_2 regularization considerably enhance the performance in all datasets. However, the L_2 regularization is slightly more effective for the ASCAD-R and Ascon-Unprotected datasets, while L_1 is more effective for AES-HD. This observation



(a) ASCAD-R (b) AES-HD (c) Ascon-Unprotected

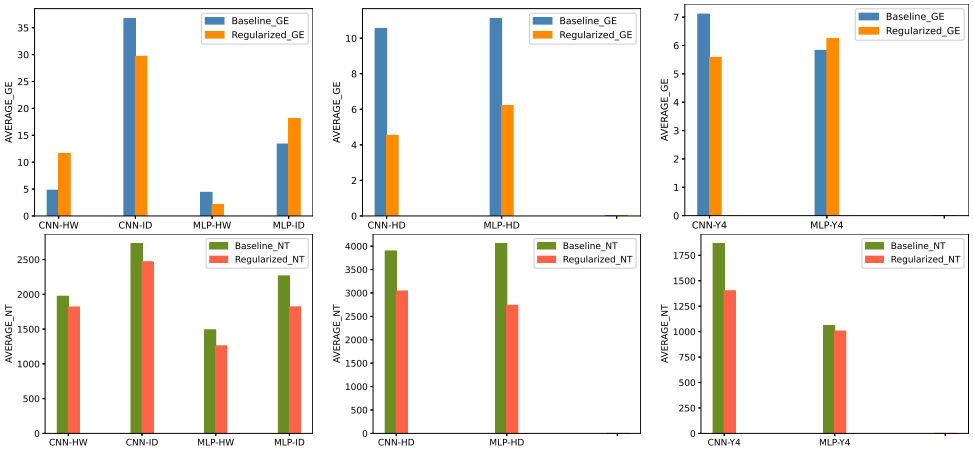
Figure 4.3: The average performance with and without L_2 regularization, for ASCAD-R, AES-HD and Ascon-Unprotected datasets. The AVERAGE-GE (left) is calculated for baseline (blue) and regularized (orange) models. The AVERAGE-NT is calculated for baseline (green) and regularized (red) models.

stems from the distinct effect of these regularization techniques and the nature of noise in the considered datasets. L_1 bypasses the influence of irrelevant features by implicit feature selection while L_2 considers almost all the input features. The input in the ASCAD-R dataset is a narrowed window of the entire measurement, including the time samples corresponding to the first round Sbox calculation. The input in the Ascon-Unprotected dataset is with time samples restricted to the first round of permutation in the initialization phase. Quite the contrary, in the AES-HD dataset, the input contains all the samples collected during the AES decryption operation. Therefore, input includes many irrelevant samples collected during pre-processing, ten rounds of AES, and the final processing. As a result, the AES-HD dataset needs stronger feature selection to confine the effect of these irrelevant time samples. The results indicate that both regularization techniques have different but equally valuable properties.

Dropout

As mentioned in Section 4.2, the dropout technique used in different neural network layers depends on the layer type. In our experiment, we used typical dropout after every dense layer in MLP and CNN models and spatial dropout after every convolution layer in CNN. Figure 4.4a shows the AVERAGE-GE and the AVERAGE-NT for the baseline and regularized models for the ASCAD-R dataset. Looking at Figure 4.4a, one can see that the average *GE* for CNN-HW and MLP-ID combinations increased after applying dropout. This observation indicates that adding dropout

may cause inferior performance in many cases. Dropout, in essence, is changing the architecture (a subset of the neural network hyperparameters) of the model. In contrast, other regularization techniques used here do not change the structural hyperparameter but only affect the learning process. Adding dropout is equivalent to changing the set of hyperparameters that work well together in every epoch, which can justify the deterioration of the model after using dropout. However, the deterioration effect of dropout is still being studied in the deep learning domain. A closer look at the *GE* and *NT* measurements for each baseline and dropout regularized model shows that many models deteriorate after applying dropout (more description is in Section 4.4.2). However, we still can see a decrease in the AVERAGE-NT for these two combinations, which indicates the potential of dropout when it is effective. The effectiveness of this technique when it does not deteriorate a model is so significant that it can compensate for the increase in the required number of attack traces imposed by deteriorated models. In the other two combinations (CNN-ID and MLP-HW), the AVERAGE-GE and AVERAGE-NT decrease slightly, showing model deterioration happens here as well. Still, it is less compared to CNN-HW and MLP-ID combinations.



(a) ASCAD-R (b) AES-HD (c) Ascon-Unprotected

Figure 4.4: The average performance with and without dropout regularization for ASCAD-R, AES-HD, and Ascon-Unprotected datasets. The AVERAGE-GE (left) is calculated for baseline (blue) and regularized (orange) models. The AVERAGE-NT is calculated for baseline (green) and regularized (red) models.

Figure 4.4b shows the results for the AES-HD dataset. With the decrease in the AVERAGE-GE and the AVERAGE-NT of regularized models, the deterioration effect is not detectable here. Still, one can see that the AVERAGE-GE and AVERAGE-NT improvement after applying dropout is less compared to L_1 and L_2.

Figure 4.4c shows the AVERAGE-GE and AVERAGE-NT for baseline and reg-

ularized models using dropout for Ascon-Unprotected dataset. Considering the AVERAGE-GE, one can see that after adding dropout, the performance improved for the CNN-ID combination, while it declined slightly for the MLP-ID combination (because of the deterioration of some models). For the AVERAGE-NT, still, the average for the MLP-ID combination improved a bit because the number of models that deteriorated was scant, and they could not degrade the average influence of adding dropout.

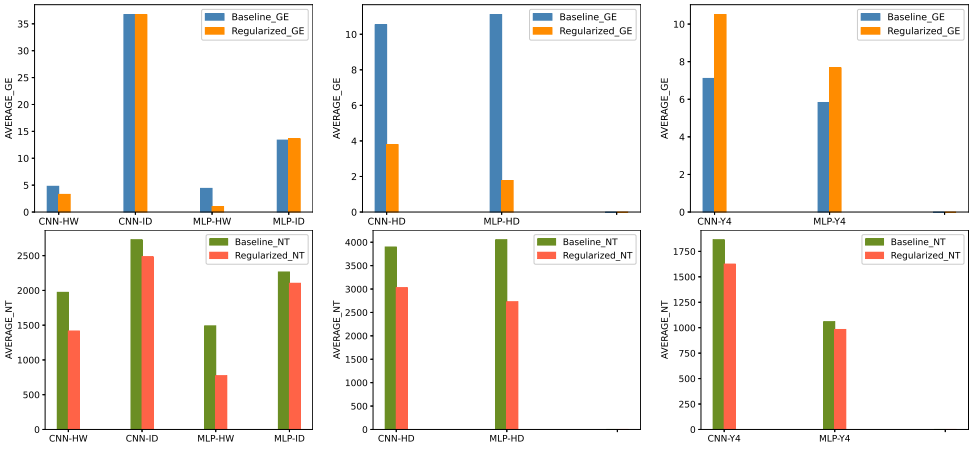
Dropout works better when using MLP with fewer output classes (HW or HD leakage models), which is the indirect effect of the deep learning model size. In the case of the ID leakage model, there are 256 output classes, while the number of output classes for HW and HD leakage models is 9. However, since the number of input samples and training examples is the same for all leakage models, significantly larger models cannot be used for the ID leakage model. As a result, dropout regularization cannot produce enough distinct smaller networks to reflect the ensemble effect for 256 output classes.

Early Stopping

Early stopping controls overfitting by manipulating the number of training iterations (epochs). This technique is adequate when the initial number of epochs is significantly larger than what the model needs to learn the underlying leakage distribution. Figure 4.5a indicates the AVERAGE-GE and AVERAGE-NT for baseline and regularized models for the ASCAD-R dataset. Based on the results, the AVERAGE-GE improvement is minimal in the ASCAD-R dataset, which means early stopping cannot improve models that do not converge without early stopping. The effect is more evident in CNN-ID and MLP-ID settings. If a baseline model does not reach $GE = 1$ without early stopping, using this technique does not significantly help the model to reach $GE = 1$. On the other hand, the epoch-wise evolution of GE shows that when GE reaches 1, it rarely increases again. Early stopping seems helpful regarding the required number of attack traces. The regularized models managed to achieve smaller AVERAGE-NT in all four settings. Stopping a model as soon as it reaches $GE = 1$ reduces overfitting and helps the model find the key with fewer attack traces.

Next, Figure 4.5b depicts the AVERAGE-GE and AVERAGE-NT for baseline and regularized models for the AES-HD dataset. The early stopping effectiveness on this dataset is similar to L_1 and L_2 regularization. This outcome shows that even early stopping regularization can improve GE and the required number of attack traces in a noisy dataset like AES-HD. This technique is considerably helpful in settings with MLP and limited output classes (HW and HD).

Finally, Figure 4.5c demonstrates the AVERAGE-GE and AVERAGE-NT before and after using early stopping for Ascon-Unprotected. As mentioned in Section 4.4.1, Ascon-Unprotected is considered an easy dataset. Therefore, neural networks generally need fewer epochs to learn the pattern. As expressed in Table 4.2, we used 30 epochs in this dataset. The results show that this value is sufficient because a large number of models could converge with this number of epochs. This smaller value for the number of epochs imposes a smaller value for the patience hyperparameter for early stopping. As shown in Table 4.3, we used three different values for



(a) ASCAD-R (b) AES-HD (c) Ascon-Unprotected

Figure 4.5: The average performance with and without early stopping regularization for ASCAD-R, AES-HD, and Ascon-Unprotected datasets. The AVERAGE-GE (left) is calculated for baseline (blue) and regularized (orange) models. The AVERAGE-NT is calculated for baseline (green) and regularized (red) models.

Table 4.4: The deterioration rate.

	ASCAD-R				**AES-HD**		**Ascon**	
	CNN-HW	CNN-ID	MLP-HW	MLP-ID	CNN-HD	MLP-HD	CNN-Y4	MLP-Y4
L_1	1.5%	4.5%	0%	6.5%	3%	0.5%	0.5%	9%
L_2	0%	2%	0%	1.5%	2%	0.5%	2%	1.5%
Dropout	29.5%	19%	7.5%	23.5%	22.5%	10.5%	10%	5.5%
Early stopping	8.5%	18%	0.5%	19%	14%	4%	9.5%	2%

the patience hyperparameter. However, comparing the AVERAGE-GE for baseline and regularized models shows that this regularization technique is not effective in the Ascon-Unprotected. Early stopping is suggested for long training or problems with smooth learning, i.e., problems with less fluctuation in validation accuracy or validation loss. In the Ascon-Unprotected, none of these conditions are fulfilled. Therefore, using early stopping does not improve model performance in many cases. A deeper look into the models shows that only a handful of non-converging baseline models converged after adding early stopping. Besides, early stopping with the used patience hyperparameters caused underfitting for some other models. These two facts justify increasing the AVERAGE-GE for regularized models. Surprisingly, the improvement provided by early stopping for the rest of the models is significant enough to promote the AVERAGE-NT and to compensate for underfitting caused by early stopping.

4.4.2. Deterioration Rate

Although all results (Figure 4.2a to Figure 4.5a) give insights into the influence of regularization techniques on DL-SCA attack performance, we can extract even more information from the experiments. One example is the percentage of models that deteriorate after using a regularization technique. We call this metric the "deterioration rate," which is the percentage of the regularized models that perform worse than their baseline counterparts. This metric shows how confident we can be that adding specific regularization techniques helps to improve the final performance. One can see the deterioration rates for L_1, L_2, dropout, and early stopping techniques in Table 4.4.

The deterioration rate for L_1 regularizer is a bit higher for CNN-ID and MLP-ID combinations in the ASCAD-R dataset and MLP-ID combination in Ascon-Unprotected dataset compared to the rest. The selected leakage model (ID) causes this higher deterioration rate. The larger number of output classes makes the model more sensitive to changes. The dispersion of the traces that we collect is fixed and independent from the leakage model that we select. When the selected leakage model leads into more output classes, we are partitioning the same data into more classes. This can make the model more sensitive to small variations in the input features. This could potentially make the model less robust to noise or other small changes in the input. Besides, with more output classes, the model generally becomes more complex, requiring more parameters to learn. This can make the model more sensitive to the nuances in the training data. Looking carefully, one can see that the columns with the ID leakage model in Table 4.4 show higher deterioration rates on average.

The deterioration rate for L_2 regularization is less than 2% for all the settings showing that L_2-regularized models almost always perform better than the baseline model regardless of the selected settings. This outcome confirms that adding L_2 regularization will improve the performance or, in the worst case, will simply not help.

The situation is different when dropout is used. Deterioration rates in Table 4.4 indicate that dropout degrades many regularized models. CNN with the HW leakage model suffers the most from applying dropout. After that, MLP with ID leakage model and CNN with HD leakage models are the combinations that worsen considerably after using dropout. The variation in the leakage models and network topology that experience the highest deterioration rate shows that the root cause of this observation is beyond the selected settings. The recognized deterioration after applying dropout is not unique to DL-SCA. In [99], Gabrin et al. reported reduced test accuracy after using dropout. Li et al. [100] showed that dropout could help accuracy but not in all cases. In [97], Srivastava et al. noted the necessity of changing the training and architectural hyperparameters to tune the model again after using dropout.

While the situation is better for early stopping compared to dropout, using it can still degrade some models. Again, CNN-ID is the combination that deteriorated the most, resulting from both neural network and leakage model selection. MLP-HW and MLP-HD show low deterioration rates after applying early stopping. However, for MLP-ID, it seems to be dependent on the dataset. Looking at Table 4.4 column-wise, MLP-HW combinations for the ASCAD-R dataset, MLP-HD combination for the AES-HD dataset and MLP-ID combination for the Ascon-Unprotected dataset

have the lowest deterioration rates for all regularization techniques. As mentioned in Section 4.3, MLP models are “generally able to find the key”, while CNN models are “potentially able to find the key”, and they should be tuned to work well. As a result, MLPs “absorb” changes in hyperparameters or added regularization techniques while applying small changes prevents CNNs from finding the key. In essence, the changes imposed on MLP models by regularization techniques do not deteriorate the models. This is why the combinations containing MLP show more improvements after applying regularization techniques, especially when models are smaller, i.e., when the number of output classes is less.

4.4.3. Profiling Time Changes

The average profiling time is the last considered metric that gives us useful information about the influence of regularization techniques on the models’ performance. Table 4.5 shows the calculated profiling time for baseline and regularized models. The numbers show that early stopping can reduce profiling time. The only change this technique imposes on the models is forcing them to stop the training as soon as the accuracy does not change for a number of epochs. This way, it can stop the training process earlier and reduce the profiling time. All the other techniques increase the profiling time considerably. The smaller numbers for Ascon-Unprotected dataset can be justified using two facts. Firstly, the number of epochs in this dataset is 30, while it is 200 for the other two datasets. Besides, the number of input time samples for this dataset is 772, while it is almost twice as much in the other two datasets (1400 for the ASCAD-R and 1250 for the AES-HD dataset). These two together will result in an average less training time for both combinations for the Ascon-Unprotected dataset.

Table 4.5: Average profiling time (in seconds) for baseline and regularized models with different regularization techniques.

	ASCAD-R				**AES-HD**		**Ascon**	
	CNN-HW	CNN-ID	MLP-HW	MLP-ID	CNN-HD	MLP-HD	CNN-Y4	MLP-Y4
Baseline models	933	986	651	663	825	465	16	12
L_1	3739	4620	1854	2603	6027	4832	23	18
L_2	3934	3958	1396	2097	6881	4194	28	17
Dropout	2420	3299	1209	1626	3759	1967	25	16
Early stopping	632	898	165	579	432	256	14	9

4.5. Discussion

So far, the experiments have investigated the influence of different regularization techniques on DL-SCA. Based on the experiments, the overall view confirms the dependency of regularized model improvement on different factors like the level of the dataset’s noise, leakage model, and neural network topology. In other words,

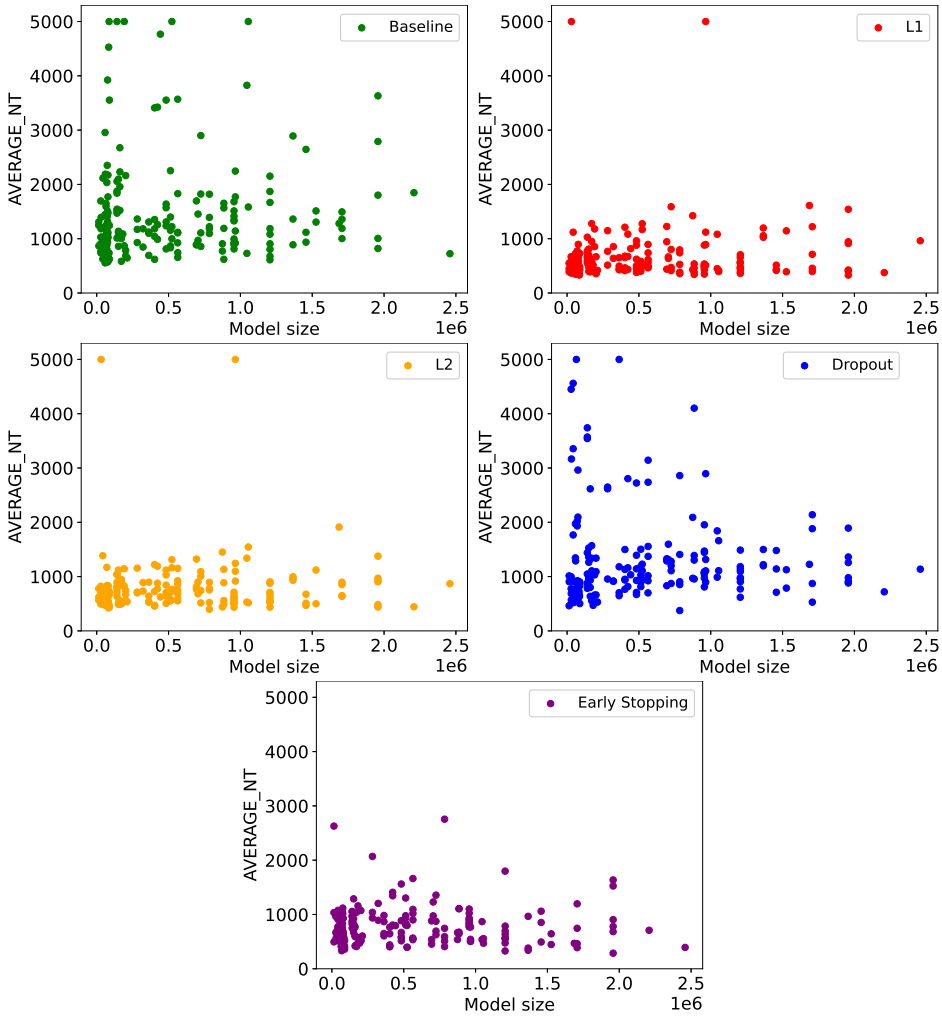


Figure 4.6: The dispersion of baseline and regularized models *NT* over their size in ASCAD-HW-MLP combination.

the improvement that a specific regularization technique offers differs per model and depends on the model's characteristics. However, it is still relevant to answer these two general questions:

- What is the most effective regularization technique among L_1, L_2, dropout, and early stopping?
- When does a regularization technique work at its best?

This section tries to find an answer to these two questions.

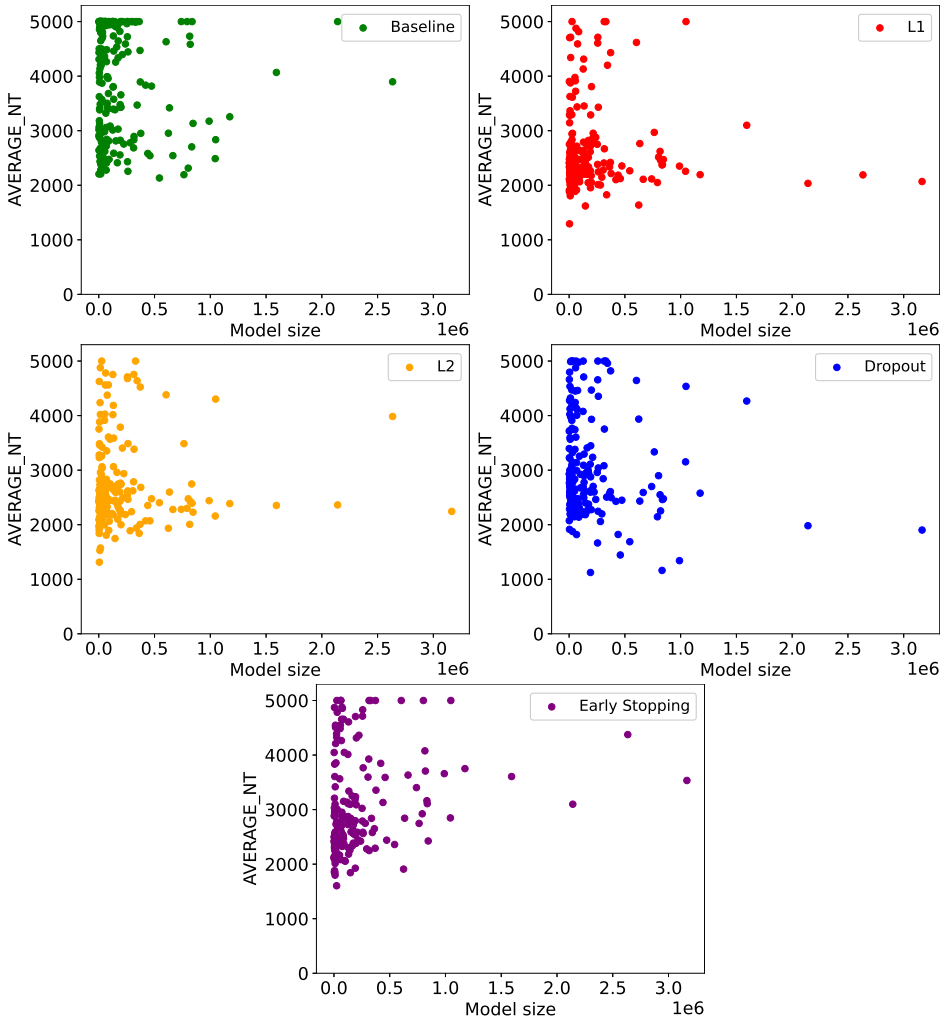


Figure 4.7: The dispersion of baseline and regularized models *NT* over their size in AES-HD-CNN combination.

4.5.1. Different Techniques Effectiveness in General

Considering the results in Section 4.4, it is not easy to say which regularization technique is the most effective among the four experimented ones. As mentioned earlier, the effectiveness of a regularization technique depends on different factors. However, Figures 4.6 and 4.7 try to give an overall comparison of L_1, L_2, dropout, and early stopping effectiveness in DL-SCA. The plots present the required number of attack traces (NT) over the size of all the baseline or regularized models for ASCAD-HW-MLP and AES-HD-CNN combinations.

As shown in Figure 4.6, the baseline models spread almost all around the plot (the

green spots) for the ASCAD-HW-MLP combination. Adding L_1 (red) or L_2(yellow), regularization pushes the spots to the bottom part of the plot so that the NT is less than 2000 traces for all L_1 and L_2-regularized models. In contrast, while adding dropout (blue) increases the density at the bottom of the plot, many spots remain on the upper side, which means it cannot improve many models as good as L_1 and L_2. Early stopping (purple) influence is considerably better than dropout but not as good as L_1 and L_2. Based on the dispersion of spots, L_1 improves the models slightly more than L_2.

The plots in Figure 4.7 (AES-HD-CNN combination) confirm the mentioned conclusion. In this combination, most of the selected models are small.[4] Therefore, the spots group in the left part of the plots. However, one can still see the sparseness of baseline models along the y-axis. Adding L_1 and L_2 regularization pushes the red and yellow spots to the lower corner. The better effectiveness happens for L_1 again. Early stopping pushes down the purple spots, and its effectiveness is close to L_2 regularization. In the case of dropout, while there are a few spots lower than 1500 (the other three regularization techniques could not push more than one spot under 1500), the rest of the spots mostly spread from 2000 to around 3500. This spreading range shows the poorer effectiveness of dropout.

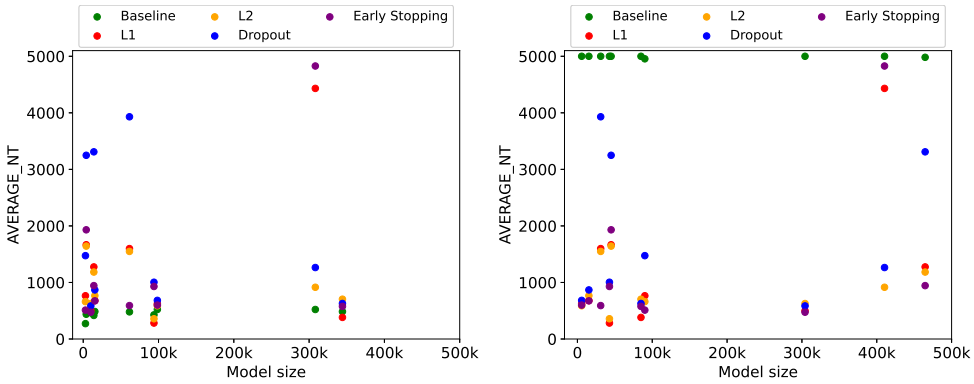
4.5.2. Implicit and Explicit Regularization Comparison

Recent works have studied implicit and explicit regularizations and their connections. Here, we use simplified definitions of those terms without getting into the details to specify when it is a good idea to use regularization techniques. Implicit regularization[5] is the effect imposed by the characteristics of the neural network architecture and the learning algorithm. In other words, it is the regularization that can be provided with a model and the learning algorithm. Designing a model using hyperparameter tuning along with deep and detailed knowledge about the dataset in hand can result in finding the optimal hyperparameters and parameter set for the problem, which provides a model with sufficient implicit regularization. This regularization does not change the objective function [101] [102]. The gradient descent algorithm (and stochastic gradient descent as its extension) offers implicit regularization inherently [101] [103]. On the other hand, explicit regularization modifies the expected loss and objective function and reduces the effective capacity of a given model to reduce overfitting. Explicit regularization is mostly provided by regularization techniques like dropout and norm penalties [102].

Implicit and explicit notions and the best and the worst baseline models in each combination are used to state when applying regularization techniques is effective. Among the baseline models selected for each combination, some models work very well and can rank the correct key in the first place with a few attack traces. These are the models close to carefully designed networks and offer adequate implicit regularization by themselves. They reduce overfitting and increase generalization

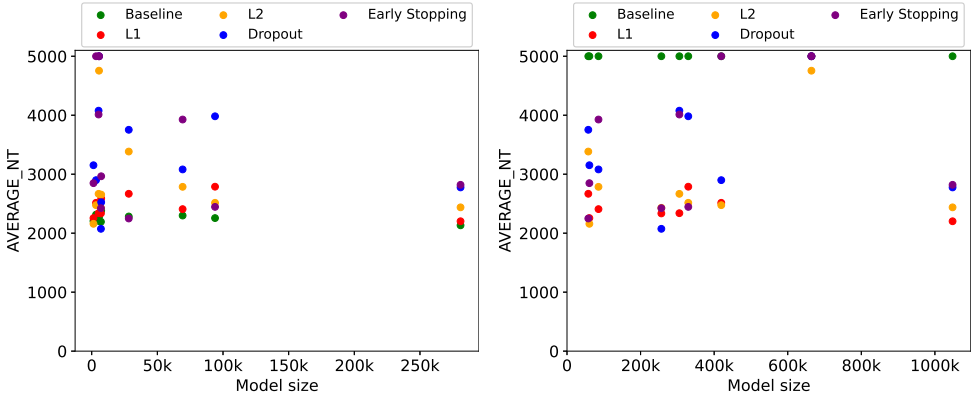
[4] The small size of neural networks is imposed during the hyperparameter search and baseline model selection. In general, hyperparameter tuning imposes selecting smaller models that offer a better implicit regularization.

[5] Also, algorithmic regularization.



(a) Best baseline models (b) Worst baseline models

Figure 4.8: (a) Ten best baseline models in ASCAD-HW-CNN combination along with their L_1, L_2, dropout, and early stopping regularized counterparts. Baseline models have better performance than their regularized counterparts. (b) Ten worst baseline models in ASCAD-HW-CNN combination along with their L_1, L_2, dropout, and early stopping regularized counterparts. Regularized models have better performance than their baseline counterparts



(a) Best baseline models (b) Worst baseline models

Figure 4.9: (a) Ten best baseline models in AES-HD-CNN combination along with their L_1, L_2, dropout, and early stopping regularized counterparts. Baseline models have better performance than their regularized counterparts. (b) Ten worst baseline models in AES-HD-CNN combination along with their L_1, L_2, dropout, and early stopping regularized counterparts. Regularized models have better performance than their baseline counterparts

sufficiently with the learning algorithm. Also, some other models can only find the key with a significant number of traces or cannot rank the key in the first place even after using the maximum available attack traces. The implicit regularization of these models is insufficient, and they usually need extra regularization to reduce overfitting (and increase generalization).

In Figure 4.8, one can see the AVERAGE-NT for ten baseline models that perform the best (green spots in Figure 4.8a) and ten baseline models that perform the worst (green spots in Figure 4.8b) among 200 selected baseline models in ASCAD-HW-CNN combination. In Figure 4.8a, one can see that the best ten selected baseline models have an acceptable performance before applying any regularization techniques. On the other hand, their counterpart regularized models perform worse in almost all cases. Figure 4.8a indicates the worst ten selected baseline models in the ASCAD-HW-CNN combination. As the opposite of best-selected models, the AVERAGE-NT for these baseline models is around 5000, while regularized models' performance is far better. In many cases, the performance of the worst models after applying a regularization technique is comparable with the best baseline models. Figure 4.9 shows the same behavior for the AES-HD-CNN combination. As depicted in Figure 4.9, the best ten baseline models worsen after adding regularization techniques. At the same time, the worst ten baseline models show good performance after applying regularization techniques.

This observation shows that regularization techniques are more effective when the selected baseline model does not offer enough regularization by itself. Thus, it seems efficient to use regularization techniques specially when the model is selected randomly and does not provide excellent performance.

4.5.3. Regularizer Benefit When *GE* Is One

As mentioned in Section 4.4.1, more than 200 of the randomly generated models in Ascon-ID-MLP and Ascon-ID-CNN scenarios converged to $GE = 1$ (In fact, their final *GE* was less than two). With a view of this, after selecting the 200 best models in each scenario, the AVERAGE-GE was less than two for baseline models. Since all the baseline models could find the key in the first place, inspecting the effect of regularization techniques in such a situation is compelling. Figure 4.10 shows the AVERAGE-GE and AVERAGE-NT for the baseline and regularized models after using L_1, L_2, dropout, and early stopping. Interestingly one can see that while the AVERAGE-GE increased most of the time, the AVERAGE-NT decreased in almost all the cases. It means that even if a model is good and can recover the key without any regularization techniques, we can still improve the performance by adding a regularization technique that appears as less required attack traces. Compared to the rest, the improvement is more pronounced for L_1 and L_2 regularization for the CNN-ID combination. For the CNN-ID combination, the required attack traces decreased up to two times for regularized models with L_1 and L_2, even the AVERAGE-NT decreased slightly. The improvement for this combination after using dropout and early stopping is not as sharp as L_1 and L_2 but is better compared to the MLP-ID combination. The improvement of AVERAGE-NT for the MLP-ID combination is

minimal for all the regularization techniques.

A closer look into the modified baseline models' deterioration rates in Ascon-Unprotected dataset in Table 4.4 and the deterioration rates for the primary selected models (the ones with $AVERAGE - GE < 2$) in Table 4.6 approves that in both schemes more or less similar number of models deteriorated.

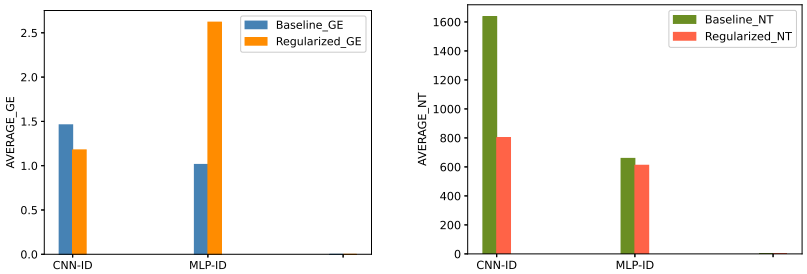
Table 4.6: The deterioration rate for primary baseline models.

	Ascon-Unprotected	
	CNN-Y4	MLP-Y4
L_1	1%	9%
L_2	1.5%	1.5%
Dropout	7%	7.5%
Early stopping	6.5%	2%

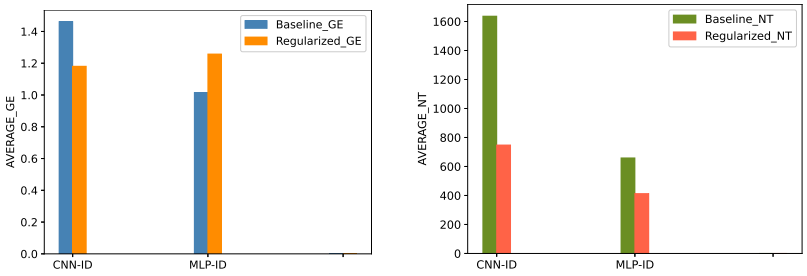
4.6. Conclusions and Future Work

This work provides an in-depth study of L_1, L_2, dropout, and early stopping influence on the performance of DL-SCA (eight different combinations of datasets, leakage models, and deep learning network topologies). Our experimental results show that while all these techniques can improve the DL-SCA performance, some of them are more effective than others. Considering the average required attack traces (AVERAGE-NT), the average guessing entropy (AVERAGE-GE), and the deterioration rate, we observe that L_1 and L_2 are the most effective regularization techniques. While early stopping has moderate effectiveness, it can reduce training time. In comparison, other techniques increase training time considerably. Since the dropout deterioration rate is very high compared to the other techniques and it increases the training time, we recommend using it carefully. Overall, there is potential in using regularization techniques to resolve the overfitting issue in SCA, but they should be used with care and consideration for their strength and weakness.

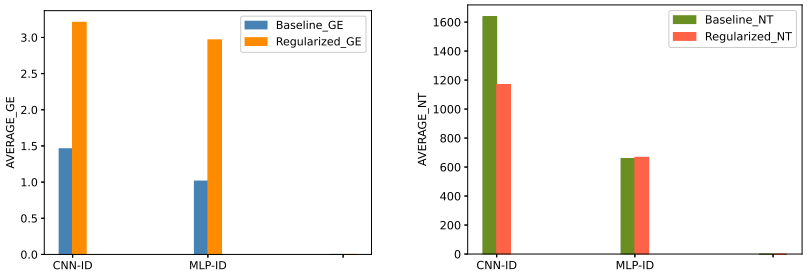
In future work, it would be interesting to compare the influence of other, more advanced regularization techniques, like batch normalization and data augmentation, with the current work results. Another interesting direction is combining different techniques and checking if adding two or more regularization techniques can improve the performance further. We tried it already for a limited number of models with dropout and L_2 regularization together. The results showed that a model using both dropout and L_2 performed better than the baseline or regularized model with only dropout or L_2. However, to generalize the observation, a more comprehensive study is needed. Besides, while we used ASCAD-R, AES-HD, and Ascon-Unprotected for our experiments, other datasets, especially from public cryptography, can be targeted to further investigate regularization techniques' effectiveness.



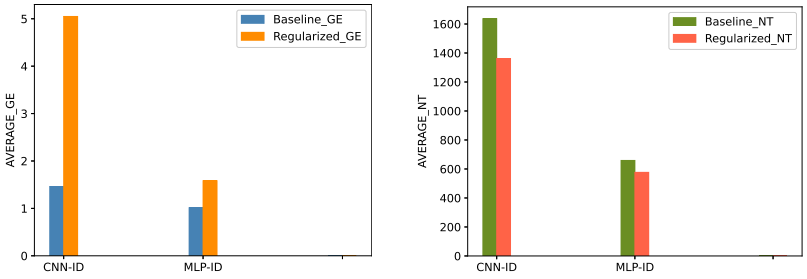
(a) L_1 regularization



(b) L_2 regularization



(c) Dropout



(d) Early stopping

Figure 4.10: The average performance with and without L_1, L_2, dropout and early stopping regularization, for Ascon-Unprotected dataset. The AVERAGE-GE (left) is calculated for baseline (blue) and regularized (orange) models. The AVERAGE-NT is calculated for baseline (green) and regularized (red) models.

5

Jump, It Is Easy: JumpReLU Activation Function in Deep Learning-based Side-channel Analysis

One of the main directions that the DL-SCA community should explore is how to design efficient architectures that can break the targets with as little as possible attack traces, but also how to consistently build such architectures. This chapter, explores the usage of the JumpReLU activation function, which was designed to improve the robustness of neural networks. Intuitively speaking, improving the robustness seems a natural requirement for side-channel analysis, as hiding countermeasures could be considered as adversarial attacks. In this chapter we will see using JumpReLU is a good option to improve the stability of attack results.

5.1. Introduction

Finding high-performing neural network architectures is challenging as it involves selecting from a long list of hyperparameters. From these hyperparameters, in the DL-SCA domain, the activation function is normally selected to be either ReLU or SeLU. ReLU is the most commonly used activation function, mainly due to its simplicity and effectiveness. Despite research on novel activation functions specifically designed for side-channel analysis, such as the proposal by Knežević et al. [104], ReLU remains a standard choice. However, this does not mean that more suitable activation functions cannot be designed.

In the field of adversarial learning, Erichson et al. [105] presented the JumpReLU activation function. The authors describe it as a very simple and inexpensive strategy that can be used to "retrofit" a previously trained network to improve its resilience to adversarial attacks (i.e., attacks on machine learning that aim to cause misclassific-

ations of the machine learning algorithm).

In DL-SCA, traces are very noisy for different reasons including added countermeasures. Implementation countermeasures can be seen as a manipulation to deceive the model into making incorrect predictions. Thus, the natural questions that arise are: Can JumpReLU be used to improve model efficincy in the SCA domain? Can JumpReLU be used as in [105] on already-trained models to enhance their accuracy? Which threshold value should be used? Can we expect randomly generated models to perform better using JumpReLU instead of ReLU and SeLU?

In this chapter, we look for answers to these questions by performing a comprehensive list of experiments with the ASCAD-F and ASCAD-R datasets.
In summary, the main contributions are:

1. Providing a benchmark between random models using ReLU, SeLU, and JumpReLU.

2. Verifying whether JumpReLU can be used in already-trained SCA models to increase their performance.

3. Verifying how performance is affected when an existing architecture is taken and its activation function is replaced with JumpReLU.

Related Work: Since the work of Maghrebi et al. [5], where the first published results of deep learning techniques applied to the domain of side-channel analysis are presented, multiple research works showcased the importance of hyperparameter tuning and how identifying high-performing models can be challenging due to the large number of hyperparameters that must be considered. As such, multiple works considered how to find high-performing neural network architectures efficiently, see, e.g., [76, 78, 79, 106, 107].

Regarding the impact of activation functions, Benadjila et al. [7] have studied the effect of the activation function on the performance of the neural network, and they reported that their best results were obtained with ReLU, tanh, and softsign (a variation of tanh), but they chose ReLU due to its state-of-the-art results and because its computation time is lower than the other two functions. Knežević et al. [104] used evolutionary algorithms to evolve new activation functions for side-channel analysis. Their experiments with the ASCAD-R database showed that this approach is highly effective compared to results obtained with standard activation functions and that it can match the state-of-the-art results from the literature. The authors evaluated two HW and ID leakage models and MLP and CNN architectures. Moreover, Do et al. [108] analyzed the effect of the activation function in MLP- and CNN-based deep learning models for non-profiled side-channel analysis. For their MLP-based models, using the ELU activation function provided better performance than ReLU in fighting against noise generation-based hiding countermeasures.

5.1.1. Activation Functions

Activation functions are an essential component of neural networks, introducing non-linearity. They enable the learning of complex patterns and relationships in data.

Without activation functions, a neural network would be equivalent to a single-layer model, regardless of the number of layers, limiting its ability to capture complicated dependencies. Activation functions help determine how neurons respond to inputs and control the flow of gradients during backpropagation, affecting convergence speed and overall training stability [109].

ReLU: The Rectified Linear Unit (ReLU) activation function is a widely used non-linear function in artificial neural networks [110, 111], particularly in deep learning models. It is defined by:

$$g(z) = max\{0, z\}. \tag{5.1}$$

ReLU is the default recommendation in modern neural networks [94].

SeLU: The Scaled Exponential Linear Units (SeLU) [112] activation function is defined by:

$$f(x) = \lambda \begin{cases} x, & \text{if } x > 0. \\ \alpha(e^x - 1), & \text{if } x \leq 0, \end{cases} \tag{5.2}$$

with $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$.

JumpReLU: The JumpReLU [105] (Jump Rectified Linear Unit) activation function is a variant of the standard ReLU function with a jump discontinuity, yielding piece-wise continuous functions. It introduces a positive threshold parameter κ such that the neuron remains inactive until its input exceeds κ. The magnitude of the jump κ is a parameter the user must define.

$$J(z) = zH(z - \kappa) = \begin{cases} 0, & \text{if } z \leq \kappa. \\ z, & \text{if } z > \kappa. \end{cases} \tag{5.3}$$

Here, H denotes the discrete Heaviside unit step function. The JumpReLU activation function introduces robustness and an additional amount of sparsity, controlled via the jump value κ. Thus, JumpReLU suppresses small positive signals. We depict the operation of the JumpReLU activation function in Figure 5.1.

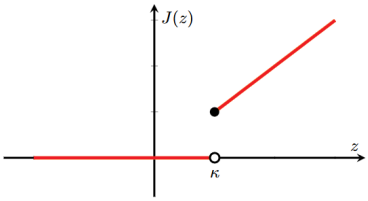


Figure 5.1: JumpReLU activation function.

When presented, JumpReLU was proposed as a strategy to improve resilience to adversarial attacks. Moreover, it was concluded that it can be used in models already trained using ReLU. In this scenario, it provides a trade-off between robustness and classification accuracy, which the user can control in a post-training stage by tuning the threshold value κ.

Another application where JumpReLU has shown improved results versus other activation functions is in sparse autoencoders for language models. Here, JumpReLU enables more faithful reconstructions than competing methods, such as Gated or TopK Sparse AutoEncoders [113].

5.2. Analysis Methodology

The goal of this section is to examine how the use of JumpReLU affects the performance of DL-SCA. To clarify this, we investigate two different scenarios.

- We explore whether substituting the activation function in well-known architectures with JumpReLU can potentially enhance their performance. This is explored in two settings:
  - Retrofit: Substitution of the activation function on an already trained model, i.e., JumpReLU is used only at the inference phase.
  - Substitution: Substitution of the activation function and training of the model, i.e., JumpReLU is employed during both the training and inference phases.
- We investigate whether the average performance of random CNN and MLP models for DL-SCA is better using JumpReLU compared to the other activation functions.

In the following, we introduce the steps required for each scenario.
In the **first scenario**, we would like to see if replacing the activation function of well-known architectures with JumpReLU results in improving the performance of those models. To verify this, our methodology is straightforward. We take the following steps:

- **Acquiring baseline models:** We train well-known models reported in previous works. Training is done using identical hyperparameters and datasets to ensure comparable results. The models we used are listed in Table 5.1. The works listed in this table reported one or more CNN and/or MLP neural networks for at least one of the datasets mentioned earlier. We call them “baseline models”. The activation functions in baseline models are either ReLU or SeLU, but none use JumpReLU.
- **Record the baseline model’s performance:** After training each model and using it to perform the attack, we use guessing entropy (*GE*) and the number of attack traces (*NT*) to report the model’s performance for the target dataset the model was developed for.
- **Replacing the activation function with JumpReLU:** There are two ways to apply JumpReLU as the activation function in this scenario. The first and simpler approach is to replace the activation function only during the attack phase (**Retrofit**). The second approach is to replace the activation function of

Table 5.1: The list of works/architectures we consider in our experiments. We depict $*$ if the work reported a well-performing CNN and $\bullet$ if the work reported a well-performing MLP.

Covered datasets	ASCAD-F			ASCAD-R		
	$N^{[0]} = 0$	$N^{[0]} = 50$	$N^{[0]} = 100$	$N^{[0]} = 0$	$N^{[0]} = 50$	$N^{[0]} = 100$
[7]	$*, \bullet$	$*, \bullet$	$*, \bullet$	$*, \bullet$	$*, \bullet$	$*, \bullet$
[76]	$*$	$*$	$*$			
[106]	$*$	$*$	$*$			

the baseline model with JumpReLU and retrain the model from scratch. In this case, all other hyperparameters are kept unchanged, and only the activation function is replaced (**Substitution**).

As discussed in Section 5.1.1, JumpReLU introduces an additional hyperparameter, κ, which must be tuned for each neural network. To tune this parameter, we evaluate five different values for each baseline model, namely $\kappa \in \{0.001, 0.002, 0.003, 0.004, 0.005\}$. The model achieving the best performance among these configurations is referred to as the *JumpReLU-equivalent model*.

- **Record the JumpReLU-equivalent models' performance:** In the final step, we evaluate the performance of models that use the JumpReLU activation function and compare their performance with the baseline models. The results obtained by replacing the activation function only during the inference phase are reported as **Retrofit**. In contrast, the results obtained by replacing the activation function during both training and inference are reported as **Substitution**. The goal of this step is to assess whether replacing the activation function of previously well-performing models with JumpReLU can further improve their performance.

In the **second scenario**, we would like to see if using JumpReLU activation function can improve the performance of the models on average. To evaluate this, we compare the average performance of neural networks using two common activation functions (ReLU and SeLU being randomly selected for each neural network combination of hyperparameters) against their performance when their activation function is fixed to JumpReLU. The methodology for comparing performance is described in the following steps.

- **Acquiring baseline models:** We generate 500 neural networks of the specific topology (MLP or CNN) using a random search. This is the pool of "baseline models". The searching ranges for hyperparameters of MLP and CNN are listed in Table 5.2. These ranges are chosen based on those reported in the previous works [7, 76, 89]. Since those 500 neural networks are generated randomly,many of them fail to decrease guessing entropy to $GE = 1$.

Table 5.2: Searched range of MLP and CNN hyperparameters. For both MLP and CNN dense layers, we used the ranges shown in *dense layers* part of the table.

Hyperparameters	Range
MLP dense layers	
Number of neurons	[10, 30, 50, 70, 90, 120, 150, 200, 250, 300, 400, 500]
Number of layers	[2, 8], step = 1
CNN convolution and dense layers	
Number of neurons in dense layer	[50, 100, 150, 200, 300, 400, 500]
Number of dense layers	[2, 4], step = 1
Number of convolution layers	[2, 4], step = 1
Kernel size	[4, 20], step = 2
Number of filters	[4, 24], step = 4
i^{th} layer filter size	$((i-1)^{th} filter\_size)^2$
Pooling	"Average", "Max"
Pooling size	[2, 10], step = 2
Pooling stride	[2, 10], step = 2
Learning hyperparameters	
Optimizer	"Adam"
Weight initialization	"random_uniform", "he_uniform", "glorot_uniform",
Activation function	"ReLU", "SeLU"
Batch size	$[128, 256, 512]$
Learning rate	$[1e-3, 5e-4, 1e-4, 5e-5, 1e-5]$
Epochs	100

- **Performance of the converging baseline models:** First, we report how many models, out of 500 baseline models, are able to reach $GE = 1$ within 4000 attack traces. We call them "converging baseline models". Reporting this number allows us later to assess whether the use of JumpReLU helps models, in general, to achieve better generalization. We also use two metrics, *AVERAGE_NT* and *MINIMUM_NT*, to report the overall performance of the converging-baseline-models. The *AVERAGE_NT* represents the average number of attack traces for the MLP or CNN baseline models when they reach $GE = 1$. The *MINIMUM_NT* corresponds to the minimum number of attack traces required by the best-performing model among the converging baseline models.

- **Re-training baseline models with JumpReLU:** In step one, we generated the pools of baseline models with 500 different models (MLP or CNN neural networks). Now, we replace the activation function of the baseline models with JumpReLU and re-train them for $\kappa \in \{0.001, 0.002, 0.003, 0.004, 0.005\}$ jumping thresholds. Again not every model converges to $GE = 1$, therefore, for each κ value, we focus on the models that can converge to $GE = 1$ for further investigation and call them the converging JumpReLU-equivalent models.

- **The average performance of JumpReLU-equivalent models:** The AV-

ERAGE_NT and the MINIMUM_NT are calculated as performance metrics for the converging JumpReLU-equivalent models in each pool with $\kappa \in \{0.001, 0.002, 0.003, 0.004, 0.005\}$. We compare the AVERAGE_NT and MINIMUM_NT of the converging-baseline-models and the converging JumpReLU-equivalent models. We also investigate if the number of converging models increased or decreased after replacing the original activation function with JumpReLU. This way, the influence of using JumpReLU on average DL-SCA performance can be observed.

5.3. Experimental Results

5.3.1. First Scenario: Effect on Well-known Models

Retrofit

To answer if the behavior observed by Erichson et al., where JumpReLU can be used to "retrofit" already trained models, is also true for SCA, we used two architectures described in [7]: MLP_{best} and CNN_{best}. For both architectures, ten models were trained for each dataset with each $N^{[0]}$ using its original activation function (ReLU), and at the inference moment, ReLU and JumpReLU were used. As there is no previous data on which threshold range κ would provide the best performance, three different sets of values were tested $S_a = \{0.001k \mid k \in [1..9]\}$, $S_b = \{0.01k \mid k \in [1..9]\}$ and $S_c = \{0.1k \mid k \in [1..9]\}$.

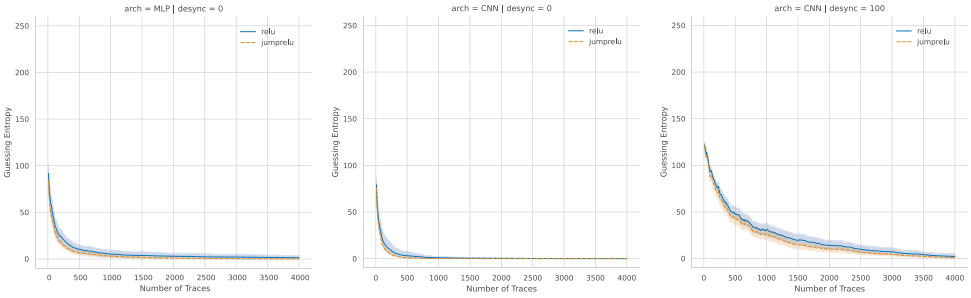
MLP_{best}-based models only achieved generalization on the ASCAD-F dataset with $N^{[0]} = 0$. All CNN_{best}-based models achieved generalization on both datasets and with the three evaluated $N^{[0]}$ values.

The experimental results show that using JumpReLU at inference time with the already-trained models did not provide any relevant performance variation. When the value of κ was from S_a or S_b, the performance of the models was almost identical to that of ReLU; for κ values in S_c equal to or greater than 0.5, the performance was even slightly worse than with ReLU. These results are consistent across both datasets and the three $N^{[0]}$ corresponding values.

Substitution

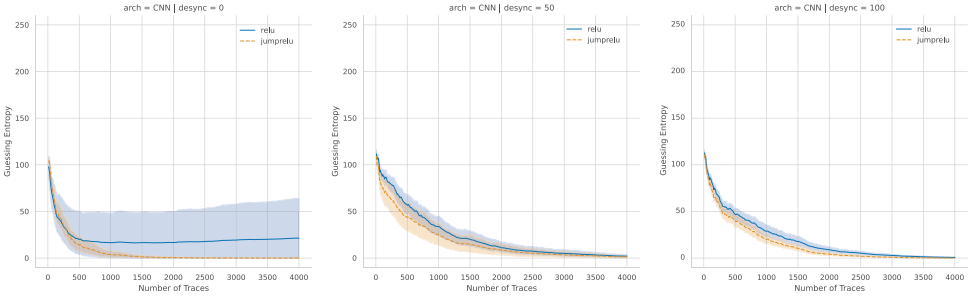
Having seen that simply using JumReLU at inference time does not provide the desired performance gain, we investigate the impact of substituting the original activation function with JumpReLU. Again, we used MLP_{best} and CNN_{best} based models, but this time, these are trained using JumpReLU. Concerning model generalization, the results are the same as described in the previous experiments. However, in this scenario, the value of κ has a more pronounced impact on the performance of the models. Here, κ values from S_a provide a slight improvement for some models (see Figures 5.2 and 5.3), while values from S_b generally provide worse performance, and finally, values from S_c mostly lead to models not generalizing. Note that the figures depict the average behavior in darker lines and the standard deviation by shaded area in the same color.

With the information that JumpReLu can provide a small performance gain on architectures that already have shown good performance, we proceed to evaluate the performance of other architectures that do not use the ReLU function but SeLU. The tested architectures are from [76] and [106]. Both works used a technique called One Cycle Policy [114] to choose the learning rate hyperparameter. However, as we want to isolate the effect of substituting the activation function to JumpReLU, this technique is not used in our experiments, and only three different learning rate values were used: $\{0.0005, 0.00025, 0.0001\}$. The experimental results show that for these architectures, SeLU presented the best results, and JumpReLU did not provide any improvement.



(a) MLP; $N^{[0]}$=0; κ=0.001 (b) CNN; $N^{[0]}$=0; κ=0.004 (c) CNN; $N^{[0]}$=100; κ=0.003

Figure 5.2: ASCAD-F observed performance improvement.



(a) CNN; $N^{[0]}$=0; κ=0.001 (b) CNN; $N^{[0]}$=50; κ=0.001 (c) CNN; $N^{[0]}$=100; κ=0.001

Figure 5.3: ASCAD-R observed performance improvement.

5.3.2. First Scenario: Effect on Random Models

Our observations show that substituting the activation function of existing well-performing architectures with JumpReLU cannot provide a significant advantage. Thus, the remaining scenario where JumpReLU is considered is to find new well-performing architectures. We explore this scenario by performing a random model search.

ASCAD-F

The results for the ASCAD-F dataset are given in Table 5.3. For MLP-based models, $GE = 0$ was only reached with $N^{[0]} = 0$, with 163 models using ReLU and 104 using SeLU. CNN-based models did reach $GE = 0$ for all $N^{[0]}$, with 68 and 52 models for ReLU and SeLU activation functions, respectively. Note that we do not show any results with MLP and desynchronization as we could not find any architecture breaking the target with the given number of attack traces.

Table 5.3: Experimental results for the ASCAD-F dataset. The column Models denotes the number of models that reached GE equal to 0. $N^{[0]}$ denotes the desynchronization rate, Arch denotes the architecture type, and AF denotes the activation function used (in the last layer is always softmax).

$N^{[0]}$	**Arch**	**AF**	**Threshold**	**Models**	**Avg NT**	**Min NT**
0	cnn	SeLU	-	52	2202.7	460
0	cnn	ReLU	-	68	2682.8	570
0	cnn	JumpReLU	0.001	118	2289.2	440
0	cnn	JumpReLU	0.002	116	2430.6	390
0	cnn	JumpReLU	0.003	119	2457.7	430
0	cnn	JumpReLU	0.004	115	2437.2	420
0	cnn	JumpReLU	0.005	117	2494.5	490
50	cnn	SeLU	-	3	3123.3	2690
50	cnn	ReLU	-	1	4000.0	4000
50	cnn	JumpReLU	0.002	1	3990.0	3990
100	cnn	JumpReLU	0.001	2	3980.0	3970
100	cnn	JumpReLU	0.002	2	3860.0	3720
100	cnn	JumpReLU	0.005	2	3915.0	3830
0	mlp	ReLU	-	163	705.1	210
0	mlp	SeLU	-	104	902.0	180
0	mlp	jumpReLU	0.001	344	732.8	220
0	mlp	JumpReLU	0.002	344	754.6	210
0	mlp	JumpReLU	0.003	334	745.7	210
0	mlp	JumpReLU	0.004	337	729.6	230
0	mlp	JumpReLU	0.005	334	724.0	190

For JumpReLU, we see that both MLP and CNN architectures reach good results. More precisely, for CNNs, the threshold variations do not cause many differences, and all settings are stable: a similar number of models breaking the target and a similar minimal number of traces to break the target. Moreover, we observe that breaking a synchronized target is relatively simple, while after added desynchronization, the task becomes significantly more difficult (with only a handful of architectures breaking the target). An additional observation is that CNN with SeLU activation function actually reaches a better best result (only 2690 attack traces needed vs. 3990 for the case with JumpReLU). Still, CNNs with SeLU and ReLU do not manage to break the unsynchronized version with the 100 desynchronization level at all, showcasing

that JumpReLU provides more robustness. On the other hand, for MLP, we see that using JumpReLU allows for the majority of the tested models to break the target, and the best results are comparable with the SeLU case (180 vs. 190 attack traces). To conclude, JumpReLU allows more architectures to break the target, with minimal influences from the selected threshold level, and allows comparable results on the minimal number of attack traces required to break the target.

ASCAD-R

Table 5.4: Experimental results for the ASCAD-R dataset. The column Models denotes the number of models that reached GE equal to 0. $N^{[0]}$ denotes the desynchronization rate, Arch denotes the architecture type, and AF denotes the activation function used (in the last layer is always softmax).

$N^{[0]}$	Arch	AF	Threshold	Models	Avg	Min
0	cnn	SeLU	-	17	2956.5	470
0	cnn	ReLU	-	29	2656.6	350
0	cnn	JumpReLU	0.001	71	2349.4	170
0	cnn	JumpReLU	0.002	70	2527.4	210
0	cnn	JumpReLU	0.003	69	2463.2	300
0	cnn	JumpReLU	0.004	69	2505.8	290
0	cnn	JumpReLU	0.005	66	2378.6	240
50	cnn	SeLU	-	1	3800.0	3800
50	cnn	JumpReLU	0.001	1	2930.0	2930
50	cnn	JumpReLU	0.003	1	3980.0	3980
100	cnn	SeLU	-	1	4000.0	4000
100	cnn	JumpReLU	0.003	1	4000.0	4000
0	mlp	ReLU	-	75	2333.2	710
0	mlp	SeLU	-	42	1670.7	340
0	mlp	JumpReLU	0.001	143	2153.7	420
0	mlp	JumpReLU	0.002	132	2249.5	480
0	mlp	JumpReLU	0.003	131	2346.5	600
0	mlp	JumpReLU	0.004	136	2515.8	560
0	mlp	JumpReLU	0.005	128	2333.0	700
50	mlp	SeLU	-	1	3990.0	3990
100	mlp	SeLU	-	1	4000.0	4000

The results for the ASCAD-R dataset are given in Table 5.4. With this dataset, for all $N^{[0]}$ values, at least one model reached $Ge = 0$. Considering JumpReLU, we can again observe that the threshold level does not play a significant role. Moreover, as before, JumpReLU allows more architectures to break the target than ReLU and SeLU. Still, with MLP-based models with desynchronization levels of 50 and 100, we break the target using SeLU, while we cannot do it with JumpReLU. However, since there is only one such architecture, it is difficult to assess the relevance of

such a result. For CNNs without desynchronization, we also observe that JumpReLU reduces the minimal number of attack traces, giving additional advantage to the usage of JumpReLU.

5.4. Conclusions and Future Work

This chapter investigates how the JumpReLU activation function can improve the performance of DL-SCA models. Finding efficient models is a significant challenge for DL-SCA, and the process is still largely reliant on the designer's expertise and the computing resources at their disposal, as these factors determine how much experimentation and fine-tuning can be conducted within a given timeframe. Finding one model that shows good performance with a given combination of hyperparameters does not imply that small changes to any specific hyperparameter will lead to a predictable improvement or degradation in performance. Hyperparameter tuning is often highly context-dependent, with interactions between parameters influencing the overall model behavior. With this in mind and based on our experimental results, we can conclude that JumpReLU can be considered a promising option when constructing new architectures for DL-SCA. More precisely, we see especially encouraging results when the JumpReLU is given as one of the options during the hyperparameter tuning. The architectures with it seem to improve the performance from two aspects: 1) more architectures breaking the target and 2) fewer attack traces required to break the target for the most performant architectures.

In future work, we plan to explore whether JumpReLU can bring advantages against other hiding countermeasures like Gaussian noise or jitter. Moreover, JumpReLU showed very good performance when combined with sparse autoencoders [113], which could be another interesting research direction for DL-SCA.

6

One for All, All for Ascon: Ensemble-based Deep Learning Side-channel Analysis

One of the successful methods that reduce the effort required to identify an optimal model is ensemble learning. While ensemble methods have demonstrated their effectiveness with AES-based datasets, their efficacy in analyzing symmetric-key cryptographic primitives with different operational mechanics remains unexplored.

Ascon was announced in 2023 as the winner of the NIST lightweight cryptography competition. This announcement leads to broader use of Ascon and a crucial requirement for thorough side-channel analysis of its implementations. With these two considerations in view, this chapter utilizes an ensemble of deep neural networks to attack two implementations of Ascon. Using an ensemble of five multilayer perceptrons or convolutional neural networks,the secret key for the Ascon-protected implementation can be retrieved with less than 3000 *traces. To the best of my knowledge, this is the best currently known result on the implementation which proves the efficiency of DL-SCA and particularly ensemble learning for evaluating cryptography implementations. For the unprotected implementation, the correct key can be retrieve with less than 100 traces using ensemble learning, which is on par with the state-of-the-art results.*

6.1. Introduction

Deep learning-based side-channel analysis has become a research hot spot from 2016 [5]. The studies have reported many advantages for this approach. However, despite all the advantages of DL-SCA, it is frequently emphasized that the principal challenge for using it is the selection of a neural network model that is tailored

to the specific nuances of the problem at hand [4]. To overcome that challenge, various approaches, including hyperparameter tuning [78, 79, 115], regularization techniques [39], or designing a methodology for model selection [76] have been suggested.

An interesting and effective strategy proposed to circumvent (or, at least alleviate) the challenge of finding an optimal model is the utilization of ensemble techniques [77], where multiple sub-optimal neural network models combine to enhance the overall performance of DL-SCA. While the results presented in that work demonstrate the utility of the ensemble method in enhancing attack performance, a gap in generalization across various cryptographic primitive implementations is evident.

Until recently, the publicly available datasets for symmetric-key cryptography were centered around the AES primitive, as discussed in [4]. Consequently, the effectiveness of many proposals, including the ensemble, has been validated using only AES-based datasets. This raises a question about the efficiency of diverse proposals in DL-SCA for AES when considering other cryptographic primitives. Ascon, the NIST lightweight cryptography competition winner, is currently being standardized for broad public use. Therefore, it is an ideal subject for such investigation.

On the other hand, introducing the Ascon family as a new standard for authenticated encryption [46] has raised interest in the available implementations that could be used in embedded devices to secure them. Then, evaluating the physical security of cryptographic implementations against SCA is a crucial step in developing secure embedded devices. This shift toward considering Ascon as a benchmark in DL-SCA research not only aligns with its growing use but also provides a broader perspective on the adaptability and efficiency of DL-SCA across different cryptographic primitives.

In this chapter, we attack two software implementations of the Ascon primitive using the ensemble method. The key contributions of this chapter are:

- **Extension beyond AES-based experiments:** Previous research demonstrated the effectiveness of the ensemble method in DL-SCA, only focusing on the AES primitive. This chapter extends this, showing that the ensemble method is also effective for other cryptographic primitives. The improved attack performance on protected Ascon implementations should be particularly highlighted as the challenge is more significant, and the attacks have not been very successful to the date of publishing the paper related to this chapter. The successful results with ensembles give more evidences that other DL-SCA proposals aiming at AES may generalize for other cryptographic primitives.
- **Exploring Ascon in the context of side-channel analysis:** With Ascon being standardized by NIST and its usage expected to increase, there is a pressing need for comprehensive side-channel analysis of its implementations. In this chapter, we successfully attack both protected and unprotected Ascon implementations using the ensemble method. The attack using the ensemble method outperforms the state-of-the-art results, emphasizing the necessity of designing and implementing adequate countermeasures for vulnerable operations in Ascon's implementation.

Related Work: Two research streams are closely related to the work presented

in this chapter. The first stream employs ensemble methods to improve SCA. The second stream targets implementations of Ascon. The following briefly summarizes what has been done so far.

Ensemble methods were used in the SCA domain as soon as the community started to use machine learning. For example, Picek et al. in [116] used Random Forest (which is an ensemble of decision trees), Rotation Forest, and MultiBoosting, all methods that use ensembling to improve the accuracy of predictions. In [117] and [5], researchers again used Random Forest, which is one of the most popular options for machine learning-based SCA (next to Support Vector Machines). In a recent work, which should be considered as the first and the most relevant from the DL-SCA perspective, Perin et al. used bagging of multiple deep neural networks for attacking different AES implementation [77]. One can find more details about [77] in Section 6.2.

Several works have analyzed Ascon's side-channel resistance since its submission to the NIST lightweight competition. In [118], a method named SCARL is used to recover the secret key of an Ascon Artix-7 FPGA implementation. SCARL uses LSTM autoencoders for dimensionality reduction of S-box operations power measurements and reinforcement learning for clustering key candidates. In [119], transfer learning is used from gate-level power simulation traces for an Ascon software implementation running on a custom-made RISC-V SoC to improve the performance of DL-SCA using raw power traces as input (measured from a chip prototype of the same design). In [71], multi-task learning is used to evaluate the side-channel resistance of protected and unprotected Ascon datasets.

6.2. Ensembles

Ensemble techniques combine multiple predictors (machine learning models or deep neural networks) to reduce generalization error [94]. The predictor can be a simple machine learning method like a decision tree or an advanced one like a deep neural network. Ensemble techniques work because different predictors may capture various aspects of the data, and by combining them, one can often achieve better performance than every single model contributing to the ensemble. There are different techniques for ensemble predictors, including voting, bagging, boosting, and stacking. These techniques are different in how they create and combine the models. For example, bagging involves training multiple models independently and averaging their predictions. This method is useful for reducing variance and overfitting. Boosting, on the other hand, trains models sequentially, with each new model focusing on the data points that previous models miss-classified, aiming to improve the predictive performance iteratively. In deep learning, ensemble methods typically involve different architectures or configurations of neural networks, such as varying numbers of layers, nodes, or activation functions [77].

The ensemble method has also been used in the domain of SCA (see related works in Section 6.1). The ensembling approach used here is aligned with the one Perin et al. used in [77]. Their technique is specialized bagging (bootstrap aggregating), where the models in the ensemble are selected through a random

search, and each model is trained on the entire dataset (the "bag" used for training every single model is equal to the whole training dataset). The models are trained independently. We diverge from common practices like majority voting or averaging to integrate the models' output. That is because, in the context of DL-SCA, models often provide uncertain predictions about the class of a trace[1]. The accuracy of models in an attack phase is marginally better (or sometimes even worse) than a random guess [77]. Consequently, techniques like majority voting or averaging are ineffective in enhancing attack performance.

6.3. Methodology

This section provides the methodology to inspect the ensemble method's effectiveness for DL-SCA of Ascon primitive implementations. To evaluate the efficacy of ensembles, we compare the performance of the ensemble (a group of the neural network models) with the performance of *the best model*. To show that the results can be generalized, the experiments are conducted on Ascon-Unprotected and Ascon-Protected datasets introduced in Section 2.6.3. To assure that the results are valid for various neural network topologies, we employed MLP and CNN models combined with the attack point introduced in Section 2.2.2. We aim to retrieve x_1 or k_1, which is eight bytes of the sixteen-byte Ascon secret key. We use a divide-and-conquer strategy, i.e., we repeat the following steps eight times for each combination of two neural network topologies and two datasets, and each time, we retrieve a sub-key of size one byte.

- **Acquiring best predictor:** In [78], Wu et al. showed that random search can reach neural network models with top performance when one attacks relatively easy datasets. Considering this, we generate fifty different models using random search. The range of hyperparameters for the random search is given in Table 6.1). Then, we use guessing entropy to compare the performance of these fifty models and take the model with the best performance as *the best model*. It is worth mentioning that the selected model is not the best possible model. Other, more advanced hyperparameter tuning techniques (like reinforcement learning [79] or Bayesian optimization [78]) or searching with a wider range of hyperparameters with more randomly generated models can lead to models with better performance. Hence, our experiments aim not to find an optimal model, and we only want to investigate whether the ensemble performs better than the single best model. We report the best model's guessing entropy (GE-Best) and its number of attack traces (NT-Best) as the performance of the best model.

- **Acquiring ensemble:** To benefit from the ensemble method in general, a group of neural networks that individually can learn the problem and give predictions better than random guesses is needed. Since accuracy is not a good metric to judge the performance of a model in the SCA domain, we use

[1] Trace is the whole or part of the measurement given as input to the neural network

Table 6.1: Random search range for MLP and CNN hyperparameters.

Hyperparameter	Range
MLP's Architecture Hyperparameters	
Number of neurons	[30, 40, 50, 60, 70, 80, 90, 100, 120, 150]
Number of layers	[2, 8], step = 1
CNN's Architecture Hyperparameters	
Number of convolution layers	[2, 4], step = 1
Number of filters	[4, 20], step = 2
First layer's filter size	[4, 24], step = 2
$i^{(th)}$ layer filter size	$((i-1)filter\_size)^2$
Stride	[2, 10], step = 2
Pooling	"Average", "Max"
Pooling size	[4, 10], step = 2
Pooling stride	[4, 10], step = 2
Number of dense layers	[2, 4], step = 1
Number of neurons in dense layers	[50, 100, 150, 200, 300, 400, 500]
Common Learning Hyperparameters in MLP and CNN	
Learning rate	random.uniform(0.0001, 0.001)
Activation function	"relu", "tanh", "selu", "elu"
Optimizer	"Adam"
Weight initialization	"he_uniform"
Batch size	128
Epochs	10

guessing entropy to select models that perform the best among the randomly generated models. We take five[2] models with the smallest guessing entropy from the pool of randomly generated models to be used in the ensemble.

The selected models do not necessarily need to find the key (reach $GE = 1$); they only need to reduce the GE to small values. In Section 2.6.2, we have seen that guessing entropy can be calculated by accumulating the probability that the neural network gives for each key hypothesis over the attack traces. In the case of ensemble-based DL-SCA, we sum up the probabilities for each key hypothesis from all individual models in the ensemble and accumulate that over the attack traces. The final guessing entropy is reported as the ensemble guessing entropy and is referred as GE-Ensemble. The number of attack traces can be calculated using the GE-Ensemble, which we call NT-Ensemble.

[2]This number can vary depending on the problem, the complexity of individual models, and the desired balance between performance and complexity. In the experiments, it observed that five models could already offer good performance improvement.

- **Comparing the best model and the ensemble performance:** The final step is comparing the performance of the best-acquired models (GE-Best and NT-Best) and the performance of the ensemble model (GE-Ensemble and NT-Ensemble). The selected group of predictors for the ensemble always includes the best predictor, and improved performance means that the ensemble method was effective.

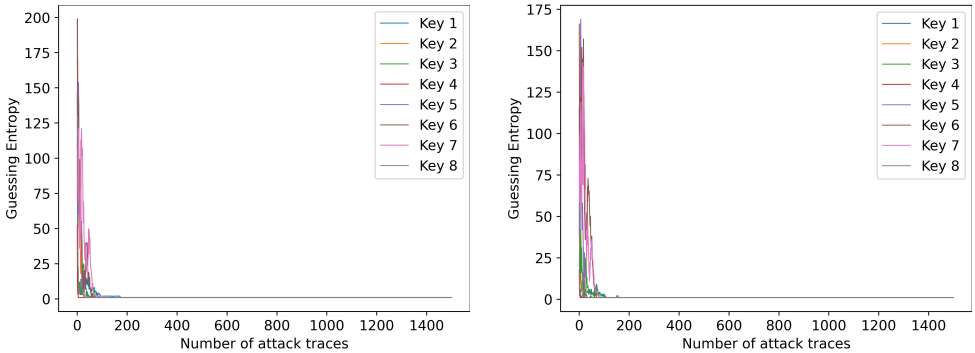
6.4. Experimental Results

The objective of this section is to demonstrate the effectiveness of using the ensemble method for side-channel analysis of the Ascon primitive implementation. As noted in Section 6.1, the ensemble method has only been utilized to attack AES primitive implementations. Thus, its effectiveness for other primitives is unclear. The experiments in this section demonstrate that the ensemble technique enables successful attacks on both unprotected and protected implementations of the Ascon primitive. The results confirm that the ensemble method is the most efficient technique to attack Ascon's protected implementation so far. Moreover, the success of the ensemble method attacking Ascon's unprotected implementation matches the success of a model selected through Bayesian optimization [71], again confirming that the ensemble of weaker learners can match the performance of a single model selected through an advanced hyperparameter tuning process.

6.4.1. Ascon-Unprotected

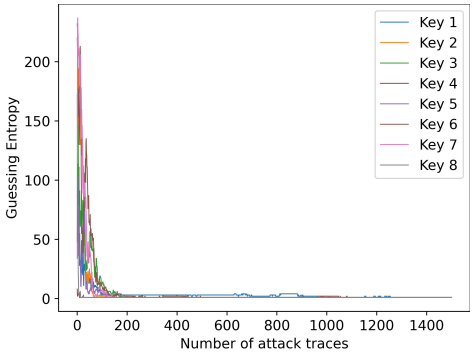
This section presents experimental results when attacking Ascon-Unprotected, an unprotected software implementation of Ascon, using the ensemble method, and compares the results with the performance of the best-found MLP and CNN models. Figure 6.1a shows the evolution of guessing entropy using the ensemble method for eight sub-keys. For each sub-key, the ensemble combines the five best MLP neural networks selected among fifty randomly generated ones. In contrast, Figure 6.1b depicts the guessing entropy evolution for the same sub-keys but employing the best-found MLP model. Observe that the reduction in guessing entropy is generally fast, though slightly slower for certain sub-keys. Figure 6.2a offers a clearer view of the impact of using the ensemble method. The effectiveness is most evident for key 3, where the number of attack traces drops from 100 to 70. However, in half of the cases, using the ensemble method slightly increased the number of attack traces. This observation is not unusual in scenarios where the problem tackled by the deep neural networks is relatively straightforward. For instance, a closer look into the performance of all randomly generated MLPs for sub-key 4 (key 4 in Figure 6.1a) shows that more than 80% of the models could reveal the key with fewer than 10 traces, indicating that the attack is relatively easy for all the generated models. Consequently, finding an optimal model for this sub-key does not need much effort, and using the ensemble method does not offer additional performance benefits.

Turning to CNNs, Figure 6.1c and Figure 6.1d show the guessing entropy evolution

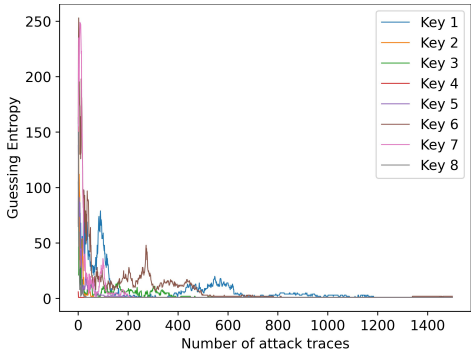


(a) Ensemble of five MLPs.

(b) The best randomly generated MLP.



(c) Ensemble of five CNNs.

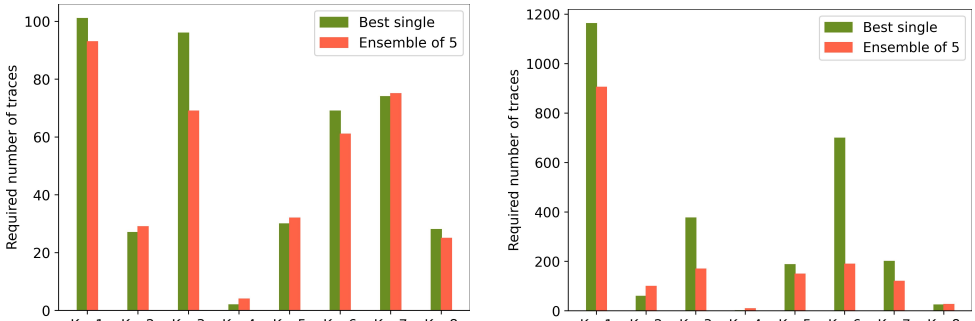


(d) The best randomly generated CNN.

Figure 6.1: Guessing entropy for Ascon-Unprotected. On top, each color shows the evolution of guessing entropy for ensemble of five MLPs (a) and the best-found MLP (b) selected from a pool with fifty randomly generated MLP for each sub-key. On the bottom, each color shows the evolution of guessing entropy for ensemble of five CNNs (c) and the best-found CNN (d) among fifty randomly generated ones for each sub-key.

for the same eight sub-keys, using an ensemble of the five best CNNs and the best-found CNN among fifty randomly generated ones for each sub-key. The ensemble's overall performance generally surpasses that of the best CNNs. However, when comparing MLP and CNN performances, it is apparent that either the best MLP or the ensemble of MLPs is typically more effective in key recovery. This observation has been mentioned in Chapter 4 as the "general ability of MLP models to find the key" and the "potential ability of CNN models to find the key". As discussed in 4.3, a limited number of MLP neural networks are more successful in reducing guessing entropy on average than the same number of CNN neural networks. Yet, with a more detailed architecture search, usually the best-found CNN outperforms the best-found MLP in key recovery.

A comparison of Figure 6.2a and Figure 6.2b reveals that the best CNN requires



(a) Unprotected-Ascon-MLP. (b) Unprotected-Ascon-CNN.

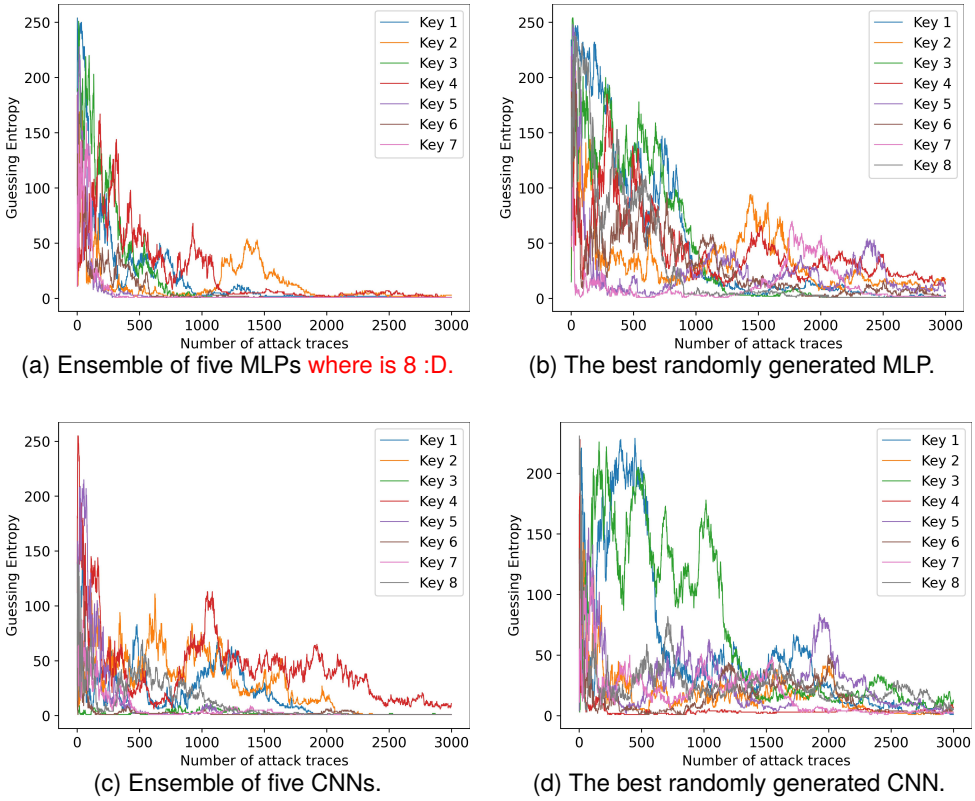
Figure 6.2: The number of attack traces with and without ensemble method in the Ascon-Unprotected dataset. On the left side, the number of attack traces using the best MLP (green) and the ensemble of five MLPs (orange) is compared. On the right side, the number of attack traces using the best CNN (green) and the ensemble of five CNNs (orange) is compared.

at least five times more traces than the best MLP to recover a key. This observation suggests that our search within the hyperparameter space detailed in Table 6.1 was not detailed enough, with no CNN model coming close to the optimal solution among the randomly generated models. However, the ensemble of CNN models could improve the attack performance compared to the best-found CNN, indicating that the ensemble is more helpful when dealing with a group of weak models rather than a group of powerful models. Comparing the results to the multi-task model on the Unprotected-Ascon dataset from previous work [71], we can see that the ensemble method with CNNs is on par with the multi-task model, recovering the key with around 1000 traces. However, the ensemble method with MLPs can recover the key with about 100 traces, which is significantly better than the multi-task model where for some sub-keys more than 1000 traces were needed.

6.4.2. Ascon-Protected

Next, we outline experimental results when attacking Ascon-Protected, a first-order protected software implementation of Ascon. The experiments in this section present a more challenging test for the efficacy of the ensemble method, particularly because the considered dataset is not easy to break [71]. Figure 6.3a illustrates the evolution of guessing entropy using an ensemble of five MLP neural networks. Figure 6.3b shows the same attack using the best-found MLP for each sub-key. Comparing these two figures shows that the reduction in guessing entropy using the ensemble method is much faster than the best-found MLP. The superior performance of the ensemble method is highlighted when analyzing the number of attack traces. Figure 6.4a compares the number of attack traces for both the ensembles and the best MLP. Clearly, the best MLP could only reveal sub-key 3 (key 3 in Figure 6.4a) and sub-key 8 (key 8 in Figure 6.4a), whereas the ensemble of MLPs successfully recovered all

the sub-keys except sub-key 2 (key 2 in Figure 6.4a).[3]



(a) Ensemble of five MLPs where is 8 :D.

(b) The best randomly generated MLP.

(c) Ensemble of five CNNs.

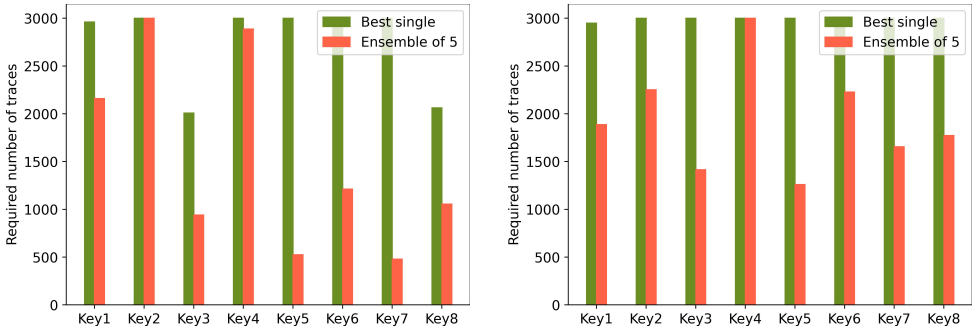
(d) The best randomly generated CNN.

Figure 6.3: Guessing entropy for Ascon-Protected. On top, each color shows the evolution of guessing entropy for ensemble of five MLPs (a) and the best-found MLP (b) selected from a pool with fifty randomly generated MLP for each sub-key. On the bottom, each color shows the evolution of guessing entropy for ensemble of five CNNs (c) and the best-found CNN (d) among fifty randomly generated ones for each sub-key.

Similar observations apply to the CNN models, as shown in Figure 6.3c and Figure 6.3d. The ensemble method allows for the reduction of all sub-keys guessing entropy to $GE = 1$, except for sub-key 4 (key 4 in Figure 6.3c), while none of the best-found CNNs in the pools of randomly generated CNN models could reduce GE to one. The stark contrast is further evident in Figure 6.4b.

Considering the results from the ensemble learning on the Ascon-Protected and

[3]This observation again emphasizes that extracting some sub-keys is more challenging than others. This difficulty stems from the difference in the amount of leakage for each sub-key. This difference in leakage can come from the architecture of the target (related to the hardware) or the implementation of the algorithm (related to the software). However, this is a common phenomenon in SCA, and to justify it, we need to get deeper into hardware and software implementations.



(a) Protected-Ascon-MLP. (b) Protected-Ascon-CNN.

Figure 6.4: The number of attack traces with and without ensemble method in the Ascon-Protected dataset. On the left side, the number of attack traces using the best MLP (green) and the ensemble of five MLPs (orange) is compared. On the right side, the number of attack traces using the best CNN (green) and the ensemble of five CNNs (orange) is compared.

Ascon-Unprotected datasets, we can conclude that the ensemble method is significantly more effective for challenging datasets, where finding optimal models is more difficult. This conclusion can be supported by the similar performance of both the ensembles and the best-found MLP model in the Ascon-Unprotected dataset and the considerably improved results using the ensemble method in the Ascon-Protected dataset. The difference in the effectiveness of using the ensemble method in these two datasets stems from the difficulty of finding optimal and sub-optimal neural network models. Since it is relatively easy to find powerful models for the Ascon-Unprotected dataset, the ensemble method does not offer much improvement. In contrast, in the case of Ascon-Protected, almost all the best-found models performed poorly. However, combining those weak models through the ensemble method could still significantly improve the attack performance.

It is worth mentioning that using an ensemble of good models is more effective compared to an ensemble of poor models (as expected). While the ensemble method can offer better performance even using poor models, combining good models provides more performance benefits [120]. One should consider that with a “good model”, we do not mean an optimal model but a sub-optimal one that can still find the key or reduce the guessing entropy to small values. In the case of the Ascon-Protected dataset, most of the best-found models in our experiments were not good enough to break the target. To find individual models with better performance, we could extend the range of the hyperparameters outlined in Table 6.1 and increase the number of models in the random models’ pool to increase the chance of finding better models.

The result from our ensemble method on the Ascon-Protected dataset significantly improved over the previous work [71], where the authors could not recover all the bits of the key with their multi-task model. We can recover the key with less than 3000 traces using the ensemble method.

6.4.3. Comparing with attacking AES:

As mentioned in Section 6.1, most of the published research in the DL-SCA domain focused on AES-based datasets. Here, we mention some similarities and differences between the side-channel analysis of AES and Ascons to make the attack easier to understand.

The most highlighted difference between attacking Ascon and AES stems from these algorithms' structure differences. AES-128 has ten rounds of S-box, shift rows, mix columns (except for the last round), and add round key, while Ascon has a sponge-based construction with initialization and finalization phases. In the AES primitive, the SCA target the first or last rounds because only these two rounds operate on a known value (plaintext in the first round and ciphertext in the last round) and the key. The structure of the Ascon algorithm is entirely different from AES, and only the initialization and finalization phases of this algorithm seem to be vulnerable against SCA. While the known values in the case of AES are plaintext or ciphertext, they are nonce and the tag in the case of Ascon. Consequently, there are fundamental differences in the S-box implementation of Ascon and AES. For instance, in the case of AES, S-box input is a combination of key and plaintext, while it is a combination of two bits of the key, two bits of the nonce, and one bit of the initial value in the case of the Ascon. However, in both cases, the S-box output seems the most effective point to attack as this point offers non-linearity. Also, divide-and-conquer is helpful in both cases.

6.5. Conclusions and Future Work

This chapter investigates the effectiveness of applying an ensemble method to attack both protected and unprotected implementations of the Ascon primitive. While the ensemble method was considered before in DL-SCA, its effectiveness for symmetric-key primitives was only validated using AES-based datasets, leading to questions about its applicability to primitives with different operational logic. This chapter demonstrated the successful application of ensemble methods to Ascon implementations as a solution to reduce the costs for finding a sufficiently good suboptimal model. Besides, using the ensemble of neural network models, we improved state-of-the-art attacks on Ascon's protected implementation, underscoring that future implementations should consider the current vulnerabilities and that stronger countermeasures are needed to prevent DL-SCA. The experimental results show that with an ensemble of (only) five neural network models, it is possible to extract the secret key with less than 3000 traces from the protected implementation and, at most, with 100 traces from the unprotected implementation both running on a widely used Arm cortex m4 microcontroller. One possible future work in this direction is using better (and more) models for the ensemble, where we stipulate it can improve the final performance even further.

The next step, can be investigating whether an ensemble of neural networks of different types (ensemble of different topologies like MLP and CNN) trained using different leakage models can improve the attack performance. The motivating intuition is that a model with a particular topology trained with the same leakage model tends

to generate less diverse predictions than models with a different topology trained with different leakage models. Indeed, when we use a dataset and a specific combination of neural network topologies and leakage models, the acquired models are less diverse and mostly focus on similar leakage (points of interest). By integrating diverse neural network types and leakage models into our ensemble, we aim to extract a richer spectrum of information from individual traces, potentially leading to more potent and efficient DL-SCA.

7

Leakage Model-flexible Deep Learning-based Side-channel Analysis

Profiling side-channel analysis has gained widespread acceptance in both academic and industrial realms due to its robust capacity to unveil protected secrets, even in the presence of countermeasures. To harness this capability, an adversary must access a clone of the target device to acquire profiling measurements, labeling them with leakage models. The challenge of finding an effective leakage model, especially for a protected dataset with a low signal-to-noise ratio or weak correlation between actual leakages and labels, often necessitates an intuitive engineering approach, as otherwise, the attack will not perform well.

This chapter, introduces a deep learning approach with a flexible leakage model, referred to as the multi-bit model. Instead of trying to learn a pre-determined representation of the target intermediate data, we utilize the concept of the stochastic model to decompose the label into bits. Then, the deep learning model is used to classify each bit independently. This versatile multi-bit model can adjust to existing leakage models like the Hamming weight and Most Significant Bit while also possessing the flexibility to adapt to complex leakage scenarios. To further improve the attack efficiency, we extend the multi-bit model to profile all 16 subkey bytes simultaneously, which requires negligible computational effort. The experimental results show that the proposed methods can efficiently break all key bytes across four considered datasets while the conventional leakage models fail. Our work signifies a significant step forward in deep learning-based side-channel attacks, showcasing a high degree of flexibility and efficiency with the proposed leakage model.

7.1. Introduction

The effectiveness of deep learning in SCA stems from its impressive flexibility in identifying and characterizing leakages, then correlating these leakages with variations in the data being processed. However, the leakage may not directly correlate with the data depending on the implementations and executed hardware. The wise selection of the leakage model should reflect the feature of actual leakages, thus strengthening the links between leakage features and data. The stochastic attack estimates the significance (represented by a coefficient) of each bit [51]. Such a method efficiently extracts the feature's linear part while the non-linear part is missing [54]. A more common method is to adopt pre-defined leakage modes for all bits (such as Hamming weight or Identity) or specific bits (such as, the most or least significant bit) [76, 78, 106, 107]. However, the existing leakage models may not match the actual leakages feature considering the diversity of actual implementations, measurement setup, and leakage pre-processing techniques. Besides, they impose a degree of pre-existing conditions on bit significance, which may inadvertently stifle the flexibility of the learning process.

Stochastic models [51] could be a good candidate for adapting to the physical leakages, but they should incorporate dimension reduction techniques to reduce the computation complexity; besides, when facing a more challenging dataset or profiling dataset number is insufficient, their performance could be mediocre. We provide more details in Section 7.3.1.

In this chapter, we proposes a flexible leakage model, denoted as *multi-bit model*, that circumvents the constraints of traditional pre-defined leakage models, allowing for dynamic adjustment of the leakage model in real-time during the profiling phase of DL-SCA. Our extensive analysis underscores the potential of the multi-bit model to evolve into an optimal leakage model that fits the physical leakages, facilitating outstanding attack performance and effective leakage assessment. This flexible leakage model combined with multitask learning takes a significant step in simplifying the conventional profiling SCA process. Typically, SCA requires attacking each secret byte separately to reveal the entire secret, such as in the case of AES, where 16 separate attacks are needed. We streamline this process into a single model training session through *multi-byte multi-bit model*, which simultaneously targets all subkeys. This model stands out for its capability to branch from a primary framework into multiple sub-branches, enabling concurrent attacks on various subkeys. To the time of publication of the paper related to this chapter, this was the first time in the field where all 16 secret key bytes could be attacked simultaneously, marking a major improvement from the state-of-the-art methods that focus on individual byte attacks in computation efficiency and attack performance. Together with the multi-bit model that is free from the constraints of pre-defined models and assumptions, the proposed method is a highly promising and robust solution in DL-SCA.

The main contributions of this chapter are:

- We analyze the limitations of the existing profiling attacks, then propose the multi-bit model that allows more flexibility and gives insight into leakage assessment.

- We propose a new attack method, multi-byte multi-bit DL-SCA, that can simultaneously profile all subkeys (in our case, 16 AES subkeys). We perform case studies to showcase the analysis, which is further validated on four publicly available datasets and an advanced hardware AES implementation [121] from CHES CTF 2023.

- We provide a hyperparameter study on several relevant factors for the proposed method: data augmentation, batch size, and the number of training epochs. The results confirm the robustness of the method to diverse settings, and data augmentation is a crucial factor in mounting powerful multi-bit model-based attacks. Finally, the multi-bit model provides a novel leakage assessment method.

The source code is available in https://github.com/lichao-wu9/MMB. The experimental results in this chapter focus exclusively on AES implementations.

Related Work: The side-channel analysis domain has been immersed in the study of profiling attacks for over two decades. Chari et al. pioneered this area by introducing the concept of the template attack (TA), which is considered the most powerful approach from an information-theoretic perspective [34]. Schindler et al. proposed stochastic models, also known as linear regression-based profiling attacks, where the authors approximated the real leakage function within a suitable vector subspace [51]. To address the issue of insufficient measurements per class for TA, pooling all covariance matrices into a single one is a viable solution [122]. Choudhary and Kuhn, for example, investigated the pooled template attack and achieved performance improvements in secret recovery and computational cost [123]. A feature engineering phase is often necessary to reduce the number of points of interest, thus avoiding the need to profile with complicated templates. This can be achieved using machine learning-based feature selection [124], dimensionality reduction methods such as Principal Component Analysis (PCA) [125–127], Linear Discriminant Analysis (LDA) [128], or Sum of Squared Pairwise T-differences (SOST) [54], or a combination of these techniques [129].

The landscape of profiling attacks has evolved significantly by incorporating machine learning (ML) techniques. Initial ML methodologies incorporated techniques such as random forest [117], support vector machines [130] and naive Bayes [131]. The performance of these techniques often outperformed (or at least matched) that of the template attack and stochastic models, laying the groundwork for the advent of more complex ML approaches. The focus of the SCA community began to pivot toward deep learning in 2016, following the seminal work of Maghrebi et al. [5]. Incorporating deep learning alleviated some challenges related to countermeasures and feature engineering, yet it also introduced difficulties associated with tuning deep learning algorithms. Despite this, early research by Cagli et al. [6] and Kim et al. [35] highlighted the potential of convolutional neural networks (CNNs) in breaking protected targets. Techniques for improving attack performance using regularization were also explored [35, 39, 102]. Subsequent works [76, 106, 107] delved deeper into the design methodologies for CNNs, achieving unprecedented attack performance

on datasets secured by masking and hiding countermeasures.

Unfortunately, most of studies discussed so far employ pre-defined leakage models to label leakage traces. These may not necessarily align well with the target dataset. For example, Perin et al. failed to use the Identity leakage model to break the CHES-CTF dataset [107]. Wu et al. pointed out that the different fixed keys in the training and validation sets for the CHES-CTF dataset result in the inefficiency of the Identity leakage model [67], then incorporated label distribution into the profiling phase to address this limitation. Besides, the leakage modeling was explored with the stochastic model [51]. Stochastic models assume that the leakage function can be formed as the sum of a deterministic component and a random one. During the profiling phase, these two components of the leakage function are approximated independently. Building on this work, Zaid et al. developed a conditional variational autoencoder methodology for stochastic attacks [132]. This approach mitigates the black-box aspect of deep learning and facilitates a more straightforward process for architectural design. Recently, Zhang et al. introduced a multi-label deep learning-based SCA that treats each bit in a byte as a separate label [133]. Then use all these labels separately as a single-bit leakage model to decide about a byte of intermediate value. Their main contribution is leveraging an ensemble using multiple labels. However, the reasoning behind the application and the method of usage is different from ours. Finally, the performance improvement is insignificant (detailed in Section 7.5).

So far, the predominant focus of the research has been treating the optimization of the labeling function (leakage model) as a task distinct from training the profiling model. This leaves a research gap as the unification of the profiling model and labeling function has not been properly addressed. Moreover, the performance enhancements over methods based on pre-defined leakage models are unclear, particularly for complex datasets incorporating countermeasures. For details about DL-SCA and challenges to be addressed, we refer readers to [4].

7.2. Multi-Task Learning

Multi-task learning (MTL) is a machine learning paradigm where multiple tasks are trained in parallel using a shared model architecture. MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks [134]. The underlying assumption is that related tasks contain complementary information that can be exploited to improve generalization on individual tasks. By leveraging shared representations, MTL reduces the need for redundant parameter learning and enhances efficiency. This concept, introduced by Caruana in [134], has widely applied in deep learning across numerous application domains.

Many advantages have been counted for MTL including data amplification, attribute selection, eavesdropping, and representation bias. Data amplification refers to the reinforcement of the signal through shared features across tasks, which reduces the impact of noise. This effect facilitates attribute selection, as relevant inputs become easier to identify within cleaner feature representations. Eavesdropping describes the situation in which a difficult-to-learn feature for one task becomes accessible by

leveraging another task that learns it more readily. Representation bias arises from the multi-objective optimization process, which constrains the model to adopt more consistent and stable representations compared to single-task training.

In deep learning, MTL is commonly implemented through hard or soft parameter sharing. Hard parameter sharing, uses a shared set of hidden layers across all tasks while keeping task-specific output layers separate. This approach is computationally efficient and significantly reduces the risk of overfitting, as the shared representation must generalize across multiple tasks. In contrast, soft parameter sharing assigns each task its own model with independent parameters, while regularization techniques (e.g., L_2) are applied to encourage similarity between the parameters. This offers more flexibility than hard sharing, as it allows tasks to maintain distinct representations while still benefiting from cross-task regularization. In this chapter we use hard parameter sharing scheme. Figure 7.1 shows a visualization of hard parameter sharing.

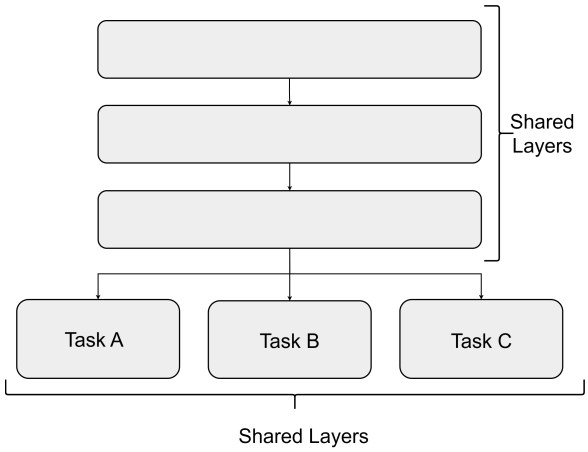


Figure 7.1: Schematic of hard parameter sharing in MLT.

7.3. Leakage Model-flexible

7.3.1. Physical Leakages Estimation

In Section 2.3 we have seen how an unknown leakage function L is estimated using numerical approximation (for instance, stochastic attacks) or hypothesized into models like Hamming weigh or Identity value of the targeted sensitive variable. We have also seen that for DL-SCA the physical leakage is hypothesize. Depending on the leakage model being used, the architecture of deep learning differs, especially the output layer. Figure 7.2 shows different characterizations of the output model, where a representation of a sub-byte (HW or ID) is attacked directly, and the bit model, where a single bit is attacked. The state-of-the-art DL-SCA relies on a good estimation of leakage models, but the current forms of DL-SCA do not attempt to

characterize/assess the leakage directly as they were constructed based on assumptions about the true leakage model, such as the HW model or ID model. For instance, DL-SCA cannot answer the question: “Is bit 2 of target data leaking?”. With an imperfect leakage model as the label of a deep learning model, one can hardly reach the optimal attack performance, meaning that the estimated leakage model used for labeling and the true leakage model in the targeted cryptographic operation match with high probability. A common practice is brute-forcing possible leakage models and selecting the best ones to report. Such an approach is problematic because of its time-consuming nature. Meanwhile, conventional profiling SCA relies on, for instance, setting a fixed key on the profiling device to assess each leakage model, potentially leading to stronger attack assumptions. Although evaluation metrics, such as loss, accuracy, or SCA metrics [32, 67, 81, 135], could be an option to assess the black-box attack, the result is not as indicative as, for instance, guessing entropy, which directly represents the attack performance [61].

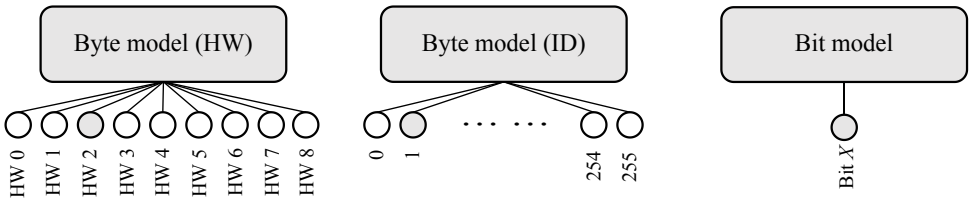


Figure 7.2: Conventional DL-SCA models.

Acknowledging the limitations of the aforementioned methods, an ideal profiling model would require two core capabilities: 1) learning both linear and non-linear aspects of leakage features and 2) enhanced flexibility in learning leakages, avoiding rigid adherence to a pre-defined leakage model. In light of these requirements, a more innate resolution emerges, which involves harnessing the potential of deep learning models. These models, noted for their adeptness in drawing out both linear and non-linear features, can be tailored to ascertain the importance of each bit independently, thus adequately fulfilling both requirements.

7.3.2. Multi-bit Model

The multi-bit model disassembles a byte into separate bits, where each bit is learned individually. Formally, assuming byte m is attacked, the learning objective from Eq. (2.17) in Section 2.3, can be rewritten as:

$$\boldsymbol{\theta}_m = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i}^{N} loss(f_{\boldsymbol{\theta}}(\mathbf{t}_i), b(f(k_{i,m}, d_{i,m}))), \tag{7.1}$$

where function b binarizes an intermediate data to a finite set $\mathbb{F}_2^n = \{0, 1\}^n$; $k_{i,m}$ and $d_{i,m}$ stand for the m-th byte in a key vector $\mathbf{k}_i$ and a plaintext vector $\mathbf{d}_i$, respectively.

Therefore, Eq. (7.1) can be rewritten as:

$$\boldsymbol{\theta}_m = \begin{cases} \arg\min_{\boldsymbol{\theta}_m} \frac{1}{N}\sum_i^N L(f_{\boldsymbol{\theta}_m}(\mathbf{t}_i), b(f(k_{i,m}, d_{i,m}))[0]), \\ \arg\min_{\boldsymbol{\theta}_m} \frac{1}{N}\sum_i^N L(f_{\boldsymbol{\theta}_m}(\mathbf{t}_i), b(f(k_{i,m}, d_{i,m}))[1]), \\ \dots \\ \arg\min_{\boldsymbol{\theta}_m} \frac{1}{N}\sum_i^N L(f_{\boldsymbol{\theta}_m}(\mathbf{t}_i), b(f(k_{i,m}, d_{i,m}))[n]). \end{cases} \tag{7.2}$$

Given a probability of each bit with a leakage trace $\mathbf{t}_i$, the probability of intermediate data y can be represented by:

$$\mathbf{p}(y|\mathbf{t}_i) = \prod_j^n \mathbf{p}(b(f(k_{i,m}, d_{i,m}))[j]|\mathbf{t}_i; \boldsymbol{\theta}_m). \tag{7.3}$$

Eq. (7.2) and Eq. (7.3) are illustrated in Figure 7.3. The multi-bit model can be considered a concatenation of bit models that cover all n bits from a side-channel perspective (here, we assume $n = 8$, targeting a single byte). Thanks to its ability to learn each bit, the proposed multi-bit model embodies characteristics shared with both byte (e.g., HW and ID) and bit models (e.g., LSB and MSB) discussed in the previous section: it bears similarities to byte models in that all bits within a byte are taken into account, and akin to bit mode, the bits are treated individually.

On the other hand, the distinguishing features render the multi-bit model particularly advantageous for DL-SCA. Indeed, unlike byte models, the multi-bit model does not enforce any pre-conditions on bit importance. The multi-bit model offers flexibility to DL-SCA in learning and weighing each bit, thus, sharing more similarity to the stochastic model discussed in Section 7.3.1. It can easily adapt to any of the existing leakage models, such as the Hamming weight model, where the probability of each bit should be above 50% with the same value, or the LSB leakage model, where only $\mathbf{p}(b(k_{i,m}, d_{i,m})[0]|\mathbf{t}_i; \boldsymbol{\theta}_m)$ moving beyond 50%, while the rest remains unchanged.

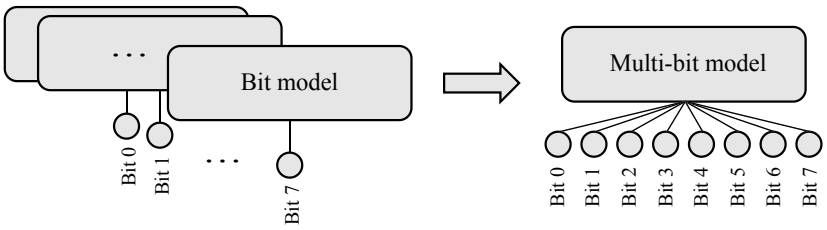


Figure 7.3: Multi-bit model.

To demonstrate the effectiveness of the multi-bit model, we present attack results

with a simulated dataset comprising different leakages following Eq. (7.4).

$$leakage = \begin{cases} Case1 : y \\ Case2 : hw(y) \\ Case3 : \sum_{i=2}^{5} b(y)[i] \\ Case4 : \prod_{i=0}^{3} b(y)[i] + \prod_{i=4}^{7} b(y)[i] \end{cases} + Z, \tag{7.4}$$

where y represents the target sensitive data $y = Sbox(d_i \oplus k^*)$, $d_i \in \mathcal{D}$; d_i and k^* denote a random plaintext byte and a fixed key byte, respectively. Function b is a binarization function. Noise Z is added to all features with $Z \sim \mathcal{N}(0, \sigma^2)$. To ensure the noise has the same effect on each test case, the leakage is normalized between 0 and 1. A total of 10 000 traces were simulated for each test case.

Template attack is used for the benchmark thanks to its interpretable nature. The multi-bit profiling model is constructed by building a Gaussian template on each bit separately[1]; the bit predictions form the probability of a byte following Eq. (7.3).

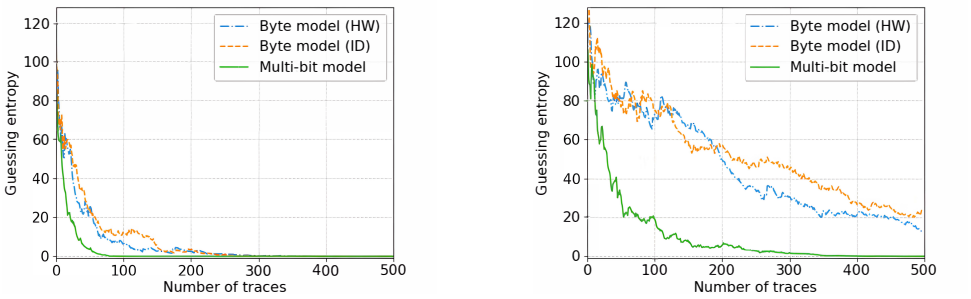
We first test the flexibility of the multi-bit model in generalizing to leakage that only leaks ID and HW (cases 1 and 2 in Eq. (7.4)). σ is set to 0.1 for the low noise setting. The accuracy of each bit is shown in Table 7.1. Aligned with our expectations, the HW leakages lead to similar accuracy of each bit, indicating the equal contribution to the actual leakage; bit_7 of the ID leakage model has the highest accuracy; the rest follows descending order. These observations confirm the ability of the multi-bit model to adjust to different leakage models. Moreover, this result confirms our claim that the commonly used leakage models, including HW and ID, have assumptions on the leakages of each bit.

Table 7.1: Bit accuracy for ID (case 1) and HW (case 2) leakages.

	bit_7	bit_6	bit_5	bit_4	bit_3	bit_2	bit_1	bit_0
ID	0.93	0.60	0.55	0.51	0.50	0.51	0.50	0.50
HW	0.61	0.60	0.60	0.60	0.60	0.60	0.60	0.61

Then, we consider a more realistic scenario for cases 3 and 4, where leakage comes from only specific bits or a combination of bits. Additionally, σ is increased to 0.4 to simulate the realistic noise effect, increasing the attack difficulties. The attack result is shown in Figure 7.4. Here, the flexibility of the multi-bit model allows it to outperform the HW and ID leakage models, with the leakage models becoming non-ideal in case 3 and case 4. Especially when the real leakage is modeled complexly, as in case 4, the performance gap between the multi-bit and byte models becomes larger. Indeed, the results thus clearly show the advantage of using a multi-bit model in adjusting and extracting complex features.

[1] Note that all bits can be modeled with one profiling model with deep learning.



(a) Case 3. (b) Case 4.

Figure 7.4: Guessing entropy for case 3 and case 4.

7.3.3. Multi-byte Multi-bit DL-SCA

From a deep learning perspective, the multi-bit model compresses multiple bit models into one model. This method aligns with multi-task learning (MTL), where multiple tasks (bit classification) are learned simultaneously. MTL is a well-studied machine learning technique that trains several learning tasks in parallel [134, 136]. Formally, given m learning tasks $\{\mathcal{T}_i\}_{i=1}^{m}$ where all the tasks or a subset of them are related, multi-task learning aims to learn the m tasks together to improve the learning of a model for each task $\mathcal{T}_i$ by leveraging information that result from the training of related tasks [134, 137].

From an SCA perspective, the side-channel leakages from a target device primarily stem from the switching activities of transistors within the integrated circuit. Considering that the modern CPU/crypto co-processor has at least an 8-bit bus width, different bit classification tasks share a common feature representation based on the original features corresponding to byte processing. multi-bit mode ensures a more powerful representation learned for all the tasks. Meanwhile, the shared representation learned by the related tasks during the training step spares the learning of redundant training parameters, improving the model's generalization performance.

Formally, Eq. (7.1) can be extended to Eq. (7.5) to attack n bytes.

$$\boldsymbol{\theta} = \begin{cases} \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_i^N L(f_{\boldsymbol{\theta}}(\mathbf{t}_i), b(f(k_{i,0}, d_{i,0}))), \\ \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_i^N L(f_{\boldsymbol{\theta}}(\mathbf{t}_i), b(f(k_{i,1}, d_{i,1}))), \\ \dots \\ \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_i^N L(f_{\boldsymbol{\theta}}(\mathbf{t}_i), b(f(k_{i,n}, d_{i,n}))). \end{cases} \tag{7.5}$$

We denote the deep learning model following Eq. (7.5) as multi-byte multi-bit DL-SCA, the corresponding leakage model is referred to as multi-byte multi-bit model (MMB). MMB provides multiple advantages. First, the profiling model can better generalize new, unseen leakages by sharing information across multiple bytes. When a profiling model learns to perform multiple tasks, it can lead to capturing common underlying patterns and features beneficial for all tasks. This characteristic makes

MMB act as a form of regularization and avoid overfitting to specific patterns or noise. Besides, It can help when individual tasks have limited data. By jointly training on multiple tasks, the profiling model can leverage the data from one task to improve its performance on another, leading to more efficient use of available data.

One can take two approaches in constructing a multi-byte multi-bit DL-SCA targeting b bytes: 1) a single model with $b * 8$ output nodes, and 2) a tree structure with a main branch and b subbranches responsible for the classification of bits in each byte. Recall that in a single n-bit multi-bit model, leakages for different bits may be found at the same time, given that in the target operation, the n bits are processed simultaneously (e.g., $n = 8$, considering a byte as the basic block). When extending to multiple bytes, we can assume that the bits of each block are processed separately (fits the target software AES implementations used in this paper). In that case, the second approach fits better, wherein a dedicated model (a subbranch) is assigned to each sub-byte to handle its bits separately. We acknowledge that the shared representation of different bytes could still exist. Thus, the main branch is introduced to extract the general features useful for all subbranches.

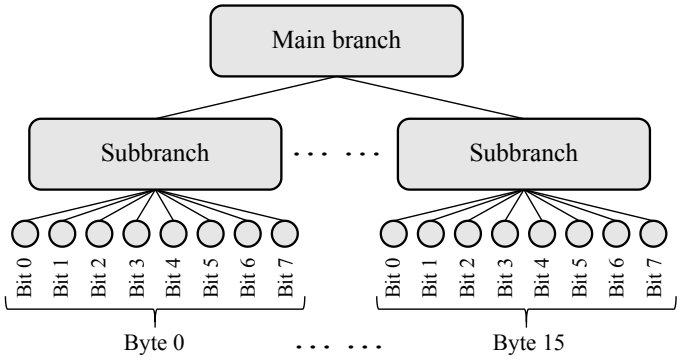


Figure 7.5: Multi-byte multi-bit DL-SCA.

The proposed architecture is shown in Figure 7.5. The main branch is responsible for leakage processing and extraction of general features. After the main branch, several multi-bit DL-SCAs construct the subbranch for each target byte. The multi-byte multi-bit DL-SCA inherits the advantages of multi-bit model, namely, the flexible leakage model. In addition, it brings several benefits. First, learning different tasks ensures that the main branch only learns useful features for all subbranches. Moreover, it is computationally efficient since the entire model has only $n * 8$ output nodes, while the model in [138] would require $n * 256$ outputs when attacking all sub-bytes. Knowing that the dimension of the output layer could influence the hyperparameter tuning of a model, our approach reduces the model size, thus increasing the learning efficiency.

Several practical aspects should be emphasized when executing real-world attacks. Like traditional DL-SCA, pre-processing leakage measurements is an indispensable step toward an effective attack. Beyond simply normalizing the data, we have identified data augmentation (Section 7.4.4) as the crucial element that drives the success

of the proposed attack. Data augmentation, used as a regularization technique, helps to deter the profiling model from focusing excessively on specific features, allowing it to concentrate instead of global features [138]. Since data leakages are confined to a few features in the realm of SCA, such methodologies can prevent the model from overfitting to non-pertinent features [139, 140]. We present a hyperparameter study on data augmentation in Section 7.4.4.

7.4. Experimental Results

This section focuses on investigating the attack performance using various leakage models. In line with Section 7.3.1, we consider HW, ID, LSB, and MSB. For the proposed methods, we include both multi-bit DL-SCA and multi-byte multi-bit DL-SCA in the benchmark. To facilitate a fair comparison and offer a comprehensive overview of the general attack performance, all 16 subkeys are attacked.

We use DL-SCA to highlight the attack capacity with the multi-bit model. For a fair comparison, the deep learning model and hyperparameters remain *constant* across all attack methods. We acknowledge that tailoring deep learning models (or hyperparameter tuning) for each dataset and method may enhance attack performance. However, this approach also introduces more variables like model complexity and training effort, which could increase the complexity of our benchmarking process significantly. According to the No Free Lunch theorem [141], the only way to know which model is best is to evaluate them all, which is impossible. As a result, when trying to see the influence of factors other than the model selection and hyperparameters, the effect of selecting optimal solutions can be neglected by picking a model that works well.

Given its excellent performance in various attack environments, we utilize a convolution neural network based on [107].[2] Since their neural network is one of the best-performing architectures based on our knowledge, we use the same neural network to attack with other leakages models, i.e., HW, ID, LSB, and MSB, for benchmarking purposes. The network structure includes two convolution blocks, each with a convolution layer (kernel numbers: 4, 32; size: 40, 8; stride: 20, 4, for each convolution layer, respectively), an average pooling layer (size: 2; stride: 2), and a batch normalization layer. This is followed by two dense layers with 32 neurons and an output layer with eight neurons. We use Scaled Exponential Linear Unit ($Selu$) for the layer activation [112], except for the final layer, which uses $Softmax$ [142] that converts a vector of n real numbers into a probability distribution of n possible outcomes. The batch size is set at 512; a hyperparameter study on its influence is given in Section 7.4.4. When the multi-byte multi-bit model is applied, the main branch includes convolution blocks, while each subbranch contains the remaining dense layers and output layers.

Regarding the training epochs (the number of iterations that allow a profiling model to adjust to input data and output labels), the DL model is trained for 200 epochs

[2]The deep learning models were implemented using Python 3.6, with the TensorFlow library version 2.6.0. Training algorithms were executed on a Nvidia GTX 1080TI GPU, managed by Slurm workload manager version 19.05.4.

for each key guess across all test cases. An evaluation of the number of training epochs can be found in Section 7.4.4. To ensure a fair comparison, we apply data augmentation to all DL-based attack methods, achieved by adding a layer right after the input layer that randomly shifts the leakage measurement within a pre-defined augmentation level (the maximum value of random shifting) of 5. We provide a detailed analysis of data augmentation in Section 7.4.4.

We use guessing entropy to evaluate the attack performance for each method. If an attack fails to break the target with the given number of attack traces, its performance is denoted with an 'x'. Otherwise, we calculate the number of attack traces to achieve a guessing entropy of one (NT). Each attack scenario is tested ten times independently to minimize the influence of random elements (e.g., random weight initialization) on the attack performance. The results are averaged to represent the general performance of an attack method.

7.4.1. Performance Evaluation

This section benchmarks the proposed methods with different pre-defined leakage models. The multi-bit and multi-byte multi-bit DL-SCA are denoted by MB and MMB, respectively. The MMB attacks 16 sub-bytes simultaneously, while the rest targets each sub-byte in sequence. The attack performance is represented by NT, the number of attack traces to each guessing entropy of one.

The performance of the proposed attack scenarios on ASCAD-F and ASCAD-R datasets indicates an effective approach to handling various leakage models. The performance across all attack scenarios is similar when attacking the first two key bytes. This is attributed to the lack of masking countermeasures on the first two bytes, allowing MMB, MB, and pre-defined leakage models to extract relevant features and break the target.

MMB and MB models significantly outperform other leakage models when dealing with Boolean masking, displaying superior efficiency in breaking all key bytes. For instance, none of the pre-defined leakage models could recover the k_{12} of ASCAD-F, but MMB and MB demonstrated remarkable capabilities by recovering it with just 219 and 59 traces, respectively. Note that our DL model for MMB is very simple. An increased DL complexity could potentially increase the attack performance. This trend is also apparent for the ASCAD-R dataset, where only MMB and MB could break the target on the same key byte (k_{12}). Only 177 and 175 attack traces are required to reach a guessing entropy of one. These observations confirm our earlier assertion in Section 7.3.2 that the proposed method can effectively handle complex leakage models. Besides, compared with LSB and MSB leakage models that solely focus on one specific bit, attacking more bits leads to significantly better performance. Due to the capability of the deep learning model to combine physical leakages of mask shares, both MMB and MB break the first-order masking effectively. Still, we note that retrieving these key bytes might be accomplished through refined deep learning architectures or increased training effort [107]. However, it is time-consuming; the tuned model may not function on different leakages. When considering real-world scenarios where the correct key is unknown, an adversary would rely on a fixed

model to attack different datasets (a common practice in the industry). In this case, MMB and MB are superior choices due to their flexibility to different leakages.

When comparing MMB and MB, their performance is comparable when attacking these two datasets. One may worry that such a simple main branch would limit the model's capability to learn so many tasks, but the results show that the features provided by the main branch of the network are sufficient for all 16 tasks. This observation suggests two conclusions. First, each sub-key shares common features and thus can be handled by MMB at once. Second, the used network still has room to be simplified when solely focusing on a single sub-byte.

Table 7.2: *NT* of each subkey for the ASCAD-F dataset.

	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
MMB	1	4	7	20	9	4	22	10	135	15	76	18	219	123	9	69
MB	1	4	5	19	9	7	23	17	185	6	37	35	59	67	9	51
HW	4	4	494	282	419	372	535	590	x	628	1 402	792	x	x	202	x
ID	2	2	x	290	66	74	2 415	111	x	126	1 272	x	x	x	23	x
LSB	17	17	1 552	43	63	81	208	57	2 683	49	138	x	x	1 084	44	61
MSB	27	33	x	1 351	1 079	983	984	866	x	623	x	517	x	x	3 544	x

Table 7.3: *NT* of each subkey for the ASCAD-R dataset.

	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
MMB	3	5	23	76	28	9	15	35	94	14	74	15	177	110	13	48
MB	2	5	23	89	23	10	17	42	80	27	71	16	145	65	15	26
HW	5	6	1 523	1 088	357	439	1 384	1 160	3 482	808	1 081	824	x	x	415	x
ID	3	2	x	x	309	59	x	x	x	x	x	x	x	x	66	x
LSB	16	11	288	118	43	55	192	98	520	39	89	116	x	391	24	54
MSB	39	60	x	x	4 128	x	1 104	x	x	1 698	x	296	x	x	1 023	x

Tables 7.4 and 7.5 show the attack results on the CHES-CTF and eShard datasets. Aligned with the previous two datasets, MMB and MB maintain outstanding performance compared with other pre-defined leakage models. Interestingly, one can observe that MB cannot retrieve all subkeys, while MMB can and performs better in attacking all keys. For instance, only MMB can recover k_4, k_8, and k_{12} of the eShard dataset, while all other methods fail. Since MMB and MB share identical main branches and subbranches (the only difference is that MMB has more subbranches for each sub-byte), aligned with the discussion in Section 7.3.3, we conclude that multi-task learning helps generalize each task, leading to robust attack performance. One should note that some attacks/profiling models could perform better than presented results, i.e., attacking different intermediate data [143] or using fine-tuned

mode [78, 107]. However, they do not influence the conclusion drawn here, as MB and MMB would also benefit from these approaches.

Table 7.4: *NT* of each subkey for the CHES-CTF dataset.

	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
MMB	89	96	58	100	227	76	48	43	114	55	50	46	3 211	50	59	35
MB	72	57	43	53	78	51	55	56	53	73	40	54	582	40	53	26
HW	2 058	x	803	1 157	x	1 369	1 715	1 121	x	3 013	1 330	4 558	x	x	1 535	1 473
ID	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
LSB	509	812	140	395	387	501	576	228	266	482	439	267	x	533	581	371
MSB	607	486	1 360	x	2 040	3 164	758	x	x	813	2 257	327	x	x	499	x

Table 7.5: *NT* of each subkey for the eShard dataset.

	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
MMB	119	504	858	248	1103	691	488	949	2 162	340	106	953	1 503	541	417	470
MB	1 173	942	908	299	x	1 706	1 894	1 085	x	876	2 420	1 459	x	1 269	1 880	1 304
HW	1 690	2 626	1 279	922	x	976	973	783	x	1 742	1 370	1 051	x	1 208	1 557	1 176
ID	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
LSB	x	x	4 592	2 321	x	x	x	4 640	x	x	x	x	x	x	x	x
MSB	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

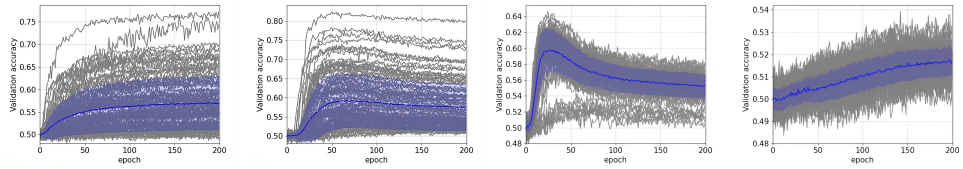
Finally, aligned with the previous observation, MB and MMB lead to better attack performance than LSB and MSB. Specifically, when attacking the eShard dataset, attacking a single bit seems non-functional; one should combine multiple bits to recover the secret. In this case, MB and MMB are the optimal choices for this task.

7.4.2. Leakage Assessment with the Multi-bit Model

Besides the key recovery, the proposed method provides insight into leakage assessment. Previous results [78, 107] show that the CHES-CTF dataset is only breakable via the HW leakage model, which our results confirm. None of the key bytes can be retrieved via the ID leakage model. Interestingly, the MSB leakage model also leads to mediocre performances. The reasoning for the poor attack performance can be explained from two perspectives. Recall in Section 7.3.2 when the simulated data has only the ID leakage, MSB is the most significant contributor to actual leakages (see Table 7.1). However, based on the attack results in Tables 7.4 and 7.5, these two datasets contains limited MSB leakages, potentially leading to the failure of the ID leakage model.

The observations can be validated with an evaluation metric. We illustrate the validation accuracy for each bit (256) of the multi-byte multi-bit model in Figure 7.6.

The mean and standard deviation are represented by the blue line and shaded blue areas, respectively. Some studies [35, 78] suggest that validation accuracy may be unreliable when working with pre-defined leakage models. However, their conclusion is drawn from the HW and ID leakage model; the bit model we used is unexplored. On the other hand, other evaluation metrics, such as precision and recall, could also evaluate each bit's prediction. However, since the two classes, 0 and 1, are balanced for each bit, validation accuracy could be considered a good metric for this task.



(a) ASCAD-F. (b) ASCAD-R. (c) CHES-CTF. (d) eShard.

Figure 7.6: Validation bit accuracy for each dataset.

As shown in Figure 7.6, for ASCAD-F and ASCAD-R, there is a sharp rise in the standard deviation, indicating a broader distribution of bit validation accuracy. The top five bits yielding the highest accuracy are either b_7 (MSB) or b_6 of a byte. This observation aligns with the results in Tables 7.2 and 7.3, indicating that the ID leakage model functions well for some bytes. On the other hand, when examining CHES-CTF and eShard, the accuracy of each bit increases relatively uniformly, as evidenced by their small standard deviation. This supports the observation of the HW accuracy made in Section 7.3.2 and also corroborates the earlier discussion about the mediocre performance of the ID model.

It is clear that the ASCAD-R and CHES-CTF datasets are prone to overfitting, a phenomenon we explore in detail in Section 7.4.4. Additionally, for the CHES-CTF dataset, one might observe that some bits emerge as 'outliers' during the early stages of training. All these outlier bits are associated with sub-byte 12, suggesting that they exhibit distinct leakage features compared to the other sub-bytes. Still, one can observe a steady increase in the validation bit accuracy of the corresponding bits, indicating that our multi-byte multi-bit model is generalized to features that all tasks can share.

7.4.3. Case Study of the SMAesH Challenge

The previous experimental results evaluate the performance of MB and MMB on 8-bit intermediate values. Next, we assess their performance on a more advanced implementation, SMAesH, used as the target for the Capture The Flag challenge in CHES 2023. SMAesH is a hardware implementation of the AES block cipher that uses masking as a countermeasure against side-channel attacks [121]. The masking technique in use, Hardware Private Circuits (HPC) [144], is a glitch-resistant hardware-specialized implementation that can compose arbitrary higher-order masking. The SMAesH dataset on Artix-7 FPGA has 2^{24} traces with random keys for

profiling and 2^{24} traces with a fixed key for the attack. There are two versions of the dataset enabling or disabling the knowledge of mask share during the profiling phase. We consider the former dataset in this section. Specifically, the plaintexts and their shares, the keys and their shares, and the random seed (the random number generator output for each cryptography operation) are given along with a simulation package that allows the generation of desired intermediate values [121].

Implementation-wise, the data is processed in 32-bit format. Besides, two intermediate shares are processed simultaneously. These characteristics complicate the leakage scenario, as a single-time sample of the trace carries information of multiple shares and bytes. In this case, using multi-task learning and the MB model is a good match. The reasons are twofold: first, learning multiple tasks simultaneously that share information across multiple intermediate values helps to avoid overfitting and learning unrelated noise patterns. Second, using deep learning and a bit model as the last layer of the neural network makes it possible to learn the complex leakage in the case of SMAesH implementation from the bit level. Specifically, aligned with the previous experimental setting, we apply two multi-bit models to attack the $Sbox$ input and output shares, respectively.[3] Then, soft analytical side-channel analysis (SASCA) [145] is applied for the key recovery.[4] The attack works under the same assumptions as profiling attacks. First, we need to profile every intermediate value we are attacking. Then, in the attack phase, we exploit the outcome probabilities for all intermediate values over multiple traces. The last step is combining the exploited information with output probabilities (acquired utilizing the profiles) employing belief propagation and factor graphs as introduced by SASCA. For the attack setting, we started by attacking 8 bits (a single byte) for each intermediate data, resulting in 16 binary outputs for each model (because we have two 8-bit shares). Then, we increased the number of attacked bits to 16, 32, and 128 key bits. Furthermore, to validate our method with different DL architectures, we employed a simple multilayer perceptron (MLP) model to attack this target.

Table 7.6: The best rank of the keys attacking the SMAesH dataset.

Attacked key size	8 bits	16 bits	32 bits	128 bits
Best key rank/total key space	$2^{5.12}/2^{8}$	$2^{9.2}/2^{16}$	$2^{17.98}/2^{32}$	$2^{66.90}/2^{128}$
Attack traces	5 000 000			

Table 7.6 shows the key rank results we reached after the attack using 5 000 000 attack traces. The key rank of the full key (128 bits) is $2^{66.9}$, which is better than the attack reported by “Morningstar-1.3” [146]. We compare our attack with theirs because it is the closest attack to our approach.[5] They employed multi-task learning with the ID leakage model to exploit the information spread over the $Sbox$ input,

[3] We have also performed attacks on unmasked intermediate data, but the performance is mediocre.

[4] SASCA includes attacking multiple cryptography operations and their input and output intermediate values simultaneously.

[5] Still, we acknowledge there are better attacks for this dataset. We refer interested readers to [121] for more details on these attacks.

Sbox output, and the transition leakage on the Sbox input wires. In their attack, they could reach the key rank of 2^{68} for recovering the whole 128 bits of the key using 5 000 000 traces. With our attack, we can reduce the key space to less than half of the key space they reached. Considering that we did not make any extra effort to optimize the neural network used to attack the SMAesH dataset, the results are surprisingly good, resulting from learning the most accurate leakage model from the bit level. One can observe a relatively lower key space when involving more intermediate data in our model. The key rank is around $2^{5.13}$ when the target is only one key byte. Assuming each byte leads to the same attack performance, the remaining key space for 16 bytes is $2^{82.08}$. Following the same assumption, we would expect that the key space was around $2^{73.60}$ when we attacked 16 bits and around $2^{71.92}$ when we attacked 32 bits. Therefore, we conclude that using multi-task learning and seeing all the outputs simultaneously is helpful for the neural network to characterize the input leakages better and extract critical and shared features that may overlap in the same time stamp.

7.4.4. Hyperparameter Study

In this section, we explore the influence of various hyperparameters on the DL model with the multi-bit model. Instead of focusing on one sub-byte, the multi-byte multi-bit DL-SCA is used for benchmarking, and all sub-bytes are considered. For a fair comparison, the attack results on 16 sub-bytes are *averaged* to demonstrate the influence of hyperparameter changes.

Data Augmentation

As discussed in Section 7.3.3, the role of data augmentation is crucial for the multi-bit DL-SCA. Consequently, we conduct a hyperparameter study specifically focusing on data augmentation, aiming to assess the impact of the augmentation level on the attack performance. The findings from this study are comprehensively presented in Table 7.7.

Table 7.7: Hyperparameter study on data augmentation (DA).

	DA-0	DA-5	DA-10	DA-20
ASCAD-F	51	46	243	1 654
ASCAD-R	137	46	88	938
CHES-CTF	274	270	428	328
eShard	x	716	395	450

When the level of data augmentation (the maximum value of random shifting) is set to zero, the multi-bit DL-SCA fails to break the eShard dataset. However, a significant performance improvement is noted with the introduction of random shifts in the datasets. Optimal results are consistently observed within the data augmentation

range of DA-5 across all test scenarios. As the level of data augmentation reaches 20, there is a notable decline in attack performance in various configurations.

From these observations, we can ascertain the critical role of data augmentation in enhancing the effectiveness of the multi-bit DL-SCA. However, employing an excessively high level of data augmentation can have adverse effects, reducing the model's attack performance. This high augmentation level can increase the complexity of fitting the model to the leakage (as the time location of the leakages becomes more random), requiring either longer training periods or larger models, thus increasing the computation effort.

Batch Size

The concept of batch size in a deep learning model pertains to the number of examples (pairings of inputs and outputs) utilized in a single training iteration. This parameter plays a significant role in shaping a deep learning model's behavior and overall performance. Notably, a discernible generalization gap exists between small and large batch sizes. Studies illustrate that smaller batch sizes can offer a regularizing effect, often resulting in superior generalization performance [147, 148]. This suggests that models trained with smaller batches may exhibit enhanced performance on unfamiliar data. As observed in Table 7.8, this aligns with our expectation that smaller batch sizes generally lead to improved attack performance in the proposed method.

Table 7.8: Study on the influence of the data size.

	BS-256	BS-512	BS-768	BS-1 024
ASCAD-F	44	46	112	177
ASCAD-R	26	45	65	79
CHES-CTF	157	270	165	190
eShard	875	715	695	673

While smaller batch sizes in deep learning models generally lead to better attack performance and superior generalization, it is crucial not to overlook the implications for computational efficiency. Larger batch sizes enable more efficient utilization of computational resources, such as GPUs, which tend to perform optimally when handling computations in larger blocks. When balancing performance against resources, one should consider that larger batch sizes can significantly reduce training time. For instance, in our experiments, a batch size of 1 024 could complete tests on all datasets four times faster than a batch size of 256 (around 8 hours). Thus, in designing and training deep learning models, careful consideration should be given to finding the optimal balance between batch size, computational efficiency, and model performance.

Training Epochs

Increasing the number of training epochs does not necessarily enhance the mapping capability of a deep learning model from input to output. On the contrary, it could diminish the model's ability to generalize on unseen datasets, a phenomenon known as 'overfitting'. Figure 7.6 clearly shows that ASCAD-R and CHES-CTF suffer from overfitting when training with 200 epochs. Table 7.9 further illustrates the performance fluctuations of the proposed method when trained with varying numbers of epochs. Indeed, training with just 50 epochs proves to be sufficient for most configurations, while an additional 150 epochs (totaling 200) yield stable attack results except the CHES-CTF, as the attack performance deteriorates with increased epoch training. Indeed, Figure 7.6c shows that training for 30 epochs results in the peak of validation accuracy for all bytes except byte 12, after which the accuracy diminishes with additional epochs. When training with 30 epochs, excluding byte 12, the required attack traces for each byte drop to an average of only 12 traces per byte for key recovery, surpassing the outcomes shown in Table 7.4. As for the eShard dataset, there is a consistent decrease in key rank value, in line with the observation in Figure 7.6d.

Table 7.9: Study on the influence of the training epoch.

	EP-30	EP-50	EP-100	EP-200	EP-300
ASCAD-F	503	137	44	46	50
ASCAD-R	88	47	40	45	46
CHES-CTF	301	119	137	270	286
eShard	x	3 889	1 081	715	761

Several strategies can be employed to mitigate overfitting. Data augmentation, as discussed in Section 7.4.4, is one effective solution. Additionally, one could apply an early stopping technique that halts model training if a monitored metric fails to increase over a certain number of epochs. Our results indicate the efficacy of validation accuracy as a metric to mitigate overfitting in the MB and MMB models.

Regarding computational efficiency, the mean training time per dataset is approximately 30 minutes. Given that both sets of 16 bytes are attacked simultaneously, the efficiency of each byte's attack (less than two minutes) is comparable with the state-of-the-art method, even with careful hyperparameter tuning [76, 79].

Our analysis highlights that hyperparameter tuning, specifically adjusting the data augmentation level, impacts the attack performance. Nonetheless, the model demonstrates resilience to hyperparameter variations, with only one hyperparameter setting (zeroing the data augmentation level) leading to a failed attack. It is important to note that we employ a single model to attack all four datasets, achieving consistent performance. This underscores the simplicity of our model's hyperparameter tuning. Furthermore, it emphasizes the robustness of the proposed multi-byte multi-bit approach, making it a reliable profiling solution across varying attack scenarios.

7.5. Discussion

The multi-bit model effectively bridges two approaches in estimating physical leakages: numerical approximation and leakage hypothesis. Typically, numerical approximations decompose the target intermediate data into bits and then estimate their coefficients. The multi-bit model follows the same approach by estimating all bits simultaneously. Integrating deep learning enables extracting non-linear features from side-channel leakages, potentially improving the estimation of physical leakages. Compared to relevant methods, such as stochastic models [51], the multi-bit leakage model can be dynamically adapted to various pre-defined leakage models during the profiling phase with different implementations. This adaptability offers enhanced flexibility, allowing the profiling model to learn more effectively from physical leakages without being constrained by suboptimal leakage model selection. Compared with [133] that profiles each bit individually, we further investigate the multi-bit model's leakage model-free characteristic. Briefly speaking, multi-label learning is familiar to the machine learning community. From our perspective, they reuse multi-label learning as a new architecture of DL-SCA with the hope of producing better results. However, from the original paper, the improvement is insignificant. We put much effort into discussing the capability of the multi-bit model to adapt to different leakage models on both the simulated and the real datasets. Based on these studies, the reason why the multi-bit model is better for SCA is well-understood. Besides, we extend the multi-bit model to the multi-byte multi-bit model, attacking all 16 bytes simultaneously. This is the first paper that can recover all key bytes at once. Another notable strategy is the One-vs-All (OvA) method, which decomposes a multi-class problem into multiple binary classification problems. For a problem with N classes, OvA establishes N distinct binary classifiers, each differentiating one class from the rest. Acharya et al. utilized the OvA method to train 256 sub-models for each key byte, aggregating these models for the final prediction [149]. However, this method assumes that each intermediate value produces distinct physical leakages that may not fit the reality. Improved label encoding, such as using Hamming Weight, could yield better OvA results. Nonetheless, potential class imbalances and high computational complexity, particularly for attacking all 16 bytes (requiring training of 256*16 models), are significant challenges. From both attack and computation perspectives, our approach demonstrates superior performance.

The multi-bit model, however, is not without its limitations. The bit decomposition of intermediate data does not guarantee that all bits can be accurately learned, leading to uncertainties in certain bits. For example, in Table 7.1, when the real leakage model is ID, bit_0 to bit_4 exhibit low accuracy, akin to random guessing. Deep learning models, which often employ gradient descent for loss function minimization, tend to learn easier features first, such as the Most Significant Bit (MSB). This preference can leave complex features, like the Least Significant Bit (LSB), less learned in the initial training stages. Gohr et al. observed similar patterns and proposed scattershot encoding to address this issue [150]. This technique involves labeling traces with the HW of random bit subsets and training multiple models on these varied encodings, ultimately recovering all bits. Although effective, this approach demands significant computational resources, particularly when attacking all 16 bytes.

Finally, the multi-bit model applies to both software and hardware implementations of symmetric cryptography. In hardware contexts, the model can adapt to scenarios where the power consumption of a circuit is influenced by the number of bit transitions in a register [151]. Since hardware crypto implementations often reuse gates for round calculations to conserve chip area [152], the proposed multi-byte multi-bit model (MMB) is also effective in attacking hardware implementations, showcased in Section 7.4.3. As mentioned, MMB can be easily adapted to different implementations. For instance, when the data bus is 32-bit, one can adapt the output of each subbranch from 8 to 32.

7.6. Conclusions and Future Work

This chapter introduces a novel multi-bit model that learns each bit separately. Unlike conventional profiling attacks, the method introduced in this chapter is adaptable to any specific leakage model, which offers increased flexibility in fitting the actual leakages. Simultaneously, multi-task learning is employed in this chapter to attack multiple sub-bytes concurrently, leading to efficient key recovery without the need to attack each byte separately. By applying this framework to four publicly available masked AES datasets, we obtain profiling attack results that significantly surpass models using pre-defined leakage models for leakage labeling. Importantly, no effort is expended in hyperparameter tuning, demonstrating its generality across different attack scenarios.

There are several potential avenues of investigation. First, the deep learning network could be enhanced, e.g., through residual networks, to strengthen the connection between the input and each task. Specifically, the shortcut could directly connect with the model's subbranch, potentially reducing the reliance on the main branch and its feature extraction capability. Second, while the current method treats each bit independently, exploring methods to reinforce inter-bit connections would be worthwhile. For instance, building an interconnection between each sub-branch could be interesting to explore. Lastly, it will be interesting to explore the capability of the proposed method in attacks without masking knowledge, e.g., testing on another version of the SMAesH dataset without mask shares.

8

Breaking the Blindfold: Deep Learning-based Blind Side-channel Analysis

Physical side-channel analysis (SCA) operates on the foundational assumption of access to known plaintext or ciphertext. However, this assumption can be easily invalidated in various scenarios, ranging from common encryption modes like Offset CodeBook (OCB) to complex hardware implementations, where such data may be inaccessible. Blind SCA addresses this challenge by operating without the knowledge of plaintext or ciphertext. Unfortunately, prior such approaches have shown limited success in practical settings.

This chapter introduces the Deep Learning-based Blind Side-channel Analysis (DL-BSCA) framework, leveraging deep neural networks to recover secret keys in blind SCA settings. In addition, it proposes a labeling method, Multi-point Cluster-based (MC) labeling, accounting for dependencies between leakage variables by exploiting multiple sample points for each variable, improving the accuracy of trace labeling. The proposed approach is validated across four datasets, including symmetric key algorithms (AES and Ascon) and a post-quantum cryptography algorithm, Kyber, with platforms ranging from high-leakage 8-bit AVR XMEGA to noisy 32-bit ARM STM32F4. Notably, previous methods failed to recover the key on the same datasets. The chapter also demonstrate the first successful blind SCA on a desynchronization countermeasure enabled by DL-BSCA and MC labeling. All experiments are validated with real-world SCA measurements, highlighting the practicality and effectiveness of the approach.

8.1. Introduction

Critical assumption underlying both profiled and non-profiled attacks is that analyzers/attackers have access to known data, such as plaintext or ciphertext. This

dependency on known input or output is foundational to most SCA techniques, but is not always valid in real-world settings where access to such data may be restricted. For example, common encryption modes like Offset CodeBook (OCB) or XTS [153] limit access to the input/output of the encryption block. Another example arises during key derivation in the EMV payment scheme, where the ciphertext cannot be recovered and only a small portion of the plaintext is known. To fully recover the session key in this context, one must use the blind setting, without access to either full plaintext or ciphertext [154].These constraints present a unique challenge for adversaries, leading to the emergence of techniques targeting implementations where input/output data is unknown—a scenario referred to as **blind SCA [155, 156]**.

In blind SCA, attackers rely exclusively on side-channel traces, captured as the device processes confidential data tied to the secret key, without access to plaintext or ciphertext. This significantly complicates the attack process, making blind SCA a compelling focus for advancing the field of side-channel research.

Despite some advancements in the last few years, most blind SCA research remains confined to simulated environments. Practical demonstrations are typically limited to platforms such as the 8-bit AVR microcontroller [155, 156] targeting AES-128 with no side-channel countermeasures. These platforms are known for their exceptionally high signal-to-noise ratio (SNR), which makes them less representative of real-world conditions. Attempts to extend blind SCA to countermeasures like masking have been limited to simulated environments, as in [156]. While Ravi et al. [154] reported blind SCA targeting Kyber on 32-bit STM32F3, it assumes a strong adversary with access to a clone device to train classifiers related to precise knowledge of secret inputs and trace leakage samples. Furthermore, all practical implementations tested to date have not been hardened against SCA.

Ref.	Board	Target
[155]	8-bit AVR ATMega	AES-128
[156]	8-bit AVR ATMega	AES-128
[154]	32-bit STM32F3	Kyber (profiled)
This work	8-bit AVR XMEGA	AES-128
This work	8-bit AVR XMEGA	Protected AES-128 (desynchronized)
This work	32-bit STM32F3	Kyber
This work	32-bit STM32F4	Ascon

Table 8.1: Blind SCA works with practical demonstrations.

This chapter addresses key limitations in existing blind SCA techniques by employing a deep learning-based approach. In this chapter we will see blind SCA across a broad range of platforms, cryptographic algorithms, and desynchronization countermeasures, validating all attacks with real-world measurements and significantly advancing the practical applicability of blind SCA. This is compared with prior works in Table 8.1. More precisely, blind SCA's main challenge lies in inferring labels for

each measurement, as it does not rely on known plaintext or ciphertext. We model this as a deep learning problem with noisy labels and show that deep neural networks effectively identify the underlying distribution of these measurements, outperforming traditional techniques in blind SCA performance.

To emphasize the real-world applicability, we consider various cryptographic algorithms: including AES, a widely used encryption standard; Ascon, the CAESAR competition winner and NIST's choice for lightweight cryptography; and Kyber, the post-quantum key encapsulation mechanism selected by NIST. Moreover, we deploy the implementations on various platforms, including an ARM Cortex-M4 chip. We emphasize that ARM Cortex-M represents the highest market share among all ARM products, leading with a market share of USD 6.0 billion in 2023, projected to grow to USD 10.5 billion by 2032, reflecting its dominance in embedded applications and low-power devices [44]. Finally, ARM Cortex-M4 is the preferred platform for the NIST post-quantum cryptography competition and the associated public pqm4 library [157].

The main contributions of this chapter are as follows:

- We consider blind SCA as a deep learning problem with noisy labels, formalized under the proposed Deep Learning-based Blind Side-channel Attacks (DL-BSCA) framework.
- We introduce an efficient unsupervised labeling scheme called Multi-point Cluster-based (MC) labeling for identifying labels from side-channel traces. This is the first multivariate labeling technique proposed within the context of blind SCA.
- We validate the approach on three devices and four datasets using real measurements. Unlike prior work, which focused on low-noise simulations or 8-bit AVR ATMEGA platforms, we demonstrate that blind SCA is practical on various platforms. Notably, previous methods failed to recover the key on the same datasets.
- We present the first successful blind SCA on a desynchronization countermeasure, exploiting the MC labeling technique to utilize multiple leakage samples, also known as points of interest (PoIs).

The source code is available at https://zenodo.org/search?q=parent.id%3A15612978&f=allversions%3Atrue&l=list&p=1&s=10&sort=version

Related Work

The blind SCA framework was first introduced by Linge et al. [155], which also proposed the first labeling technique, slicing. Initially, distance-based approaches were used to compare the experimental distribution with the theoretical distributions to recover the secret key. Subsequently, Le Bouder [158] introduced a more advanced approach based on the maximum likelihood criterion. Later, Clavier and Reynaud [156] further confirmed that the maximum likelihood criterion outperformed distance-based methods in terms of efficiency. They also proposed another labeling

technique called VA labeling that leveraged variance analysis for Hamming weight estimation. This attack was practically demonstrated on an 8-bit AVR microcontroller (Arduino Uno) with a high signal-to-noise ratio running an unprotected AES implementation.

Blind SCAs are also explored on popular SCA countermeasures, masking, in [156] and [159]. Clavier and Reynaud [156] assume the same mask is used for the input and output bytes, whereas [159] extends the attack to scenarios where masks are reused across rounds but applied uniformly to all bytes within a single round. However, these settings represent weak masking schemes, as mask reuse is generally considered poor practice in secure implementations. Both studies were validated only in simulated environments, highlighting the complexity of attacking protected implementations with blind SCA (although [156] includes a practical demonstration, it relies on known plaintext, making it incompatible with the blind SCA). Moreover, all the aforementioned blind SCA approaches only exploit one single PoI. As a result, these methods are not inherently effective against hiding countermeasures such as desynchronization, which introduce additional challenges by obscuring the leakage's temporal alignment.

Recently, Ravi et al. [154] demonstrated blind SCA on a newly standardized PQC algorithm, Kyber, making it the first blind SCA on a public key cryptosystem. The attack targets the decapsulation procedure, where the secret key is manipulated. The work was validated on the STM32F3 platform but required access to a clone device. Access to a clone device was required to train Random Forest classifiers related to precise knowledge of secret inputs and precise PoI selection. In other words, although the attacker only requires side-channel traces without knowledge of input ciphertexts during the attack phase, the adversary still needs knowledge of secret and known inputs from the clone device to train the Random Forest classifier, which may not be a realistic assumption in practice.

Separately, blind SCAs have also been investigated for another cryptographic scheme, namely, authenticated encryption with associated data (AEAD). The works [160, 161] proposed theoretical blind SCA targeting the LFSR-based counter on Elephant and Sparkle. Both of these are validated solely in a simulated environment.

In contrast, the methods proposed in this paper address the major limitations of existing blind SCA by leveraging deep learning. Table 8.2 compares the work in this chapter with [155, 156].[1] The proposed approach successfully demonstrates blind SCA on various platforms, cryptographic algorithms, and countermeasures, validated through real-world experiments. This significantly enhances the practical applicability of blind SCA.

8.2. Blind Side-channel Attacks

In a blind SCA scenario, the attackers lack access to any known data (e.g., plaintext/ciphertext) and can only use the target device's side-channel measurements to deduce the key. This poses a significantly harder task than usual SCAs. We focus

[1] Other works are excluded from comparison as they were either simulations only or fully profiled.

Table 8.2: Comparison of proposed results on different datasets/devices with prior works. We highlight successful attacks in ✓ and failed attacks in ×.

	CW	CW (desync)	Kyber	Ascon
Linge et al. [155]	×	×	×	×
Clavier & Reynaud [156]	×	×	×	×
MLP + MC	×	✓	✓	✓
MLP + MC + Dropout	✓	✓	×	✓
CNN + MC	✓	✓	✓	✓
CNN + MC + Dropout	✓	✓	✓	✓

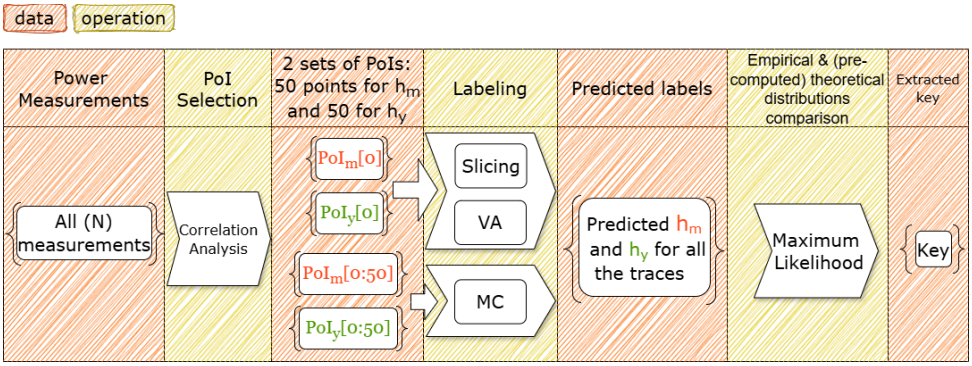


Figure 8.1: Blind Side-channel Analysis Framework.

on the work by Linge et al. [155] and Claiver et al. [156] as we consider the same scenario. Both [155] and [156] consider the blind SCA framework. This framework comprises three phases, which are discussed below. One can see the attack flow of the blind SCA framework in Figure 8.1.

Phase 1: Pre-computing theoretical joint distribution. Suppose m is a public variable (plaintext or ciphertext) while y is the sensitive intermediate variable (e.g., $y = Sbox(m \oplus k^*)$ where $Sbox$ is a Substitution Box, and k^* is the secret key). Assuming that the leakage is following the Hamming weight (HW) model, the key observation is that the joint distributions $(HW(m), HW(y))$ of the public variable and the sensitive intermediate variables are distinct for every secret key k^*. Therefore, in this phase, the theoretical joint distribution, denoted as (h^*_m, h^*_y), is computed for all key candidates. This is done by iterating through all the keys and m to count the number of times $(HW(m), HW(y))$ tuples appear. Then, we normalize the frequencies to obtain a probability distribution, $Pr((h^*_m, h^*_y)|k)$, for given hypothesis k. This can be pre-computed beforehand since it is independent

of the measurements collected from the device. For a clearer understanding of the theoretical joint distribution calculation for different targeted algorithms in this work, please refer to Section 8.4.

Phase 2: Labeling traces to obtain empirical distribution. The goal of this phase is to acquire the empirical distribution. We further divide it into two substeps for clarity.

- **PoIs Selection:** To attain the empirical distribution, one first needs to identify a suitable PoI that represents $HW(m)$ and another suitable PoI that represents $HW(y)$. Both [155] and [156] use one PoI for each targeted variable, i.e., m and y. Both works assume that the adversary can precisely locate the PoIs related to targeted intermediate variables. Consequently, finding precise PoIs is crucial as these points are used to estimate the actual Hamming weights (i.e., $HW(m)$ and $HW(y)$) and obtain the empirical joint distribution (denoted as (h_m, h_y)). We follow the assumption of prior works that the adversary can precisely locate the PoIs.[2] We achieve this by assuming that the adversary uses correlation between the measurements and the Hamming weight of the targeted variable to locate the PoIs. Accordingly, we select sample points that exhibit the highest correlation. Note that this is the only part where the public variable is required, and it is a non-blind setting. It is worth mentioning that even with this assumption in selecting PoIs, classical techniques fail to recover the secret key (see Tables 8.3 to 8.6).

- **Labeling Traces:** Next, one needs to obtain the empirical distribution using the selected PoIs. This is achieved by labeling the traces. Two labeling methods are proposed in [155] and [156]. We recall both labeling methods: Slicing labeling [155] and Variance Analysis (VA) labeling [156].
  - **Slicing labeling [155]:** Linge et al. [155] decided on the Hamming weight value, (h_m, h_y), based on the amplitude of the measurements at selected PoI, which they call slicing, thus *Slicing* labeling. The underlying assumption is that if the amplitude of the consumed power at the considered PoI is small, then the Hamming weight of the corresponding intermediate value is small.

    Let N be the number of traces to be labeled and let l_v denote the amplitude of the consumed power at the selected PoI that reflects $HW(v)$ where $v = m$ or $v = y$. They first sort the traces according to their amplitude values, l_v, in ascending order. Then, it is reasonable to assign the smallest values to the Hamming weight $h_v = 0$, then the next smallest values to $h_v = 1$, and so on. If the targeted variable has $\mathcal{B}$ bits, its corresponding Hamming

[2] [156] used the highest peak from the traces' variances along with reasonable assumptions about the implementation to locate the PoIs. However, based on our initial experiments, extracting the key was not successful using this method.

weight can take values between 0 (when all bits are 0) and $\mathcal{B}$ (when all bits are 1). To assign the correct number of traces to each Hamming weight, they fragmented the traces based on the distribution of the Hamming weight. The proportion of the different Hamming weight from 0 to $\mathcal{B}$ can be calculated using $\binom{\mathcal{B}}{h_v}$. With the assumption of a uniform distribution for cryptographic data, among the N traces, theoretically, $(\frac{N}{2^{\mathcal{B}}}\binom{\mathcal{B}}{h_v})$ of them should have the Hamming weight equal to h_v, $0 \leq h_v \leq \mathcal{B}$. The technique is applied on both l_m and l_y separately, providing the labels (h_m, h_y) for each trace.

– **VA labeling [156]:** Similar to [155], Clavier and Reynaud [156] also assume the knowledge of two suitable PoIs for $HW(m)$ and $HW(y)$. They proposed two linear regression methodologies to label the traces instead of using slicing to label their traces. However, the first method requires knowledge of the intermediate byte values, which is not practical because of the plaintext/ciphertext inaccessibility. Thus, we only recall the second linear regression known as *Variance Analysis*(VA) labeling. First, the authors assume the noisy leakage of the sample point as $l_v = \alpha_v HW(v) + \beta_v + \epsilon_v$ where α_v, β_v are constants to be determined, ϵ_v is the Gaussian noise, and v is the sensitive variable to be considered (i.e., $v = m$ or $v = y$). Hence, the variance can be written as $Var(l_v) = \alpha^2 Var(HW(v)) + Var(\epsilon_v)$. $HW(v)$ can be considered as binomial distribution $B(\mathcal{B}, p)$, with $\mathcal{B}$ being the number of bits and p being the probability of having value 1. Hence, the variance can be calculated as $\mathcal{B}p(1-p)$. Since the distribution of bits 0 and 1 is uniform, $p = 0.5$, and as such, $Var(HW(v)) = 0.25\mathcal{B}$. For an 8-bit implementation, $Var(HW(v)) = 2$. Thus, we can obtain α_v and β_v as follows:

$$\begin{aligned} \alpha_v &= \sqrt{(Var(l_v) - Var(\epsilon_v))/(0.25\mathcal{B})}, \\ \beta_v &= \mathbb{E}(l_v) - 4\alpha_v. \end{aligned} \tag{8.1}$$

Then, the estimated Hamming weight will be $h_v = \frac{l_v - \beta_v}{\alpha_v}$, from which the labels for the traces (h_m, h_y) are obtained.

Notably, these labeling methods are not designed to handle more than one PoI for each variable.

Phase 3: Comparing the empirical distribution with the theoretical joint distribution. Various methodologies were proposed to compare an empirical distribution with its theoretical joint distribution. The authors in [155] considered different metric-based comparisons like χ^2 distance, inner product, or harmonic mean. Le Bouder proposed to use the maximum likelihood criterion to compare instead [158]. The authors in [156] compared these techniques and found that the maximum likelihood criterion obtained better results. Thus, we recall the maximum likelihood criterion next.

Let (h_m^*, h_y^*) denote the true Hamming weight tuple of (m, y), while the recovered Hamming weight tuple using labeling technique is $(h_m, h_y) = (h_m^* + \epsilon_m, h_y^* + \epsilon_y)$ where ϵ_m and ϵ_y are Gaussian noise with standard deviations σ_m and σ_y, respectively. Based on the Bayes formula, the probability of the key k given a single observation (h_m, h_y) is given as

$$Pr(k|(h_m, h_y)) = \frac{Pr((h_m, h_y)|k) \cdot Pr(k)}{Pr((h_m, h_y))}. \tag{8.2}$$

The denominator $Pr((h_m, h_y))$ is a normalization term independent of the key, so we can ignore it. Moreover, $Pr(k)$ is assumed to be uniformly distributed. The probability of the key given the set of observation $((h_m, h_y)_i)_{i=1,\dots N}$ is denoted as

$$\begin{aligned} &Pr(k|((h_m, h_y)_i)_{i=1,\dots,N}) \\ &\quad = Pr((h_m, h_y)_N|k) \cdot Pr(k|((h_m, h_y)_i)_{i=1,\dots,N-1}). \end{aligned} \tag{8.3}$$

Then, by the law of total probability, we can rewrite

$$\begin{aligned} &Pr((h_m, h_y)_i|k) \\ &\quad = \sum_{h_m^*, h_y^*} Pr((h_m, h_y)_i|(h_m^*, h_y^*)) \cdot Pr((h_m^*, h_y^*)|k) \end{aligned} \tag{8.4}$$

for each i. For the second term, we use the theoretical joint distribution from before, while for the first term, we can rewrite it as the probability of its noise, which we assume follows a Gaussian distribution:

$$\begin{aligned} &Pr((h_m, h_y)_i|(h_m^*, h_y^*)) \\ &\quad = Pr(\epsilon_{m,i} = h_m^* - h_{m,i}) \cdot Pr(\epsilon_{y,i} = h_y^* - h_{y,i}) \\ &\quad = \left(\frac{1}{\sigma_m\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_{m,i}-h_m^*}{\sigma_m}\right)^2}\right) \cdot \left(\frac{1}{\sigma_y\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_{y,i}-h_y^*}{\sigma_y}\right)^2}\right). \end{aligned} \tag{8.5}$$

The values of $(h_m, h_y)_i$ are obtained by applying the labeling techniques, and are subsequently used as empirical labels for i^{th} trace in the computation of Eq. (8.5).

8.2.1. Gaussian Mixture Model

Here, we recall the well-known clustering technique called Gaussian Mixture Model (GMM) [162], which will be used in our proposed labeling technique. The GMM clustering technique is a probabilistic method that assumes the data can be represented as a combination of several Gaussian distributions. This assumption is often valid for side-channel measurements because the noise in these measurements can typically be approximated by a Gaussian distribution [34].

To apply the GMM clustering method, we must first define the number of clusters the model should generate. In our case, this is straightforward because we know

the possible Hamming weights that the values of m and y can take for various cryptographic algorithms. The GMM clustering begins by initializing the parameters for each cluster, which include the mean, covariance, and mixing coefficients.[3] Once initialized, the model uses the Expectation-Maximization (EM) algorithm to refine these parameters iteratively.

The EM algorithm operates in two main steps.

- **Expectation Step:** Compute the probability that each data point belongs to each Gaussian distribution, given the current estimates of the model's parameters.
- **Maximization Step:** Update the parameters (mean, covariance, and mixing coefficients) to maximize the expected likelihood of the data, given these probability estimates.

We repeat both steps until the model converges or until a specified number of iterations is reached, providing a set of parameters for the GMM clustering method to generate clusters. We will use this approach in our proposed labeling method described in Section 8.3.3.

8.3. Methodology

8.3.1. Threat Model

We follow the same threat model as blind SCA presented in both [155] and [156]. The adversary has no prior knowledge about the data being processed (like plaintexts or ciphertexts) and must solely rely on SCA measurements to infer the secret key. This presents a considerably more challenging scenario compared to traditional SCAs, where each SCA trace has a known plaintext or ciphertext associated with each trace. Additionally, similar to the blind SCA framework, we assume the adversary has knowledge of the most informative PoIs. The relevant PoIs are obtained by analyzing the correlation between the measurements and the actual Hamming weight of the target variables.

8.3.2. Deep Learning-based Blind Side-channel Attack

Within the blind SCA framework, the main problem is to label the traces (Phase 2 of Figure 8.1). If the traces are labeled correctly, the attack will be successful. But if there are too many mislabeled traces, the attack will fail. Classical blind SCA approaches typically rely on the assumption that larger Hamming weights lead to higher power consumption and use this to assign labels. In practice, though, such relationships are often obscured in raw SCA traces. This is reflected by the poor performance of classical methods (shown later in Tables 8.3–8.6). By contrast, DNNs have been shown to generalize unseen data, even with mislabeled data [163–165]. To

[3] Mixing coefficients, denoted by π_l, represent the contribution of each Gaussian distribution l to the overall mixture: $p(\mathbf{x}) = \sum_{k=l}^{L} \pi_l \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)$. Since we work with more than one PoI, we use multivariate Gaussian distributions.

harness the capabilities of DNNs in blind SCA, we propose the Deep Learning-based Blind Side-channel Analysis (DL-BSCA) framework. Our framework consists of the same three phases outlined in Section 8.2. Phases 1 and 3 remain unchanged; the modification has been introduced in Phase 2. We depict the attack flow of DL-BSCA in Figure 8.2. To label traces and obtain the empirical distribution, we divide Phase 2 into three substeps:

(a) PoI Selection.

(b) Labeling a subset of the traces and training the DNN with it.

(c) Predict the labels of remaining traces using DNN to obtain empirical distribution.

Unlike the blind SCA framework, which consists of one set of traces, DL-BSCA splits the traces into two sets of traces. One set of traces is labeled using a certain labeling technique to train the DNN, which we denote as *training traces*, while the other set of traces is passed through the trained DNN to obtain the empirical distribution, which we call *attack traces*. We denote N_t and N_a as the number of training traces and attack traces used.

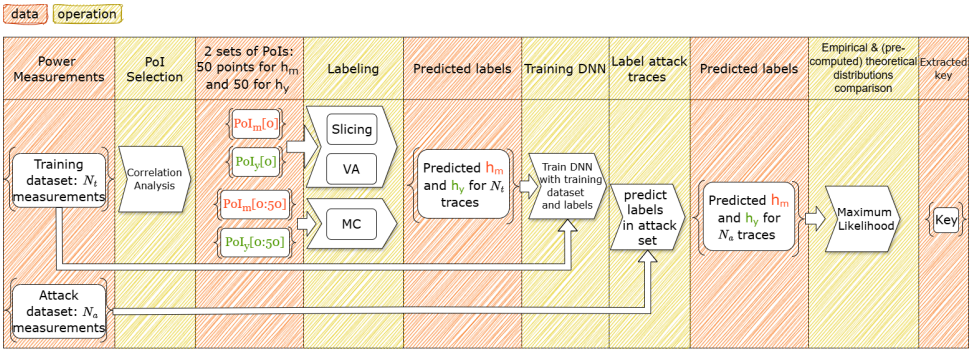


Figure 8.2: DL-BSCA Framework.

Since the labeling technique may result in many mislabeled traces, the DL-BSCA can be viewed as training a DNN with noisy labels. The issue of "noisy labels" is well-recognized in the deep learning community [165–167].[4] An interesting observation for DNNs is that they tend to learn simpler patterns first and memorize instances that do not show the straightforward relation between input features and labels later in training [163]. This behavior implies that the network can capture the core data patterns from correctly labeled data early in training, even with noisy labels.

The problem of training DNNs with noisy labels is also not completely new for SCA. In [164], Perin et al. proposed an iterative framework to improve the percentage of correct labels using accurately labeled traces slightly better than a random guess

[4] Noisy labels can originate from various sources, such as the complexity of determining accurate labels, non-expert labeling, or even adversarial manipulation.

within the context of a horizontal attack on public datasets.[5] However, in blind SCA, determining the correct measurement labels is highly challenging and often impractical.[6]

Therefore, the labeling technique we use is critical, as it determines the number of mislabeled traces within the dataset used for training the DNN. In this context, both Slicing labeling and VA labeling, which were previously proposed, can be applied. In the following, we also propose a new labeling method called MC labeling.

8.3.3. Multi-point Cluster-based Labeling

As mentioned above, the labeling technique is a key aspect of the framework. Unlike Slicing labeling and VA labeling, which consider only one PoI for $HW(m)$ and $HW(y)$ each, we use a set of PoIs to represent better $HW(m)$ and $HW(y)$ when labeling. By selecting multiple PoIs, we can better account for noise introduced by the environment and device, as each PoI reflects the same value for the target variable. We take 50 PoIs for $HW(m)$ and another 50 PoIs for $HW(y)$. These points are selected as the top 50 sample points with the highest correlation to $HW(m)$ and $HW(y)$, respectively. Our experiments show that using 50 PoIs for each variable is suitable as it provides enough information for accurate clustering despite the noise (see Section 8.5). At the same time, 50 PoIs is a manageable number, allowing us to consolidate all the information from the points and assign a unique label to each trace. We refer to the selected 50 PoIs for $HW(m)$ as PoI_m and the other selected 50 PoIs for $HW(y)$ as PoI_y. We truncate the traces into traces with 100 sample points in total (i.e., $|PoI_m| + |PoI_y| = 100$, where $|PoI_\ell|$ denotes the number of sample points in the set PoI_ℓ for $\ell = m, y$). The MC labeling is executed in two stages: **Produce Clusters** and **Provide Labels**.

Produce Clusters. In this step, the truncated traces with 100 sample points are given to a clustering technique. Our technique clusters the traces considering all the target variables at once. The number of clusters is specified with the number of bits for a target variable, $\mathcal{B}$, and the number of variables considered, n_b. Therefore, the number of clusters equals $(\mathcal{B}+1)^{n_b}$. In other words, instead of clustering the traces based on the sample points from PoI_m and PoI_y separately, we cluster the traces considering both sets of sample points from PoI_m and PoI_y together and cluster the traces with the same $HW(y)$ and $HW(m)$ into one cluster. Figure 8.3 illustrates the cluster generation step in the MC labeling process.

For example, in AES, we group traces into 81 clusters and consider both $HW(y)$ and $HW(m)$ simultaneously. This is to capture the interaction between m and y and provide a more comprehensive view of the data, which is particularly useful when the variables are interdependent (in the AES example, $y = Sbox(m \oplus k)$ is a function

[5] We conducted various experiments using the iterative framework proposed by Perin et al. [164] within the context of DL-BSCA. However, it yielded suboptimal results, likely due to the significantly higher number of mislabeled traces compared to the setup in [164].

[6] The accurate labeling when using Slicing labeling and VA labeling on CW datasets is only around 2%, while the setting in [164] has a dataset with 52% accuracy.

of m and the secret key). If we cluster the traces based on the sample points from PoI_m and PoI_y separately, we will lose information on the relation between m and y. Therefore, we consider both PoI_m and PoI_y together (a total of 100 sample points) when applying the clustering technique.

In our work, we use GMM as the clustering technique since we found it the most successful compared to other clustering techniques. We use the *scikit-learn* library when applying the GMM for clustering. Then, we use the *predict* function to obtain the clusters of traces.

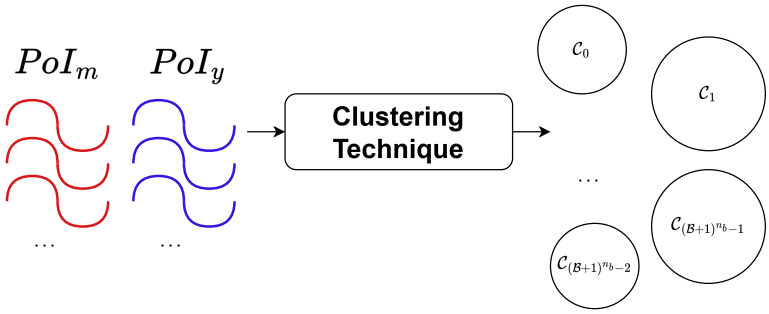


Figure 8.3: Pictorial illustration of the MC labeling step to produce clusters.

Provide Labels. The clustering processes group traces based on similarities, but the clusters still lack labels. Thus, we must map each cluster $\mathcal{C}$ to an associated label, $(Y_{\mathcal{C}}^{(m)}, Y_{\mathcal{C}}^{(y)})$. $(Y_{\mathcal{C}}^{(m)}, Y_{\mathcal{C}}^{(y)})$ will represent the label (h_m, h_y).

We provide the label of each cluster based on the steps:

- We first compute the 'center' of each cluster. Let $CT_{\mathcal{C}} \in \mathbb{R}^{|PoI_m|+|PoI_y|}$ be the center of cluster $\mathcal{C}$. $CT_{\mathcal{C}}$ can be attained by averaging each PoI of all the traces within the cluster $\mathcal{C}$. Formally, we have

$$CT_{\mathcal{C}}[i] = \frac{1}{N_{\mathcal{C}}} \sum_{t=0}^{N_{\mathcal{C}}-1} trace_{\mathcal{C}}[t, i] \tag{8.6}$$

  where $trace_{\mathcal{C}}[t, i]$ is the i^{th} sample point of the t^{th} trace from the cluster $\mathcal{C}$ and $N_{\mathcal{C}}$ is the total number of traces in cluster $\mathcal{C}$.

- We split the PoIs of $CT_{\mathcal{C}}$ based on PoI_m and PoI_y, denoted $CT_{\mathcal{C}}^{PoI_m}$ and $CT_{\mathcal{C}}^{PoI_y}$, respectively. Now, suppose there are M different clusters. We define

$$\begin{aligned} CT^{PoI_m}[i] &= \{CT_{\mathcal{C}_j}^{PoI_m}[i] | j \in \{0, \ldots, M\}\} \text{ and} \\ CT^{PoI_y}[i] &= \{CT_{\mathcal{C}_j}^{PoI_y}[i] | j \in \{0, \ldots, M\}\} \end{aligned} \tag{8.7}$$

where $CT_{\mathcal{C}_j}^{PoI_\ell}[i]$ denote the i^{th} sample point within the PoI_ℓ portion of the center trace $CT_{\mathcal{C}_j}$ for $\ell = m$ or y.

We describe the methodology for CT^{PoI_m} to obtain $Y_{\mathcal{C}_j}^{(m)}$ for all clusters $\mathcal{C}_j$. The technique can be analogously applied to CT^{PoI_y} to attain $Y_{\mathcal{C}_j}^{(y)}$ for all clusters $\mathcal{C}_j$.

- For each sample point i in PoI_m, we label $CT_{\mathcal{C}_j}^{PoI_m}[i]$ of all $\mathcal{C}_j$ when applying the slicing labeling on the $CT^{PoI_m}[i]$. This yields a collection of possible labels for a $CT_{\mathcal{C}_j}^{PoI_m}$, one from each sample point. More precisely, for each $\mathcal{C}_j$, we have $|PoI_m|$ number of possible labels. This is depicted on the left side of Figure 8.4.
- To obtain one label for the cluster, we apply the weighted majority voting to obtain the overall label $Y_{\mathcal{C}_j}^{(m)}$ for all clusters $\mathcal{C}_j$. From the collection of possible labels, we apply the weighted majority voting as follows:

$$Y_{\mathcal{C}_j}^{m} = \arg max_{h \in \{0,\dots,\mathcal{B}\}} \left(\frac{|CT_{\mathcal{C}_j}^{PoI_{m,h}}|}{\binom{\mathcal{B}}{h}} \right) \tag{8.8}$$

with $\mathcal{B}$ being the number of bits for a target variable, h being the corresponding Hamming weight, and $|CT_{\mathcal{C}_j}^{PoI_{m,h}}|$ being the number of sample points in $CT_{\mathcal{C}_j}^{PoI_m}$ labeled as the Hamming weight h through slicing labeling. We consider the weighted version because the nature of the Hamming weight and the slicing labeling causes an imbalance within the collection of labels. Indeed, the proportion of appearance of, e.g., Hamming weights 4 and 5 is higher than the Hamming weights 0 and 8 for AES [61]. To compensate for that, we assign more weight to the extreme values of Hamming weights 0 and 8. Thus, we prevent the more occurring Hamming weights from dominating the decision-making process. The weights corresponding to each Hamming weight are calculated using $\binom{\mathcal{B}}{h}^{-1}$.

- We repeat the same steps 8.3.3 and 8.3.3 for $CT_{\mathcal{C}_j}^{PoI_y}$ to obtain $Y_{\mathcal{C}_j}^{(y)}$. Lastly, we set all the traces within the cluster $\mathcal{C}_j$ to the same label $(h_m, h_y) = (Y_{\mathcal{C}_j}^{(m)}, Y_{\mathcal{C}_j}^{(y)})$.

We illustrate steps 8.3.3 and 8.3.3 in Figure 8.4.

The MC labeling introduced here predicts the Hamming weight for two variables, h_m and h_y (e.g., AES and Kyber in Section 8.4). However, the MC labeling technique can be generalized to predicts the Hamming weight for three or more variables, as shown for Ascon in Section 8.4.3. The proposed MC labeling can be used directly with the previously proposed blind SCA [155, 156] to label all the traces and obtain the empirical distribution. Alternatively, MC labeling can be combined with DL-BSCA, where MC labeling is used to label the traces for training the DNN.

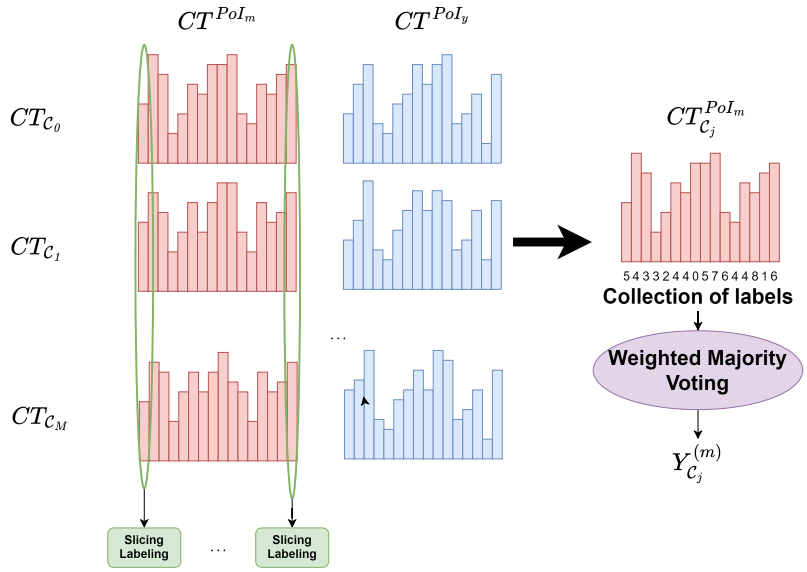


Figure 8.4: Pictorial illustration of MC labeling steps 8.3.3 and 8.3.3 to provide label $Y^{(m)}_{\mathcal{C}}$.

8.4. Experimental Results

I this section we will see the DL-BSCA on four datasets. The datasets are ChipWhisperer (CW), Reference-PPM, and Ascon-Unprotected, which are introduced in Section 2.6.3 and ChipWhisperer-desynchronized that we introduce here. We use Guessing Entropy (*GE*) and the number of attack traces (*NT*) as the metrics to report the efficincy of the attack.

8.4.1. ChipWhisperer

Joint Distribution Variables. For AES, the attack point for the DL-BSCA is the Sbox output of the first round, as it represents the nonlinear part of the algorithm. Consequently, the two interesting variables to build the joint distribution in the AES primitive are the plaintext, m, and the Sbox output, $y = Sbox(m \oplus k^*)$. Using these two variables, we estimate the joint distribution of $(HW(m), HW(y))$ to carry out the attack on AES. Algorithm 1 shows how to calculate the theoretical joint distributions of the interesting variables considered for AES.

Dataset. The dataset consists of 10 000 traces with fixed key. Each trace includes 5000 sample points. We use $N_t = 8,000$ traces for labeling and training the DNN and $N_a = 2,000$ for the attack. The reported values for the guessing entropy are the average values over repeating the attack 100 times using 1, 700 random traces

Algorithm 1 $(HW(m), HW(y))$ Joint Distribution Calculation for AES

1: **for** fixed k, $k \in \{0, \ldots, 255\}$ **do**
2: **for** each m, $m \in \{0, \ldots, 255\}$ **do**
3: Calculate $y = Sbox(k \oplus m)$
4: Calculate $HW(m)$ and $HW(y)$
5: Record occurrence of $(HW(m), HW(y))$ tuple
6: **end for**
7: Count the frequency of each tuple $(HW(m), HW(y))$
8: Divide by the total number of observations
9: Save values obtained in line 8 as expected theoretical joint distribution while using key k to be used later
10: **end for**

from the attack set.[7] As for the classical blind SCA, we use $N = 8,000$ traces to label and attack.

Experimental Results for the CW Dataset. First, we applied the method from Linge et al. [155] and Clavier and Reynaud [156] on the CW dataset and could not recover the secret key despite the relatively high SNR (see Figure 8.5). This shows previous attacks are not necessarily practical even with higher SNR. We also applied MC labeling on its own without using DNN and observed that the secret key could not be retrieved.

Then, we employ the DL-BSCA framework with various architectures with or without dropout and different labeling techniques. Table 8.3 provides the overall NT and GE obtained in the various tested scenarios. Figures 8.6 and 8.7 illustrate GE for MLP and CNN, respectively. We observe that when using the DL-BSCA framework, GE converges below 10 for all cases except for the CNN with MC labeling. This shows the effectiveness of DL-BSCA compared to previous works. In fact, for each labeling technique, when used together with a DNN, it allows us to achieve $GE = 1$ for at least one scenario (DNN + {Slicing, VA, MC + Dropout}).

The best results are obtained when we apply DL-BSCA with VA labeling; we achieve NT value of 901.

8.4.2. Kyber

Joint Distribution Variables. As introduced in Section 2.2, one of the common attack points to extract the secret key of Kyber is pair-pointwise Multiplication in decapsulation procedure, where the coefficients of the u part of the ciphertext multiplies with the secret key coefficients in the NTT domain. As divide and conquer

[7] Guessing entropy measures the **average rank** of the key across multiple attacks. Since the attack set has limited traces, we randomly select a portion of these traces for each attack iteration. This approach helps make the results less dependent on specific traces and more reflective of realistic attack scenarios.

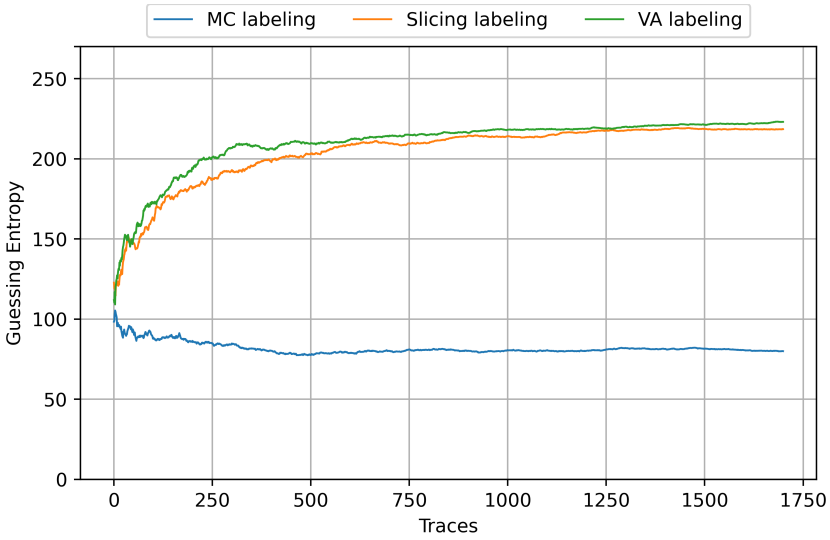


Figure 8.5: Guessing Entropy on CW dataset using classical blind SCA framework without DNN.

Table 8.3: Performance for the CW dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or NT when $GE = 1$).

	Without DNN	**CNN**	**MLP**
Linge et al. [155]	$GE = 218$	–	–
Clavier & Reynaud [156]	$GE = 223$	–	–
MC labeling	$GE = 79$	–	–
DNN + Slicing	-	$GE = 7.11$	1475
DNN + Slicing + Dropout	-	$GE = 6.14$	$GE = 0.14$
DNN + VA	-	$GE = 7.13$	901
DNN + VA + Dropout	-	$GE = 7.24$	$GE = 8.21$
DNN + MC	-	$GE = 3.05$	$GE = 15.05$
DNN + MC + Dropout	-	$GE = 1$	1455

strategy can be used to extract secret key coefficients we only target the first secret key coefficient as is shown in Eq. (8.9):

$$\begin{aligned} m_0 &= s_0 u_1 + s_1 u_0 \\ m_1 &= s_0 u_0 + s_1 u_1 \zeta, \end{aligned} \tag{8.9}$$

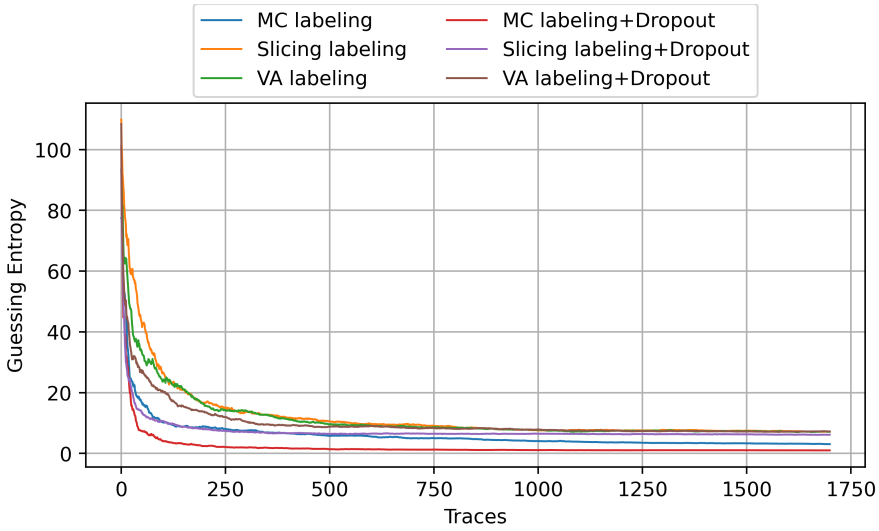


Figure 8.6: Guessing Entropy for the CW dataset with CNN.

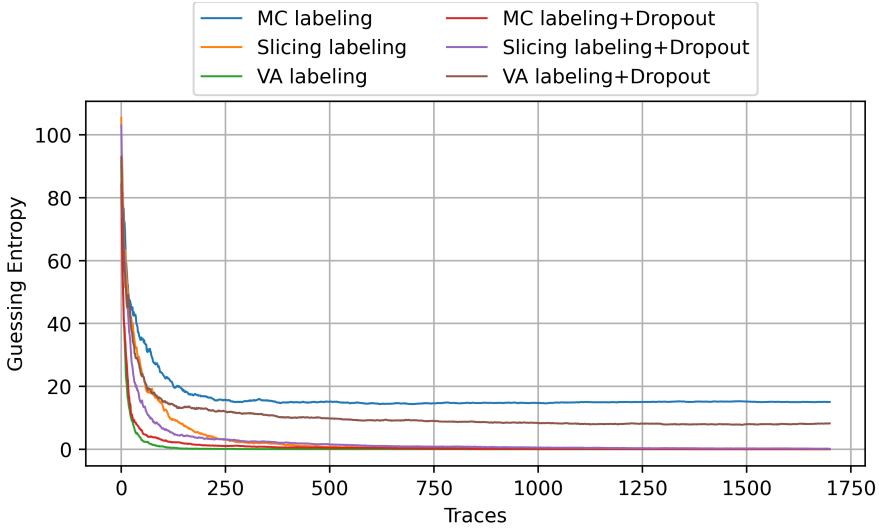


Figure 8.7: Guessing Entropy for the CW dataset with MLP.

where s_0 and s_1, and u_0 and u_1 are coefficients of first two polynomials of s and u in the NTT domain, respectively. ζ is the root of unity corresponding to the first two polynomials.

We consider $m = u_0$ and $y = s_0 u_0$ (highlighted in red in Eq. (8.9)) as the interesting variables for Kyber. The theoretical joint distribution (h_m^*, h_y^*) for two Kyber secret key coefficients differs. For example, the theoretical joint distribution

for the secret key coefficient $s_0 = 52$ differs significantly from the theoretical joint distribution for $s_0 = 2056$ as shown in Figure 8.8. This means we can compare the empirical joint distribution of $m = u_0$ and $y = s_0 u_0$ with the theoretical ones to recover the secret key coefficient, s_0.

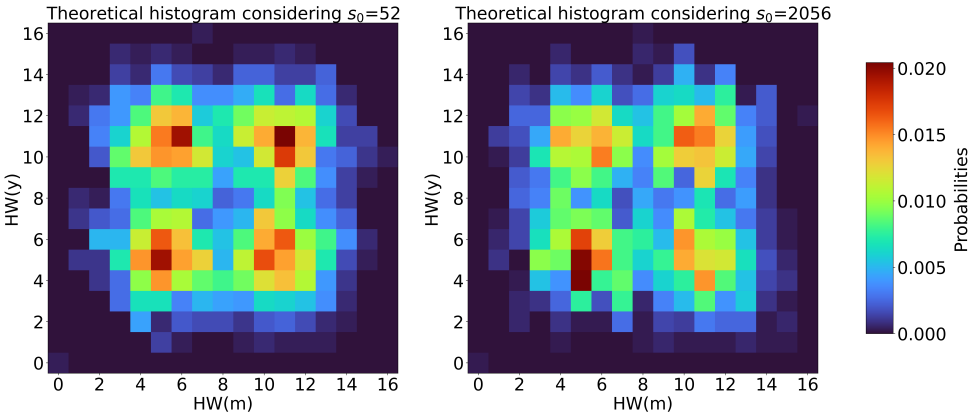


Figure 8.8: Two theoretical joint distribution of $(HW(m), HW(y))$ considering two different secret key in Kyber.

Algorithm 2 shows how to calculate the theoretical joint distributions of the two variables considered for joint distribution in Kyber.

Algorithm 2 $(HW(u_0), HW(w_0))$ Joint Distribution Calculation for Kyber

1: **for** fixed s_0, $s_0 \in \{0, \ldots, q\}$ **(With** $q = 3329$**) do**
2: **for** each u_0, $u_0 \in \{-\frac{q}{2}, \ldots, \frac{q}{2}\}$ **do**
3: Calculate $w_0 =$ reduced$(u_0.s_0)$
4: Calculate $HW(u_0)$ and $HW(w_0)$
5: Record occurrence of $(HW(u_0), HW(w_0))$ tuple
6: **end for**
7: Count the frequency of each tuple $(HW(u_0), HW(w_0))$
8: Divide by the total number of observations
9: Save values obtained in line 8 as expected theoretical joint distribution while using key s_0 to be used later
10: **end for**

Dataset. The Reference-PPM dataset used for attack is similar to the one from [154]. It is based on the reference implementation of Kyber KEM taken from *pqm4* [168] library. In the original dataset, each trace involves 50.000 time samples. To reduce the number of samples in the dataset, we used the window resampling technique from [107], which was shown to be effective in previous works. The final traces after window resampling have 10 000 samples each. We use N_t =80 000 traces for

labeling and training the neural network, and the remaining $N_a = 20\,000$ traces are used to compute the empirical distribution. The reported values for the guessing entropy are the average values over repeating the attack 100 times using 5000 random traces from the attack set. With regards to the classical blind SCA, we use $N = 80\,000$ traces to label and attack.

Experimental Results on Kyber. We first tried the classical blind SCA with various labeling techniques without using DNN. We could not recover any of the keys in these settings (see Table 8.4). Next, we run experiments with various scenarios of DL-BSCA and record the performance in Table 8.4. Figures 8.9 and 8.10 provide the performance results for CNN and MLP, respectively. Only MC labeling with DNN could bring GE to values smaller than 10. This shows that with the Kyber dataset, MC labeling results in excellent performance.

Table 8.4: Performance for the Kyber dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or NT when $GE = 1$).

	Without DNN	**CNN**	**MLP**
Linge et al. [155]	$GE = 1242$	–	–
Clavier & Reynaud [156]	$GE = 2415$	–	–
MC labeling	$GE = 1146$	–	–
DNN + Slicing	–	$GE = 165$	$GE = 255$
DNN + Slicing + Dropout	–	$GE = 57$	$GE = 58$
DNN + VA	–	$GE = 2369$	$GE = 1372$
DNN + VA + Dropout	–	$GE = 2356$	$GE = 1143$
DNN + MC	–	$GE = 2.01$	$GE = 9.2$
DNN + MC + Dropout	–	$GE = 8.04$	$GE = 16.2$

8.4.3. Ascon

Joint Distribution Variables. We take the Sbox output of the first round of the permutation as the attack point as shown in Section 2.2. Similar to [41], we consider the leakage from the first 8 bits of Y_4 as one of the variables for joint distribution. After substituting the IV, key, and nonce in the first round of permutation we have:

$$y = k_1 \& (255 \oplus IV_0 \oplus m_0) \oplus m_0 \oplus m_1, \tag{8.10}$$

where IV_0 is the first 8 bits constants (the green block in Figure 8.11), m_0 and m_1 are input nonces and k_1 is the 8 bit key we are trying to recover (orange block in Figure 8.11 on the X_1 block). As seen from eq 8.10, for Ascon, there are three variables

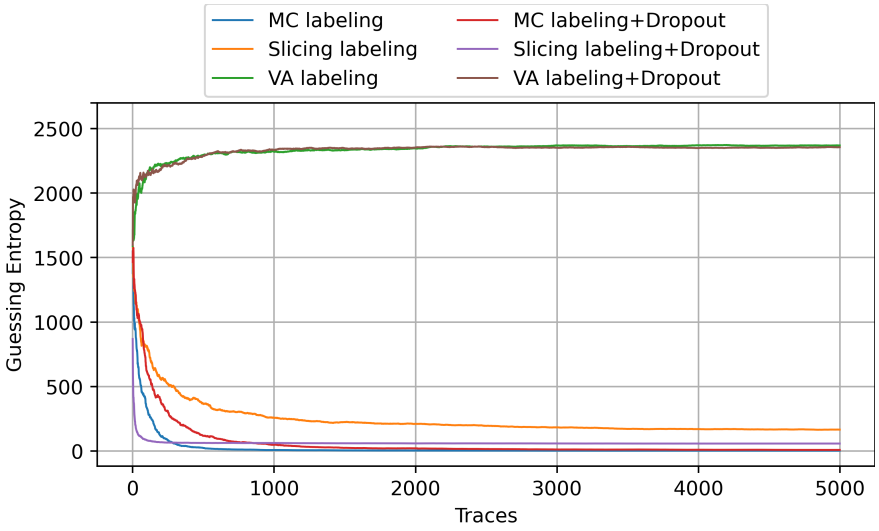


Figure 8.9: Guessing Entropy for the Kyber dataset using CNN.

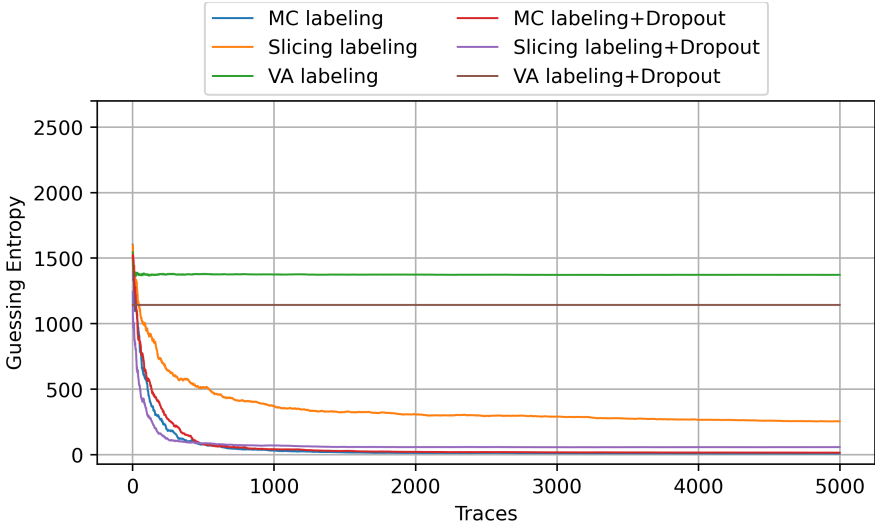


Figure 8.10: Guessing Entropy for the Kyber dataset using MLP.

for the attack point instead of two. Therefore, we consider both parts of the nonce and the Sbox output in the joint distribution, i.e., $(HW(m_0), HW(m_1), HW(y))$, for both theoretical and empirical.

Algorithm 3 shows how to calculate the theoretical joint distributions of the three interesting variables considered for Ascon.

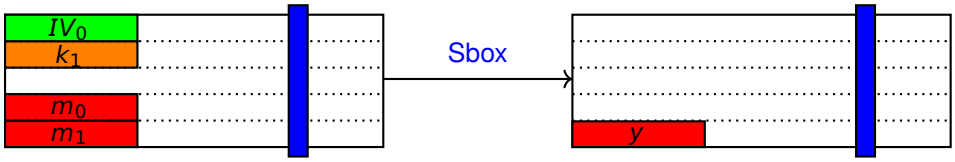


Figure 8.11: Inputs to the Ascon substitution layer and the targeted output y_4. Sbox operation takes 5 bits input in a column-wise manner, one bit from each word X_i and outputs 5 bits output with one bit from each word Y_i (highlighted in blue). The green block corresponds to the 8 bits of IV used to compute the leakage. The attack points used in our DL-BSCA are highlighted in red. Each red block corresponds to 8 bits in the state. The orange block k_1 corresponds to the 8 bits key we are trying to recover.

Algorithm 3 ($HW(m_1), HW(m_2), HW(y)$ Joint Distribution Calculation for Ascon

1: Set the initial vector $iv = [128, 64, 12, 6, 0, 0, 0, 0]$
2: **for** for each fixed k, $k \in \{0, \ldots, 255\}$ **do**
3: **for** each m_1, $m_1 \in \{0, \ldots, 255\}$ **do**
4: **for** each m_2, $m_2 \in \{0, \ldots, 255\}$ **do**
5: Calculate $y = k_1 \& (255 \oplus IV_0 \oplus m_0) \oplus m_0 \oplus m_1$
6: Calculate $HW(m_1)$, $HW(m_2)$, $HW(y)$
7: Record occurrence of triple $(HW(m_1), HW(m_2), HW(y))$
8: **end for**
9: **end for**
10: Count the frequency of each triple $(HW(m_1), HW(m_2), HW(y))$
11: Divide by the total number of observations
12: Save values obtained in line 8 as expected theoretical joint distribution while using key k to be used later
13: **end for**

Dataset. The Ascon-Unprotected dataset is introduced in Section 2.6.3. From the last 100 000 traces with fixed key in this dataset we used N_t =80 000 traces for training, while the remaining N_a =20 000 are used as attack traces. The reported values for the guessing entropy are the average values over repeating the attack 100 times using 5000 traces randomly picked from the attack set. For methods using the classical blind SCA framework, we use the $N = 80\,000$ traces.

Experimental Results for Ascon. Applying DL-BSCA on Ascon implementation is interesting because we need to extend the framework for three variables: both nonce m_0 and m_1, and the Sbox output y_4. As mentioned earlier, the theoretical and empirical joint distributions should also include HW of these three variables, and the labeling method should obtain all three variables, i.e. $(h_{m_0}, h_{m_1}, h_{y_4})$. Slicing labeling and VA labeling are extended by applying their technique on the

Table 8.5: Performance for the Ascon dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or NT when $GE = 1$).

	Without DNN	CNN	MLP
Linge et al. [155]	$GE = 77$	–	–
Clavier & Reynaud [156]	$GE = 77$	–	–
MC	$GE = 45$	–	–
DNN + Slicing	–	$GE = 81$	$GE = 49$
DNN + Slicing + Dropout	–	$GE = 12.5$	$GE = 51$
DNN + VA	–	$GE = 48$	$GE = 54$
DNN + VA + Dropout	–	$GE = 11.5$	$GE = 18$
DNN + MC (50 PoIs each)	–	$GE = 2$	$GE = 39$
DNN + MC (50 PoIs each) + Dropout	–	$GE = 19.5$	$GE = 32$
DNN + MC (7 PoIs each)	–	$GE = 5.7$	$GE = 3.5$
DNN + MC (7 PoIs each) + Dropout	–	$GE = 4$	$GE = 0.64$

additional variable. As for MC labeling, since these three variables are in bytes (8 bits), the number of clusters that the GMM clustering technique generates is $(\mathcal{B}+1)^3 = (8+1)^3 = 729$. The difference in the case of three variables compared to two variables is that we provide PoIs for all three variables at once to GMM clustering technique, and the $CT_{\mathcal{C}}$ is divided into three portions $CT_{\mathcal{C}}^{PoI_{m_0}}$, $CT_{\mathcal{C}}^{PoI_{m_1}}$, and $CT_{\mathcal{C}}^{PoI_{y_4}}$ after clustering. The rest of the technique remains the same as described earlier in Section 8.3.3.

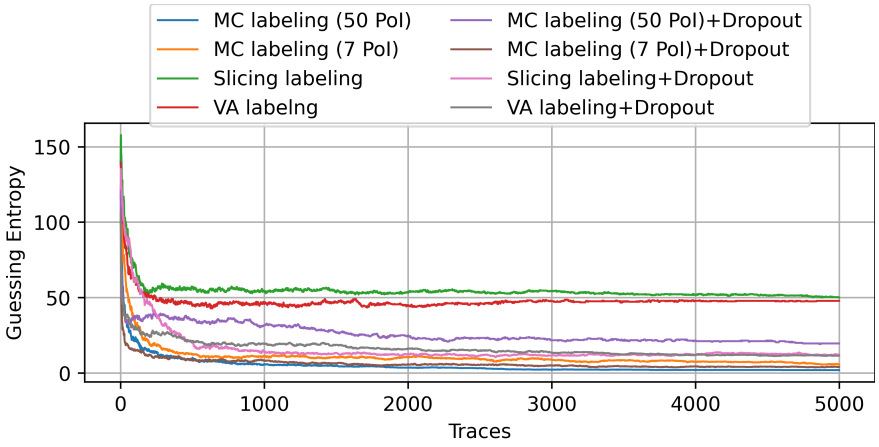


Figure 8.12: Guessing Entropy for Ascon dataset using CNN.

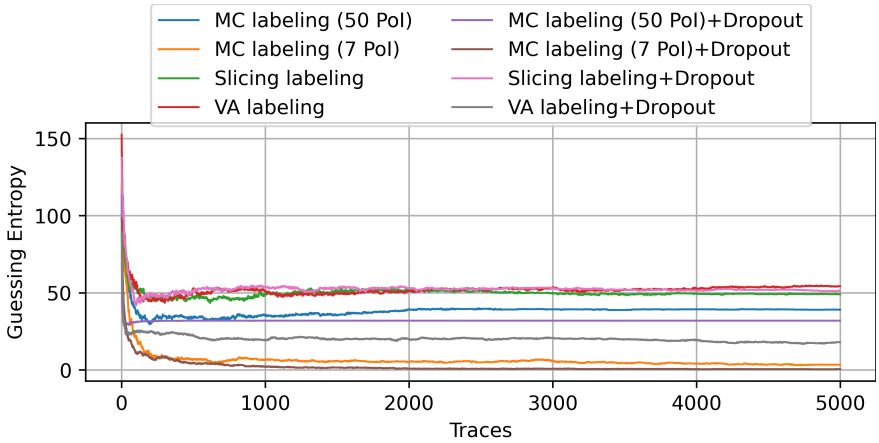


Figure 8.13: Guessing Entropy for Ascon dataset using MLP.

We applied both methodologies from [155] and [156] but were unable to recover the secret key practically (see Table 8.5). Thus, we will focus exclusively on scenarios using DL-BSCA. We run experiments for the various settings of DL-BSCA and show the results in Table 8.5, and Figures 8.12 and 8.13. We could not recover the secret key except for CNN combined with MC labeling when using DL-BSCA. This poor performance could be due to the leakage of y, m_0, and m_1 overlapping in multiple sample points, resulting in many more mislabeled traces. Figure 8.14 shows the correlation analysis of the variables involved in the attack with the traces in Ascon-Unprotected dataset.

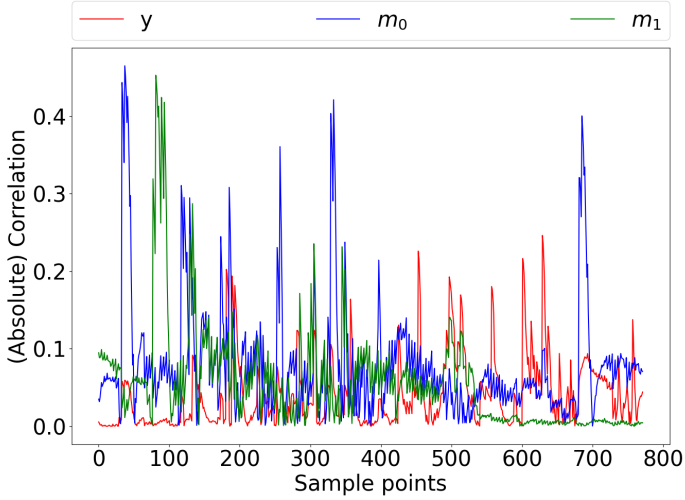


Figure 8.14: Correlation of various leakage for the Ascon dataset.

As can be seen, some points with high correlation for y are overlapped with m_0 or m_1. Examples are points around samples 200 and 500. Since the MC algorithm uses all the PoIs to decide about the *HW* of all three variables simultaneously, this overlapping can confuse the clustering technique, resulting in more mislabeled traces. This will result in very similar labels between y, m_0 and m_1 when labeling all their $CT_{C_j}^{PoI_l}[i]$ in step 8.3.3 of the MC labeling. Therefore, it resulted in more mislabeled traces.

To confirm this, we select 7 sample points for each leakage with the highest correlation to their corresponding leakages with minor overlap. We observe that *GE* decreases to under 10 for both CNN and MLP with and without dropout when we use MC labeling. This observation shows that one should consider non-overlapping PoIs when using MC labeling with DL-BSCA for key recovery. Overall, the performance improves significantly when using the DL-BSCA framework instead of just the BSCA framework.

8.4.4. Targeting Countermeasures: Desynchronized CW Dataset.

Previous blind SCA approaches selected a single PoI for each variable. Naturally, a hiding countermeasure like desynchronization, which hampers the alignment of the traces, prevents an adversary from selecting a single relevant PoI. Thus, previous blind SCA methods were never applied to desynchronization (or other hiding) countermeasures. The proposed DL-BSCA framework considers 50 PoIs per variable. Moreover, the ability of DNN to handle desynchronization is already demonstrated in prior works [6, 169, 170].

Table 8.6: Performance for the desynchronized CW dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or NT when $GE = 1$).

	Without DNN	**CNN**	**MLP**
Linge et al. [155]	$GE = 67$	–	–
Clavier & Reynaud [156]	$GE = 208$	–	–
MC labeling	$GE = 55$	–	–
DNN + Slicing	–	$GE = 45$	$GE = 20.5$
DNN + Slicing + Dropout	–	$GE = 41$	$GE = 19.5$
DNN + VA	–	$GE = 186$	$GE = 209$
DNN + VA + Dropout	–	$GE = 198$	$GE = 128$
DNN + MC	–	$GE = 2.4$	$GE = 3.5$
DNN + MC + Dropout	–	$GE = 2.1$	$GE = 1.41$

The desynchronized dataset is derived by applying random desynchronization

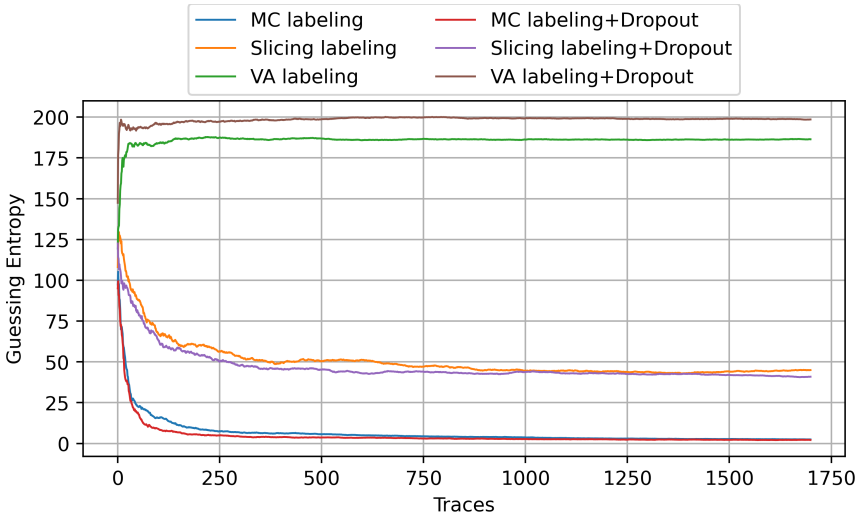


Figure 8.15: Guessing Entropy for the CW dataset protected with desynchronization using CNN models.

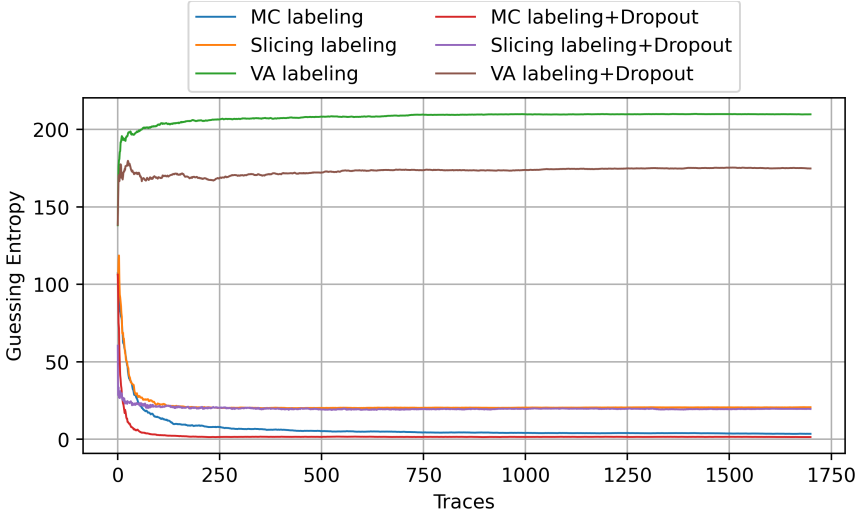


Figure 8.16: Guessing Entropy for the CW dataset protected with desynchronization using MLP models.

of up to 10 samples (in either direction) to the CW dataset. We use 8000 training traces and 2000 attack traces. The results are averaged over 100 experiments using 1700 random attack traces. Then, we run the experiments for the different settings with DL-BSCA. The results are provided in Table 8.6, and Figures 8.15 and 8.16. As before, without using DNN, we cannot recover the secret key. Furthermore, with

both Slicing and VA labeling, $GE > 10$. We can obtain $GE < 4$ in scenarios when using DL-BSCA with MC labeling. This shows the effectiveness of MC labeling with the DL-BSCA framework in recovering the secret key, even when desynchronization is used to protect the dataset. In fact, this is the first time that a viable attack on a desynchronized target is demonstrated in the realm of blind SCA, which was previously considered impossible.

8.5. Discussion

8.5.1. Accuracy

We investigate how the proportion of mislabeled data used to train the DNN affects its performance in key recovery.

MLP. We begin with completely random training labels and gradually replace a portion of them with the correct labels. Using this set of traces, we train 100 MLPs and compute the average GE of the top 10 best-performing MLPs. The results are depicted as a grey line in Figure 8.17. The trained MLPs obtained an average $GE \leq 10$ when the training labels are above approximately 15% correctly labeled. Next, we label the training traces using the various labeling techniques, achieving an accuracy of 1.2% with MC labeling, 3.3% with Slicing labeling, and 2.7% with VA labeling. Similarly, for each labeling method, we train 100 MLPs and compute their average GE for the top 10 best-performing MLPs. The results are plotted in Figure 8.17 with green for MC labeling, red for Slicing labeling, and blue for VA labeling. Notably, only the MC labeling achieved an average GE of 8, despite a low training accuracy of 1.2%. This discrepancy highlights the inadequacy of accuracy as a metric for evaluating side-channel attack effectiveness, a finding consistent with work by Picek et al. [61] and subsequent research [4, 75]. The reason is that accuracy assesses per-trace performance, whereas GE reflects the cumulative ability to recover the secret key.

We further examine the accuracy for each class for h_m and h_y when applying various labeling techniques on the training traces. As shown in Figure 8.18 for h_m and Figure 8.19 for h_y, with Slicing or VA labeling, the accuracy for marginal classes ($HW(v) = 0, 1, 7$ and 8 for $v = m, y$) is negligible. This is expected given the binomial distribution of Hamming weights, leading to a lower frequency of these marginal classes than the central classes (3, 4, 5). In contrast, MC labeling provides a more balanced representation across all classes, including the marginal ones. The presence of even a few correct examples in less frequent classes enables the DNN to learn and generalize more effectively to unseen data.

CNN. Similarly, we train 100 CNNs models and compute the average GE of the top 10 best-performing models. CNN yields similar results, as MLP. The results are shown in Figure 8.20 where MC labeling is represented in green, slicing in red, and VA labeling in blue. Notably, only the MC labeling method achieves an average $GE \leq 10$, despite labeling accuracy of just 1.2%. These results demonstrate that DNNs trained

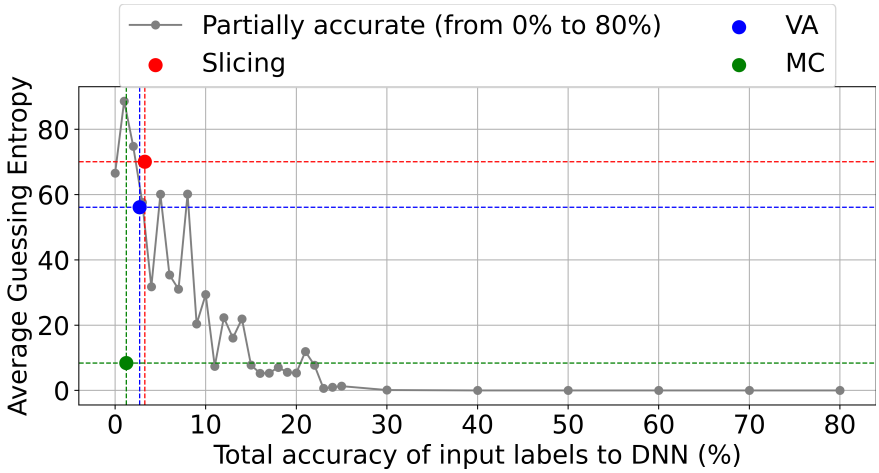


Figure 8.17: Average Guessing Entropy vs Accuracy within the training data. The average *GE* is computed over the top 10 best-performing MLPs.

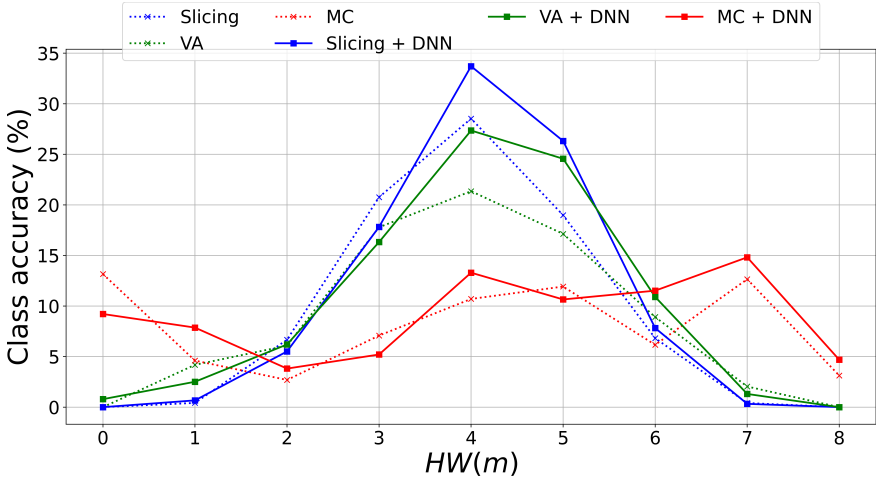


Figure 8.18: The effect of using MLP on the class accuracy of h_m. The dotted line is the class accuracy provided by the labeling technique on the training traces, and the solid line shows how the accuracy for each class changes after training the DNN and using it to evaluate the same training traces.

on data labeled using MC labeling can achieve performance comparable to models trained with randomly labeled data containing approximately 10% correct labels. This confirms the effectiveness of MC labeling in producing informative training labels that facilitate successful key recovery within the DL-BSCA framework a similar

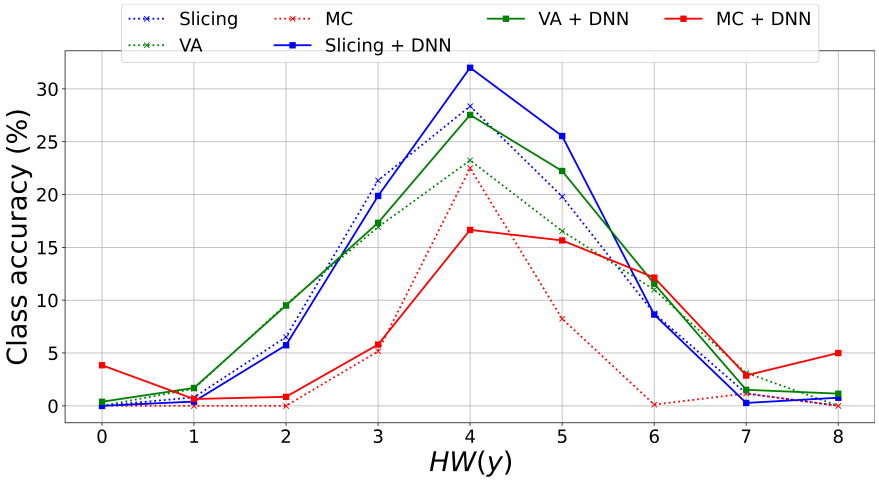


Figure 8.19: The effect of using MLP on the class accuracy of h_y. The dotted line is the class accuracy provided by the labeling technique on the training traces, and the solid line shows how the accuracy for each class changes after training the DNN and using it to evaluate the same training traces.

conclusion obtain from the experiments on MLP.

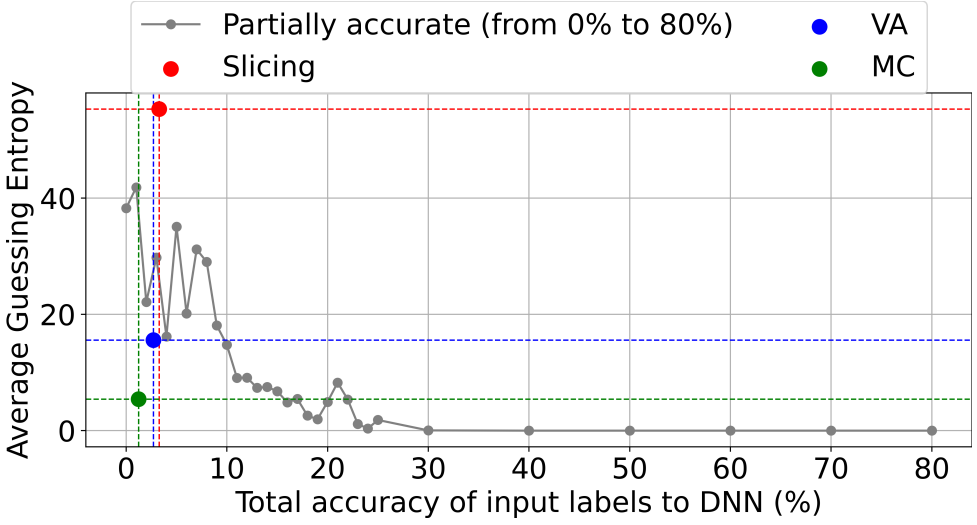


Figure 8.20: Average Guessing Entropy vs Accuracy within the training data. The average *GE* is computed for the top 10 best-performing CNNs.

8.5.2. Tuning the number of PoIs

Next, we evaluate how employing multiple PoIs can enhance the performance of DL-BSCA. For the CW dataset[8], we compare scenarios using MC labeling with a varying number of PoIs to generate training labels for DNN. For each scenario, we train 100 different models and take the top 10 best-performing DNNs to compute the average *GE*. Table 8.7 summarizes the average *GE* for CNN and MLP.

Table 8.7: Average *GE* for MC labeling for DL-BSCA with different numbers of PoIs.

Number of PoIs (per variable)	1	10	20	50	70	100	200
AVG-GE (CNN)	49	79	40	3	27	62	67
AVG-GE (MLP)	52	60	58	7	18	69	73

Tuning the number of PoIs used is crucial for successful key recovery. Using too few PoIs may fail to capture sufficient information about the targeted variables, as seen in the cases with 1, 10, and 20 PoIs. Conversely, too many PoIs, such as 70 to 200, introduce irrelevant points and deteriorate performance. In our experiments, 50 PoIs gave the best results, but this number is dataset- and model-dependent.

8.5.3. Limitations

Selection of PoIs in Blind Setting. One of the limitations is that PoIs' knowledge is a prerequisite for our and all prior works. Existing methods for PoI selection working in non-blind settings have been shown to be effective [68, 171, 172]. In our work, the PoIs are selected based on their correlation between the measurements and the actual Hamming weight of the target variables. In this step, the public variable m is required (a non-blind setting). PoI selection in full blind selection is an open challenge, and we leave it as future work.

Masking. Previous works analyze masking in a weak setting [156, 159]. Moreover, all blind SCA on masking are reported in a simulated setting. Naive application of the DL-BSCA framework to masked implementations is not feasible. The randomized shares obscure the original leakage values, rendering it impossible to produce informative labels essential for training the DNN within DL-BSCA. Therefore, attacking the masked implementation in a practical setting remains an open problem that is left for future work.

8.6. Conclusions and Future Work

This chapter proposes DL-BSCA, a novel framework for deep learning-based blind side-channel analysis. DL-BSCA harnesses the power of deep learning to effectively handle noisy datasets, addressing key challenges in blind SCA. Unlike prior works

[8]The results in other datasets are aligned with the reported results here. As such, we only present results for CW Dataset.

focusing on simulated environments or high-leakage devices, we demonstrate successful attacks across diverse platforms, validating our approach on four datasets of real measurements from symmetric key and post-quantum cryptography algorithms. Another key innovation in our work is the MC labeling method, which improves trace labeling by considering dependencies between secret and public variables. This approach outperforms prior techniques, particularly in scenarios where leakage points do not overlap, as shown by our results on AES and Kyber datasets. Finally, we report the first successful blind SCA on hiding countermeasures like desynchronization, showcasing the versatility of the DL-BSCA framework.

Despite these advances, there are open challenges. Previous works considered weak masking and experimented with simulated traces only. Existing studies, including ours, assume that adversaries can locate PoIs. Addressing these limitations would significantly broaden the applicability of blind SCA, paving the way for more robust evaluations of cryptographic implementations.

9 Discussion

At this point, it is helpful to revisit the thesis's focus. The central goal of this work was to explore how evaluators can effectively leverage deep learning in side-channel analysis to improve the performance of their attacks while keeping computational and practical costs low.

As discussed in Chapter 1, an evaluation process aims to be as thorough as possible within limited resources. In practice, this means considering advanced attacks and powerful adversaries while keeping the evaluation effort within a reasonable cost range. One powerful tool for evaluating against advanced attacks is deep learning. At the same time, because of its ability to break protected implementations by countermeasures, it is also an attractive and powerful tool in the hands of adversaries. These capabilities make DL-SCA both a powerful evaluation method and an essential benchmark for assessing the robustness of cryptographic implementations.

This thesis examined the use of DL-SCA from two perspectives:

1. The **benefits** that using deep learning can bring to the evaluation process.
2. The **challenges** that using deep learning introduces to the evaluation process.

In the following, we discuss these two perspectives in terms of the costs they introduce to or eliminate from the evaluation process.

Reducing Evaluation Costs with Deep Learning. Deep learning can simplify or even eliminate certain preprocessing steps, reducing the overall cost of measurement analysis. Beyond preprocessing, however, conventional side-channel evaluation methods are affected by additional cost drivers. In this thesis, we investigate how deep neural networks can be used to mitigate the costs associated with two major factors: (1) the need to assume a specific leakage model, and (2) the reliance of both profiling and non-profiling analyses on known data, such as plaintexts or ciphertexts, in scenarios where obtaining such data is difficult or expensive.

Aligned with the first factor, in Chapter 7, we proposed a multi-byte, multi-bit DL-SCA approach based on multitask learning. This method combines bit-level and overall leakage modeling within a single network structure, allowing it to adapt flexibly

to real leakage patterns without repeated evaluations under different assumptions. This qualification reduces the cost imposed by a poor evaluation, as the evaluator may not detect the actual leakage, or by repeating the evaluation with different leakage models.

The experimental results demonstrate that the proposed multi-bit and multi-byte multi-bit DL-SCA approaches consistently outperform traditional predefined leakage models such as HW, ID, LSB, and MSB across multiple masked AES datasets, including ASCAD-F, ASCAD-R, CHES-CTF, and eShard. Using a fixed CNN architecture and identical hyperparameters for all attacks to ensure fairness, the experiments show that multi-bit and multi-byte are particularly effective in scenarios with Boolean masking, where conventional leakage models often fail to recover certain key bytes. In several cases, such as subkey number 12 in ASCAD-F and eShard, only multi-bit and multi-byte successfully recover the key with significantly fewer traces, confirming their superior ability to capture complex leakage patterns by jointly exploiting multiple bits and shares. Moreover, the results highlight the benefits of multi-task learning in multi-bit-multi-byte, which enables simultaneous sub-key recovery without sacrificing performance, indicating strong feature sharing across tasks. Overall, the findings validate that flexible, model-agnostic multi-bit approaches provide more robust and scalable attack performance than bit-specific leakage models, especially in realistic masked implementations.

In Chapter 8, we showed that it is possible to use deep learning where accessing known data like plaintext or ciphertext is expensive or impossible (a scenario known as blind SCA), and reformulated SCA as a classification problem with noisy labels. By combining deep learning and clustering, we demonstrated the first practical blind SCA in real-world scenarios for implementations that do not allow access to the known data.

The experimental results show that the proposed DL-BSCA framework, together with the new Multi-point Cluster-based labeling method, enables successful blind side-channel attacks in scenarios where all prior approaches failed. Across multiple real-world datasets, including AES on ChipWhisperer, Kyber decapsulation on STM32, and ASCON implementation, classical blind SCA and standalone MC labeling were unable to recover the key. In contrast, combining deep neural networks with the proposed labeling strategy consistently reduced the guessing entropy, demonstrating practical key recovery without plaintext or ciphertext knowledge. The results highlight that MC labeling is particularly effective when paired with DNNs, and that multi-point labeling significantly improves robustness in noisy or misaligned traces, enabling—for the first time—a successful blind SCA attack against a target protected with desynchronization.

Managing the Costs and Pitfalls of Using Deep Learning. While deep learning can reduce several costs, it also introduces new challenges and costs that evaluators must manage carefully. Two primary concerns in DL-SCA are the need for a costly hyperparameter tuning procedure and the reduced evaluation reliability due to overfitting. The final goal of hyperparameter tuning is to improve the performance of the models. In the realm of deep learning, model improvement is equivalent to

enhancing the model's accuracy and/or reducing its loss on the test dataset. However, in the SCA realm, we are more interested in reducing the *GE* and *NT* in the attack phase. Therefore, we seek alternative techniques and strategies to improve model performance on these two metrics while limiting consecutive costs low.

In Chapters 4 and 6, we demonstrated how model enhancement techniques, specifically regularization and an ensemble of models, can improve the performance of DL-SCA by reducing *GE* and *NT*.

Regularization methods, especially L_1 and L_2, reduce overfitting by controlling the model's capacity, resulting in robustness during both training and attack. However, regularization methods generally increase training time,[1] slightly raising computational costs. Ensembles of models, on the other hand, achieve similar improvements in performance and reliability with lower additional cost. The reuse of models already trained during hyperparameter tuning helps to avoid extra training effort while enhancing evaluation completeness. Still, one should not abandon the use of regularization techniques merely because of their higher training cost, as we have seen that evaluation must always find a balance between completeness and cost.

The ensemble models used in our work were derived through a random search strategy. Prior research has shown that random search is an effective method for identifying strong models in DL-SCA tasks on publicly available datasets. This approach tends to yield diverse models, both in terms of architecture and the representations they learn. Consequently, ensembling such heterogeneous models is beneficial, as it allows the combination of complementary information. However, an open research question remains: would ensembling still provide benefits if the individual models were obtained through more structured hyperparameter optimization methods, such as Bayesian optimization? This is a valid concern because such tuning techniques typically converge toward a single local optimum, resulting in models that are similar in their characteristics. In that case, the output probability distributions of these models may contain little complementary information, and thus, their ensemble may not offer additional benefits.

Both regularization and ensemble learning illustrate complementary strategies for enhancing DL-SCA: the former improves a single model's generalization by constraining its learning dynamics, while the latter improves decision-making by aggregating diverse models. Taken together, these techniques strengthen DL-SCA in terms of reliability and efficiency, both essential for practical evaluation.

We also explored the influence of more training examples and using robust activation function strategies on improving DL-SCA performance in Chapters 3 and 5. In Chapter 3, we have seen that the influence of increasing the number of training examples on the models' performance depends on the models' operational regimes. However, most of the models performed better after adding more examples to the training dataset. From this observation, we derive a practical strategy: evaluators can perform hyperparameter tuning on a smaller dataset to reduce cost, and then train the final selected model with more traces for the final evaluation. There is a good chance that the model's performance improves. However, it is still not clear

[1] Early stopping can reduce the training time. However, its effectiveness in the final model's performance, in terms of *GE* and *NT*, is significantly lower than that of the other tested methods.

how the final improvement of the improved models relates to the additional training traces. Some models show a significant improvement after training with more traces, while others only show a slight improvement, which can be considered a trade-off between completeness and cost.

In Chapter 5, we further analyzed the effect of robust activation functions, i.e., JumpReLU, on model performance and generalization, showing that architectural improvements can complement data-based strategies in achieving stable and reliable evaluations. Motivated by this, one interesting research direction is designing customized activation functions, loss functions, or even neural network architectures that are more tailored for specific side-channel analysis usage.

One should note that the techniques and strategies mentioned above are not intended to replace hyperparameter tuning. Hyperparameter tuning remains the main step before any evaluation using deep learning; however, the available budget determines the extent to which the tuning process can be rigorous and comprehensive. It is only after that that the enhancement techniques and strategies can come into play and provide further improvement.

An important advantage of the enhancement techniques investigated in this thesis is that they are not mutually exclusive. Each method addresses different aspects of model improvement, meaning that using one does not make the others impractical or redundant. Instead, they can be combined in a complementary manner, allowing evaluators to tailor a balanced strategy that leverages multiple forms of improvement within the constraints of their evaluation budget.

9.1. Distinct Characteristics of Deep Learning Usage in the SCA Domain

Strive to Fail. A fundamental difference between DL-SCA and common deep-learning domains, such as natural language processing or computer vision, lies in how “success” is interpreted. In most deep learning applications, a model is successful when it performs its task as accurately as possible (whether that means classifying images, translating text, or recognizing speech), and an accurate prediction is the desired outcome. In DL-SCA, however, success has two perspectives: from the attacker’s (or evaluator’s) viewpoint, DL-SCA is successful when a model is able to recover secret-dependent information from side-channel traces; from the defender’s viewpoint, a countermeasure is successful when even strong and carefully designed DL-SCA attacks fail to do so. In many cases, the DL-SCA failure is reduced to “inappropriate” deep learning model. This asymmetry creates an evaluation challenge: when a model fails to recover the secret, it is not immediately clear whether this indicates a secure implementation or simply an “inappropriate” model. This is highlighted more when the evaluator is using DL-SCA for the purpose of improving the implementation before releasing the product.

In typical learning tasks, one can define a performance threshold (for example, achieving a specific accuracy level) and declare the task complete once the model reaches it. In contrast, in SCA,if a trained model fails to extract the secret, evaluation cannot stop there. The only valid statement the analyst can make is that the *tested*

model did not succeed. Even when analysts explore a broad and diverse range of neural-network architectures and attack configurations, this still represents only a finite subset of all possible distinguishers.

To illustrate this further, let us imagine a universe where deep learning is the only distinguisher for a power side-channel (the only physical side-channel). Let us also imagine that, in this hypothetical universe, evaluating an implementation using all conceivable neural networks is possible. If none of these models could recover the secret, the analyst could confidently conclude that "**the implementation does not leak exploitable secret information**." This statement is highly desirable for both manufacturers and end users.

However, the real world is far from this imaginary universe. While manufacturers and users would still like to hear that "**the implementation does not leak exploitable secret information**," evaluating it against all possible distinguishers, including the infinite space of neural networks, is impossible. Consequently, an analyst cannot formally prove the absence of leakage through empirical evaluation, since it is infeasible to test every possible attack. This is similar to the leakage-assessment process: the fact that no leakage is detected does not necessarily mean that an implementation cannot be attacked using another distinguisher (for example, a higher-order moment). Likewise, the fact that none of the tested models could break the target does not prove that the implementation cannot be broken by other models or attacks. In practice, evaluators therefore aim for a pragmatic conclusion: "**Given the best attacks and models that can realistically be designed within a limited resource budget, no exploitable information leakage was detected**."

This is where the importance of model enhancement techniques and strategies becomes clear. Such methods allow analysts to design stronger and more representative attacks and models without significantly increasing resource requirements, thereby enhancing the credibility of the evaluation.

DL-SCA is not the only domain where the 'strive to fail' principle applies. Similar evaluation philosophies exist in other fields where systems are tested through intentionally strong but undesirable outcomes. For instance, in adversarial robustness evaluation for deep neural networks, researchers design the most effective adversarial attacks, hoping the model resists them. The same mindset applies to software fuzzing, where testers attempt to crash programs while developers hope no crash occurs. Formal verification is another example, where analysts search for specification violations while ideally finding none. In all these cases, the evaluator desires failure: the system's resistance under the strongest possible tests defines its strength.

Added noise by design. Another important difference of DL-SCA from the common usage of deep learning lies in the presence and purpose of noise. In common deep learning applications, noise in training and test data is typically undesired and unavoidable. The noise in the common usages arises naturally from environmental variability or data acquisition imperfections. Users do not appreciate its presence, but

must train models to be robust against it. The environmental and architectural noise [2] are also present in the side-channel measurements. These noises are unavoidable, but here they are desirable, unlike in common deep learning applications, as they help hide leakage. However, the importance of noise in side-channel analysis extends further. Indeed, a part of the noise in the side-channel traces is intentionally added by using masking or hiding countermeasures in the implementation of cryptography algorithms to obfuscate sensitive information that might leak through side channels.

To make this difference clear, consider two image classification tasks. In the first task, the model must determine whether there is a dog in a photo of any random landscape, where the background can vary widely and include many irrelevant objects. In the second, the model must detect a dog in images that always have a fixed background, such as the entrance of a building captured by a stationary camera. The first model faces a much more challenging problem, as it must learn to recognize the dog regardless of distracting or variable backgrounds. During training, it is therefore important to ensure that the final model becomes robust to background noise and irrelevant details. Whatever complexity is added is to make the model more robust for the actual application, but not to the point where it becomes unrecognizable. The second task is simpler. Since all images share the same background, the model can more easily focus on identifying the presence or absence of a dog. These two examples illustrate problems with two different levels of undesirable environmental noise. However, in side-channel analysis, the objective is quite the opposite: designers of countermeasures purposely aim to increase noise and blur the information that could reveal secret data. The ideal situation is one where no information related to the secret can be extracted from the device. Thus, while practitioners continuously design stronger countermeasures, evaluators still employ deep learning to test their effectiveness. This also presents an additional challenge regarding the “strive-to-fail” dynamic in this domain. Of course, we continuously strive to design stronger countermeasures, but we also still want to evaluate their robustness using deep learning.

Different performance metrics. This is something we have already mentioned in Section 2.6.2. In SCA, performance metrics such as guessing entropy and the required number of attack traces evaluate the attacker’s ability to recover the correct secret key, whereas deep learning and machine learning metrics like accuracy, precision, or F1-score measure how well a classifier predicts class labels. As shown by Picek et al. [61], these two categories of metrics capture fundamentally different aspects of performance. Machine/deep learning metrics like accuracy, quantify per-example classification correctness, treating each prediction independently, whereas SCA metrics aggregate information across multiple examples (traces) to estimate the likelihood of the attack revealing the correct key. In other words, accuracy measures how well the model classifies intermediate values, whereas *GE* directly reflects the adversary’s success in key recovery. These metrics often diverge: a classifier with high accuracy can still perform poorly in key recovery, and vice versa.

[2] Noise originating from other unrelated activities on the device, which affect the recorded traces.

Machine/deep learning metrics generally ignore the probabilistic structure exploited in SCA. Both accuracy and loss consider only the true class of each trace. Although the loss is typically aggregated over multiple measurements, it still focuses solely on the probability (in the case of using $softmax$) assigned to the correct class and does not account for how the probabilities of the incorrect classes contribute to the attack. However, in SCA, these probabilities are highly informative: combining them across traces provides useful evidence that strengthens the ranking of the correct key candidate during the attack phase. By contrast, *GE* and, consequently, *NT* explicitly depend on the accumulation of output probabilities across all classes and all traces, which better reflects the true objective of key recovery.

Consequently, while deep learning can enhance feature extraction and classification performance, SCA evaluations should always rely on domain-specific metrics such as *GE* and *NT* rather than general-purpose machine/deep learning indicators.

It is important to note, however, that this observation primarily concerns *multi-trace* attacks such as DPA, where the attacker aggregates information from many traces, and the evaluation objective is inherently tied to cumulative key-ranking metrics. In Single-Trace or SPA scenarios, the attack often relies on a single prediction event, and the link between classification metrics and attack success may therefore be stronger. Nevertheless, even in SPA settings, security claims should ultimately be expressed in terms of key-recovery success (e.g., probability of correct key identification), because a model may achieve good classification accuracy while still offering insufficient confidence for a reliable or repeatable attack. In this sense, *GE*, SR, or closely related key-focused indicators remain the most informative metrics for assessing the real cryptographic risk, whereas accuracy and loss should be regarded only as auxiliary training or diagnostic measures rather than primary evaluation criteria.

9.2. Future Directions

Reducing the gap between academic research and DL-SCA applications in evaluation. A persistent challenge in the field of SCA is the gap between academic research and its practical use in industrial or certification-level evaluations. The evaluation frameworks commonly adopted in industry (as introduced in Section 1.3.1) are either based on leakage analysis, like FIPS 140 [2], or on the Common Criteria. These frameworks tend to evaluate the security claim of the producer rather than ensuring the security of the implementation against all possible attacks. They emphasize structured testing procedures, resource limitations, and clear success criteria that make results comparable across laboratories and devices. In contrast, academic research often follows a working-backward strategy, where the main objective is to exploit possible vulnerabilities using maximum resources, assuming the most powerful adversary. This experimental approach is crucial for scientific progress: by pushing these limits, the academic community highlights weaknesses that motivate new countermeasures and improved software and hardware design.

However, while evaluation labs (especially those that use common criteria) may draw inspiration from published academic attacks, they cannot directly apply them to their structured evaluation process. In particular, three main reasons can be counted

for that non-applicability.

- **Resource availability:** Academic studies often employ vast computational resources. The researcher works on the target for a long time and has a high motivation to exploit the vulnerability in the end. In contrast, evaluation labs operate under strict time and cost constraints, where each analysis must be completed within a defined budget.

- **Data quality and experimental control:** Publicly available academic datasets are usually collected under idealized laboratory conditions using educational targets that provide clean, low-noise traces and stable environmental settings. Industrial devices, however, often exhibit high variability, measurement noise, and additional layers of protection, which can make even strong academic attacks ineffective in practice.

- **Evaluation goals:** Research papers generally aim to demonstrate feasibility, showing that an attack can work under certain conditions, whereas evaluators must assess whether the implementation can pass a security level that is defined with some specific characteristics. The latter demands repeatability, bounded assumptions, and cost justification, which are rarely quantified in academic works.

These differences are particularly pronounced in DL-SCA, where the effectiveness of deep learning models depends heavily on data quality, hyperparameter tuning, and computational budgets. As a result, many published DL-SCA results, although scientifically valuable, are not directly usable as evaluation tools.

Bridging this gap requires efforts from both sides. Evaluation laboratories could strengthen collaboration with academia by sharing anonymized case studies, realistic traces, and evaluation protocols that reflect industrial constraints. Such collaborations would help researchers design methods that are both scientifically sound and practically relevant. On the academic side, researchers could aim to publish not only performance metrics but also detailed reports on the resources consumed, such as training time, hardware specifications, and data collection costs, to provide evaluators with a more realistic sense of the attack's feasibility. Moreover, a new generation of benchmark datasets could be established to reflect real-world conditions better, incorporating noise, environmental variability, and diverse hardware architectures.

Ultimately, reducing the academic–industrial gap is not merely a matter of transferring knowledge; it requires aligning objectives, constraints, and success criteria. The long-term goal should be a feedback loop where academic findings continuously inform evaluation methodologies, and industrial challenges, in turn, shape the next generation of research questions in DL-SCA.

Unlocking the real power of deep learning and artificial intelligence for SCA. Currently, DL-SCA remains highly implementation-specific: for each new device or countermeasure, evaluators must engage in time-consuming hyperparameter tuning and model selection. The problem is so fundamental that we face not only a

transferability issue but even a portability problem.[3] This dependence on exhaustive tuning increases both the cost and complexity of evaluations. One promising path forward is to reduce this dependency through techniques such as transfer learning[4] or the development of models with stronger generalization capabilities.

To better understand the potential of such advances, it helps to look at progress in other AI domains. Generative Artificial Intelligence (AI) has demonstrated that deep learning systems can extend far beyond task-specific optimization, learning highly general and transferable representations of data. For instance, models like ChatGPT, DALL·E, and GitHub Copilot are capable of generating diverse, human-like content because they have been trained on massive and heterogeneous datasets that capture broad patterns of language, vision, and reasoning. Similarly, foundation models such as VGG, ResNet, and DenseNet for vision; BERT and RoBERTa for text; and Whisper, wav2vec 2.0, and HuBERT for speech demonstrate that deep learning architectures can learn features general enough to be reused across tasks with minimal fine-tuning.

A similar breakthrough could occur in the SCA domain. Imagine models trained not merely to attack a single implementation, but to understand the general characteristics of side-channel leakage, including different structures, noise patterns, and countermeasure artifacts, across different platforms, cryptographic algorithms, and measurement conditions. Such "foundational" SCA models could then be fine-tuned for specific devices or evaluation contexts, reducing the effort and cost required for hyperparameter tuning. This would also enhance reproducibility and standardization in evaluations, as different laboratories could build upon the same base model rather than developing models from scratch for every target. Regardless of pursuing this direction, there is a requirement for parallel research on adaptive countermeasures that can detect or resist attacks from generalized models trained outside the evaluation environment. That is because such models can sooner or later be developed either by evaluators or adversaries, as they have been developed for other classification tasks.

At the same time, achieving such transferability has a dual-use implication. Developing broadly generalizable SCA models would empower evaluators by allowing for faster and more unified assessments; however, the same models could also be misused by adversaries to attack devices more efficiently. The existence of general-purpose, pretrained SCA models could lower the technical barrier for conducting sophisticated attacks, potentially giving attackers an advantage in the ongoing competition between attacks and defenses.

[3] In current DL-SCA practice, researchers often use a single device for both the profiling and attack phases [87]. While this design choice aligns with the working-backwards academic strategy and represents a worst-case adversary assumption, it avoids an even more basic challenge: ensuring that a trained model performs consistently across multiple instances of the same device type, produced by the same manufacturer, and using the same architecture. This intra-device portability issue arises even before tackling the broader problem of model transferability across devices with different manufacturing variations, hardware architectures, or acquisition setups. Addressing both portability and transferability remains one of the most critical steps toward making DL-SCA a practical evaluation tool.

[4] Limited studies have explored this idea [173–175], but they have not yet yielded models with robust and widely transferable performance.

Moving toward more automation in DL-SCA. Building on the previous discussion, another key research direction is to increase the level of automation in DL-SCA. Despite relying on deep learning, the current use of AI in SCA is far from automated. DL-SCA remains an interdisciplinary field that demands expertise across several domains, including cryptography, software and hardware implementations, statistics, and deep learning. Conducting a successful DL-SCA still requires significant manual effort and expert judgment at every stage of the process.

Currently, the analyst must manually interpret the source code or hardware implementation to identify potential leakage points, select relevant signal regions, and take appropriate preprocessing steps prior to the profiling phase. Even after the deep neural network generates predictions for the attack traces, the attack phase involves additional tools, such as SASCA, and manual postprocessing to compute metrics like *GE*. In other words, the process is not end-to-end: it remains a semi-manual pipeline where human expertise fills in many critical gaps.

A deeper issue is conceptual: many DL-SCA studies use deep learning to solve problems whose solutions are already partially known. For example, analysts often restrict the neural network input to a specific time window of the traces because they already know, through prior leakage detection methods, such as correlation analysis, where the targeted intermediate variable leaks. This means that the neural network is not truly "discovering" leakage from raw data, but rather optimizing within a region preselected by the analyst. Consequently, DL-SCA remains highly dependent on human expertise and intuition, much like traditional statistical approaches such as the Gaussian templates attack.

Achieving a higher degree of automation would require redefining the DL-SCA workflow to make such manual steps an intrinsic part of the learning process itself. A fully automated DL-SCA pipeline would reduce the need for extensive human intervention, making the process faster, more reproducible, and more accessible to non-experts. More importantly, it would allow evaluators to focus on interpreting results and defining threat models rather than manually engineering each analysis step. Automation, therefore, represents not only a practical improvement but also a conceptual evolution toward truly intelligent SCA systems.

Bibliography

[1] I. O. for Standardization. *Information Technology; Security Techniques; Evaluation Criteria for IT Security: Part 1: Introduction and General Model*. International Organization for Standardization, 2009.

[2] National Institute of Standards and Technology (NIST). *Security Requirements for Cryptographic Modules (FIPS PUB 140-3)*. FIPS Publication FIPS PUB 140-3. U.S. Department of Commerce, National Institute of Standards and Technology, Information Technology Laboratory, 2019. DOI: 10.6028/NIST.FIPS.140-3. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf.

[3] M. Azouaoui, D. Bellizia, I. Buhan, N. Debande, S. Duval, C. Giraud, É. Jaulmes, F. Koeune, E. Oswald, F. Standaert and C. Whitnall. 'A Systematic Appraisal of Side Channel Evaluation Strategies'. In: *Security Standardisation Research - 6th International Conference, SSR 2020, London, UK, November 30 - December 1, 2020, Proceedings*. Ed. by T. van der Merwe, C. J. Mitchell and M. Mehrnezhad. Lecture Notes in Computer Science. Springer, 2020, pp. 46–66. DOI: 10.1007/978-3-030-64357-7_3. URL: https://doi.org/10.1007/978-3-030-64357-7%5C_3.

[4] S. Picek, G. Perin, L. Mariot, L. Wu and L. Batina. 'SoK: Deep Learning-based Physical Side-channel Analysis'. In: *ACM Comput. Surv.* 55.11 (2023), 227:1–227:35. DOI: 10.1145/3569577. URL: https://doi.org/10.1145/3569577.

[5] H. Maghrebi, T. Portigliatti and E. Prouff. 'Breaking Cryptographic Implementations Using Deep Learning Techniques'. In: *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*. Ed. by C. Carlet, M. A. Hasan and V. Saraswat. Lecture Notes in Computer Science. Springer, 2016, pp. 3–26. DOI: 10.1007/978-3-319-49445-6_1. URL: https://doi.org/10.1007/978-3-319-49445-6%5C_1.

[6] E. Cagli, C. Dumas and E. Prouff. 'Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without pre-processing'. In: *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer. 2017, pp. 45–68. DOI: 10.1007/978-3-319-66787-4_3.

[7] R. Benadjila, E. Prouff, R. Strullu, E. Cagli and C. Dumas. 'Deep learning for side-channel analysis and introduction to ASCAD database'. In: *Journal of Cryptographic Engineering* 10.2 (2020), pp. 163–188.

[8] S. Picek, A. Heuser, G. Perin and S. Guilley. 'Profiled Side-Channel Analysis in the Efficient Attacker Framework'. In: *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*. Ed. by V. Grosso and T. Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Springer, 2021, pp. 44–63. DOI: 10.1007/978-3-030-97348-3_3. URL: https://doi.org/10.1007/978-3-030-97348-3%5C_3.

[9] A. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN: 0-8493-8523-7. DOI: 10.1201/9781439821916. URL: http://cacr.uwaterloo.ca/hac/.

[10] J. Daemen and V. Rijmen. 'AES proposal: Rijndael'. In: (1999).

[11] N. I. of Standards and Technology. *About NIST*. https://www.nist.gov/about-nist. U.S. Department of Commerce. (Visited on 29/09/2025).

[12] National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard (AES)*. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[13] R. L. Rivest, A. Shamir and L. Adleman. 'A method for obtaining digital signatures and public-key cryptosystems'. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[14] N. Koblitz. 'Elliptic curve cryptosystems'. In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[15] V. S. Miller. 'Use of Elliptic Curves in Cryptography'. In: *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*. Ed. by H. C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426. DOI: 10.1007/3-540-39799-X_31. URL: https://doi.org/10.1007/3-540-39799-X%5C_31.

[16] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, R. V. Keer and B. Viguier. 'KangarooTwelve: Fast Hashing Based on Keccak-p'. In: *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*. Ed. by B. Preneel and F. Vercauteren. Vol. 10892. Lecture Notes in Computer Science. Springer, 2018, pp. 400–418. DOI: 10.1007/978-3-319-93387-0_21. URL: https://doi.org/10.1007/978-3-319-93387-0%5C_21.

[17] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche. 'Duplexing the sponge: single-pass authenticated encryption and other applications'. In: *Selected Areas in Cryptography: 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers 18*. Springer. 2012, pp. 320–337.

[18] C. Dobraunig, M. Eichlseder, F. Mendel and M. Schläffer. 'Ascon v1.2: Lightweight Authenticated Encryption and Hashing'. In: *J. Cryptol.* 34.3 (2021), p. 33. DOI: 10.1007/S00145-021-09398-9. URL: https://doi.org/10.1007/s00145-021-09398-9.

[19] P. W. Shor. 'Algorithms for Quantum Computation: Discrete Logarithms and Factoring'. In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, November 20-22, 1994*. IEEE Computer Society, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700. URL: https://doi.org/10.1109/SFCS.1994.365700.

[20] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler and D. Stehlé. 'CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM'. In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 2018, pp. 353–367. DOI: 10.1109/EUROSP.2018.00032. URL: https://doi.org/10.1109/EuroSP.2018.00032.

[21] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler and D. Stehlé. 'CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme'. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.1 (2018), pp. 238–268. DOI: 10.13154/TCHES.V2018.I1.238-268. URL: https://doi.org/10.13154/tches.v2018.i1.238-268.

[22] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld and P. Schwabe. 'The SPHINCS$^+$ Signature Framework'. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by L. Cavallaro, J. Kinder, X. Wang and J. Katz. ACM, 2019, pp. 2129–2146. DOI: 10.1145/3319535.3363229. URL: https://doi.org/10.1145/3319535.3363229.

[23] L. K. Grover. 'A Fast Quantum Mechanical Algorithm for Database Search'. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*. Ed. by G. L. Miller. ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[24] F. Standaert. 'Introduction to Side-Channel Attacks'. In: *Secure Integrated Circuits and Systems*. Ed. by I. M. R. Verbauwhede. Integrated Circuits and Systems. Springer, 2010, pp. 27–42. DOI: 10.1007/978-0-387-71829-3_2. URL: https://doi.org/10.1007/978-0-387-71829-3%5C_2.

[25] S. Mangard and K. Schramm. ‘Pinpointing the side-channel leakage of masked AES hardware implementations’. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2006, pp. 76–90.

[26] T. Moos, A. Moradi and B. Richter. ‘Static power side-channel analysis—An investigation of measurement factors’. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.2 (2019), pp. 376–389.

[27] G. Provelengios, C. Ramesh, S. B. Patil, K. Eguro, R. Tessier and D. Holcomb. ‘Characterization of long wire data leakage in deep submicron FPGAs’. In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2019, pp. 292–297.

[28] I. Giechaskiel and J. Szefer. ‘Information leakage from FPGA routing and logic elements’. In: *Proceedings of the 39th international conference on computer-aided design*. 2020, pp. 1–9.

[29] T. S. Messerges, E. A. Dabbish and R. H. Sloan. ‘Power Analysis Attacks of Modular Exponentiation in Smartcards’. In: *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES’99, Worcester, MA, USA, August 12-13, 1999, Proceedings*. Ed. by Ç. K. Koç and C. Paar. Vol. 1717. Lecture Notes in Computer Science. Springer, 1999, pp. 144–157. DOI: 10.1007/3-540-48059-5_14.

[30] S. Chari, C. S. Jutla, J. R. Rao and P. Rohatgi. ‘Towards Sound Approaches to Counteract Power-Analysis Attacks’. In: *Advances in Cryptology - CRYPTO ’99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by M. J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 398–412. DOI: 10.1007/3-540-48405-1_26.

[31] E. Brier, C. Clavier and F. Olivier. ‘Correlation Power Analysis with a Leakage Model’. In: *Cryptographic Hardware and Embedded Systems - CHES 2004*. Ed. by M. Joye and J.-J. Quisquater. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29. ISBN: 978-3-540-28632-5.

[32] B. Gierlichs, L. Batina, P. Tuyls and B. Preneel. ‘Mutual information analysis’. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2008, pp. 426–442. DOI: 10.1007/978-3-540-85053-3_27.

[33] P. C. Kocher, J. Jaffe and B. Jun. ‘Differential Power Analysis’. In: *Advances in Cryptology - CRYPTO ’99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by M. J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25. URL: https://doi.org/10.1007/3-540-48405-1%5C_25.

[34] S. Chari, J. R. Rao and P. Rohatgi. 'Template Attacks'. In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by B. S. K. Jr., Ç. K. Koç and C. Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 13–28. DOI: 10.1007/3-540-36400-5_3.

[35] J. Kim, S. Picek, A. Heuser, S. Bhasin and A. Hanjalic. 'Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), pp. 148–179.

[36] Federal Office for Information Security (BSI). *Guidelines for Evaluating Machine-Learning based Side-Channel Attack Resistance*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_AI_guide.pdf?__blob=publicationFile&v=6. Technical Report AIS 46. Feb. 2024.

[37] L. Masure, N. Belleville, E. Cagli, M.-A. Cornelie, D. Couroussé, C. Dumas and L. Maingault. 'Deep Learning Side-Channel Analysis on Large-Scale Traces: A Case Study on a Polymorphic AES'. In: *European Symposium on Research in Computer Security*. Springer. 2020, pp. 440–460.

[38] **A. Rezaeezade**, G. Perin and S. Picek. 'To overfit, or not to overfit: improving the performance of deep learning-based SCA'. In: *International Conference on Cryptology in Africa*. Springer. 2022, pp. 397–421.

[39] **A. Rezaeezade** and L. Batina. 'Regularizers to the rescue: fighting overfitting in deep learning-based side-channel analysis'. In: *Journal of Cryptographic Engineering* (2024), pp. 1–21. DOI: 10.1007/s13389-024-00361-5.

[40] A. Basurto-Becerra, **A. Rezaeezade** and S. Picek. 'Jump, It Is Easy: JumpReLU Activation Function in Deep Learning-based Side-channel Analysis'. In: *Cryptology ePrint Archive* (2025).

[41] **A. Rezaeezade**, A. Basurto-Becerra, L. Weissbart and G. Perin. 'One for All, All for Ascon: Ensemble-Based Deep Learning Side-Channel Analysis'. In: *Applied Cryptography and Network Security Workshops*. Ed. by M. Andreoni. Cham: Springer Nature Switzerland, 2024, pp. 139–157. ISBN: 978-3-031-61486-6.

[42] L. Wu, **A. Rezaeezade**, A. Alipour, G. Perin and S. Picek. 'Leakage Model-flexible Deep Learning-based Side-channel Analysis'. In: *IACR Commun. Cryptol.* 1.3 (2024), p. 41. DOI: 10.62056/AY4C3TXOL7. URL: https://doi.org/10.62056/ay4c3txol7.

[43] **A. Rezaeezade**, T. Yap, D. Jap, S. Bhasin and S. Picek. 'Breaking the Blindfold: Deep Learning-based Blind Side-channel Analysis'. In: *Cryptology ePrint Archive* (2025).

[44] A. Dhapte. *ARM Microcontroller Market Research Report By Microcontroller Architecture*. 2025. URL: https://www.marketresearchfuture.com/reports/arm-microcontroller-market-32787.

[45] M. J. Dworkin, E. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, J. F. Dray Jr *et al.* 'Advanced encryption standard (AES)'. In: (2001).

[46] NIST Information Technology Laboratory. 'NIST Lightweight Cryptography Standardization Process'. In: *The National Institute of Standards and Technology* (2023). https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon.

[47] M. Eichlseder. *tikz*. https://extgit.isec.tugraz.at/meichlseder/tikz. GitLab project, Institute of Applied Information Processing and Communications, Graz University of Technology. 2020.

[48] C. Dobraunig, M. Eichlseder, F. Mendel and M. Schläffer. 'Ascon v1. 2'. In: *Submission to the CAESAR Competition* 5.6 (2016), p. 7.

[49] R. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler and D. Stehlé. 'CRYSTALS-Kyber (version 3.0): Algorithm specifications and supporting documentation (October 1, 2020)'. In: *Submission to the NIST post-quantum project* (2020).

[50] F. Hu, H. Wang and J. Wang. 'Multi-leak deep-learning side-channel analysis'. In: *IEEE Access* 10 (2022), pp. 22610–22621.

[51] W. Schindler, K. Lemke and C. Paar. 'A Stochastic Model for Differential Side Channel Cryptanalysis'. In: *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*. Ed. by J. R. Rao and B. Sunar. Lecture Notes in Computer Science. Springer, 2005, pp. 30–46. DOI: 10.1007/11545262_3. URL: https://doi.org/10.1007/11545262%5C_3.

[52] M. O. Choudary and M. G. Kuhn. 'Efficient stochastic methods: Profiled attacks beyond 8 bits'. In: *Smart Card Research and Advanced Applications: 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers 13*. Springer. 2015, pp. 85–103. DOI: 10.1007/978-3-319-16763-3_6.

[53] F.-X. Standaert, F. Koeune and W. Schindler. 'How to compare profiled side-channel attacks?' In: *Applied Cryptography and Network Security: 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings 7*. Springer. 2009, pp. 485–498. DOI: 10.1007/978-3-642-01957-9_30.

[54] B. Gierlichs, K. Lemke-Rust and C. Paar. 'Templates vs. stochastic methods'. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2006, pp. 15–29. DOI: 10.1007/11894063_2.

[55] J.-S. Coron, E. Prouff and M. Rivain. 'Side channel cryptanalysis of a higher order masking scheme'. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2007, pp. 28–44.

[56] T. Zijlstra, K. Bigou and A. Tisserand. 'FPGA Implementation and Comparison of Protections Against SCAs for RLWE'. In: *Progress in Cryptology–INDOCRYPT 2019: 20th International Conference on Cryptology in India, Hyderabad, India, December 15–18, 2019, Proceedings 20*. Springer. 2019, pp. 535–555. DOI: 10.1007/978-3-030-35423-7_27.

[57] Z. Chen, Y. Ma and J. Jing. 'Low-Cost Shuffling Countermeasures Against Side-Channel Attacks for NTT-Based Post-Quantum Cryptography'. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.1 (2023), pp. 322–326. DOI: 10.1109/TCAD.2022.3174142.

[58] R. C. Rodriguez, F. Bruguier, E. Valea and P. Benoit. 'Correlation Electromagnetic Analysis on an FPGA Implementation of CRYSTALS-Kyber'. In: *2023 18th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*. IEEE. 2023, pp. 217–220. DOI: 10.1109/prime58259.2023.10161764.

[59] R. Hierons. *Machine learning. Tom M. Mitchell. Published by McGraw-Hill, Maidenhead, UK, International Student Edition, 1997. ISBN: 0-07-115467-1, 414 pages.* 1999.

[60] Y. LeCun, Y. Bengio and G. Hinton. 'Deep learning'. In: *nature* 521.7553 (2015), pp. 436–444.

[61] S. Picek, A. Heuser, A. Jovic, S. Bhasin and F. Regazzoni. 'The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.1 (2019), pp. 1–29.

[62] J. Massey. 'Guessing and entropy'. In: *Proceedings of 1994 IEEE International Symposium on Information Theory*. 1994, pp. 204–. DOI: 10.1109/ISIT.1994.394764.

[63] B. Köpf and D. Basin. 'An information-theoretic model for adaptive side-channel attacks'. In: *Proceedings of the 14th ACM conference on Computer and communications security*. 2007, pp. 286–296.

[64] F.-X. Standaert, T. G. Malkin and M. Yung. 'A unified framework for the analysis of side-channel key recovery attacks'. In: *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*. Springer. 2009, pp. 443–461. DOI: 10.1007/978-3-642-01001-9_26.

[65] K. Papagiannopoulos, O. Glamočanin, M. Azouaoui, D. Ros, F. Regazzoni and M. Stojilović. 'The Side-Channel Metrics Cheat Sheet'. In: *ACM Comput. Surv.* 55.10 (2023). ISSN: 0360-0300. DOI: 10.1145/3565571. URL: https://doi.org/10.1145/3565571.

[66] Colin O'Flynn and Zhizhang (David) Chen. 'ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research'. In: *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*. Ed. by E. Prouff. Vol. 8622. Lecture Notes in Computer Science. Springer, 2014, pp. 243–260. DOI: 10.1007/978-3-319-10175-0_17. URL: https://doi.org/10.1007/978-3-319-10175-0%5C_17.

[67] L. Wu, L. Weissbart, M. Krček, H. Li, G. Perin, L. Batina and S. Picek. 'Label Correlation in Deep Learning-based Side-channel Analysis'. In: *IEEE Transactions on Information Forensics and Security* (2023). DOI: 10.1109/tifs.2023.3287728.

[68] T. Yap, S. Picek and S. Bhasin. 'OccPoIs: Points of Interest Based on Neural Network's Key Recovery in Side-Channel Analysis Through Occlusion'. In: *Progress in Cryptology - INDOCRYPT 2024 - 25th International Conference on Cryptology in India, Chennai, India, December 18-21, 2024, Proceedings, Part II*. Ed. by S. Mukhopadhyay and P. Stanica. Vol. 15496. Lecture Notes in Computer Science. Springer, 2024, pp. 3–28. DOI: 10.1007/978-3-031-80311-6_1. URL: https://doi.org/10.1007/978-3-031-80311-6%5C_1.

[69] A. Vasselle, H. Thiebeauld and P. Maurine. 'Spatial dependency analysis to extract information from side-channel mixtures: extended version'. In: *Journal of Cryptographic Engineering* (2023), pp. 1–17. DOI: 10.1007/s13389-022-00307-9.

[70] M. Schläffer, C. Dobraunig, J. Großschädl, L. C. dos Santos, F. Bachmann and M. Eichlseder. *ASCON-C Implementation*. Github repository. https://github.com/ascon/ascon-c. 2020.

[71] L. Weissbart and S. Picek. 'Lightweight but Not Easy: Side-channel Analysis of the Ascon Authenticated Cipher on a 32-bit Microcontroller'. In: *IACR Cryptol. ePrint Arch.* (2023), p. 1598. URL: https://eprint.iacr.org/2023/1598.

[72] H. Groß, R. Iusupov and R. Bloem. 'Generic Low-Latency Masking in Hardware'. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.2 (2018), pp. 1–21. DOI: 10.13154/TCHES.V2018.I2.1-21. URL: https://doi.org/10.13154/tches.v2018.i2.1-21.

[73] NewAE Technology. *ChipWhisperer*. https://newae.com/tools/chipwhisperer.

[74] **A. Rezaeezade**, T. Yap, D. Jap, S. Bhasin and S. Picek. 'Side-Channel Power Trace Dataset for Kyber Pair-Pointwise Multiplication on Cortex-M4'. In: *Cryptology ePrint Archive* (2025).

[75] L. Masure, C. Dumas and E. Prouff. 'A Comprehensive Study of Deep Learning for Side-Channel Analysis'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.1 (2020), pp. 348–375. DOI: 10.13154/tches.v2020.i1.348-375.

[76] G. Zaid, L. Bossuet, A. Habrard and A. Venelli. 'Methodology for Efficient CNN Architectures in Profiling Attacks'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.1 (2020), pp. 1–36. DOI: 10.13154/tches.v2020.i1.1-36.

[77] G. Perin, L. Chmielewski and S. Picek. 'Strength in Numbers: Improving Generalization with Ensembles in Machine Learning-based Profiled Side-channel Analysis'. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.4 (2020), pp. 337–364. DOI: 10.13154/tches.v2020.i4.337-364. URL: https://doi.org/10.13154/tches.v2020.i4.337-364.

[78] L. Wu, G. Perin and S. Picek. 'I Choose You: Automated Hyperparameter Tuning for Deep Learning-Based Side-Channel Analysis'. In: *IEEE Trans. Emerg. Top. Comput.* 12.2 (2024), pp. 546–557. DOI: 10.1109/TETC.2022.3218372. URL: https://doi.org/10.1109/TETC.2022.3218372.

[79] J. Rijsdijk, L. Wu, G. Perin and S. Picek. 'Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), pp. 677–707.

[80] F. Durvaux, F. Standaert and N. Veyrat-Charvillon. 'How to Certify the Leakage of a Chip?' In: *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*. Ed. by P. Q. Nguyen and E. Oswald. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 459–476. DOI: 10.1007/978-3-642-55220-5_26.

[81] O. Bronchain, J. M. Hendrickx, C. Massart, A. Olshevsky and F.-X. Standaert. 'Leakage certification revisited: Bounding model errors in side-channel security evaluations'. In: *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*. Springer. 2019, pp. 713–737. DOI: 10.1007/978-3-030-26948-7_25.

[82] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak and I. Sutskever. 'Deep Double Descent: Where Bigger Models and More Data Hurt'. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=B1g5sA4twr.

[83] M. Belkin, D. Hsu, S. Ma and S. Mandal. 'Reconciling modern machine-learning practice and the classical bias–variance trade-off'. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[84] A. Krizhevsky, I. Sutskever and G. E. Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger. 2012, pp. 1106–1114.

[85] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le and Z. Chen. 'GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism'. In: *CoRR* abs/1811.06965 (2018). arXiv: 1811.06965. URL: http://arxiv.org/abs/1811.06965.

[86] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. 'Going deeper with convolutions'. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[87] S. Bhasin, A. Chattopadhyay, A. Heuser, D. Jap, S. Picek and R. R. Shrivastwa. 'Mind the Portability: A Warriors Guide through Realistic Profiled Side-channel Analysis'. In: *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[88] Agence nationale de la sécurité des systèmes d'information (ANSSI). *ASCAD*. Github repository. https://github.com/ANSSI-FR/ASCAD. 2018.

[89] G. Perin and S. Picek. 'On the influence of optimizers in deep learning-based side-channel analysis'. In: *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*. Springer. 2021, pp. 615–636.

[90] P. L. Bartlett, A. Montanari and A. Rakhlin. 'Deep learning: a statistical viewpoint'. In: *CoRR* abs/2103.09177 (2021). arXiv: 2103.09177. URL: https://arxiv.org/abs/2103.09177.

[91] G. Perin, I. Buhan and S. Picek. 'Learning When to Stop: A Mutual Information Approach to Prevent Overfitting in Profiled Side-Channel Analysis'. In: *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*. Ed. by S. Bhasin and F. D. Santis. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 53–81. DOI: 10.1007/978-3-030-89915-8_3. URL: https://doi.org/10.1007/978-3-030-89915-8%5C_3.

[92] D. Robissout, G. Zaid, B. Colombier, L. Bossuet and A. Habrard. 'Online Performance Evaluation of Deep Learning Networks for Profiled Side-Channel Analysis'. In: *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Revised Selected Papers*. Ed. by G. M. Bertoni and F. Regazzoni. Vol. 12244. Lecture Notes in Computer Science. Springer, 2020, pp. 200–218. DOI: 10.1007/978-3-030-68773-1_10.

[93] S. Ioffe and C. Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by F. R. Bach and D. M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[94] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[95] A. Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[96] A. Krogh and J. A. Hertz. 'A Simple Weight Decay Can Improve Generalization'. In: *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*. Ed. by J. E. Moody, S. J. Hanson and R. Lippmann. Morgan Kaufmann, 1991, pp. 950–957. URL: http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.

[97] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. 'Dropout: a simple way to prevent neural networks from overfitting'. In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. DOI: 10.5555/2627435.2670313. URL: https://dl.acm.org/doi/10.5555/2627435.2670313.

[98] J. Tompson, R. Goroshin, A. Jain, Y. LeCun and C. Bregler. 'Efficient object localization using Convolutional Networks'. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 648–656. DOI: 10.1109/CVPR.2015.7298664. URL: https://doi.org/10.1109/CVPR.2015.7298664.

[99] C. Garbin, X. Zhu and O. Marques. 'Dropout vs. batch normalization: an empirical study of their impact to deep learning'. In: *Multim. Tools Appl.* 79.19-20 (2020), pp. 12777–12815. DOI: 10.1007/s11042-019-08453-9. URL: https://doi.org/10.1007/s11042-019-08453-9.

[100] X. Li, S. Chen, X. Hu and J. Yang. 'Understanding the Disharmony Between Dropout and Batch Normalization by Variance Shift'. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA,*

USA, June 16-20, 2019. Computer Vision Foundation / IEEE, 2019, pp. 2682–2690. DOI: 10.1109/CVPR.2019.00279.

[101] S. Arora, N. Cohen, W. Hu and Y. Luo. 'Implicit regularization in deep matrix factorization'. In: *Advances in Neural Information Processing Systems* 32 (2019).

[102] A. Hernández-García and P. König. 'Data augmentation instead of explicit regularization'. In: *CoRR* abs/1806.03852 (2018). arXiv: 1806.03852. URL: http://arxiv.org/abs/1806.03852.

[103] D. G. T. Barrett and B. Dherin. 'Implicit Gradient Regularization'. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=3q5IqUrkcF.

[104] K. Knežević, J. Fulir, D. Jakobović, S. Picek and M. Đurasević. 'Neuro-SCA: Evolving Activation Functions for Side-Channel Analysis'. In: *IEEE Access* 11 (2023). Conference Name: IEEE Access, pp. 284–299. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3232064. URL: https://ieeexplore.ieee.org/document/9998512 (visited on 09/12/2024).

[105] N. B. Erichson, Z. Yao and M. W. Mahoney. 'Jumprelu: A retrofit defense strategy for adversarial attacks'. In: *arXiv preprint arXiv:1904.03750* (2019).

[106] L. Wouters, V. Arribas, B. Gierlichs and B. Preneel. 'Revisiting a methodology for efficient CNN architectures in profiling attacks'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), pp. 147–168.

[107] G. Perin, L. Wu and S. Picek. 'Exploring Feature Selection Scenarios for Deep Learning-based Side-channel Analysis'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022), pp. 828–861. DOI: 10.46586/tches.v2022.i4.828-861.

[108] N.-T. Do, V.-P. Hoang, V. S. Doan and C.-K. Pham. 'On the performance of non-profiled side channel attacks based on deep learning techniques'. en. In: *IET Information Security* 17.3 (Dec. 2022), pp. 377–393. ISSN: 1751-8709. DOI: 10.1049/ise2.12102. URL: http://dx.doi.org/10.1049/ise2.12102.

[109] S. R. Dubey, S. K. Singh and B. B. Chaudhuri. 'Activation functions in deep learning: A comprehensive survey and benchmark'. In: *Neurocomputing* 503 (2022), pp. 92–108.

[110] P. Ramachandran, B. Zoph and Q. V. Le. 'Searching for activation functions'. In: *arXiv preprint arXiv:1710.05941* (2017).

[111] V. Nair and G. E. Hinton. 'Rectified linear units improve restricted boltzmann machines'. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[112] G. Klambauer, T. Unterthiner, A. Mayr and S. Hochreiter. 'Self-normalizing neural networks'. In: *Advances in neural information processing systems*. 2017, pp. 971–980. DOI: 10.5555/3294771.3294864.

[113] S. Rajamanoharan, T. Lieberum, N. Sonnerat, A. Conmy, V. Varma, J. Kramár and N. Nanda. *Jumping Ahead: Improving Reconstruction Fidelity with JumpReLU Sparse Autoencoders*. 2024. arXiv: 2407.14435 [cs.LG]. URL: https://arxiv.org/abs/2407.14435.

[114] L. N. Smith. 'Cyclical learning rates for training neural networks'. In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 464–472.

[115] R. Y. Acharya, F. Ganji and D. Forte. 'InfoNEAT: Information Theory-based NeuroEvolution of Augmenting Topologies for Side-channel Analysis'. In: *CoRR* abs/2105.00117 (2021). arXiv: 2105.00117. URL: https://arxiv.org/abs/2105.00117.

[116] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic and N. Mentens. 'Side-channel analysis and machine learning: A practical perspective'. In: *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*. 2017, pp. 4095–4102.

[117] L. Lerman, G. Bontempi and O. Markowitch. 'A machine learning approach against a masked AES'. In: *Journal of Cryptographic Engineering* 5.2 (2015), pp. 123–139.

[118] K. Ramezanpour, P. Ampadu and W. Diehl. 'SCARL: side-channel analysis with reinforcement learning on the ascon authenticated cipher'. In: *arXiv preprint arXiv:2006.03995* (2020).

[119] D. Shanmugam and P. Schaumont. 'Improving Side-channel Leakage Assessment Using Pre-silicon Leakage Models'. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2023, pp. 105–124.

[120] A. Mohammed and R. Kora. 'A comprehensive review on ensemble deep learning: Opportunities and challenges'. In: *J. King Saud Univ. Comput. Inf. Sci.* 35.2 (2023), pp. 757–774. DOI: 10.1016/J.JKSUCI.2023.01.014. URL: https://doi.org/10.1016/j.jksuci.2023.01.014.

[121] SIMPLE-Crypto. *SMAesH: Technical Documentation*. https://www.simple-crypto.org/activities/smaesh/. Accessed: 2024-07-02. 2023.

[122] R. Johnson and D. Wichern. *Applied multivariate statistical analysis*. 5. ed. Upper Saddle River, NJ: Prentice Hall, 2002. XVIII, 767. ISBN: 0-13-092553-5. DOI: 10.1007/978-3-540-72244-1.

[123] O. Choudary and M. G. Kuhn. 'Efficient template attacks'. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 253–270.

[124] S. Picek, A. Heuser, A. Jovic and L. Batina. 'A Systematic Evaluation of Profiling Through Focused Feature Selection'. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.12 (2019), pp. 2802–2815. DOI: 10.1109/TVLSI.2019.2937365.

[125] S. Wold, K. Esbensen and P. Geladi. 'Principal component analysis'. In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52. DOI: 10.1016/0169-7439(87)80084-9.

[126] C. Archambeau, E. Peeters, F.-X. Standaert and J.-J. Quisquater. 'Template attacks in principal subspaces'. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2006, pp. 1–14. DOI: 10.1007/11894063_1.

[127] L. Batina, J. Hogenboom and J. G. van Woudenberg. 'Getting more from PCA: first results of using principal component analysis for extensive power analysis'. In: *Topics in Cryptology–CT-RSA 2012: The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27–March 2, 2012. Proceedings*. Springer. 2012, pp. 383–397. DOI: 10.1007/978-3-642-27954-6_24.

[128] F.-X. Standaert and C. Archambeau. 'Using subspace-based template attacks to compare and combine power and electromagnetic information leakages'. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2008, pp. 411–425. DOI: 10.1007/978-3-540-85053-3_26.

[129] G. Cassiers, H. Devillez, F.-X. Standaert and B. Udvarhelyi. 'Efficient regression-based linear discriminant analysis for side-channel security evaluations: Towards analytical attacks against 32-bit implementations'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.3 (2023), pp. 270–293. DOI: 10.46586/tches.v2023.i3.270-293.

[130] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede and J. Vandewalle. 'Machine learning in side-channel analysis: a first study'. In: *Journal of Cryptographic Engineering* 1.4 (2011), pp. 293–302.

[131] S. Picek, A. Heuser and S. Guilley. 'Template attack versus Bayes classifier'. In: *Journal of Cryptographic Engineering* 7.4 (Sept. 2017), pp. 343–351. DOI: 10.1007/s13389-017-0172-7. URL: https://doi.org/10.1007/s13389-017-0172-7.

[132] G. Zaid, L. Bossuet, M. Carbone, A. Habrard and A. Venelli. 'Conditional variational autoencoder based on stochastic attacks'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023), pp. 310–357. DOI: 10.46586/tches.v2023.i2.310-357.

[133] L. Zhang, X. Xing, J. Fan, Z. Wang and S. Wang. 'Multilabel deep learning-based side-channel attack'. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.6 (2020), pp. 1207–1216. DOI: 10.1109/TCAD.2020.3033495.

[134] R. Caruana. 'Multitask learning'. In: *Machine learning* 28 (1997), pp. 41–75. DOI: 10.1007/978-1-4615-5529-2_5.

[135] J. Zhang, M. Zheng, J. Nan, H. Hu and N. Yu. 'A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), pp. 73–96. DOI: 10.46586/tches.v2020.i3.73-96.

[136] S. Ruder. 'An overview of multi-task learning in deep neural networks'. In: *arXiv preprint arXiv:1706.05098* (2017).

[137] Y. Zhang and Q. Yang. 'A survey on multi-task learning'. In: *IEEE Transactions on Knowledge and Data Engineering* 34.12 (2021), pp. 5586–5609. DOI: 10.1109/TKDE.2021.3070203.

[138] H. Maghrebi. 'Deep learning based side-channel attack: a new profiling methodology based on multi-label classification'. In: *Cryptology ePrint Archive* (2020).

[139] S. Picek, I. P. Samiotis, J. Kim, A. Heuser, S. Bhasin and A. Legay. 'On the performance of convolutional neural networks for side-channel analysis'. In: *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2018, pp. 157–176. DOI: 10.1007/978-3-030-05072-6_10.

[140] Y.-S. Won, D. Jap and S. Bhasin. 'Push for more: On comparison of data augmentation and SMOTE with optimised deep learning architecture for side-channel'. In: *Information Security Applications: 21st International Conference, WISA 2020, Jeju Island, South Korea, August 26–28, 2020, Revised Selected Papers 21*. Springer. 2020, pp. 227–241. DOI: 10.1007/978-3-030-65299-9_18.

[141] Y.-C. Ho and D. L. Pepyne. 'Simple explanation of the no-free-lunch theorem and its implications'. In: *Journal of optimization theory and applications* 115 (2002), pp. 549–570. DOI: 10.1023/a:1021251113462.

[142] J. S. Bridle. 'Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition'. In: *Neurocomputing: Algorithms, architectures and applications*. Springer, 1990, pp. 227–236. DOI: 10.1007/978-3-642-76153-9_28.

[143] A. Gohr, S. Jacob and W. Schindler. 'Subsampling and knowledge distillation on adversarial examples: New techniques for deep learning based side channel evaluations'. In: *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*. Springer. 2021, pp. 567–592. DOI: 10.1007/978-3-030-81652-0_22.

[144] G. Cassiers, B. Grégoire, I. Levi and F. Standaert. 'Hardware Private Circuits: From Trivial Composition to Full Verification'. In: *IEEE Trans. Computers* 70.10 (2021), pp. 1677–1690. DOI: 10.1109/TC.2020.3022979. URL: https://doi.org/10.1109/TC.2020.3022979.

[145] N. Veyrat-Charvillon, B. Gérard and F. Standaert. 'Soft Analytical Side-Channel Attacks'. In: *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*. Ed. by P. Sarkar and T. Iwata. Vol. 8873. Lecture Notes in Computer Science. Springer, 2014, pp. 282–296. DOI: 10.1007/978-3-662-45611-8_15. URL: https://doi.org/10.1007/978-3-662-45611-8%5C_15.

[146] S. Crypto. *SMAesH Challenge Leaderboard*. Accessed: 2024-07-08. 2024. URL: https://smaesh-challenge.simple-crypto.org/leaderboard.html.

[147] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy and P. T. P. Tang. 'On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima'. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=H1oyRlYgg.

[148] D. Masters and C. Luschi. 'Revisiting small batch training for deep neural networks'. In: *arXiv preprint arXiv:1804.07612* (2018).

[149] R. Y. Acharya, F. Ganji and D. Forte. 'Information theory-based evolution of neural networks for side-channel analysis'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022). DOI: 10.46586/tches.v2023.i1.401-437.

[150] A. Gohr, F. Laus and W. Schindler. 'Breaking masked implementations of the clyde-cipher by means of side-channel analysis: A report on the ches challenge side-channel contest 2020'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022), pp. 397–437. DOI: 10.46586/tches.v2022.i4.397-437.

[151] Y. Fukuda, K. Yoshida, H. Hashimoto, K. Kuroda and T. Fujino. 'Profiling Deep Learning Side-Channel Attacks Using Multi-Label against AES Circuits with RSM Countermeasure'. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 106.3 (2023), pp. 294–305. DOI: 10.1587/transfun.2022cip0015.

[152] P. Hamalainen, T. Alho, M. Hannikainen and T. D. Hamalainen. 'Design and implementation of low-area and low-power AES encryption hardware core'. In: *9th EUROMICRO conference on digital system design (DSD'06)*. IEEE. 2006, pp. 577–583. DOI: 10.1109/dsd.2006.40.

[153] 'IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices'. In: *IEEE Std 1619-2007* (2008), pp. 1–40. DOI: 10.1109/IEEESTD.2008.4493450.

[154] P. Ravi, D. Jap, S. Bhasin and A. Chattopadhyay. 'Machine Learning Based Blind Side-Channel Attacks on PQC-Based KEMs-A Case Study of Kyber KEM'. In: *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE. 2023, pp. 01–07.

[155] Yanis Linge and Cécile Dumas and Sophie Lambert-Lacroix. 'Using the Joint Distributions of a Cryptographic Function in Side Channel Analysis'. In: *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*. Ed. by E. Prouff. Vol. 8622. Lecture Notes in Computer Science. Springer, 2014, pp. 199–213. DOI: 10.1007/978-3-319-10175-0_14. URL: https://doi.org/10.1007/978-3-319-10175-0%5C_14.

[156] C. Clavier and L. Reynaud. 'Improved Blind Side-Channel Analysis by Exploitation of Joint Distributions of Leakages'. In: *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Ed. by W. Fischer and N. Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 24–44. DOI: 10.1007/978-3-319-66787-4_2. URL: https://doi.org/10.1007/978-3-319-66787-4%5C_2.

[157] M. J. Kannwischer, J. Rijneveld, P. Schwabe and K. Stoffelen. 'pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4'. In: *IACR Cryptol. ePrint Arch.* (2019), p. 844. URL: https://eprint.iacr.org/2019/844.

[158] H. Le Bouder. 'Un Formalisme Unifiant Les Attaques Physiques Sur Circuits Crytographiques Et Son Exploitation Afin De Comparer Et Rechercher De Nouvelles Attaques'. Theses. Ecole Nationale Supérieure des Mines de Saint-Etienne, Oct. 2014. URL: https://theses.hal.science/tel-01140014.

[159] C. Clavier, L. Reynaud and A. Wurcker. 'Quadrivariate Improved Blind Side-Channel Analysis on Boolean Masked AES'. In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by J. Fan and B. Gierlichs. Cham: Springer International Publishing, 2018, pp. 153–167. ISBN: 978-3-319-89641-0.

[160] Awaleh Houssein Meraneh and Christophe Clavier and Hélène Le Bouder and Julien Maillard and Gaël Thomas. 'Blind Side Channel on the Elephant LFSR'. In: *Proceedings of the 19th International Conference on Security and Cryptography, SECRYPT 2022, Lisbon, Portugal, July 11-13, 2022*. Ed. by S. D. C. di Vimercati and P. Samarati. SCITEPRESS, 2022, pp. 25–34. DOI: 10.5220/0011135300003283. URL: https://doi.org/10.5220/0011135300003283.

[161] M. Sarry, H. L. Bouder, E. Maaloouf and G. Thomas. 'Blind Side Channel Analysis Against AEAD with a Belief Propagation Approach'. In: *Smart Card Research and Advanced Applications - 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14-16, 2023, Revised Se-*

lected Papers. Ed. by S. Bhasin and T. Roche. Vol. 14530. Lecture Notes in Computer Science. Springer, 2023, pp. 127–147. DOI: 10.1007/978-3-031-54409-5_7. URL: https://doi.org/10.1007/978-3-031-54409-5%5C_7.

[162] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0-387-31073-8.

[163] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. C. Courville, Y. Bengio and S. Lacoste-Julien. 'A Closer Look at Memorization in Deep Networks'. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 233–242. URL: http://proceedings.mlr.press/v70/arpit17a.html.

[164] G. Perin, L. Chmielewski, L. Batina and S. Picek. 'Keep it Unsupervised: Horizontal Attacks Meet Deep Learning'. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.1 (2021), pp. 343–372. DOI: 10.46586/TCHES.V2021.I1.343-372. URL: https://doi.org/10.46586/tches.v2021.i1.343-372.

[165] H. Song, M. Kim, D. Park, Y. Shin and J. Lee. 'Learning From Noisy Labels With Deep Neural Networks: A Survey'. In: *IEEE Trans. Neural Networks Learn. Syst.* 34.11 (2023), pp. 8135–8153. DOI: 10.1109/TNNLS.2022.3152527. URL: https://doi.org/10.1109/TNNLS.2022.3152527.

[166] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. W. Tsang and M. Sugiyama. 'Co-teaching: Robust training of deep neural networks with extremely noisy labels'. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett. 2018, pp. 8536–8546. URL: https://proceedings.neurips.cc/paper/2018/hash/a19744e268754fb0148b017647355b7b-Abstract.html.

[167] Z. Zhang and M. R. Sabuncu. 'Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels'. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett. 2018, pp. 8792–8802. URL: https://proceedings.neurips.cc/paper/2018/hash/f2925f97bc13ad2852a7a551802feea0-Abstract.html.

[168] M. J. Kannwischer, J. Rijneveld, P. Schwabe and K. Stoffelen. *PQM4: Post-quantum crypto library for the ARM Cortex-M4*. https://github.com/mupq/pqm4.

[169] T. Yap, S. Bhasin and L. Weissbart. 'Train Wisely: Multifidelity Bayesian Optimization Hyperparameter Tuning in Deep Learning-Based Side-Channel Analysis'. In: *Selected Areas in Cryptography - SAC 2024 - 31st International Conference, Montreal, QC, Canada, August 28-30, 2024, Revised Selected Papers, Part II*. Ed. by M. Eichlseder and S. Gambs. Vol. 15517. Lecture Notes in Computer Science. Springer, 2024, pp. 294–315. DOI: 10.1007/978-3-031-82841-6_12. URL: https://doi.org/10.1007/978-3-031-82841-6%5C_12.

[170] Suvadeep Hajra and Siddhartha Chowdhury and Debdeep Mukhopadhyay. 'EstraNet: An Efficient Shift-Invariant Transformer Network for Side-Channel Analysis'. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024.1 (2024), pp. 336–374. DOI: 10.46586/TCHES.V2024.I1.336-374. URL: https://doi.org/10.46586/tches.v2024.i1.336-374.

[171] Y. Zheng, Y. Zhou, Z. Yu, C. Hu and H. Zhang. 'How to Compare Selections of Points of Interest for Side-Channel Distinguishers in Practice?' In: *Information and Communications Security - 16th International Conference, ICICS 2014, Hong Kong, China, December 16-17, 2014, Revised Selected Papers*. Ed. by L. C. K. Hui, S. H. Qing, E. Shi and S. Yiu. Vol. 8958. Lecture Notes in Computer Science. Springer, 2014, pp. 200–214. DOI: 10.1007/978-3-319-21966-0_15. URL: https://doi.org/10.1007/978-3-319-21966-0%5C_15.

[172] S. Bhasin, J. Danger, S. Guilley and Z. Najm. 'NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage'. In: *IACR Cryptol. ePrint Arch.* (2013), p. 717. URL: http://eprint.iacr.org/2013/717.

[173] D. Thapar, M. Alam and D. Mukhopadhyay. 'Deep Learning assisted Cross-Family Profiled Side-Channel Attacks using Transfer Learning'. In: *22nd International Symposium on Quality Electronic Design, ISQED 2021, Santa Clara, CA, USA, April 7-9, 2021*. IEEE, 2021, pp. 178–185. DOI: 10.1109/ISQED51717.2021.9424254. URL: https://doi.org/10.1109/ISQED51717.2021.9424254.

[174] D. Thapar, M. Alam and D. Mukhopadhyay. 'Transca: Cross-family profiled side-channel attacks using transfer learning on deep neural networks'. In: *Cryptology ePrint Archive* (2020).

[175] H. Yu, H. Shan, M. Panoff and Y. Jin. 'Cross-device profiled side-channel attacks using meta-transfer learning'. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2021, pp. 703–708.

[176] P. Dobias, **A. Rezaeezade**, Ł. Chmielewski, L. Malina and L. Batina. ‘SoK: Reassessing Side-Channel Vulnerabilities and Countermeasures in PQC Implementations’. In: *Cryptology ePrint Archive* (2025).

Acknowledgements

This thesis brings together the research I carried out during my PhD, but it does not tell the story of my PhD journey. That is why I see these last few pages as a good place to reflect a little on that journey: both by thanking the people who walked with me along the way, and by looking back at what I went through myself. I believe both belong to the meaning of acknowledgement, right?

For me, the PhD journey was not only about research. It was also a period of developing new skills, learning more about myself, and becoming more comfortable with being pushed beyond what felt familiar and safe. Along the way, I learned not only how to approach scientific problems, but also how to handle uncertainty, work independently, explain my thoughts more clearly, and build a life in a new environment. As is almost always the case, this learning did not happen in a vacuum, but through interactions with people and situations along the way.

First of all, I should thank my supervisors. I owe a great deal to you, Stjepan. Among many other things, you constantly pushed me out of my comfort zone and encouraged me to go beyond what I thought I was capable of. You also taught me not to take everything too seriously and to leave room for fun and jokes, even if my face may still not always show it. I would also like to thank you, Lejla. I learned a great deal from you as well, but what I especially appreciate is the way you supported me whenever I needed guidance, while also giving me the freedom to find my own way when I could. Inald, although our interactions were fewer, you were also very influential in this journey. I hope I managed to learn a little from you about how to ask good questions and how to think like a researcher.

My PhD journey started during the lockdown. That sentence alone already says enough. Still, to give a sense of what that meant, imagine arriving in a new country, while even going to the office required making sure that no one else was there. This is why I want to thank Huimin, Sandra, Lukasz, Luka, Marina, Jing, and Guilherme for receiving me with such kindness. Despite all the restrictions and social distancing, you found ways to make me feel less alone: by meeting me outside, checking in on me, visiting me at home, or inviting me to your homes. For me, these acts of kindness meant a lot.

This PhD journey also meant being separated from my family and friends, and from the culture that felt familiar to me. While this separation brought a sense of loss, it also opened up space for learning a new culture and making new friends. I was very lucky to join a group with a large Iranian community. This gave me the chance to sometimes feel at home while I was trying to integrate into a completely new environment. For this, I would like to thank Zahra, Shahram, Behrad, Behnam, Omid, Vahid, the two Parisas, and Shiva. I remember all the afternoons when we stayed late, played foosball, or shared food together. Those moments made the

difficulty of being far away from my family and close friends much more bearable.

About a year and a half into my PhD, I finally started to feel enthusiastic and happy in the Netherlands. That was also the time when life slowly began to feel normal again after the COVID period. Travelling became possible again, coffee breaks were full of colleagues and sometimes homemade cakes, and lunch breaks became fun and energising. I would chat with colleagues until late in the afternoon, and we had borrels every Friday. All the laughter and shared moments in the office made the journey lighter and turned the demanding work of research into something much more enjoyable. For this, I would like to thank many people, including Asmita, Yanis, Abraham, Leo, Tom, Peter, Thomas, Trevor, Estuardo, Krijn, Durba, Alex, Silvia, Ileana, Monika, Lars, Runny, Janet, Paulus, Shanly, Mario, Konstantina, Charlotte, Tijs, Gorka, Cris, Melvin, Patrik, and many others. I cannot mention every wonderful memory we shared together, but I am sure you can recall some of them yourselves.

But that was still not the whole story. I was happy here, but I did not yet feel at home. I knew very little about this new country and its culture. I did not know what the norms and values, traditions, and communication style in the Netherlands were. I also did not know all the small, funny, and slightly *gekke* details that *Nederlanders* grow up with. To feel more integrated, I needed to learn them from the source. This is where you really helped me: Olga, Cas, Lizzy, Amber, Gerard, and Bart. All the conversations, practical help, and shared moments made me feel accepted into this new society, which I have come to experience as peaceful and kind.

I was also very lucky to have always had the support of my family, not only during this PhD, but also throughout other parts of my life. I would like to write the next few lines in Persian, the language in which I can most naturally thank them.

پدر و مادر عزیزم، این بخش از تز من به تشکر از کسانی اختصاص دارد که در مسیر گرفتن مدرک دکترا همراه و پشتیبان من بودند؛ و در این میان، من خود را بیش از همه مدیون شما می‌دانم. در این چند سطر می‌خواهم از شما تشکر کنم؛ برای همه‌ی چیزهای خوبی که به من آموختید، برای ایمانی که همیشه به من داشتید، و برای اینکه در تمام این سال‌ها تکیه‌گاه من بودید. می‌دانم که هر دوی شما فداکاری‌های بسیاری کردید تا من بتوانم رویاهایم را دنبال کنم، و من خود را بسیار خوشبخت می‌دانم که تا امروز، در سایه‌ی مهر و حمایت شما، به بسیاری از آن‌ها رسیده‌ام. آفاق، پروین و محمدجواد عزیزم، از روزی که به هلند آمدم، روزی نبوده که به شما فکر نکنم. تلخ‌ترین بخش این مسیر برای من دور بودن از شما بوده است. می‌دانم که به عنوان خواهر بزرگ‌تر، چیزهای زیادی به شما بدهکارم؛ مثل اینکه نزدیکتان نیستم تا بیشتر با شما وقت بگذرانم، در شادی‌هایتان شریک باشم، و در روزهای سخت همراهیتان کنم. امیدوارم مرا برای این دوری ببخشید. هنوز هم امیدوارم روزی دوباره همه‌ی ما کنار هم جمع شویم.

And finally, you, Esmaeil. This journey would not have been as joyful, fulfilling, or successful without your support, encouragement, and understanding. I would not have made it this far without you listening every day to all my exciting, stressful, frustrating, and complaining stories from the office. Thank you for saying yes to this journey from the very beginning, without complaint, even though it meant giving up many things. Thank you for making sure that I did not have to worry about other things, and for giving me the space to focus on what I felt I needed during this journey. This journey would not have been possible without you being my “hamsafar”.

List of Publications

8. P. Dobias, **A. Rezaeezade**, Ł. Chmielewski, L. Malina and L. Batina. 'SoK: Reassessing Side-Channel Vulnerabilities and Countermeasures in PQC Implementations'. In: *Cryptology ePrint Archive* (2025)

7. **A. Rezaeezade**, T. Yap, D. Jap, S. Bhasin and S. Picek. 'Side-Channel Power Trace Dataset for Kyber Pair-Pointwise Multiplication on Cortex-M4'. In: *Cryptology ePrint Archive* (2025)

6. **A. Rezaeezade**, T. Yap, D. Jap, S. Bhasin and S. Picek. 'Breaking the Blindfold: Deep Learning-based Blind Side-channel Analysis'. In: *Cryptology ePrint Archive* (2025)

5. A. Basurto-Becerra, **A. Rezaeezade** and S. Picek. 'Jump, It Is Easy: JumpReLU Activation Function in Deep Learning-based Side-channel Analysis'. In: *Cryptology ePrint Archive* (2025)

4. **A. Rezaeezade** and L. Batina. 'Regularizers to the rescue: fighting overfitting in deep learning-based side-channel analysis'. In: *Journal of Cryptographic Engineering* (2024), pp. 1–21. DOI: 10.1007/s13389-024-00361-5

3. **A. Rezaeezade**, A. Basurto-Becerra, L. Weissbart and G. Perin. 'One for All, All for Ascon: Ensemble-Based Deep Learning Side-Channel Analysis'. In: *Applied Cryptography and Network Security Workshops*. Ed. by M. Andreoni. Cham: Springer Nature Switzerland, 2024, pp. 139–157. ISBN: 978-3-031-61486-6

2. L. Wu, **A. Rezaeezade**, A. Alipour, G. Perin and S. Picek. 'Leakage Model-flexible Deep Learning-based Side-channel Analysis'. In: *IACR Commun. Cryptol.* 1.3 (2024), p. 41. DOI: 10.62056/AY4C3TXOL7. URL: https://doi.org/10.62056/ay4c3txol7

1. **A. Rezaeezade**, G. Perin and S. Picek. 'To overfit, or not to overfit: improving the performance of deep learning-based SCA'. In: *International Conference on Cryptology in Africa*. Springer. 2022, pp. 397–421

Cryptographic algorithms are essential for protecting sensitive data in digital systems, but their security does not depend only on their mathematical design. Once implemented on real devices, their execution may reveal data-dependent physical behavior. Side-channel analysis exploits such unintended leakage to infer secrets such as cryptographic keys. For this reason, evaluating the side-channel resistance of cryptographic implementations is an important task for security laboratories, yet it is also demanding because evaluators must work within limited time, budget, data, and computational resources.

This thesis focuses on the role of deep learning in making such evaluations more effective and practical. Deep learning-based side-channel analysis (DL-SCA) can reduce evaluation costs by relaxing certain assumptions and reducing the need for some manual effort. However, it also brings its own challenges. The main goal of this thesis is to study how DL-SCA can be used in a more reliable and cost-aware way: by lowering the effort needed for model selection, improving generalization, and reducing dependence on fixed leakage models or access to known data.