Implementation Attacks Powered by Artificial Intelligence

Krcek, M.

DOI
10.4233/uuid:bc7aec5a-1d38-4c8d-8222-9f8630fdb36f

Publication date
2024

Document Version
Final published version

Citation (APA)
Krcek, M. (2024). *Implementation Attacks Powered by Artificial Intelligence*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:bc7aec5a-1d38-4c8d-8222-9f8630fdb36f

Implementation Attacks Powered by Artificial Intelligence

IMPLEMENTATION ATTACKS POWERED BY ARTIFICIAL INTELLIGENCE

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. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
Monday 21 October 2024 at 12:30 o'clock

by

Marina KRČEK

Master of Science in Computing,
University of Zagreb, Croatia,
born in Zagreb, Croatia.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. R.L. Lagendijk,	Delft University of Technology, *promotor*
Dr. S. Picek,	Delft University of Technology, *copromotor*
Independent members:	
Prof. dr. P.A.N. Bosman	Delft University of Technology & CWI, The Netherlands
Prof. dr. ir. N. Mentens	Leiden University, The Netherlands & KU Leuven, Belgium
Prof. dr. ir. P.R. Schaumont	Worcester Polytechnic Institute, USA
Prof. dr. G. Smaragdakis	Delft University of Technology, The Netherlands
Dr. ir. R.C. Hendriks	Delft University of Technology, The Netherlands

Keywords: Implementation Attacks, Deep Learning, Evolutionary Algorithms

Printed by: ProefschriftMaken

Cover by: Microsoft Copilot (DALL-E 3) & Marina Krček

ISBN 978-94-6384-649-3

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

CONTENTS

ACRONYMS

List of Figures

List of Tables

SUMMARY

In an era of increasing reliance on digital technology, securing embedded and interconnected devices, such as smart cards or Internet of Things (IoT) devices, against emerging threats becomes crucial, highlighting the need for advanced security measures. Cryptographic algorithms, essential for secure communication, data storage, and transaction integrity, are often employed to develop secure systems. However, the practical implementation of these algorithms in software and hardware introduces vulnerabilities, exposing sensitive information to risks. Implementation attacks, such as fault injection (FI) and side-channel analysis (SCA), belong to a category of security threats that exploit these vulnerabilities occurring during the cryptographic algorithms' execution.

Security evaluation and certification assess the product's security features against industry best practices and regulatory standards. These processes aim to independently verify the claims made about the product's security, fostering and maintaining trust among users. Given the evolving landscape of security threats and increasing security concerns, the need for more efficient and resource-effective security evaluations has become evident. Fault injection and side-channel analysis are commonly conducted as part of this assessment, and recent studies have demonstrated that integrating artificial intelligence (AI) methods can significantly enhance their performance. Moreover, this integration can provide more automated and optimized attacks for security evaluation.

This thesis aims to advance AI-based implementation attacks by investigating current AI frameworks, with the objective of improving the efficiency and effectiveness of these attacks across various scenarios. We target specific challenges within AI-based fault injection (AIFI) and deep learning-based SCA (DLSCA), addressing gaps in the current methodologies and proposing solutions that significantly impact their performance and efficiency. We focus on hyperparameter tuning of the utilized AI methods, portability of the attacks, and alternative evaluation metrics within the AI frameworks.

Hyperparameter tuning is critical but can be a time-intensive process. By investigating specific hyperparameters, we can identify those crucial for the performance, guiding a more efficient tuning process. This thesis focuses on initialization methods, revealing no universally optimal initialization method. Instead, we offer a strategic approach to selecting initialization methods that can lead to improved and more reliable performance in specific scenarios. Next, we provide practical AI-based solutions to enhance the portability of FI parameter search results across different samples of the same target and SCA profiling models across different public datasets (targets). This approach makes security evaluation more efficient by leveraging data and findings to expedite evaluations on other targets. Furthermore, this enables future efforts to develop universal methods to help standardize the AI-based implementation attacks for security evaluation. Lastly, we revisit and refine evaluation metrics within the AI-based implementation attacks, proposing new metrics better aligned with the considered objectives. We present new

metrics for evaluating the performance of AI-based FI parameter search to find distant vulnerable regions of the target alongside algorithms for this objective. On the other hand, we improve the training process of DLSCA by introducing a training scheme involving the redefinition of the labels and a metric that can evaluate the generality of the profiling model, enabling better assessment for early stopping and model tuning.

Through its exploration of AI-based implementation attacks, this thesis offers valuable insights and practical solutions that significantly enhance the field. By improving the efficiency and effectiveness of AI-based implementation attacks, this research not only aids security analysts but also offers a foundation for future standardization efforts of these attacks for security evaluation.

SAMENVATTING

In een tijdperk waarin we toenemend afhankelijk zijn van digitale technologie, wordt het beveiligen van *embedded devices*, zoals *smartcards* en *Internet of Things (IoT)*-apparaten, tegen opkomende bedreigingen cruciaal, wat de behoefte aan geavanceerde beveiligingsmaatregelen benadrukt. Cryptografische algoritmen, essentieel voor beveiligde communicatie, data-opslag en integriteit van transacties, worden vaak gebruikt om beveiligde systemen te ontwikkelen. Echter, de praktische implementatie van deze algoritmen in *software* en *hardware* introduceert kwetsbaarheden, waardoor gevoelige informatie aan risico's wordt blootgesteld. Aanvallen op implementaties, zoals *fault injection (FI)* en *side-channel analysis (SCA)*, behoren tot een categorie van beveiligingsbedreigingen die zulke kwetsbaarheden tijdens de uitvoering van de cryptografische algoritmen exploiteren.

De evaluatie en certificering van beveiliging beoordeelt de beveiligingskenmerken van een product aan de hand van de beste praktijken in de industrie en aan de geldende wettelijke normen. Zulke certificeringen hebben tot doel de *claims* over de beveiliging van het product onafhankelijk te verifiëren, waardoor het vertrouwen onder gebruikers wordt bevorderd en behouden. Gezien het veranderende landschap van beveiligingsbedreigingen en toenemende beveiligingszorgen, is er een duidelijke behoefte aan efficiëntere en kosteneffectievere beveiligingsevaluaties. *Fault injection* en *side-channel analysis* worden vaak uitgevoerd als onderdeel van zulke beoordelingen. Recent heeft onderzoek aangetoond dat het integreren van kunstmatige intelligentie (AI) in dergelijke evaluaties hun prestaties aanzienlijk kan verbeteren. Bovendien kan deze integratie zorgen voor meer geautomatiseerde en geoptimaliseerde aanvallen voor beveiligingsevaluatie.

Dit proefschrift heeft als doel AI-gebaseerde implementatie-aanvallen te verbeteren door huidige AI *frameworks* te onderzoeken, met als doel de efficiëntie en effectiviteit van deze aanvallen in verschillende scenario's te verbeteren. We richten ons op specifieke uitdagingen binnen AI-gebaseerde FI (AIFI) en *deep learning*-gebaseerde SCA (DLSCA), waarbij we hiaten in de huidige methodologieën aanpakken en oplossingen voorstellen die hun prestaties en efficiëntie aanzienlijk beïnvloeden. We richten ons op *hyperparameter tuning* van de gebruikte AI methoden, de draagbaarheid van de aanvallen en alternatieve evaluatiemetrieken binnen deze AI *frameworks*.

Hyperparameter tuning is cruciaal, maar kan een tijdrovend proces zijn. Door specifieke *hyperparameters* te onderzoeken, kunnen we identificeren welke cruciaal zijn voor de prestaties, wat leidt tot een efficiënter tuningproces. Dit proefschrift richt zich op initialisatiemethoden, waarbij geen universeel optimale initialisatiemethode wordt onthuld. In plaats daarvan bieden we een strategische benadering voor het selecteren van initialisatiemethoden die kunnen leiden tot verbeterde en meer betrouwbare prestaties in specifieke scenario's. Vervolgens bieden we praktische AI-gebaseerde oplossingen om

de draagbaarheid van FI-parameterzoekresultaten over verschillende monsters van hetzelfde doel en SCA-profileringsmodellen over verschillende openbare *datasets* (doelen) te verbeteren. Deze benadering maakt beveiligingsevaluatie efficiënter door gegevens en bevindingen te benutten om evaluaties van andere doelen te versnellen. Bovendien stelt dit toekomstige inspanningen in staat om universele methoden te ontwikkelen die helpen bij het standaardiseren van AI-gebaseerde implementatie-aanvallen voor beveiligingsevaluatie. Ten slotte herzien en verfijnen we evaluatiemetrieken binnen de AI-gebaseerde implementatie-aanvallen en stellen we nieuwe metrieken voor die beter zijn afgestemd op de beschouwde doelstellingen. We presenteren nieuwe metrieken voor het evalueren van de prestaties van AI-gebaseerde FI-parameterzoekalgoritmen om goed verstopte kwetsbaarheden van het doelwit te vinden, samen met algoritmen voor dit doel. Eveneens verbeteren we het trainingsproces van DLSCA door een trainingsschema te introduceren dat de herdefiniëring van de *labels* omvat en een metriek die de generaliteit van het profileringsmodel kan evalueren, waardoor een betere beoordeling voor vroegtijdig stoppen en voor *model tuning* mogelijk is.

Door de verkenning van AI-gebaseerde implementatie-aanvallen biedt dit proefschrift waardevolle inzichten en praktische oplossingen die het veld aanzienlijk verbeteren. Door de efficiëntie en effectiviteit van AI-gebaseerde implementatie-aanvallen te verbeteren, helpt dit onderzoek niet alleen beveiligingsanalisten, maar biedt het ook een basis voor toekomstige inspanningen voor standaardisatie van deze aanvallen voor beveiligingsevaluatie.

1

INTRODUCTION

Technological advancements, from the first computers to the inception of the Internet, have formed an era characterized by the ubiquitous integration of technology into our daily lives, from personal computers to the widespread use of embedded computing devices. Embedded devices, known for their compact size and low power consumption, are well-suited for systems constrained by size, weight, or power limitations. Typically, these devices are assigned specific functions and can either be integrated into a larger computing system or operate independently. Such devices range from smart cards, biometric devices, and security keys to smartphones and numerous Internet of Things (IoT) devices, including wearables like smartwatches and fitness trackers and smart home appliances like vacuum cleaners or refrigerators. In 2022, the number of global IoT connections grew by 18% from the previous year, reaching 14.3 billion active IoT devices. Projections suggest that by 2027, there will be approximately 29.7 billion connected IoT devices worldwide, equating to around four devices per person [1]. Inadequate security measures in embedded devices can have far-reaching and catastrophic consequences across various sectors and for different stakeholders, including users, manufacturers, and society at large. The use of these devices in sectors like healthcare and autonomous vehicles can potentially result in severe life-threatening consequences. In the financial industry, security failures can lead to significant monetary losses through unauthorized access to banking systems or fraudulent transactions. This not only affects individual customers but can also undermine the stability of financial institutions and the broader economy. For manufacturers, the consequences of poor security measures include loss of consumer trust, legal liabilities, and financial penalties, which can ultimately lead to a decline in market share and profitability. Moreover, the interconnected nature of modern technology means that a breach in one device can potentially provide access to broader network infiltration, increasing the damage. Therefore, ensuring robust security in embedded devices is not just a technical necessity but a critical component of maintaining trust, safety, and functionality in an increasingly interconnected world.

Cryptographic algorithms, ensuring secure communication, data storage, and transaction integrity, play a crucial role in preserving the confidentiality and integrity of sensitive information. Consequently, they are often employed in designing

and producing secure devices. Despite being carefully designed within theoretical frameworks, these algorithms face significant challenges when implemented using software and hardware for practical deployment. A sophisticated category of security threats, known as implementation attacks, targets vulnerabilities that emerge during the cryptographic algorithms' execution. These attacks thereby compromise the robustness of systems designed to protect confidential processes and information.

Therefore, manufacturers commonly comply with standards set by governmental and private certification bodies designed for the devices' intended purpose and functionality. During the manufacturing process, these devices undergo an internal security evaluation. The depth and techniques employed in this evaluation vary, depending on the manufacturer's judgment and discretion. In response to growing concerns about the security of connected devices and their data, manufacturers increasingly search for assistance from independent experts. Accredited security laboratories are crucial in assessing a product's security features to ensure they align with industry best practices and comply with existing and emerging laws, regulations, and baseline requirements. These third-party evaluations facilitate product certification, enhancing user trust. Two key standards in this field are the Common Criteria for Information Technology Security Evaluation, an international standard (ISO/IEC 15408) for computer security certification [2], and the U.S. government computer security standard, Federal Information Processing Standard (FIPS) 140-3 [3]. FIPS 140-3 outlines specifications for cryptographic modules, including requirements for protection against Fault Injection (FI) and guidance for testing against non-invasive attacks, like Side-channel Analysis (SCA). Given the global increase in reliance on interconnected secure devices and the evolving threat landscape, the security evaluation process must be both efficient and reliable, providing confidence in the observed results. This thesis aims to contribute to this field by enhancing implementation attacks, focused on fault injection and side-channel analysis, to support security analysts in performing more efficient and reliable security evaluations.

Artificial Intelligence (AI) has revolutionized various domains, from healthcare and finance to autonomous vehicles and beyond, demonstrating an impressive capacity to analyze large amounts of data, recognize patterns, and make informed decisions [4, 5]. The success of AI in these areas stems from its ability to automate and enhance tasks beyond the scope of manual human effort, leading to improved efficiency, accuracy, and insights. AI has been shown to provide the same benefits within cybersecurity, like Denial-of-service (DoS) and Distributed Denial-of-service (DDoS) intrusion and malware detection [6]. Considering its remarkable achievements, it is reasonable to broaden the application of AI to implementation attacks. Moreover, certain AI methods align well with the execution processes of the two considered implementation attacks, promising notable enhancements. In particular, evolutionary algorithms and deep learning methods have been explored for fault injection and side-channel analysis. Evolutionary Algorithms (EAs) are optimization techniques, and since fault injection includes a parameter search, which can be easily translated into an optimization problem, these algorithms were a natural choice. They are particularly effective for this problem because they can efficiently navigate complex,

multidimensional search spaces to identify optimal or near-optimal parameters for successful attacks, adapting to the problem's landscape without requiring precise mathematical formulations or gradient information. EAs were primarily used for fault injection to help find more vulnerabilities on devices compared to commonly used methods like random search through FI parameter search. However, we note that research on AI-based fault injection is still relatively new and unexplored. At the same time, investigating deep learning methods for side-channel analysis is already more extensive in scope. Integration of Deep Learning (DL) was relatively straightforward as the supervised learning scheme and profiling side-channel analysis both consist of a two-step approach. The benefits of this integration were significant considering the attack performance [7]. This thesis further explores AI methods for implementation attacks, leveraging various AI methods for more efficient execution of Artificial Intelligence-based Fault Injection (AIFI) and DLSCA, which implicitly makes security evaluation more efficient.

This introduction chapter first explains the cryptography concepts necessary to understand some technical chapters later in the thesis. The cryptography section is followed by explanations of implementation attacks and their existing challenges. Then, artificial intelligence is introduced to help understand the AI-based frameworks for the two implementation attacks, which are the main focus of the thesis. Note that these explanations are high-level, while detailed descriptions are covered within each chapter as necessary. This chapter ends with a problem statement, contributions, and a thesis outline.

1.1. CRYPTOGRAPHY

Cryptography is the practice and study of securing communication and protecting data against adversaries, aiming to ensure information confidentiality, integrity, and authenticity. Therefore, it is crucial for providing secure online communication, data storage, and many other applications. As technology evolves, cryptographic techniques continue to advance to address new challenges and maintain the security of digital systems. Modern cryptography encompasses a range of techniques, including digital signatures, authentication, and secure computation [8].

Fundamental concepts and components of cryptography include encryption and decryption. Encryption converts readable plaintext into ciphertext (encrypted data), while decryption reverses this process, turning ciphertext back into plaintext. Modern cryptography is commonly divided into symmetric-key and asymmetric-key (public-key) cryptography and hash functions. In symmetric-key cryptography, the same key is used for encryption and decryption, which makes the management and distribution of this secret key a critical part of these cryptosystems. On the other hand, asymmetric-key (public-key) cryptography uses a pair of keys - a public key for encryption and a private key for decryption. The public key can be freely distributed, but the private key must be kept secret. The encrypted data with a corresponding public key can be decrypted only using the correct private key. Asymmetric-key cryptography does not require secure communication to establish the secret keys, as there is a public-private key pair, and enables message authenticity checking.

However, these algorithms are slower than their symmetric counterparts. Hash functions compute a digest of an input, a fixed-length output referred to as a hash value, without using keys. The hash value represents a unique representation of the input data, which finds its application for digital signature schemes, message authentication codes, password hashes, and similar. In practice, these algorithms are often used together for cryptographic applications because of their specific strengths and weaknesses in what is referred to as hybrid systems.

1.1.1. ADVANCED ENCRYPTION STANDARD (AES)

In 1997, the U.S. National Institute of Standards and Technology (NIST) initiated a search for a new Advanced Encryption Standard (AES) to replace the previous Data Encryption Standard (DES). NIST specifically required submissions to be symmetric key block ciphers that encrypt data in fixed-size blocks, supporting 128-bit block sizes and key sizes of 128, 192, and 256 bits. In 2000, NIST selected the Rijndael algorithm [9] as the AES, meeting all the specified criteria [10]. The AES algorithm has since been widely adopted in various applications, from wireless security to processor security and file encryption. Given that this thesis partially relies on understanding the AES cryptographic algorithm, we describe the AES cipher's main characteristics and operational flow. AES is a symmetric block cipher algorithm, encrypting and decrypting data in blocks of 128 bits arranged in 16 bytes in a four-by-four matrix, known as the *state* of the AES algorithm. The AES encryption process involves several rounds of transformation, with the number of rounds depending on the key size: 10 rounds for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys. Each round consists of several steps, with the final round slightly differing from the others.

AES operational flow. The AES algorithm starts with the initial round before the main rounds begin. The initial round consists only of the `AddRoundKey` operation, where the round key is combined with the plaintext block by a simple bitwise XOR (exclusive or) operation between the plaintext and the key. For each round, a new round key consisting of 128 bits is derived from the primary key using a key schedule. `AddRoundKey` operation introduces the key into the AES state at the beginning, ensuring that subsequent operations depend on the key. This intermediate state is further processed in the following main rounds. Each of the main rounds consists of the following steps:

- `SubBytes` (Substitution Layer): This step uses a Substitution box (S-box) to perform a byte-by-byte substitution of the state. Each byte of the state is replaced with another byte according to the S-box, which is designed to be non-linear and resistant to known cryptographic attacks. This step obscures the relationship between the key and ciphertext, constituting the confusion operation.

- `ShiftRows` (Permutation Layer): This step shifts the rows of the state cyclically. The first row is not shifted, while the second row is shifted one byte to the

left, the third two bytes to the left, and the fourth three bytes to the left. This step disperses the bytes of the state, contributing to diffusion, which aims to hide the statistical properties of the plaintext by spreading the influence of one plaintext symbol over many ciphertext symbols.

- `MixColumns`: This step is a linear transformation that mixes each column of the state matrix. Each four-byte column is multiplied by a fixed four-by-four matrix with constant entries. Since every input byte influences four output bytes, the `MixColumns` operation is the major diffusion element.

- `AddRoundKey`: The round key is added to the state. As with the initial round, this is done via a bitwise XOR.

The final round, the 10th round considering the AES-128, follows the same steps, excluding only the `MixColumns` operation. The decryption process in AES is similar but uses the inverse operations of each step, ensuring that the original plaintext is retrievable only with the correct key. More on the AES algorithm can be found in [9, 11].

1.1.2. CRYPTOANALYSIS

Cryptanalysis is the study and practice focused on analyzing and understanding hidden aspects of information systems. It aims to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the cryptographic key is unknown. Essentially, cryptanalysis involves deciphering information, discovering the secret keys, finding vulnerabilities, and establishing the robustness of the cryptosystem. The practice dates back to ancient times but has become increasingly complex with the evolution of more advanced cryptographic techniques. Central to the field is Kerckhoffs' principle, which states that a cryptographic system should be secure even if everything about the system, except the key, is public knowledge [12]. Cryptanalysis employs various methods to examine and challenge the security of cryptographic algorithms [8]. Classical approaches like analytical and brute-force attacks challenge the theoretical resilience of cryptographic algorithms. However, the real-world deployment of these systems reveals additional vulnerabilities. For instance, social engineering targets the human element, manipulating users to bypass security measures. More subtly, implementation attacks focus on the flaws not in the cryptographic algorithms themselves but in how they are implemented and executed in hardware and software. These vulnerabilities can arise from a range of issues, including poor coding practices, inadequate system configuration, and physical susceptibilities of the hardware. These techniques are not only for malicious purposes but are crucial in testing and reinforcing the security of cryptographic methods. By identifying potential vulnerabilities, cryptanalysis helps ensure that the cryptographic systems are resilient against various attacks, aligning with Kerckhoffs' principle and contributing to the overall security and reliability of communications and data storage.

1

1.2. IMPLEMENTATION ATTACKS

Implementation attacks exploit vulnerabilities in the implementation of cryptographic algorithms rather than theoretical weaknesses. These attacks are particularly interesting as they target the gap between the theoretical design and the implementation. Two primary forms of these attacks are fault injection and side-channel analysis.

1.2.1. FAULT INJECTION (FI)

Fault injection attacks deliberately induce errors in a system's hardware or software to cause it to malfunction in ways that compromise security or reveal sensitive information. For instance, a successful FI attack might cause a system to output incorrect data, such as encryption and decryption outputs, bypass security checks, or expose cryptographic keys. This thesis focuses on physical fault injection, where external sources are used to expose hardware beyond its intended operating limits to provoke this erroneous behavior.

Physical fault injections can be categorized into non-invasive, semi-invasive, and invasive types of fault injection. Non-invasive do not require physical modifications to the target. Techniques like voltage and clock glitching or temperature variations can induce faults without direct contact. A semi-invasive setup requires the chip to be exposed to the injection source but does not involve direct contact with the internal components. Thus, depackaging might be necessary, and there are two main methods - chemical decapsulation, using acids to dissolve the layers covering the silicon die, and mechanical decapsulation, which uses mechanical milling devices to reach the target surface [13]. Invasive attacks involve direct interference with the chip's internal components, often requiring depackaging and physical modification of the hardware.

Semi- and non-invasive fault injection attacks like Laser Fault Injection (LFI), Electromagnetic Fault Injection (EMFI), and voltage glitching are commonly executed in two phases. Initially, the attacker defines injection parameters to cause the device's erroneous behavior. Therefore, this is a parameter search to find suitable FI parameters to obtain specific types of faults, known as fault models. Fault models can range from bit flips, where the bit value is modified to its opposite, and stuck-at faults to instruction skips. The impact of faults varies depending on their duration, and they can be categorized as either transient or persistent. LFI offers high precision and repeatability in inducing errors, assuming highly capable attackers, making it a common choice for security testing in the industry [14]. Therefore, this work focuses on LFI, which falls under the optical FI type introduced by Skorobogatov et al. [15]. In the second step of the attack, the attacker performs fault analyses to exploit the induced behavior. Some of the popular fault analysis methods used for FI attacks are Differential Fault Analysis (DFA) [16], Fault Sensitivity Analysis (FSA) [17], Statistical Fault Attack (SFA) [18], Ineffective Fault Attacks (IFA) [19], and Statistical Ineffective Fault Attacks (SIFA) [20]. These fault analysis methods can use the FI effects differently [21, 22]. For example, considering attacks on cryptographic algorithms, using DFA, the attacker can deduce information about secret keys from the pairs

of correct (non-faulty) and incorrect (faulty) outputs (ciphertexts). In contrast, with FSA, only faulty outputs are analyzed.

CHALLENGES AND NEW DEVELOPMENTS FOR FI

In fault injection, attackers constantly advance their techniques, bypassing existing defenses to target emerging technologies. Researchers must proactively design innovative detection and mitigation strategies to protect against these developments. Meanwhile, these attacks are also a part of security evaluations, necessitating preparation for the worst-case scenarios. Thus, research also focuses on the FI attacks, aiming to understand and simulate the most realistic and challenging conditions.

Challenges within the FI attacks include creating realistic fault models that accurately replicate potential threats, which is crucial for understanding and preparing for possible attacks. Fault injection experiments are particularly time-consuming and resource-intensive, underscoring the need for developing more efficient and scalable methods. As the effectiveness of FI varies with environmental conditions, ensuring robustness across diverse scenarios is imperative. Furthermore, as systems become more complex, the manual FI process becomes highly impractical, highlighting the urgency of automating these processes for enhanced efficiency. Evaluating the security of a system under FI threat is an essential but complex task, necessitating clear metrics and methodologies to understand and clearly communicate the security implications of injected faults.

Determining optimal parameters for fault injection is a non-trivial task, given the large search space and the unique responses different systems may have to the injected faults. For example, the authors in [23] reported the exhaustive FI parameter search would take 29 203 years. Security analysts often decrease parameter ranges, requiring knowledge about the target's internal design. This limitation can lead to inadequate testing or missed vulnerabilities. Within security evaluation, specific parameters can be explored in a grid-like manner, while others are fixed to particular values, and this can be done for several fixed combinations. The fixed parameters can be set based on previous experience and knowledge, where information about the target layout is commonly unavailable to attackers. That, however, can have an opposite effect if the target or used equipment and setup are different [24]. As an alternative, random search was commonly applied as it can be less time-consuming. However, this search is unreliable, and the results obtained might not accurately represent the system's vulnerability level. Therefore, there is a need to enhance the FI parameter search process to identify potential vulnerabilities more effectively and efficiently, enabling it to find more potential vulnerabilities within the same amount of time and resources or less.

To address these challenges, researchers increasingly turn to artificial intelligence, a promising tool for automating the fault injection process, optimizing parameter selection, and analyzing results more effectively. The problem of selecting suitable fault injection parameters can be easily translated into an optimization problem, for which EAs, such as Genetic Algorithms (GAs), have been shown to perform well. There are several papers where evolutionary techniques are used to optimize fault

injection attacks. Genetic algorithms were used for voltage glitching [25, 26], as well as memetic algorithms [27], which were also used for EMFI [23]. This thesis also relies on these algorithms, so we explain them in Section 1.3.1 and corresponding chapters that utilize them.

The process of finding FI parameters that lead to possible vulnerabilities has to be repeated when the target is changed or the bench and the setup used for performing the injections. Moreover, even when different samples of the same target are used, the obtained results might be misleading, and the evaluation has to be repeated. This raises the issue of the portability[1]. Portability, in this context, refers to the ability to apply the findings and FI parameters between different setups, targets, or samples while still obtaining consistent results. It also includes the adaptability of algorithms to modifications, ensuring they perform well under various changes. For example, the authors in [24] presented a method for fast characterization of the laser fault injection settings on the target. The technique can be used for other semi-invasive fault injection attacks and is transferable to different samples of the same target, as shown by the experiments. The authors achieved their results using a curve generator method that used binary search and machine learning methods, specifically Artificial Neural Networks (ANNs), to predict the target behavior from the tested injections. This example showcases the benefits of integrating AI in FI attacks for more robust and adaptable fault injection methodologies for tackling portability, staying ahead of attackers, and strengthening the security of increasingly complex systems.

1.2.2. SIDE-CHANNEL ANALYSIS (SCA)

Side-channel analysis represents a form of passive implementation attack that exploits unintended side-channel information emitted during the execution of algorithms on targeted devices. This side-channel information, including power consumption [28], electromagnetic emissions [29], execution time [30], or even sound [31], can be exploited to uncover sensitive data such as cryptographic keys. Side-channel analysis aims to decipher the relationship between the observed side-channel emissions and the underlying instructions and data being processed by the device. At the same time, the side-channel attacks focus on the strategic exploitation of this information to compromise the device's security. SCA is categorized into non-profiling and profiling (two-stage) SCA, each with distinct security assumptions and methodologies.

Non-profiling SCA explores the statistical dependency between leaked side-channel information and secret cryptographic keys. Examples include techniques such as Simple Power Analysis (SPA) [30], Differential Power Analysis (DPA) [28], and Correlation Power Analysis (CPA) [32]. These methods rely on observing the system's physical outputs over multiple operations and running the attack over all possible key hypotheses. The attacker uses statistical techniques (e.g., Pearson correlation, difference-of-means, or mutual information) to identify the most probable key.

On the other hand, a two-stage or profiling SCA assumes a more powerful

[1] Related research uses the terms portability and transferability interchangeably.

adversary with access to an open device identical to the target, which allows the creation of a detailed profiling model. Therefore, the profiling SCA is more common in security evaluations, which leads us to choose this type of SCA for exploration within this thesis. These attacks commonly have two phases. The first phase creates a model using the clone device under control, and in the second phase, the model is utilized to obtain the secret from the targeted device. One such attack called Template Attack (TA) [33] creates a statistical model or 'template' that describes the leakage and noise of the open (clone) device under control. The adversary or security analysts then use this template to analyze the measurements collected from the target device, which should be an identical (clone) device, and distinguish the most likely key used on the targeted device.

As SCA techniques have evolved, so too have the countermeasures designed to protect against them. Commonly, side-channel countermeasures are categorized into masking or hiding. Masking introduces random values (masks) into the computation, obscuring the relationship between the computation and the side-channel emissions. Hiding countermeasures aim to disrupt the correlation by, for example, implementing a constant-time execution to prevent timing attacks or injecting noise in collected side channels through random delays in the execution of operations. These countermeasures are critical in maintaining the security of cryptographic devices against side-channel attacks, but designing and implementing them can be challenging. The effectiveness of these countermeasures depends on their ability to sufficiently break the correlation between side-channel information and processed data, ensuring that even if the side-channel information is intercepted, it cannot be used to compromise the system.

While hiding and masking techniques have been crucial in hindering first-order attacks, their effectiveness has inadvertently led to the emergence of higher-order attacks. For instance, in a system protected by masking, a second-order DPA can uncover the secret key by analyzing the correlation between two different masked operations [34]. The development of higher-order attacks underscores the continuous arms race in cryptographic security, as it calls for innovation in countermeasures to protect against emerging attacks.

CHALLENGES AND NEW DEVELOPMENTS FOR SCA

As mentioned, designing and implementing countermeasures against side-channel attacks remains an open challenge. Many systems vulnerable to these attacks, such as IoT devices and embedded systems, operate under strict resource constraints. Consequently, developing effective countermeasures that do not compromise performance or energy efficiency is a crucial and complex problem. Furthermore, during the development phase of cryptographic implementations, there is a lack of insight into potential side-channel leakage, making identifying and addressing vulnerabilities early a non-trivial challenge. Additionally, attackers continually adapt their strategies, complicating the prediction and mitigation of evolving side-channel attack methodologies. This progression requires the consistent development of adaptive defense mechanisms that can keep up with new exploitation techniques. Security evaluations must also improve to incorporate more advanced attack

methods, providing a more accurate measure of a device's security.

While template attacks, known for their effectiveness from an information theoretical perspective, require significant feature extraction and numerous traces to model device leakage accurately, research is increasingly looking towards advanced methods such as Machine Learning (ML) to enhance side-channel analysis. Initial explorations utilized traditional machine learning techniques such as Support Vector Machines (SVM) [35, 36] and Random Forest [37], followed by the application of Multi-layer Perceptron (MLP) [38, 39]. More on the traditional methods can be found in [40]. The introduction of deep learning-based profiling SCA marked a significant development, employing various architectures like MLP, Stacked Autoencoder, Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) to analyze side-channel data [7]. Since this work, many publications have considered different methods and architectures, establishing a well-defined framework for DLSCA, which we explain in Section 1.4.2. However, this framework requires further improvement to enable efficient and robust DL-based side-channel attacks during security evaluation. The use of DL enables automatic feature extraction, but further investigation is necessary to understand the trade-off between a separate, dedicated process for feature extraction and an integrated, automatic feature extraction within DL techniques. Inherent to DL, the DLSCA framework now includes new time-intensive operations. For example, hyperparameter tuning can significantly influence the performance of DLSCA but is time-consuming due to the large search space. Further, the trained models often demonstrate poor generalization even to the clone devices used for training [41–43]. Thus, improving model portability and robustness can contribute to more effective DLSCA, leading to more efficient security evaluations. Further exploration of the training process and utilized metrics could help reduce training time and computational resources. The difference between DL metrics and SCA evaluation metrics can be further studied to improve convergence speed and attack success. Moreover, ML methods were not only applied to finding the correct key but have found their application in data processing, like denoising, for both non-profiling [44] and profiling side-channel analysis [45]. More on different DL methods and their applications within the SCA are discussed in Chapter 5.

The major advantages of DL methods for SCA include the ability to process traces without specific feature extraction or preprocessing, as these methods inherently distinguish relevant features and train models from them. This capability allows for bypassing traditional requirements like trace alignment, which typically improves the correlation between data and leakage. Additionally, DL methods have demonstrated proficiency against masking countermeasures by learning from second-order leakages, suggesting potential effectiveness against even higher-order leakages [46, 47]. These developments and the broad application of ML methods at various steps within SCA signify a shift in how side-channel attacks are understood and addressed. As the field continues to evolve, these new methodologies promise to enhance the efficiency and robustness of side-channel analysis, offering potential modifications to the future of standard security evaluations and defense strategies.

1.3. ARTIFICIAL INTELLIGENCE (AI)

Artificial intelligence is a study within computer science that explores and develops intelligent machines or software to mimic the problem-solving and decision-making capabilities of humans and animals [48]. This broad field encompasses various methods for diverse domains, ranging from expert systems and fuzzy logic to evolutionary computation and machine learning. Within the scope of this thesis, we utilized evolutionary algorithms from evolutionary computation and specific techniques from the machine learning domain. Note that motivation for the used methods and their detailed descriptions are part of dedicated chapters, while here, we provide a more generic overview of the field and shared concepts.

1.3.1. EVOLUTIONARY ALGORITHMS (EAS)

Evolutionary algorithms draw inspiration from the mechanisms of natural evolution, precisely the concept of survival of the fittest [49]. Analogous to the biological process, EAs maintains a population of potential solutions to the problem at hand, treating them as individuals subject to an evaluation process utilizing a fitness function. This function measures the performance of solutions within the population, guiding the evolution towards increasingly better generations. Inspired by the fitness-based survival principle, selection mechanisms determine the individuals contributing to the next generation. During each iteration, selected individuals undergo genetic operations, such as crossover and mutation, mimicking the genetic recombination and variation observed in nature. This process ensures a balanced exploration of the solution space, exploiting promising areas, but also exploring new possibilities. The adaptability and robustness of EAs make them particularly effective in optimizing various problems across disciplines, including engineering [50] and finance [51]. Moreover, they are successful in fine-tuning complex systems, such as machine learning models [52]. In the field of robotics, these algorithms are used for robot path planning [53], and automatic creation of autonomous robots [54] where trajectory planning and robot control are part of the process.

Genetic algorithms are among the most well-known types of EAs. GA mimics the biological process of selection, reproduction, and mutation and is particularly effective for solving optimization problems by evolving solutions of the population [55]. Memetic Algorithms (MAs) extend the GA by incorporating local search procedures to refine the individual solutions in the population, combining global and local search capabilities, often leading to improved solution quality [56]. Evolution Strategy (ES) relies only on mutation and selection for the evolution process and is often used for continuous optimization problems [57]. Beyond these, there are several other evolutionary techniques, such as Genetic Programming (GP) [58], Evolutionary Programming (EP) [59], and Differential Evolution (DE) [60]. The algorithms mentioned are not fully explained here but are detailed in the chapters where they are used.

1

1.3.2. MACHINE LEARNING (ML)

Machine learning is a subfield of AI that enables computers to learn from data and improve their performance on a specific task over time without being explicitly programmed. It is generally divided into three main categories: supervised, unsupervised, and Reinforcement Learning (RL), each designed for different types of tasks [61]. Supervised learning involves training algorithms on a labeled dataset, where the input data is paired with the corresponding desired output. The algorithm aims to learn the mapping from input to output. This type of learning is commonly used for tasks such as classification and regression. Classification involves assigning labels to input data (e.g., spam or not spam), while regression predicts a continuous output (e.g., house prices). Support vector machines, decision trees, linear regression, and neural networks are algorithms that support supervised learning. Unsupervised learning works on unlabeled data, discovering hidden patterns or structures within the data without explicit guidance on the desired output. Typical applications include clustering, dimensionality reduction, and density estimation. Clustering groups similar data points together, dimensionality reduction simplifies the dataset by reducing the number of features, and density estimation models the data distribution. Examples of unsupervised learning include k-means clustering and autoencoders. Lastly, reinforcement learning is focused on training agents to make sequences of decisions in an environment to maximize cumulative rewards. The agent learns by receiving feedback through rewards or penalties based on its actions. This type of learning is suitable for tasks that involve decision-making and sequential actions, such as game-playing, robotics, and autonomous systems. Methods for RL include Q-learning, deep Q networks, and policy gradient methods [62]. Many machine learning algorithms can be classified into one or more of these types depending on their underlying principles and learning mechanisms.

Furthermore, we distinguish between traditional machine learning and deep learning, though the boundary between them is not always straightforward. Deep learning is characterized by its ability to construct models that exhibit a more complex hierarchy of learned functions or concepts than those found in traditional machine learning approaches [63]. This capability to perform advanced feature extraction and transformation allows for more effective high-dimensional data processing. Traditional machine learning models, such as linear regression, logistic regression, naive Bayes, and decision trees, are designed to learn from data in a straightforward manner, often requiring manual feature selection and data pre-processing. ANNs originate from the domain of traditional machine learning, beginning with the fundamental concept of the Perceptron. This simple model, representing a single neuron, enabled the development of more complex structures such as the Multi-layer Perceptron, which consists of multiple interconnected neurons organized in layers. There is no consensus about how much depth (layers and neurons) a model requires to be considered a deep learning model. However, deep learning extends ANNs to Deep Neural Networks (DNNs), which include different architectures, such as the mentioned MLPs, but also Convolutional Neural Networks and Recurrent Neural Networks. These architectures enable the automatic extraction and learning of complex patterns in data. Additionally, deep learning

models include autoencoders, used for data compression and denoising, and deep generative models, which can generate new data samples that resemble the training data.

Machine learning's ability to analyze patterns, detect anomalies, and make intelligent decisions significantly enhances various technological fields. Successful applications include image and speech recognition, natural language processing, recommendation systems, fraud detection, autonomous vehicles, and more [63]. The field continues to evolve, contributing to the development of increasingly sophisticated and intelligent systems [64].

1.4. AI-BASED IMPLEMENTATION ATTACKS

This section examines the integration of artificial intelligence methods in implementation attacks and explains the frameworks for fault injection and side-channel analysis utilized in this thesis.

1.4.1. AI-BASED FAULT INJECTION (AIFI)

The AIFI framework primarily focuses on employing AI methods to optimize the search for effective FI parameters, constituting the first phase of the FI attacks. Despite being a novel area with relatively fewer publications, a common framework has been discerned across studies. Notable works in this domain [23, 25–27] have explored a range of algorithms, identifying genetic and memetic algorithms as particularly effective. Collectively, these studies contribute to defining the typical structure of the AI-based FI framework, as we describe it here.

In AIFI, AI algorithms are utilized as search algorithms to identify FI parameters that induce desired fault responses in target devices from a predefined FI parameter search space. Attackers and security analysts determine the scope of the FI parameter search based on the capabilities of their FI equipment and the target's characteristics. The algorithms applied for the parameter search determine the specific parameter combinations to be tested. Once the injection is performed given selected FI parameters, the device's responses are categorized into several fault classes based on observed behavior, constituting a major component of the AIFI parameter search framework. Common classification from the mentioned related work [23, 25–27] includes the following fault classes:

- MUTE: The device does not respond within a predetermined time frame, indicating time-out errors.
- RESET: The device resets after the fault injection.
- NORMAL/PASS/OK: The device operates as expected, showing no signs of successful fault injection.
- SUCCESS/FAIL: The fault injection alters the device's behavior, indicating a potential vulnerability.

The fault class names of the last category, SUCCESS/FAIL, might appear contradictory. However, in the context of related work, SUCCESS indicates the successful discovery of a potential vulnerability, while FAIL refers to the device's altered behavior. The standardization of these terms is still evolving, and variations exist. For instance, MUTE and RESET might be combined into a single class if differentiating them does not significantly impact the fault analysis. While the terminology is not universal, these classifications provide a recognized framework for distinguishing between common device responses in related research [23, 25–27]. This categorization simplifies the range of outcomes into manageable classes. Further analysis is crucial to understand the extent of the vulnerabilities from the observed responses. Moreover, depending on the objective of the FI parameter search, this classification can vary, allowing for more granular categorization, such as classes for different faulty behaviors.

Each fault class is assigned a fitness value reflecting its desirability and severity within this framework. The algorithms aim to maximize these fitness values in related work, searching for fault classes that indicate the most promising outcomes for successful attacks. Typically, the unexpected behavior (SUCCESS/FAIL fault class) is considered the desired outcome and is thus assigned the highest fitness value, followed by MUTE or RESET, and finally, PASS.

Part of the framework includes performing multiple measurements (injections) with the same FI parameter combination, as the device's responses might vary. Thus, related work injects multiple injections consecutively with the same FI parameters. This leads to the definition of an additional CHANGING fault class, which represents the variability in device responses to the same FI parameter combination.

Note that the desired outcome is defined based on the fault model and the fault analysis applied in the attack scenario. During internal security evaluations, the focus is on identifying potentially exploitable vulnerabilities under a more general setup rather than executing complete FI attacks. Evaluators aim to understand the target's reactions to the injections, conducting a characterization under a specific FI type. Knowing the target's possible outcomes and behaviors under injections enables them to understand what attacks an adversary might perform. An attacker typically requires only one FI parameter combination with repeatable faulty behavior to perform the attack. In contrast, evaluators aim to learn about all or most possible device fault behaviors that could lead to various attacks and vulnerability exploitation [65].

Lastly, the performance of the algorithms for finding effective FI parameters within the framework is generally evaluated based on the number of observed vulnerabilities, specifically the number of observed FI parameter combinations associated with a particular fault class, such as SUCCESS, leading to faulty behavior. Reported metrics include the total number of tested parameter combinations and the number or percentage of combinations associated with specific classes. The higher the number of observed faulty behavior (vulnerabilities) for a similar number of tested parameter combinations, the more effective the search algorithm is considered. Note that the number of tested FI parameter combinations is positively correlated with the execution time of the search algorithm.

1.4.2. DEEP LEARNING-BASED SIDE-CHANNEL ATTACKS (DLSCA)

Deep learning methods have already demonstrated significant advantages in side-channel analysis, as mentioned in Section 1.2.2. Moreover, in February 2024, the German Federal Office for Information Security (BSI) published an updated version of the Application Notes and Interpretation of the Scheme (AIS) 46, defining guidelines for evaluating machine learning-based side-channel attack resistance [66]. Thus, integrating DL into SCA is a well-established and widely recognized practice. This section outlines the main characteristics of the framework utilized throughout this thesis.

In DLSCA, the main objective is to train the parameters θ of deep neural networks using training data $\mathscr{D}$ by minimizing a loss function $\mathscr{L}$. Each instance of training data $\mathscr{D}$ consists of a tuple $(\mathtt{x}_i, \mathtt{y}_i)$, where $\mathtt{x}_i$ is a one-dimensional vector representing the i-th side-channel measurement (or trace) in a dataset $\mathscr{D}$. The range of i is from 0 to the size of the dataset $|\mathscr{D}|$. The term $\mathtt{y}_i$ refers to the label (or class) associated to $\mathtt{x}_i$.

Labeling the dataset requires defining a leakage model and a selection function. Common leakage models in SCA include Identity (ID), Hamming Weight (HW), Hamming Distance (HD), and bit-level models [67]. The identity model refers to the direct value of an intermediate being processed by a cryptographic algorithm, while HW and HD models refer to the Hamming weight of an intermediate and the Hamming weight of the XOR between two intermediates, respectively. Bit-level models typically focus on the most or least significant bit of an intermediate variable. The intermediate variable is defined according to a key-dependent selection function, representing an intermediate value from the cryptographic algorithm. For instance, in AES encryption, the intermediate for the i-th trace $\mathtt{x}_i$ could be an $\mathtt{S\text{-}Box}$ output byte in the first encryption round, i.e., $\mathtt{y}_i = \mathtt{S\text{-}Box}(\mathtt{d}_j \oplus \mathtt{k}_j)$, where $\mathtt{d}_j$ and $\mathtt{k}_j$ are the j-th plaintext and key bytes ($j \in [0, 15]$ for a 128-bit key), respectively. In this thesis, we commonly consider AES algorithm. Thus, we explain the rest of the framework using the notion of key bytes and intermediate byte values. However, we note that the framework can be utilized for algorithms that process data in different chunk sizes, not only bytes.

From the training set $\mathscr{D}$, a subset $\mathscr{V}$ is selected to validate the trained model, where both come from the clone device in profiling SCA. The trained model is subsequently tested on a separate dataset $\mathscr{A}$, known as the attack set, collected from the targeted device. Since the goal is to extract the secret key (or a single byte of it) from $\mathscr{A}$, commonly Guessing Entropy (GE) or Success Rate (SR) are used to evaluate attack performance [68]. The best neural network model is the one that requires minimal attack complexity, measured in the minimum number of attack traces necessary to successfully recover the key [69].

Guessing entropy is computed by initially predicting labels for the validation or attack set and obtaining class probabilities $\mathtt{p}_{i,\mathtt{y}_i}$ for each trace i. As labels $\mathtt{y}_i$ are derived from a key-dependent selection function, we obtain the log-likelihood l_k of

a particular key byte $\mathtt{k}_j \in [0,255]$:

$$l_k = \sum_{i=0}^{N_a-1} \log \mathtt{p}_{i,\mathtt{y}_i}, \tag{1.1}$$

where N_a denotes the number of traces in the predicted set. This process is repeated for all possible key byte hypotheses, with each hypothesis generating different labels $\mathtt{y}_i$ for each trace. The key rank of the correct key $\mathtt{k}^*$ is determined by sorting all l_k values in descending order (most likely candidates to first positions), obtaining a key guessing vector g, and locating the position of l_{k^*} associated with the correct key byte k^* within the g. The GE of the correct key, $\mathtt{ge}^*$, is obtained through an empirical process where the key rank calculation is repeated multiple times with varied randomly selected subsets from the attack or validation set. The average position of the correct key $\mathtt{k}^*$ across these experiments (average g) is guessing entropy. When $\mathtt{ge}^* = 1$, the model is considered to have successfully recovered the key with N_a attack traces. The minimum number of traces to retrieve the key is denoted as $N_{\mathtt{ge}^*=1}$. Although the primary goal of training a deep neural network in the SCA context is to minimize the number of attack traces required to successfully recover the key ($\mathtt{ge}^* = 1$), models are commonly trained using a categorical cross-entropy loss function. The authors in [70] demonstrate that minimizing the categorical cross-entropy loss function is aligned with the training goal, supporting this design choice.

Another common attack performance metric is the success rate, which assumes first-order SR, while more generally, o^{th}-order SR relates to the probability that the correct key is among the o most probable key candidates based on the guessing vector g. Within one experiment, the probability is 1, meaning that the correct key k^* is among the first o candidates or 0 when it is not. Similarly to GE calculation, the process is repeated multiple times in an empirical process, obtaining the probability of the correct key k^* being in the first o positions of the g.

Due to their computational demands, GE and SR are not typically calculated during the training phase of side-channel analysis models. These metrics require an empirical evaluation process, which, if integrated into training, significantly increases its duration time. Moreover, these metrics assess the model's capability to conduct an effective side-channel attack, which requires the model to generalize to new and unseen data accurately. Employing GE or SR during training might not accurately reflect the model's performance for the attack on unseen data. Additionally, in cases where training datasets involve traces collected under varying keys, the computation of GE and SR is more complex and leads to outcomes that may not precisely represent the model's true attack efficacy.

1.5. PROBLEM STATEMENT

Security evaluation and certification cannot guarantee security but can ensure that claims about the security features of the evaluated product are independently verified. With increasing security concerns, the relevance of these processes in ensuring the integrity of secure devices becomes increasingly important. Consequently, there

is a growing need to make security evaluations more efficient, making them less time-consuming and more resource-efficient without compromising the depth and reliability of the analysis. Implementation attacks, which are typically part of security evaluations, present distinct challenges in practice. Thus, research improving these attacks contributes to more efficient security evaluation. This thesis contributes to the field by enhancing specific aspects of implementation attacks, which we detail in this section, which allow for more efficient security evaluation processes.

Given the development of novel security threats, this research focuses on the application of AI methods in implementation attacks. As discussed in Section 1.4, AI methods, particularly in AI-based fault injection and deep learning side-channel analysis, have shown promise in enhancing security evaluation processes. This thesis extends this topic of research, investigating these AI frameworks to improve further the efficiency and effectiveness of implementation attacks in various scenarios.

The main research question addressed by this thesis is

> How can AI-based implementation attacks, specifically fault injection and side-channel analysis, be further enhanced for a more efficient security evaluation of cryptographic systems?

We aim to support security analysts conducting security evaluations by providing practical solutions and insights to make the evaluation process more efficient. Since implementation attacks are a part of these evaluations, we focus on enhancing them. Specifically, our goal is to make the AIFI parameter search more efficient by finding more vulnerabilities within the same time (tested parameter).
As in related work, DLSCA is considered more efficient when it requires fewer training or attack traces to achieve a successful attack. However, we also consider the practical execution of these methods and aim to offer solutions that improve the process while maintaining successful security analysis using these implementation attacks. For example, hyperparameter tuning is necessary for AI methods, but it can be time-consuming. By eliminating the need for tuning or optimizing it, we improve the execution of AI-based implementations attack and enhance the overall efficiency of the entire security evaluations. Therefore, several sub-questions are proposed to examine the main question, each targeting specific challenges within AIFI and DLSCA processes. These sub-questions aim to address gaps in the current methodology and propose solutions that significantly impact the efficiency of implementation attacks.

Sub-question 1: Hyperparameter Tuning. AI methods commonly have parameters of their own to define, referred to as hyperparameters, that can significantly influence their performance. Hyperparameter tuning, therefore, is an essential aspect of AI applications for any problem. This process can be costly and complex, so there is an incentive to improve the tuning process, mitigate it, or provide specific methodologies to enable efficient tuning in different scenarios. Understanding the impact of individual hyperparameters on the performance of AI methods can provide insights that help define the hyperparameter search space and guide the tuning process. A sub-question that we aim to answer to aid the hyperparameter tuning process is: *What impact do initialization methods have on the performance*

of AI methods in AIFI parameter search and DLSCA, and how can their selection be optimized?

Sub-question 2: Portability. A notable issue with implementation attacks during security evaluation is limited portability across different scenarios. For example, once fault injection characterization is done on a device, the obtained results are not often transferable to another sample of the same target, even with the same setup (bench specific to FI type). The portability is even more pronounced when the target or bench is different. DLSCA faces similar challenges, where models trained on one target do not generalize effectively across other targets. The difference between targets can even come from clone devices (profiling and attack devices) and again gets more pronounced if different targets and trace acquisition processes are used. Since security evaluations are done on multiple devices, requiring repeated experiments, this thesis investigates how AI methods can mitigate these portability issues in both AIFI parameter search and DLSCA to help make security evaluation more efficient. The related sub-question is: *How can AI methods help mitigate portability challenges in AIFI parameter search and DLSCA, enhancing the efficiency and applicability of security evaluations?*

Sub-question 3: Evaluation Metrics. The success of the AI methods in AIFI parameter search is often measured using a single metric, which may not adequately represent different objectives. Similarly, in the current DLSCA framework, discrepancies exist between training metrics and actual attack evaluation metrics due to reasons discussed at the end of Section 1.4.2. While it was shown that minimizing the commonly utilized categorical cross-entropy loss is aligned with reducing the GE metric, bridging the gap between these metrics could enhance the efficiency of the whole DLSCA framework. Thus, this thesis explores alternative metrics that can improve the evaluation of AI methods in both types of implementation attacks. The sub-question is: *What alternative metrics can facilitate a more efficient AIFI parameter search and DLSCA model training, offering a comprehensive evaluation of AI methods for implementation attacks?*

These aspects represent only part of the considered frameworks, and other elements could be improved to address the efficiency of attacks and security evaluations. For example, target alignment and fitness functions for AIFI, or the trade-off between dedicated or automatic data preprocessing and feature extraction could be studied to distinguish the most efficient approach for DLSCA. However, the three aspects of AI-based implementation attacks addressed in this thesis, we distinguish as the most critical based on the literature discussed in Chapter 5, because they are the most time-consuming, or resolving them could significantly enhance overall efficiency. Additionally, industry partners from this collaboration raised these issues as the most pressing to resolve to enable efficient execution of AI-based implementation attacks for security evaluation. In conclusion, this thesis aims to address these key areas, providing efficient solutions to described aspects of AI-based implementation attacks. While not claiming to revolutionize the process entirely, the suggested improvements offer significant steps toward a more effective

and optimized security evaluation.

1.6. CONTRIBUTIONS

This thesis makes several significant contributions to the field of security evaluation and the application of AI in implementation attacks, focusing on AI-based fault injection and deep learning side-channel analysis. The key findings and contributions can be summarized as follows:

- This thesis introduces the application of memetic algorithms to laser fault injection, demonstrating the superiority of memetic algorithms over traditional random search in effectively identifying faults. This extension of memetic algorithms to laser fault injection represents a novel contribution, expanding their application beyond previous usage in voltage glitching and EMFI and further showcasing its benefits in the scope of fault injection.
- The thesis investigates the impact of different initialization methods on the performance of memetic algorithms for AIFI parameter search, offering insights into optimizing the hyperparameter selection process for more effective fault detection using these AI algorithms.
- Leveraging machine learning methods, particularly decision trees, the thesis addresses the portability issue within fault injection parameter search. This method utilizes prior knowledge to improve parameter search across different samples of the same target and adapt to minor changes in the laser bench setup. It significantly outperforms other methods in finding interesting faults, contributing to a more robust and efficient security evaluation process.
- This thesis proposes new metrics to evaluate the performance of AIFI parameter search algorithms. These metrics, focused on efforts to identify distant possible faults within a target, introduce a novel perspective for assessing algorithm effectiveness in fault injection scenarios.
- The study introduces two methods designed to promote diversity within solutions for fault injection parameter search. Although the improvements are modest compared to the transition from random search to memetic algorithms, the research identifies potential direction for further investigation and suggests that more advanced methods may result in more significant improvements.
- The thesis provides a detailed overview of recent deep learning applications in profiling side-channel analysis, summarizing the main directions and results obtained by the research community. This offers a valuable resource for understanding the potential and limitations of deep neural networks in this domain.
- The thesis presents a thorough analysis of how various weight initializers impact the performance of deep neural networks in DLSCA. By highlighting the variability in performance based on the choice of weight initializers, this

work provides crucial guidance for researchers and practitioners to conduct more informed hyperparameter tuning and deliver optimized deep learning models for side-channel analysis.

- The thesis introduces the use of autoencoders for dimensionality reduction to enhance the portability of DLSCA models between different datasets representing diverse targets and acquisition processes. This approach reduces hyperparameter tuning efforts and facilitates the portability of trained models to new datasets using transfer learning. The findings contribute to developing more adaptable and efficient security evaluations that deal with variations in target devices and setups.
- This thesis introduces novel metrics and training schemes designed for the DLSCA training process that aim to improve the attack performance, particularly in cases with limited profiling traces. These metrics bridge the gap between standard deep learning metrics and side-channel analysis evaluation metrics, significantly contributing to more effective security evaluation processes.

Through these contributions, the thesis significantly advances discussed AI-based implementation attacks. The research addresses specific problems and offers practical solutions, assisting security practitioners in performing efficient and portable AIFI parameter search and DLSCA. Valuable insights and methods from the thesis can be employed to enhance the efficiency and effectiveness of security evaluations, helping to ensure the integrity and robustness of cryptographic systems.

1.7. THESIS OUTLINE

This thesis is structured into two main sections, each addressing the same sub-questions within different contexts. The first section consists of multiple chapters focused on AI-based fault injection, while the second section explores deep learning side-channel analysis. The first section, including Chapter 2 through Chapter 4, methodically explores the sub-questions in the AIFI domain. The second section shifts the focus to DLSCA, starting with Chapter 5, which provides an overview of the state-of-the-art in DL-based SCA. Subsequent chapters, from Chapter 6 to Chapter 8, investigate the same sub-questions within the context of DLSCA. The structure of the thesis and its connection to the research questions introduced in Section 1.5 are visually represented in Figure 1.1. The thesis concludes with Chapter 9, offering a detailed discussion and reflection on the entire work.

1.7.1. AI-BASED FAULT INJECTION CHAPTERS

The chapters focusing on AI-based fault injection provide a detailed analysis of various aspects of AI application in FI:

Chapter 2. This chapter explores the application of artificial intelligence methods to laser fault injection, a type of FI with challenges similar to voltage glitching and EMFI. We introduce an innovative memetic algorithm and, for the first time,

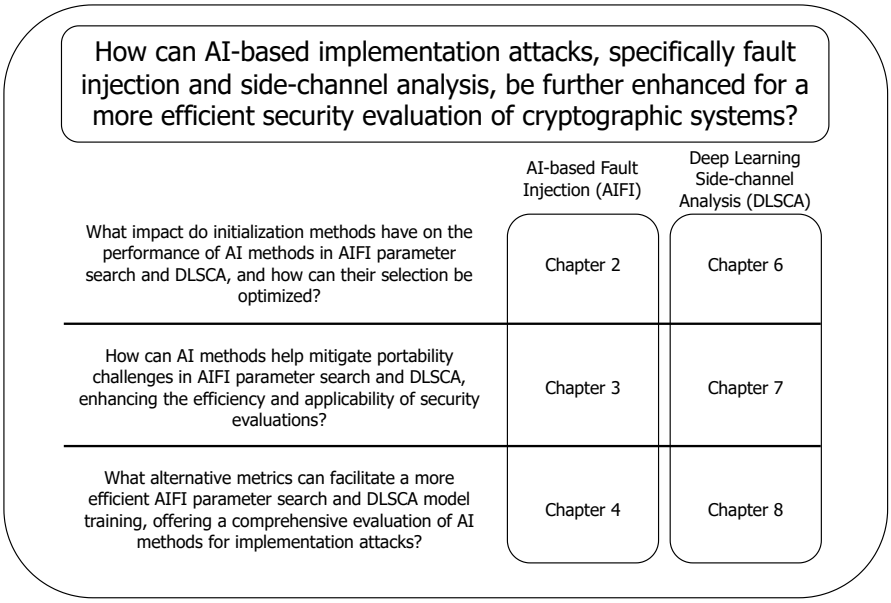


Figure 1.1.: Thesis structure overview. This diagram outlines the two main sections on implementation attacks, AIFI and DLSCA, highlighting the chapters that address the research sub-questions in each area. Chapter 5 is omitted from this overview because it does not directly address any of the three research sub-questions. However, it provides valuable context for the subsequent chapters by discussing the state-of-the-art in DLSCA.

employ an evolutionary algorithm for optimizing laser FI parameters. A comparative analysis is conducted between our proposed algorithm and the traditional random search approach. Furthermore, it investigates the significance of initialization methods, which are crucial for generating the initial population in the memetic algorithm, presenting a detailed experimental analysis across various scenarios. This research provides insights into the importance and impact of this hyperparameter to help enable efficient hyperparameter tuning. This chapter has been published as M. Krček, D. Fronte, and S. Picek. “On the importance of initial solutions selection in fault injection”. In: *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2021, pp. 1–12.

Chapter 3. In this chapter, we utilize AI methods to enable more efficient portability between different samples of the same target combined with minor modifications on the bench. We present our innovative method of integrating decision trees with a memetic algorithm. This combination allows for the efficient use of prior knowledge in the AIFI parameter search for the portability cases. Our experimental findings demonstrate that this integrated approach improves performance and offers insights into the target characterization under fault injections. This chapter has been

published as M. Krček, T. Ordas, D. Fronte, and S. Picek. "The more you know: Improving laser fault injection with prior knowledge". In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 18–29.

Chapter 4. Addressing the challenge of uncovering diverse vulnerabilities in AI-based fault injection, this chapter introduces two novel metrics designed better to evaluate algorithm performance from the perspective of security analysts. The aim is to identify multiple, distinct vulnerabilities, moving beyond the traditional focus on single optimal solutions. To achieve this, we propose two evolutionary methods that promote diversity in the search process. The chapter analyzes these methods, revealing their benefits and highlighting the need for further research on advanced diversity methods. This chapter has been published as M. Krček and T. Ordas. "Diversity Algorithms for Laser Fault Injection". In: *Applied Cryptography and Network Security Workshops*. Ed. by M. Andreoni. Cham: Springer Nature Switzerland, 2024, pp. 121–138. ISBN: 978-3-031-61486-6.

1.7.2. DEEP LEARNING SIDE-CHANNEL ANALYSIS CHAPTERS

The chapters on DLSCA provide insights into the application of deep learning in side-channel analysis:

Chapter 5. This chapter provides an overview of the application of deep learning in profiling side-channel analysis. It includes a detailed discussion of state-of-the-art techniques and the advantages and challenges of deep learning in this context. The chapter examines the evaluation metrics and the hyperparameter tuning, offering guidance in practical application and a forward-looking perspective on using deep neural networks in SCA. This chapter has been published as M. Krček, H. Li, S. Paguada, U. Rioja, L. Wu, G. Perin, and Ł. Chmielewski. "Deep learning on side-channel analysis". In: *Security and Artificial Intelligence: A Crossdisciplinary Approach*. Springer, 2022, pp. 48–71.

Chapter 6. Focusing on the crucial aspect of hyperparameter tuning in DLSCA, this chapter investigates the impact of weight initializers on the performance of deep learning models in profiling side-channel analysis. It analyzes how different initializers affect the model outcomes on various datasets and explores their connection with other hyperparameters. Our findings highlight the importance of careful selection of weight initializers in enhancing model performance, offering practical insights for security analysts conducting DLSCA. This chapter has been published as H. Li, M. Krček, and G. Perin. "A Comparison of Weight Initializers in Deep Learning-Based Side-Channel Analysis". In: *Applied Cryptography and Network Security Workshops*. Ed. by J. Zhou, M. Conti, C. M. Ahmed, M. H. Au, L. Batina, Z. Li, J. Lin, E. Losiouk, B. Luo, S. Majumdar, W. Meng, M. Ochoa, S. Picek, G. Portokalidis, C. Wang, and K. Zhang. Cham: Springer International Publishing, 2020, pp. 126–143. ISBN: 978-3-030-61638-0.

Chapter 7. This chapter addresses the challenge of architecture and profiling model portability in DLSCA across different public datasets, where the hardware, software, and trace acquisition differ, using autoencoders for dimensionality reduction. The chapter presents three distinct portability cases detailing the benefits and applications of our approach. It demonstrates architecture portability, leading to more efficient evaluations by mitigating the tuning process. Moreover, we present the portability of fully trained models on different target datasets through transfer learning, reducing training time. This chapter has been published as M. Krček and G. Perin. "Autoencoder-enabled model portability for reducing hyperparameter tuning efforts in side-channel analysis". In: *Journal of Cryptographic Engineering* (2023), pp. 1–23. DOI: 10.1007/s13389-023-00330-4.

Chapter 8. This chapter introduces a novel training technique utilizing distributed labels to enhance attack performance in scenarios where profiling traces are limited. Moreover, it proposes a new metric, the Label Correlation (LC), bridging the gap between the standard deep learning metrics and SCA-specific metrics, namely guessing entropy and success rate. The chapter demonstrates the benefits of the proposed metrics and techniques in practical scenarios, offering analysis of the use cases, such as early stopping and hyperparameter tuning, where the improvement of the correlation between training/validation metrics and the attack metrics is most relevant. This work enhances the DLSCA framework by providing a more appropriate metric that is less computationally expensive than GE and SR, enabling its application in the validation step. This chapter has been published as 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* 18 (2023), pp. 3849–3861. DOI: 10.1109/TIFS.2023.3287728.

1.7.3. FINAL INSIGHTS

Chapter 9. This final chapter provides a thorough summary of the entire thesis, discussing key findings and addressing the research questions. It critically examines the limitations of this work and proposes directions for future work to overcome them. Reflecting on the broader implications of the findings, the chapter highlights the potential impact and future directions for research in the field of AI-based implementation attacks.

1.7.4. APPENDICES

The appendices provide additional supporting materials:

Appendix A. This appendix includes supplementary data for Chapter 2, presenting orthogonal arrays used in the experiments.

Appendix B. This appendix provides supporting information for Chapter 7, including hyperparameter search spaces, results from statistical tests, and descriptions of models obtained with hyperparameter tuning.

Appendix C. This appendix offers supplementary information for Chapter 8, describing the hyperparameters of the utilized state-of-the-art models.

1.7.5. ABOUT THE THESIS

Except for the last concluding chapter, the following chapters are integral copies of publications with only minor changes. Consequently, there might be overlaps and similarities between sections, such as the introduction, background, related work, or descriptions of targets and datasets used in the experimental analysis. Note also that the terminology and notation might not be uniform between different chapters as each chapter is an independent publication.

We note that the list of co-authored published and under-review papers can be found at the end of the thesis under List of Publications, while only some are included within the thesis. Only publications that directly contribute to the narrative and focus of this thesis were included, ensuring a concise and targeted exploration of the research objectives while avoiding an overly extensive document.

The research presented in this thesis was conducted in partnership with STMicroelectronics, focusing on practical applications within the industry. The collaboration with STMicroelectronics has been crucial, particularly in research on fault injection, where the company provided direct support through access to the necessary fault injection equipment (laser bench) and targeted devices. This collaboration has also aligned the selection of publications and the focus of this thesis with the industry's challenges and needs. The emphasis has been on delivering practically viable findings, offering benefits and advancements to the industry standards and expectations.

REFERENCES

[1] S. Sinha. *State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally*. 2023. URL: https://iot-analytics.com/number-connected-iot-devices.

[2] I. J. 1. 27. *ISO/IEC 15408-1:2022: Information security, cybersecurity and privacy protection - Evaluation criteria for IT security*. 2022. URL: https://www.iso.org/standard/72891.html.

[3] NIST. *FIPS 140-3 development*. 2017. URL: https://csrc.nist.gov/projects/fips-140-3-development.

[4] J. Correia, S. Smith, and R. Qaddoura. *Applications of Evolutionary Computation: 26th European Conference, EvoApplications 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12–14, 2023, Proceedings*. Lecture Notes in Computer Science. Springer Nature Switzerland, 2023. ISBN: 9783031302299. URL: https://books.google.nl/books?id=JjG4EAAAQBAJ.

[5] P. P. Shinde and S. Shah. "A review of machine learning and deep learning applications". In: *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE. IEEE, 2018, pp. 1–6. ISBN: 9781538652572. DOI: 10.1109/ICCUBEA.2018.8697857.

[6] H. Wu, H. Han, X. Wang, and S. Sun. "Research on artificial intelligence enhancing internet of things security: A survey". In: *Ieee Access* 8 (2020), pp. 153826–153848.

[7] 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. Vol. 10076. Lecture Notes in Computer Science. Springer. Springer, Dec. 2016, pp. 3–26. ISBN: 978-3-319-49444-9. DOI: 10.1007/978-3-319-49445-6_1. URL: https://doi.org/10.1007/978-3-319-49445-6%5C_1.

[8] C. Paar and J. Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[9] J. Daemen and V. Rijmen. *The design of Rijndael*. Vol. 2. Springer, 2002.

[10] National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. en. Nov. 26, 2001. DOI: https://doi.org/10.6028/NIST.FIPS.197-upd1.

[11] C. Paar and J. Pelzl. “The advanced encryption standard (AES)”. In: *Understanding Cryptography: A Textbook for Students and Practitioners* (2010), pp. 87–121.

[12] A. Kerckhoffs. “La cryptographie militaire”. In: *Journal des Sciences Militaires* (1883), pp. 161–191.

[13] J. Breier and C.-N. Chen. “On determining optimal parameters for testing devices against laser fault attacks”. In: *2016 International Symposium on Integrated Circuits (ISIC)*. IEEE. 2016, pp. 1–4.

[14] J. Breier and X. Hou. “How practical are fault injection attacks, really?” In: *IEEE Access* 10 (2022), pp. 113122–113130.

[15] S. P. Skorobogatov and R. J. Anderson. “Optical fault induction attacks”. In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2002, pp. 2–12.

[16] E. Biham and A. Shamir. *Differential Fault Analysis of Secret Key Cryptosystems*. 1997.

[17] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta. “Fault Sensitivity Analysis”. In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Ed. by S. Mangard and F.-X. Standaert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 320–334. ISBN: 978-3-642-15031-9.

[18] T. Fuhr, E. Jaulmes, V. Lomné, and A. Thillard. “Fault Attacks on AES with Faulty Ciphertexts Only”. In: *Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. FDTC ’13. USA: IEEE Computer Society, 2013, pp. 108–118. ISBN: 9780769550596. DOI: 10.1109/FDTC.2013.18. URL: https://doi.org/10.1109/FDTC.2013.18.

[19] C. Clavier. “Secret external encodings do not prevent transient fault analysis”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2007, pp. 181–194.

[20] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas. “SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, Issue 3 (2018), pp. 547–572. DOI: 10.13154/tches.v2018.i3.547-572. URL: https://tches.iacr.org/index.php/TCHES/article/view/7286.

[21] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. “Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures”. In: *Proceedings of the IEEE* 100.11 (2012), pp. 3056–3076.

[22] A. Baksi, S. Bhasin, J. Breier, D. Jap, and D. Saha. “A survey on fault attacks on symmetric key cryptosystems”. In: *ACM Computing Surveys* 55.4 (2022), pp. 1–34.

[23] A. Maldini, N. Samwel, S. Picek, and L. Batina. “Genetic algorithm-based electromagnetic fault injection”. In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2018, pp. 35–42.

[24] L. Wu, G. Ribera, N. Beringuier-Boher, and S. Picek. “A fast characterization method for semi-invasive fault injection attacks”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2020, pp. 146–170. ISBN: 978-3-030-40185-6.

[25] R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub. “Glitch it if you can: parameter search strategies for successful fault injection”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 236–252.

[26] S. Picek, L. Batina, D. Jakobović, and R. B. Carpi. “Evolving genetic algorithms for fault injection attacks”. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2014, pp. 1106–1111.

[27] S. Picek, L. Batina, P. Buzing, and D. Jakobovic. “Fault Injection with a new flavor: Memetic Algorithms make a difference”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2015, pp. 159–173.

[28] P. C. Kocher, J. Jaffe, and B. Jun. “Differential Power Analysis”. In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. Ed. by M. Wiener. CRYPTO ’99. Springer. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 388–397. ISBN: 3540663479.

[29] J.-J. Quisquater and D. Samyde. “ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards”. In: *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*. E-SMART ’01. Springer. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 200–210. ISBN: 3540426108. URL: http://dl.acm.org/citation.cfm?id=646803.705980.

[30] P. C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO ’96. Springer. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 104–113. ISBN: 3540615121.

[31] D. Genkin, A. Shamir, and E. Tromer. “Acoustic cryptanalysis”. In: *Journal of Cryptology* 30 (2017), pp. 392–443.

[32] E. Brier, C. Clavier, and F. Olivier. “Correlation Power Analysis with a Leakage Model”. In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*. Ed. by M. Joye and J. Quisquater. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2. URL: https://doi.org/10.1007/978-3-540-28632-5%5C_2.

[33] 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. ISBN: 3-540-00409-2. DOI: 10.1007/3-540-36400-5_3. URL: https://doi.org/10.1007/3-540-36400-5%5C_3.

[34] T. S. Messerges. “Using second-order power analysis to attack DPA resistant software”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Ed. by Ç. K. Koç and C. Paar. Springer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 238–251. ISBN: 978-3-540-44499-2.

[35] 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.

[36] A. Heuser and M. Zohner. “Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines”. In: *COSADE*. Ed. by W. Schindler and S. A. Huss. Vol. 7275. LNCS. Springer, 2012, pp. 249–264. ISBN: 978-3-642-29911-7.

[37] L. Lerman, G. Bontempi, and O. Markowitch. “Side channel attack: an approach based on machine learning”. In: *Center for Advanced Security Research Darmstadt* 29 (2011), pp. 29–41.

[38] Z. Martinasek and V. Zeman. “Innovative method of the power analysis”. In: *Radioengineering* 22.2 (2013), pp. 586–594.

[39] Z. Martinasek, P. Dzurenda, and L. Malina. “Profiling power analysis attack based on MLP in DPA contest V4. 2”. In: *2016 39th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE. 2016, pp. 223–226.

[40] A. Jovic, D. Jap, L. Papachristodoulou, and A. Heuser. “Traditional machine learning methods for side-channel analysis”. In: *Security and Artificial Intelligence: A Crossdisciplinary Approach*. Springer, 2022, pp. 25–47.

[41] O. Choudary and M. G. Kuhn. “Template attacks on different devices”. In: *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*. Springer. 2014, pp. 179–198.

[42] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen. “X-DeepSCA: Cross-device deep learning side channel attack”. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. Vol. 1. New York, New York, USA: ACM Press, 2019, pp. 1–6. ISBN: 9781450367257. DOI: 10.1145/3316781.3317934. URL: http://doi.acm.org/10.1145/3316781.3317934%20http://dl.acm.org/citation.cfm?doid=3316781.3317934.

[43] S. Bhasin, A. Chattopadhyay, A. Heuser, D. Jap, S. Picek, and R. Ranjan. “Mind the portability: A warriors guide through realistic profiled side-channel analysis”. In: *NDSS 2020-Network and Distributed System Security Symposium*. 2020, pp. 1–14. URL: https://eprint.iacr.org/2019/661.pdf.

[44] D. Kwon, H. Kim, and S. Hong. “Improving non-profiled side-channel attacks using autoencoder based preprocessing”. In: *Cryptology ePrint Archive* (2020).

[45] L. Wu and S. Picek. “Remove some noise: On pre-processing of side-channel measurements with autoencoders”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), pp. 389–415.

[46] B. Timon. “Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 107–131. DOI: 10.13154/tches.v2019.i2.107-131. URL: https://doi.org/10.13154/tches.v2019.i2.107-131.

[47] L. Masure, C. Dumas, and E. Prouff. “Gradient visualization for general characterization in profiling attacks”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2019, pp. 145–167.

[48] S. J. Russell and P. Norvig. *Artificial intelligence a modern approach*. London, 2010.

[49] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Vol. 53. Springer, 2003.

[50] A. Slowik and H. Kwasnicka. “Evolutionary algorithms and their applications to engineering problems”. In: *Neural Computing and Applications* 32 (2020), pp. 12363–12379.

[51] R. Aguilar-Rivera, M. Valenzuela-Rendón, and J. Rodríguez-Ortiz. “Genetic algorithms and Darwinian approaches in financial applications: A survey”. In: *Expert Systems with Applications* 42.21 (2015), pp. 7684–7697.

[52] E. Galván and P. Mooney. “Neuroevolution in deep neural networks: Current trends and future challenges”. In: *IEEE Transactions on Artificial Intelligence* 2.6 (2021), pp. 476–493.

[53] M. A. Contreras-Cruz, V. Ayala-Ramirez, and U. H. Hernandez-Belmonte. “Mobile robot path planning using artificial bee colony and evolutionary programming”. In: *Applied Soft Computing* 30 (2015), pp. 319–328.

[54] S. Nolfi and D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[55] D. Whitley. “A genetic algorithm tutorial”. In: *Statistics and computing* 4.2 (1994), pp. 65–85.

[56] P. Moscato. “On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms”. In: *Caltech Concurrent Computation Program* (Oct. 2000).

[57] H.-G. Beyer and H.-P. Schwefel. “Evolution strategies–a comprehensive introduction”. In: *Natural computing* 1 (2002), pp. 3–52.

[58] J. R. Koza. “Genetic programming as a means for programming computers by natural selection”. In: *Statistics and computing* 4 (1994), pp. 87–112.

[59] D. B. Fogel. *Artificial intelligence through simulated evolution*. Wiley-IEEE Press, 1998.

[60] R. Storn and K. Price. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11 (1997), pp. 341–359.

[61] C. Bishop. “Pattern recognition and machine learning”. In: *Springer google schola* 2 (2006), pp. 5–43.

[62] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[63] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[64] N. Maslej, L. Fattorini, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, H. Ngo, J. C. Niebles, V. Parli, *et al.* "Artificial intelligence index report 2023". In: *arXiv preprint arXiv:2310.03715* (2023).

[65] I. Rais-Ali, A. Bouvet, and S. Guilley. "Quantifying the Speed-Up Offered by Genetic Algorithms during Fault Injection Cartographies". In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 61–72.

[66] 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.

[67] S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[68] 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*. Ed. by A. Joux. Vol. 5479 LNCS. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9. DOI: 10.1007/978-3-642-01001-9_26.

[69] O. Bronchain, J. M. Hendrickx, C. Massart, A. Olshevsky, and F. Standaert. "Leakage Certification Revisited: Bounding Model Errors in Side-Channel Security Evaluations". In: *IACR Cryptol. ePrint Arch.* (2019), p. 132. URL: https://eprint.iacr.org/2019/132.

[70] 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 (1 Nov. 2019), pp. 348–375. DOI: 10.13154/tches.v2020.i1.348-375. URL: https://tches.iacr.org/index.php/TCHES/article/view/8402.

[71] M. Krček, D. Fronte, and S. Picek. "On the importance of initial solutions selection in fault injection". In: *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2021, pp. 1–12.

[72] M. Krček, T. Ordas, D. Fronte, and S. Picek. "The more you know: Improving laser fault injection with prior knowledge". In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 18–29.

[73] M. Krček and T. Ordas. "Diversity Algorithms for Laser Fault Injection". In: *Applied Cryptography and Network Security Workshops*. Ed. by M. Andreoni. Cham: Springer Nature Switzerland, 2024, pp. 121–138. ISBN: 978-3-031-61486-6.

[74] M. Krček, H. Li, S. Paguada, U. Rioja, L. Wu, G. Perin, and Ł. Chmielewski. "Deep learning on side-channel analysis". In: *Security and Artificial Intelligence: A Crossdisciplinary Approach*. Springer, 2022, pp. 48–71.

[75] H. Li, M. Krček, and G. Perin. "A Comparison of Weight Initializers in Deep Learning-Based Side-Channel Analysis". In: *Applied Cryptography and Network Security Workshops*. Ed. by J. Zhou, M. Conti, C. M. Ahmed, M. H. Au, L. Batina, Z. Li, J. Lin, E. Losiouk, B. Luo, S. Majumdar, W. Meng, M. Ochoa, S. Picek, G. Portokalidis, C. Wang, and K. Zhang. Cham: Springer International Publishing, 2020, pp. 126–143. ISBN: 978-3-030-61638-0.

[76] M. Krček and G. Perin. "Autoencoder-enabled model portability for reducing hyperparameter tuning efforts in side-channel analysis". In: *Journal of Cryptographic Engineering* (2023), pp. 1–23. DOI: 10.1007/s13389-023-00330-4.

[77] 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* 18 (2023), pp. 3849–3861. DOI: 10.1109/TIFS.2023.3287728.

I

AI-based Fault Injection

2

On the Importance of Initial Solutions Selection in Fault Injection

Fault injection attacks require the adversary to select suitable parameters for the attack. In this work, we consider laser fault injection and parameters like the location of the laser shot (x, y), delay, pulse width, and intensity of the laser. The parameter selection process can be translated into an optimization problem. A very popular and successful method for various optimization problems is the genetic algorithm. To further improve the performance of a genetic algorithm, it is possible to combine it with local search to obtain a memetic algorithm.
We conduct several experiments comparing the performance of the memetic algorithm and the random search algorithm for finding faults. We investigate the influence of different initialization techniques on the performance of the memetic algorithm. In our experiments, the memetic algorithm is significantly better at finding faults than the random search. While evaluating different initialization techniques, we did not observe significant differences when averaging results. However, when considering the stability of the results with a memetic algorithm based on different initialization techniques, we can distinguish preferable techniques, such as LHSMDU and the Taguchi method.

This chapter has been published as M. Krček, D. Fronte, and S. Picek. "On the importance of initial solutions selection in fault injection". In: *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2021, pp. 1–12.

2.1. Introduction

Many hardware devices are used daily by millions of users, promising secure transactions. Security analysts evaluate the security of these hardware devices, and as adversaries, they have numerous possibilities of attacking them. Passive techniques exist, such as side-channel attacks, where the attacker monitors side-channel (unintended) information. Examples of the side-channel information are power consumption [2], electromagnetic radiation [3], and time [4]. On the other hand, there are also active attacks, specifically fault injection attacks, where the attacker aims to extract some secret/sensitive information from the target device by exposing the device to external interference. Doing this can cause the device to deviate from the typical behavior and correct execution, which the attacker might exploit, as in differential fault analysis (DFA) [5], fault sensitivity analysis (FSA) [6], and statistical fault attack (SFA) [7]. Attacks, such as ineffective fault attacks (IFA) [8], and statistical ineffective fault attacks (SIFA) [9], can exploit information even if induced faults do not change the output of the execution on the device. The attacks can be divided into two steps - inducing the faults and analyzing the target device's behavior and responses to achieve the attacker's goal. The focus of this work is on the first part - inducing the faults using external interferences that can come from different sources, such as optical (laser pulses) [10], electrical glitches (voltage) [11], electromagnetic (EM) radiation [12], and temperature changes [13].

The fault injection attacks can be divided by their invasiveness. Invasive attacks remove layers of the target to reach the silicon layer. In semi-invasive attacks, only the packaging is removed, and in non-invasive attacks, the target is attacked as it is, without any modifications. In this work, we consider the optical fault injection technique introduced in [10], namely Laser Fault Injection (LFI) attacks that are semi-invasive attacks. Like other fault injection attacks, for LFI to be successful, the adversary or security analyst must carefully select and tune the attack. In the case of the LFI attacks, the adversary needs to choose good parameters for successfully injecting faults, such as the location of the laser shot (x, y), focus, laser spot size, wavelength, laser trigger delay, laser pulse width, and laser intensity. These parameters are limited by the physical properties of the target and the equipment used for performing the attack. For example, x and y are limited by the target device and motorized stage (the minimum step it can make for each axis). On the other hand, the laser parameters are limited by the laser source equipment. In this work, we aim at injecting faults that lead to exploitable faulty outputs for attacks such as DFA. However, the same parameters can be considered for inducing so-called ineffective faults used in, e.g., SIFA.

The search space is often too large to perform an exhaustive search in a reasonable time, considering all the parameters and their possible values for laser fault injection. The attacker could decrease the ranges for the parameters, but this requires significant knowledge about the internal design of the target. Unfortunately, this is not often the case as the attacks are usually performed in a black-box setting. Consequently, limiting the ranges of the parameters can lead to testing many inadequate parameter combinations or failing to test parameter combinations that would be successful. Additionally, the laser effects on the target behavior can

significantly vary between two physically close positions, representing a problem to find optimal parameter sets to perform the laser attacks.

There is a clear need for a "smart" technique for parameter selection leading to a more effective security evaluation, considering the mentioned issues. The problem of selecting suitable fault injection parameters can be easily translated into an optimization problem, for which we can apply various techniques. One example would be evolutionary algorithms (EAs), such as genetic algorithms (GAs). There are several papers where evolutionary techniques are used to optimize fault injection attacks. In [14], the authors investigated a new direction based on genetic algorithms for voltage (VCC) glitching and found it suitable when not much is known about the device under attack. A continuation of this work is done in [15] where the authors compared the Monte Carlo search with the genetic algorithm. Picek et al. also compared the standard GA to the performance of an enhanced GA called a memetic algorithm (MA) for voltage glitching [16]. The memetic algorithm iterates the standard process of GA but adds a local search. The memetic algorithm is also used by Maldini et al. for Electromagnetic Fault Injection (EMFI) [17]. Another interesting approach using artificial intelligence techniques, namely machine learning, was introduced in [18]. The authors presented a method for fast characterization of the laser fault injection settings on the target. However, the technique can be used for other semi-invasive fault injection attacks and is transferable to different samples of the same target.

This work applies a genetic algorithm with Hooke-Jeeves as the local search to select laser fault injection parameters. The algorithm begins with an initial population of solutions that is evolved through a number of generations to reach better solutions for the optimization problem. A well-selected initial population of solutions is crucial because it can cause the algorithm to converge to some local optima quickly or cover more search space, which could help reach the global optimum. After comparing the performance of the memetic algorithm to the Monte Carlo approach for the laser fault injection attacks, we experiment with several different initialization methods for creating the initial population of solutions for the memetic algorithm.

Our main contributions are as follows:

1. We use evolutionary algorithms (genetic algorithms) to find parameters leading to a successful laser fault injection attack. To the best of our knowledge, this is the first time that evolutionary algorithms are used for LFI. Differing from the discussion in [18], we do not observe genetic algorithms causing issues due to their exploratory character (i.e., damaging the target). Additionally, the laser effect may greatly vary between two very close positions that are of different nature. Evolutionary algorithms work well with non-differentiable functions, making them a natural choice for such settings.
2. We propose a memetic algorithm that combines genetic algorithm and Hooke-Jeeves local search. In our opinion, this improves over related works [16, 17] with memetic algorithms as Hooke-Jeeves is a well understood and efficient derivative-free optimization algorithm.

3. We are the first to explore the influence of the initial population on the performance of the population-based search algorithms for fault injection attacks.

2.2. Background

This section briefly explains the memetic algorithm and different initialization techniques for the initial population we use in our experiments.

2.2.1. Memetic Algorithm (MA)

The memetic algorithm (MA) is a population-based hybrid genetic algorithm enhanced with an added procedure performing local refinements on individuals from the population [19]. Such algorithms were applied in many different fields, such as business analytics and data science [20], designing and training artificial neural networks [21], and robotic motion planning [22]. The memetic algorithms have shown to be better than traditional GAs for some optimization problem domains [23, 24]. The reason could be the trade-off between the exploration abilities of the underlying GA and the exploitation abilities of the local search. The improvement cost is more fitness evaluations and a fast loss of diversity within the population.

Throughout the years, there have been many versions of memetic algorithms proposed [25]. However, our work considers a simple combination of the traditional genetic algorithm and Hooke-Jeeves local search. The pseudocode of the memetic algorithm can be seen in Algorithm 1. The pseudocode is general, and it shows how the local search is integrated into the genetic algorithm - after recombination of the genetic algorithm, we perform local refinements on some of the solutions from the population. To further explain the pseudocode and memetic algorithm, we describe the genetic algorithm and our chosen local search approach - the Hooke-Jeeves algorithm.

Algorithm 1 Memetic Algorithm.

```
1:  procedure MemeticAlgorithm
2:      generate population of size N
3:      evaluate the population
4:      iteration = 0
5:      while stop condition not satisfied do
6:          new population = best solutions (elite size)
7:          select parent solutions for crossover
8:          children = crossover(parents)
9:          mutation(children)
10:         add children to new population
11:         evaluate the new population
12:         perform the local search
13:         iteration = iteration + 1
14:     return population
```

GENETIC ALGORITHM

The genetic algorithm (GA) is the most widely known type of evolutionary algorithm inspired by the process of natural selection [26, 27]. Genetic algorithms are often used for optimization problems [28].

Genetic algorithms work on a population of solutions of size N. In the early application, solutions were represented by bit strings, but representation can be adjusted for many different problems nowadays. For example, considering the LFI problem, the solution can be an array of integer and floating-point values representing the parameter set of LFI, namely the x, y, $delay$, $pulse\ width$, and $laser\ intensity$ parameters. A set of such solutions constructs the population for the genetic algorithm. After the initial population is created, the solutions are evaluated to retrieve the fitness. Fitness is a value that gives information about how 'good' or 'bad' the solution is for the problem (the definition of 'good' or 'bad' are problem-specific). In simple cases of finding an optimum of well-defined functions, the evaluation retrieves the function result based on the input, which is one solution from the population. There, the function result is the fitness value. Evaluation and fitness values are adjusted based on the problem, as is the representation of the solutions. For the LFI problem, we explain our definitions in Section 2.3.

After evaluating all the solutions in the population, we perform selection, which is usually done based on the fitness of the solutions. The idea is to choose better solutions as parents for reproduction since these solutions should have 'good' genes (solution values) to propagate into the next generation. The reproduction is done using the crossover and mutation operators by combining the genes of the selected parents and introducing minor variations. The crossover is applied to the selected parent solutions resulting in offspring solutions. There are different versions of the crossover operator, such as uniform crossover or average crossover. However, all these variations combine the genes from the parent solutions to produce a new solution (or several solutions). The new offspring solutions undergo mutation where each gene in the solution is modified with probability p_m. These variations help in the exploration of the search space and prevent fast convergence to a local optimum. The resulting intermediary population forms the next generation replacing the previous one entirely. However, complete individuals of the prior generation are transferred to the new generation without modifications when using elitism. Usually, a parameter often called the elite size defines how many solutions are transferred unmodified for the next generation.

Since we use the memetic algorithm, we select solutions that should undergo the individual improvement procedure after the described GA part. For the local improvements, we use the Hooke-Jeeves algorithm described in Section 2.2.1. We perform the local refinement on each of the selected solutions, and the algorithm starts a new iteration by evaluating the new solutions before parent selection. The process is repeated until the stop (termination) condition is met. This condition can consider different information about the progress of the algorithm. However, often simply the number of iterations, evaluations, or generations is considered. Further, one can consider the fitness values of the solutions - if we find a satisfying fitness, we could terminate the algorithm.

Hooke-Jeeves Algorithm

Hooke-Jeeves is a direct search algorithm used to search for the optimum of a nonlinear function without requiring derivatives of the function [29]. The algorithm combines exploratory moves with pattern moves. The step size can be different for each coordinate direction and change during the search in the exploratory move. The exploration starts from the initial point by exploring each coordinate direction by the specified step size. If the fitness does not improve, the opposite direction is considered. After all the coordinates are investigated, the exploration move is completed. If the exploration found a better solution, it is followed by the pattern move. Otherwise, only the step is decreased to try the exploration move again. The pattern move calculates the direction of the improvement and moves the starting point in that direction. The pattern move can be calculated as:

$$x_p{}^{k+1} = x^k + (x^k - x^{k-1}),$$

where $x_p{}^{k+1}$ is the temporary base point for a new exploratory move, x^k is the result of the exploratory move, and x^{k-1} is the previous base point. The algorithm ends when the step size cannot be reduced anymore.

2.2.2. Initialization Techniques for the Initial Population

The genetic algorithm requires an initial set of solutions called the population to start the process of evolution. In this work, we investigate different initialization techniques for the initial population of solutions. The initial population, if well-distributed, could lead to better performance of the genetic algorithm [30–32]. Usually, the initialization is done using Monte Carlo simulation, taking random values for each gene of the solution [17, 33, 34]. We additionally explore the Latin Hypercube Sampling (LHS) and a Taguchi method.

Next, we explain Latin squares and Orthogonal arrays necessary to understand the LHS and Taguchi method, respectively.

Latin Squares A Latin square is an $n \times n$ array where each of the n different symbols occurs exactly once in each row and each column [35]. Table 2.1 gives an example of a Latin square with n equal to three.

Table 2.1.: An example of a Latin square with $n = 3$.

1	2	3
3	1	2
2	3	1

A Latin hypercube is a generalization of a Latin square concept from two dimensions to an arbitrary number of dimensions.

Orthogonal Arrays Orthogonal arrays generalize the idea of mutually orthogonal Latin squares and are used in the statistical design of experiments [36],

cryptography [37], and software testing [38, 39]. An orthogonal array is an array whose elements come from a fixed finite set of symbols arranged in such a way that for every selection of t columns of the table, all ordered t-tuples of the symbols, formed by taking the elements in each row restricted to these columns, appear the same number of times [40]. An example of an orthogonal array is shown in Table 2.2.

Definition 1 (Orthogonal array [40])**.** *An $N \times k$ array A with entries from S is said to be an orthogonal array with s levels, strength t and index λ (for some t in the range $0 \leq t \leq k$) if every $N \times t$ subarray of A contains each t-tuple based on S exactly λ times as a row.*

N, k, s, t, and λ are the parameters of the orthogonal array, denoted as $OA(N, k, s, t)$[1]. We can omit the λ as the other parameters determine it.

Table 2.2.: An example of an orthogonal array of strength two. The four ordered pairs (2-tuples) formed by the rows of the first and third columns, namely (1,1), (2,1), (1,2), and (2,2), are all the possible ordered pairs of the two-element set, and each appears exactly once. The same would hold with other combinations of two columns.

1	1	1
2	2	1
1	2	2
2	1	2

Following are short descriptions of Latin Hypercube Sampling and Taguchi method techniques.

LATIN HYPERCUBE SAMPLING

The traditional technique for generating samples of parameter values is Monte Carlo simulation (MCS) that randomly samples the cumulative distributions to obtain the samples. The Latin hypercube sampling (LHS) is a technique emphasizing uniform sampling of the univariate distributions [41–43]. LHS accomplishes this by stratifying the cumulative distribution function and randomly sampling within the strata. Uniform sampling increases realization efficiency while randomizing within the strata prevents introducing a bias and avoids the extreme value effect associated with regular stratified sampling. It has been demonstrated for many applications that LHS is a more efficient sampling method compared to MCS [44].

This work also considers the Latin hypercube sampling with multidimensional uniformity (LHSMDU) [45]. This method increases the multidimensional uniformity of a sampling matrix by increasing the statistical distance between realizations. The most closely related modification of Latin hypercube sampling is the *maximinLHS* algorithm proposed by Johnson et al. [46]. The LHSMDU uses MCS to generate many realization inputs and sequentially eliminate realizations near each other in

[1]This notation is not universally accepted.

the multidimensional space. The distributed realizations are then post-processed to enforce univariate uniformity.

Taguchi Methods

Taguchi methods are statistical methods initially developed to improve the quality of manufactured goods [47], but now these methods are applied in various areas, such as engineering [48] and marketing [49]. Taguchi method presents an experimental strategy utilizing a modified and standardized form of experiment design. Additionally, it provides tools to analyze the results of the experiments to determine the design solution producing the best quality. Two primary tools used in the Taguchi method are the orthogonal arrays to design the experiments and the signal-to-noise ratio to analyze them.

The design of an experiment involves several steps - first, selecting independent variables (factors) and the number of levels for each variable. Then, the user selects the orthogonal array and assigns the variables to each of the columns. After conducting all the experiments, the user analyzes the data to find the best values for the variables. In our work, since we use the memetic algorithm and the Taguchi method to reach better distribution of tested parameter values in the initial phase, we do not conduct the Taguchi recommended analysis using a signal-to-noise ratio. Therefore, we only focus on using orthogonal arrays to create the initial set of solutions for the memetic algorithm.

2.3. Implementation

We previously explained the memetic algorithm in a general setting, but here we describe our implementation of the algorithm in detail. Explanations follow the flow of the algorithm.

2.3.1. Solution Representation and Initialization Methods

As discussed, the memetic algorithm works on a set of solutions (population), improving it to reach the optimal solution for the given problem. For the LFI problem, the solutions are arrays representing the LFI parameters - x, y, $delay$, $pulse\ width$, and $laser\ intensity$. Bounds for these parameters are user-defined by setting values for the minimum and maximum value of the parameter and the allowed step.

In our experiments, we evaluate several initialization techniques for the initial solutions of the population. First, we use random initialization. That is a Monte Carlo sampling, where parameter values are taken randomly for each solution, with each value having the same probability of being selected. We do not allow duplicate solutions in the population to enable the random initialization to cover more search space. Additionally, identical solutions do not provide more information, and the laser shots would be wasteful.

The second initialization is using the Latin Hypercube Sampling. We use the existing Python package called *pyDOE2*[2]. There are multiple options on how to

[2] `https://pypi.org/project/pyDOE2/`

sample the points, and we use the default one, which randomizes the points within the intervals. The number of samples defines the number of intervals. The LHS method requires the number of factors and the number of samples to be defined, which, in our case, are the number of LFI parameters (five) and the population size, respectively. Another similar method is Latin hypercube sampling with multidimensional uniformity (LHSMDU) from the Python package *lhsmdu*[3]. This technique should improve the sampling for more dimensions than two, compared to the previously described LHS.

Lastly, we use the Taguchi method by utilizing the Orthogonal Array (OA) package[4], which contains functionality to generate and analyze these types of designs. For generating the arrays and designs, the package uses the exhaustive enumeration algorithm from [50] and the optimization algorithm from [51].

2.3.2. EVALUATION AND FITNESS VALUES

After the initialization of the population, the solutions need to be evaluated. This is done by executing the laser shots with the given parameters. We include a sorting algorithm since the physical operations needed to adjust the laser bench for the laser shot are time-consuming, and the most expensive operations are the motorized stage movements. The motorized stage is used to change the location of the laser shot (parameters x and y). We sort the x and y coordinates using a greedy algorithm with the Manhattan distance between the points as a metric. We start at the lowest x and lowest y coordinate and look for the nearest (x, y) coordinates based on the Manhattan distance. Greedy algorithms provide a combination of the best local choices, which does not guarantee the best global solution, or in our case, the shortest path through all the points. Nevertheless, greedy algorithms are helpful because they are fast and often give good approximations of the optimum.

The evaluation consists of conducting the laser shot several times for all the solutions. Each device response is categorized into one fault class - *pass*, *mute*, *changing*, or *fail*. If the device does not respond in a given time, the response is categorized as *mute*. If the response of the device is expected, it is categorized as *pass*, otherwise as a *fail*. If we get different classes with multiple measurements, the response is categorized as the *changing* class.

We need to have a fitness value for each fault class for the memetic algorithm to distinguish the solutions based on their quality. The fitness of the *changing* class is calculated as $\frac{f_P \cdot N_P + f_M \cdot N_M + f_F \cdot N_F}{N_P + N_M + N_F}$, where f_P, f_M, f_F represent the fitness values for fault class *pass*, *mute*, and *fail*, respectively, and N_P, N_M, and N_F the number of the *pass*, *mute*, and *fail* class occurrences in the number of measurements times. Therefore, the denominator, the sum of N_P, N_M, and N_F is equal to the number of measurements per parameter set. The fitness values for other fault classes are shown in Table 2.3. The values are taken from [17] to present the preference of the fault classes and their relation. Since we are trying to maximize the fitness of the solutions, we set the fitness value for the *fail* class the highest followed by the *mute*

[3] https://pypi.org/project/lhsmdu/
[4] https://pypi.org/project/OApackage/

and *pass* response. Accordingly, the *changing* class with *mute* and *fail* classes results in a higher fitness value than a combination of *mute* and *pass* classes.

Table 2.3.: Fitness values for *pass*, *mute*, and *fail* fault classes.

Fault class	Fitness value
PASS	2
MUTE	5
FAIL	10

2.3.3. GA Operators

After the evaluation, the genetic algorithm part starts creating a new population using the GA operators. For the selection operator, we use a roulette wheel (fitness proportionate) selection. The fitness function assigns a fitness to possible solutions, and it is used to associate a probability of selection with each solution. Let f_i be the fitness of an individual i in the population, then its probability of being selected is $p_i = \frac{f_i}{\Sigma_{j=1}^{N} f_j}$, where N is the number of individuals in the population.

For recombination of the gene material from the selected parents, we use an average crossover. A child solution is created by taking the average value of the parents' values for every gene (parameter). Next is the uniform mutation, where a new random value can replace each gene (parameter) in the solution with probability p_m. We iterate this process until the whole population of size N is again created. However, we also use elitism, which means that we keep the best solutions in the next generation of the population without modifications. The elite size parameter defines the number of those solutions. Therefore, we generate $N - elite\ size$ new solutions for the new population. New solutions are evaluated before conducting the local search.

2.3.4. Local Search

We use the Hooke-Jeeves algorithm for the local search. Only solutions with a fitness larger than 85% of the maximum fitness value (fitness of the *fail* class - 10) are considered. From these solutions, at most, three solutions are randomly selected for the local search. Considering the conditions, the number of solutions taken for local search can vary from zero to three. In our first experiments, we performed a local search on all the solutions with good-enough fitness. This proved to be too many solutions going through the local search because as the population improved, more and more solutions satisfied the requirement. Thus, we limited it to only three randomly selected solutions that fulfill the condition on the fitness values. The steps of the parameter values are defined with their bounds. We only allow two iterations for the exploration phase of the Hooke-Jeeves algorithm, meaning we start with the step doubles, and if no improvements are found, we will only have one more exploration phase with the decreased step size. Again, in our initial testing, we allowed more exploration phases, where we started the local search with the larger

step size. It showed too much exploitation of local space, neglecting the exploration of the search space. Regarding the nature of implemented local search as described in Section 2.2.1, the number of evaluations can vary during the execution of the local search. Therefore, the number of total tested parameter sets can vary with each execution of the memetic algorithm.

2.3.5. TERMINATION CONDITION

Both the genetic algorithm part and local search part constitute one iteration of the memetic algorithm. The termination of the algorithm is done according to the stopping condition. In our experiments, the stopping condition considers only the number of iterations - when the maximum number of iterations is reached, the algorithm is terminated. The user sets the maximum number of iterations as it is the parameter of the memetic algorithm.

2.4. EXPERIMENTAL SETUP AND RESULTS

2.4.1. EXPERIMENTAL SETUP

We performed all the experiments on the same target, a test chip from STMicroelectronics. Considering the company policies, we may disclose only a portion of the confidential information about the target device and the utilized laser bench. The target chip is made using 40nm technology, and for our experiments, we ran an implementation in C programming language displayed in Pseudocode 2.1. The code running on the target device is a test program where data words are copied (loaded) from the non-volatile memory (NVM) into a register. The *trigger_event* function is a monitored event that is used to inject faults at the desired time - on loading a data word into a register (marked as a comment in Pseudocode 2.1). After the fault injection, the register is read, and there is a fault if the register value has changed (fault class *fail*).

Pseudocode 2.1: Pseudocode of the implementation running on the target device.

```
...
trigger_event()
load_register() // injection here
read_register()
...
```

Regarding the laser bench and the parameters, we adopted an infrared laser wavelength, and for each of the five parameters, we used a subset of the available values. We defined the subsets according to the previous knowledge of the target device. The same subsets for all the parameters were applied in all the experiments. We note there are 31 737 600 possible combinations considering the parameter values we use. Each test case is repeated five times for a better statistical representation of the results. We run an online optimization where the results are based on how much *fail* classes are found in a specific number of tested parameters instead of finding only one best parameter set (recall, there is no single best solution since all *fail*

solutions are equally good and there can be multiple distinct regions of parameter values that can lead to *fail* classes), so repeating the same setting five times gives us sufficient information. We note that having a small number of experimental runs is more common for online optimization, e.g., in the case of evolutionary fuzzing [52]. Also, we perform the laser shot for each parameter set five times (number of measurements).

First, we compare the memetic algorithm with a simple random search of the LFI parameters. The random search will test a defined number of parameter sets allowing unique combinations. We test 9 130 different parameter sets and compare the results with the memetic algorithm with the random initialization technique. Running each setup five times, we average the number of total parameter sets tested and the fault classes' percentages. As already mentioned, as the local search in the memetic algorithm is not deterministic in the number of evaluations, the number of tested parameters varies, which is not the case with the random search.

Concerning the memetic algorithm parameters, we use the population size of 36 and the elite size of 2. We use a population size of 36 because we created a mixed-level orthogonal array with 36 samples having three levels for the x, y, and $delay$ parameters and two levels for the $pulse\ width$ and $intensity$. The number of iterations (200) and mutation probability (0.05) are the same throughout all the experiments. Since we have an expensive evaluation of the solutions, we want to keep the number of iterations low, but we chose to use 200 iterations since our population size is small. The mutation probability is low because a high mutation probability can cause the algorithm to behave like a random search. All these parameters influence the algorithm's performance, and we do not claim these values to be optimal. The tuning of these parameters could be further investigated. However, while testing the setup for the experiments, these parameters led to good convergence of the memetic algorithm and were kept for comparison with random search and different initialization methods.

In the experiments with different initialization methods for the initial population of the memetic algorithm, we test with a smaller population size of 36 with elite size 2, and then a larger population of size 128 and elite size 10.

2.4.2. RESULTS

In Table 2.4, with the memetic algorithm, we find significantly more *fail* classes than with the random search having tested 9 034 parameter sets on average (over five runs). To be more precise, we find ≈ 30 times more *fail* classes, and ≈ 3 times more *changing* classes in a similar number of tested parameters (9 130 for the random search, and 9 034 for the memetic algorithm). We conclude that we benefit from using memetic algorithms instead of a simple random search when the exhaustive search is not feasible. An exhaustive search in our case with reduced parameter bounds would take ≈ 59 days if we ran each parameter set once and each evaluation lasted $\approx 0.16s$. However, if we do not reduce the parameter bounds, the exhaustive search could take years. On the other hand, one run of the memetic algorithm in our case was around two hours, with 9 034 parameter sets tested five times.

To further improve the performance of the memetic algorithm, we test different

Table 2.4.: Results on average for random search algorithm and memetic algorithm with random initialization of the population of size 36.

Algo-rithm	Tested pa-rameters	Fails (%)	Chang-ing (%)	Mute (%)	Pass (%)
Random	9 130.0	0.90	2.58	2.54	93.98
Memetic	9 034.6	31.03	7.11	3.64	58.22

initialization methods for generating the initial set of solutions. Thus, instead of generating only random initial solutions, we use techniques that try to distribute the samples taken from the allowed values over the search space and cover as many combinations as possible, considering the levels given to each parameter.

We compare four initialization methods, random initialization, LHS, LHSMDU, and the Taguchi method. For the Taguchi method, we use the orthogonal array of 36 samples, with strength two, and factor levels 3,3,3,2,2 for the variables x, y, $delay$, $pulse\ width$, and $intensity$, respectively. The array can be seen in Table A.1 in Appendix A.1. We use the same population size (36) and elite size (2) as the first experiment with the memetic algorithm.

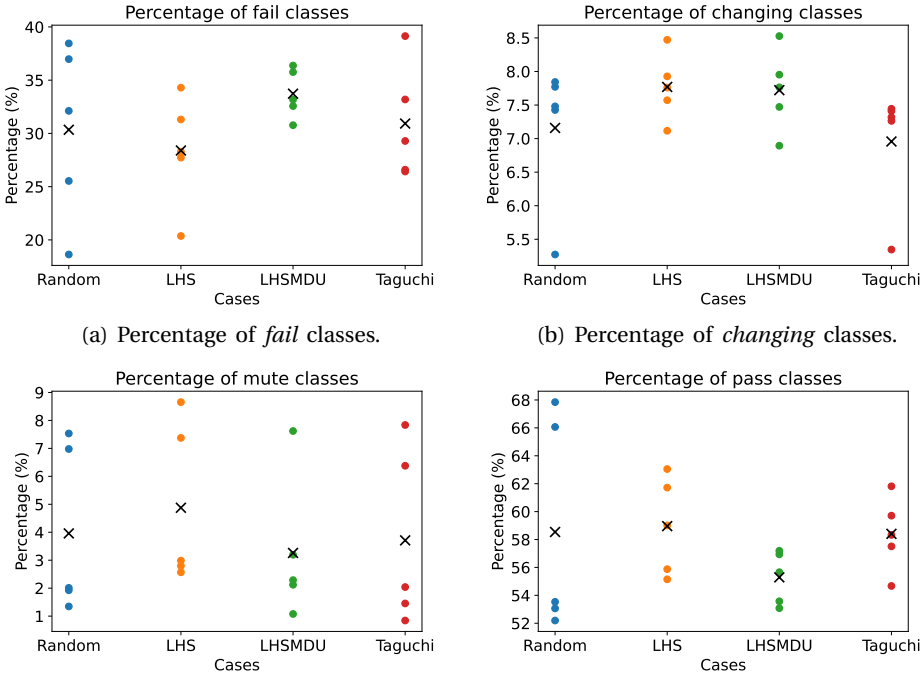
With the memetic algorithm, we are trying to maximize the fitness function. Therefore, we prefer laser injection to result in a corrupt register value which we categorize as the *fail* class. Looking at the averaged results from Table 2.5, the difference between the initialization techniques is not significant. Furthermore, because of the variance in the number of tested parameters, we see that the LHSMDU technique has the largest number of tested parameters and the highest percentage of *fail* classes. We conclude that all the initialization techniques, on average, result in similar results. Therefore, if we have time and can use the average results, we might not need to consider using some of the proposed techniques to improve the sampling for the initial population compared to the random sampling.

Table 2.5.: Results on average for all initialization techniques in experiments with population size 36.

Initiali-zation	Tested pa-rameters	Fails (%)	Chang-ing (%)	Mute (%)	Pass (%)
Random	9 034.6	31.03	7.11	3.64	58.22
LHS	8 812.8	28.80	7.73	4.59	58.88
LHS-MDU	9 605.0	33.92	7.67	3.02	55.40
Taguchi	9 070.2	31.58	6.87	3.33	58.23

However, we also looked into individual results as we are interested in the variation of the results. In Figure 2.1, we show percentages of different fault classes for settings with all the initialization techniques. For each initialization technique, we show the

percentage for each experiment to see the variation and mark the average percentage with an X. What can be seen is that even though with the random initialization of the population we can get excellent results, we can also end with the worst results considering the percentage of *fail* classes (which we prefer) in Figure 2.1a. Similarly, using the LHSMDU technique, we had the lowest variation, but the results did not reach as good results as random initialization and Taguchi in one of the experiments.



(a) Percentage of *fail* classes.

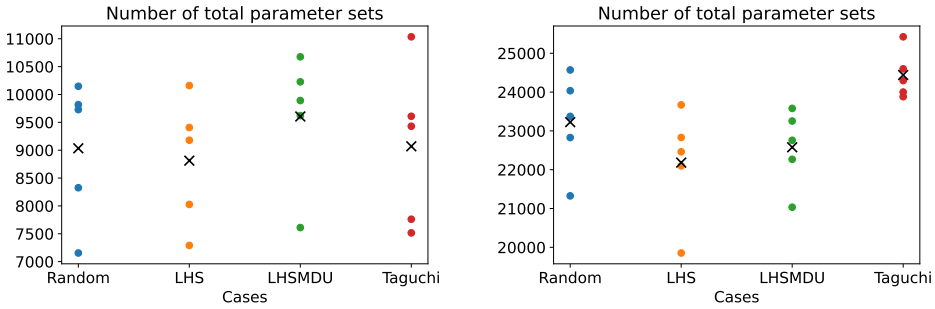
(b) Percentage of *changing* classes.

(c) Percentage of *mute* classes.

(d) Percentage of *pass* classes.

Figure 2.1.: Percentages of *fail*, *changing*, *mute*, and *pass* classes with all initialization techniques for a setting with a population size of 36 and elite size of 2.

Looking at the number of tested parameters in Figure 2.2a, we observe that, in the case with a smaller population, the ranges among the different test cases are similar. With the LHSMDU, we had a more consistent number of tested parameters, with only one outlier. By combining the percentages of all the fault classes and the number of tested parameters, we see that the stability of the results is not coming directly from the number of tested parameters. *Having tested more parameters does not necessarily lead to finding more interesting parameter sets.* For example, in the test case with the LHSMDU technique, we often tested more parameters than random initialization. However, the percentage of *fail* classes is larger with random initialization than LHSMDU in some cases. Considering the best results on the number of found *fail* classes and the stability of the results (consistency), we can see that LHSMDU is the best choice, followed by the Taguchi method.



(a) The number of tested parameter sets with population size of 36.

(b) The number of tested parameter sets with population size of 128.

Figure 2.2.: The number of tested parameter sets in a setting with a smaller population (population size of 36 and elite size of two) in Figure 2.2a and a larger population (population size of 128 and elite size of ten) in Figure 2.2b.

Next, we increase the population size to 128 and elite size to ten and run the same experiments. We generate an orthogonal array of 128 samples for the Taguchi method, with strength two and factor level two for all the variables. The array can be seen in Table A.2 in Appendix A.1.

Table 2.6 gives the averaged results, where we observe that, similarly, the experiments show, on average, no initialization technique is significantly better. Again the largest percentage of *fail* classes was found by the test case with the largest number of tested parameters. Here, that was the Taguchi method. The percentage of the *fail* classes is lower in these experiments with a larger population because only a limited number of *fail* classes can occur with our target and test program. We test more unique parameter sets since we have a larger population but the same number of iterations. The more parameters we test, the closer we get to the exhaustive search, which would only show the true distribution of the fault classes. However, we did not run the exhaustive search because, as already mentioned, it would take ≈ 59 days to run it with our reduced intervals if we ran each parameter set only once and each laser shot takes $\approx 0.16s$.

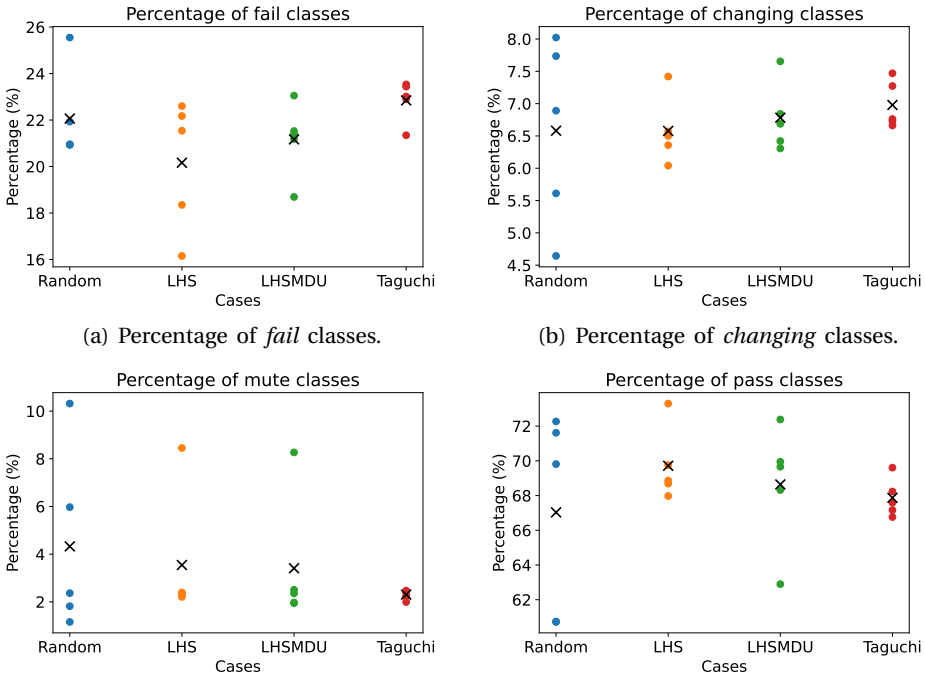
Looking at the number of tested parameters with the larger population in Figure 2.2b, with the Taguchi method, the number became more stable, and the variation is reduced. This is somewhat reflected in the results shown in Figure 2.3 presenting the variance of the percentages of different fault classes. Again, we see that the test case that tested the most parameters in one of the runs (Taguchi) did not lead to the highest percentage of *fail* classes (random initialization).

We can see that the experiments with Taguchi initialization are more stable with a larger population. Considering the results with the LHSMDU initialization, the stability is slightly worse than with a smaller population. The LHSMDU divides the interval for the parameter in the number of samples we want to have, which,

Table 2.6.: Results on average for all initialization techniques in experiments with population size 128.

Initialization	Tested parameters	Fails (%)	Changing (%)	Mute (%)	Pass (%)
Random	23 226.6	22.07	6.55	4.19	67.18
LHS	22 183.0	20.24	6.55	3.42	69.80
LHS-MDU	22 578.8	21.19	6.77	3.32	68.72
Taguchi	24 440.0	22.85	6.99	2.31	67.86

in this case, equals 128. For some smaller intervals, this means many values get mapped to the same value, which could be why the performance of this technique has deteriorated with larger population size. On the other hand, the stability of the random initialization technique is improved with a larger population as we do not allow duplicate parameter sets.



(a) Percentage of *fail* classes.

(b) Percentage of *changing* classes.

(c) Percentage of *mute* classes.

(d) Percentage of *pass* classes.

Figure 2.3.: Percentages of *fail*, *changing*, *mute*, and *pass* classes with all initialization techniques for a setting with a population size of 128 and elite size of 10.

Considering the LHS method for initialization, LHSMDU and Taguchi would be preferred over simple LHS for both smaller and larger populations. The weaker performance of this technique could be the fact that we experiment with five parameters. We have a five-dimensional search space making it harder for LHS to reach a better distribution of samples than random sampling.

We conclude that we can benefit from using sampling techniques that produce well-distributed samples, such as the LHSMDU or Taguchi method, to create the initial population. However, there could be more techniques that can provide these characteristics. *Additionally, we observe that performance with random initialization improves with a larger population, so it is more important to have a better sampling method when working with a smaller population.* Moreover, we can see that the difference between the percentage of found *fail* classes in the worst and best case decreases with the size of the population. In the test cases with a smaller population (Table 2.5), the difference between the worst (LHS) and best case (LHSMDU) is 5.12%, while with the larger population (Table 2.6), the difference between the worst (LHS) and best case (Taguchi method) is 2.61%.

2.5. CONCLUSIONS AND FUTURE WORK

This work showed that using the memetic algorithm can greatly improve the results compared to the random search algorithm. While with 9 130 tested parameters, the random search found 0.9% of the *fail* classes, a memetic algorithm with 9 034 tested parameters had 31.03% of *fail* classes. The memetic algorithm used a random initialization of the population, and since the initial population can have a significant impact on the results of the algorithm itself, we tested different techniques for initialization. We experiment with techniques that aim to achieve a well-distributed set of samples for exploring more of the search space and possible combinations: the Latin Hypercube Sampling (LHS) and Taguchi methods. There are two implementations of the LHS. First, the simple one, where we sample from equally probable intervals of the variable range, and another implementation (LHSMDU), which sequentially eliminates realizations near each other in the multidimensional space. On average, in our results, the initialization technique does not significantly influence the performance of the memetic algorithm. However, when we consider all the experiments with the same setup, we see that it is beneficial to use other techniques rather than simple random sampling. With a smaller population size, LHSMDU has excellent stability, with the random initialization being the least stable and less reliable. With a larger population size, Taguchi showed the best results, and it was stable. We recommend using techniques that promote a better distribution of randomly selected samples for initializing the population for the memetic algorithms when working with a smaller population. It could make the algorithm more robust and avoid fast convergence to a local optimum.

This work shows that memetic algorithms can be very beneficial and efficient in finding faults for laser fault injection attacks. Therefore, it offers excellent potential for improving these attacks. However, since this work is limited to one target sample, we do not consider the evaluation of the memetic algorithm complete. Therefore,

we plan to test the algorithm on different targets to test the approach in a more general setting. We also plan to distinguish between exploitable faulty outputs found by memetic algorithms since this work considers only parameter sets that lead to a faulty output (a change in the register value, in our case) that might be exploitable. Therefore, this includes performing attacks such as DFA. Our memetic algorithm can also be tested with different fault injection methods, such as voltage glitching and EMFI. Further, this work can be improved by considering more population sizes as it could benefit some of the initialization techniques. For example, decreasing the population size from 128 could benefit the LHSMDU method. For Taguchi, we can experiment with a larger number of factor levels to help improve the algorithm's performance using this technique. Another interesting research direction could consider which parameters would benefit more from a good distribution in the initial set. Here, we considered all the parameters, but there could be a subset of the parameters that would benefit more than others.

REFERENCES

[1] M. Krček, D. Fronte, and S. Picek. “On the importance of initial solutions selection in fault injection”. In: *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2021, pp. 1–12.

[2] P. C. Kocher, J. Jaffe, and B. Jun. “Differential Power Analysis”. In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. Ed. by M. Wiener. CRYPTO ’99. Springer. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 388–397. ISBN: 3540663479.

[3] J.-J. Quisquater and D. Samyde. “ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards”. In: *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*. E-SMART ’01. Springer. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 200–210. ISBN: 3540426108. URL: http://dl.acm.org/citation.cfm?id=646803.705980.

[4] P. C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO ’96. Springer. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 104–113. ISBN: 3540615121.

[5] E. Biham and A. Shamir. *Differential Fault Analysis of Secret Key Cryptosystems*. 1997.

[6] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta. “Fault Sensitivity Analysis”. In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Ed. by S. Mangard and F.-X. Standaert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 320–334. ISBN: 978-3-642-15031-9.

[7] T. Fuhr, E. Jaulmes, V. Lomné, and A. Thillard. “Fault Attacks on AES with Faulty Ciphertexts Only”. In: *Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. FDTC ’13. USA: IEEE Computer Society, 2013, pp. 108–118. ISBN: 9780769550596. DOI: 10.1109/FDTC.2013.18. URL: https://doi.org/10.1109/FDTC.2013.18.

[8] C. Clavier. “Secret external encodings do not prevent transient fault analysis”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2007, pp. 181–194.

[9] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas. “SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, Issue 3 (2018), pp. 547–572. DOI: 10.13154/tches.v2018.i3.547-572. URL: https://tches.iacr.org/index.php/TCHES/article/view/7286.

[10] S. P. Skorobogatov and R. J. Anderson. “Optical fault induction attacks”. In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2002, pp. 2–12.

[11] C. H. Kim and J.-J. Quisquater. “Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures”. In: *IFIP International Workshop on Information Security Theory and Practices*. Springer. 2007, pp. 215–228.

[12] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz. “Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller”. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE. 2013, pp. 77–88.

[13] M. Hutter and J.-M. Schmidt. “The temperature side channel and heating fault attacks”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 219–235.

[14] R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub. “Glitch it if you can: parameter search strategies for successful fault injection”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 236–252.

[15] S. Picek, L. Batina, D. Jakobović, and R. B. Carpi. “Evolving genetic algorithms for fault injection attacks”. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2014, pp. 1106–1111.

[16] S. Picek, L. Batina, P. Buzing, and D. Jakobovic. “Fault Injection with a new flavor: Memetic Algorithms make a difference”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2015, pp. 159–173.

[17] A. Maldini, N. Samwel, S. Picek, and L. Batina. “Genetic algorithm-based electromagnetic fault injection”. In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2018, pp. 35–42.

[18] L. Wu, G. Ribera, N. Beringuier-Boher, and S. Picek. “A fast characterization method for semi-invasive fault injection attacks”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2020, pp. 146–170. ISBN: 978-3-030-40185-6.

[19] P. Moscato. “On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms”. In: *Caltech Concurrent Computation Program* (Oct. 2000).

[20] P. Moscato and L. Mathieson. “Memetic algorithms for business analytics and data science: a brief survey”. In: *Business and consumer analytics: new ideas* (2019), pp. 545–608. URL: https://doi.org/10.1007/978-3-030-06222-4_13.

[21] W. Sheng, P. Shan, J. Mao, Y. Zheng, S. Chen, and Z. Wang. “An Adaptive Memetic Algorithm With Rank-Based Mutation for Artificial Neural Network Architecture Optimization”. In: *IEEE Access* PP (Sept. 2017), pp. 1–1. DOI: 10.1109/ACCESS.2017.2752901.

[22] P. Rakshit, D. Banerjee, A. Konar, and R. Janarthanan. “An Adaptive Memetic Algorithm for Multi-robot Path-Planning”. In: *Swarm, Evolutionary, and Memetic Computing*. Ed. by B. K. Panigrahi, S. Das, P. N. Suganthan, and P. K. Nanda. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 248–258. ISBN: 978-3-642-35380-2.

[23] W. E. Hart. “Adaptive global optimization with local search”. PhD thesis. Citeseer, 1994.

[24] K. Michalak. “Evolutionary algorithm with a directional local search for multiobjective optimization in combinatorial problems”. In: *Optimization Methods and Software* 31.2 (2016), pp. 392–404.

[25] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan. “A multi-facet survey on memetic computation”. In: *IEEE Transactions on Evolutionary Computation* 15.5 (2011), pp. 591–607.

[26] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Vol. 53. Springer, 2003.

[27] J. H. Holland. *Adaptation in Natural and Artificial Systems*. second edition, 1992. Ann Arbor, MI: University of Michigan Press, 1975.

[28] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[29] R. Hooke and T. A. Jeeves. ““ Direct Search” Solution of Numerical and Statistical Problems”. In: *J. ACM* 8 (1961), pp. 212–229.

[30] H. Maaranen, K. Miettinen, and A. Penttinen. “On initial populations of a genetic algorithm for continuous optimization problems”. In: *Journal of Global Optimization* 37.3 (2007), p. 405.

[31] V. Toğan and A. T. Daloğlu. “An improved genetic algorithm with initial population strategy and self-adaptive member grouping”. In: *Computers & Structures* 86.11-12 (2008), pp. 1204–1218.

[32] S. Poles, Y. Fu, and E. Rigoni. “The effect of initial population sampling on the convergence of multi-objective genetic algorithms”. In: *Multiobjective programming and goal programming*. Springer, 2009, pp. 123–133.

[33] D. Whitley. “A genetic algorithm tutorial”. In: *Statistics and computing* 4.2 (1994), pp. 65–85.

[34] F. H.-F. Leung, H.-K. Lam, S.-H. Ling, and P. K.-S. Tam. “Tuning of the structure and parameters of a neural network using an improved genetic algorithm”. In: *IEEE Transactions on Neural networks* 14.1 (2003), pp. 79–88.

[35] K. Kishen. “On Latin and hyper-Graeco-Latin cubes and hyper-cubes”. In: *Current Science* 11.3 (1942), pp. 98–99.

[36] C. R. Rao. “Some combinatorial problems of arrays and applications to design of experiments”. In: *A Survey of Combinatorial Theory*. Elsevier, 1973, pp. 349–359.

[37] C. Koukouvinos, B. Lappas, and D. Simos. “Encryption schemes using orthogonal arrays”. In: *Journal of Discrete Mathematical Sciences and Cryptography* 12.5 (2009), pp. 615–628.

[38] I. S. Dunietz, W. K. Ehrlich, B. Szablak, C. L. Mallows, and A. Iannino. “Applying design of experiments to software testing: experience report”. In: *Proceedings of the 19th international conference on Software engineering*. 1997, pp. 205–215.

[39] L. Lazić and D. Velašević. “Applying simulation and design of experiments to the embedded software testing process”. In: *Software Testing, Verification and Reliability* 14.4 (2004), pp. 257–282.

[40] A. Hedayat, N. Sloane, and J. Stufken. *Orthogonal Arrays: Theory and Applications*. Springer Science & Business Media, 1999.

[41] P. Audze and V. Eglais. “New approach to planning out of experiments”. In: *Problems of dynamics and strength (in Russian)* 35 (1977), pp. 104–107.

[42] R. Iman, J. Helton, and J. Campbell. “An Approach to Sensitivity Analysis of Computer Models: Part I—Introduction, Input Variable Selection and Preliminary Variable Assessment”. In: *J Qual Technol* 13 (July 1981), pp. 174–183. DOI: 10.1080/00224065.1981.11978748.

[43] R. L. Iman, J. M. Davenport, and D. K. Zeigler. “Latin hypercube sampling (program user’s guide). [LHC, in FORTRAN]”. In: (Jan. 1980).

[44] M. Mckay, R. Beckman, and W. Conover. “A Comparison of Three Methods for Selecting Vales of Input Variables in the Analysis of Output From a Computer Code”. In: *Technometrics* 21 (May 1979), pp. 239–245. DOI: 10.1080/00401706.1979.10489755.

[45] J. L. Deutsch and C. V. Deutsch. “Latin hypercube sampling with multidimensional uniformity”. In: *Journal of Statistical Planning and Inference* 142.3 (2012), pp. 763–772. ISSN: 0378-3758. DOI: https://doi.org/10.1016/j.jspi.2011.09.016. URL: https://www.sciencedirect.com/science/article/pii/S0378375811003776.

[46] M. Johnson, L. Moore, and D. Ylvisaker. “Minimax and maximin distance designs”. In: *Journal of Statistical Planning and Inference* 26.2 (1990), pp. 131–148. ISSN: 0378-3758. DOI: https://doi.org/10.1016/0378-3758(90)90122-B. URL: https://www.sciencedirect.com/science/article/pii/037837589090122B.

[47] G. Taguchi and A. P. Organization. *Introduction to Quality Engineering: Designing Quality Into Products and Processes*. Asian Productivity Organization, 1986. ISBN: 9789283310846. URL: https://books.google.hr/books?id=1NtTAAAAMAAJ.

[48] G. Taguchi and V. Cariapa. “Taguchi on robust technology development”. In: (1993).

[49] P. Selden. *Sales Process Engineering: A Personal Workshop*. ASQC Quality Press, 1996. ISBN: 9780873894180. URL: https://books.google.nl/books?id=ZqfAQAACAAJ.

[50] E. Schoen, P. Eendebak, and M. Nguyen VM. “Complete Enumeration of Pure-Level and Mixed-Level Orthogonal Arrays”. In: *Journal of Combinatorial Designs* 18 (Mar. 2009), pp. 123–140. DOI: 10.1002/jcd.20236.

[51] P. T. Eendebak and A. R. Vazquez. “OApackage: A Python package for generation and analysis of orthogonal arrays, optimal designs and conference designs”. In: *Journal of Open Source Software* 4.34 (2019), p. 1097. DOI: 10.21105/joss.01097. URL: https://doi.org/10.21105/joss.01097.

[52] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos. “VUzzer: Application-aware Evolutionary Fuzzing”. In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/vuzzer-application-aware-evolutionary-fuzzing/.

3

THE MORE YOU KNOW: IMPROVING LASER FAULT INJECTION WITH PRIOR KNOWLEDGE

We consider finding as many faults as possible on the target device in the laser fault injection security evaluation. Since the search space is large, we require efficient search methods. Recently, an evolutionary approach using a memetic algorithm was proposed and shown to find more interesting parameter combinations than random search, which is commonly used. Unfortunately, once a variation on the bench or target is introduced, the process must be repeated to find suitable parameter combinations anew.

To negate the effect of variation, we propose a novel method combining a memetic algorithm with a machine learning approach called a decision tree. Our approach improves the memetic algorithm by using prior knowledge of the target introduced in the initial phase of the memetic algorithm. In our experiments, the decision tree rules enhance the performance of the memetic algorithm by finding more interesting faults in different samples of the same target. Our approach shows more than two orders of magnitude better performance than random search and $\approx 60\%$ *better performance than previous state-of-the-art results with a memetic algorithm. Another advantage of our approach is human-readable rules, allowing the first insights into the explainability of target characterization for laser fault injection.*

This chapter has been published as M. Krček, T. Ordas, D. Fronte, and S. Picek. "The more you know: Improving laser fault injection with prior knowledge". In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 18–29.

3.1. INTRODUCTION

A secure device should be designed so that as little as possible secret information can leak to an attacker. Even if the algorithms (e.g., cryptographic algorithms) running on the device are mathematically secure, it does not mean the attacks are impossible. One well-known and powerful category of the attacks is called the implementation attacks, which aim at the weaknesses of the implementation and not the algorithms themselves. Two common implementation attacks are side-channel attacks (SCAs) and fault injection (FI) attacks. Side-channel attacks are passive, and the attacker tries to obtain some side-channel information from the execution on the hardware, such as time [2], power consumption [3], and electromagnetic radiation [4]. From this information, it is possible to obtain secret information. Fault injection attacks are active attacks where the attacker tries to force the device to make errors during its execution. This way, the adversary could reach some malicious goal, like passing the authentication or obtaining secret information from the target device. Both implementation attacks are prevalent in security evaluations but can be challenging to deploy in practice.

This work focuses on laser fault injections (LFI), as introduced by Skorobogatov et al. [5]. Laser fault injections are very powerful as they provide high precision for injecting faults by producing single-bit faults [6]. However, multiple parameters define the laser injections, such as location on the device (x and y coordinates), distance from the device to the microscope lens, lasers settings like *intensity*, *delay*, and *pulse width*. The attacker's goal is to find the parameters that lead to successful fault injection causing the device to skip instructions, change values in memory, etc. Afterward, the attackers can use various techniques, e.g., differential fault analysis (DFA) [7] to reach their malicious goals.

As mentioned, there are *many parameters* to be defined and *many possibilities* to consider. Consequently, the search space is large, and finding exploitable faults is difficult, making this attack challenging to perform. While an exhaustive search is commonly not reasonable, the location on the target is often searched in a grid-like manner using the same laser settings. The laser settings are usually based on previous experience, which might be lacking if the target or the bench is new. This process is often time-consuming, so a random search is applied as an alternative. However, both approaches could *miss interesting parameter sets* that lead to faults. Additionally, there is a problem with the *transferability of the results* in fault injection attacks [8, 9]. Results obtained with one setup are not necessarily easily reproduced on another, and changing the target can cause changes in the optimal parameters for the injection. Thus, the attackers need to repeat the same process of finding the optimal parameters for every change, increasing the difficulty of a successful FI.

There is a need for better, automated approaches to efficiently search the parameter space for FI. Evolutionary algorithms were proposed for laser fault injections [10], as well as other types of injections such as voltage glitching [11–13] and electromagnetic fault injection [14] since laser fault injections are not the only type of injections suffering from the previously described issues. Methods applied are either genetic algorithm (GA) or memetic algorithm (MA), which combines a genetic algorithm with local search. Besides evolutionary approach, hyperparameter

optimization techniques [15] and reinforcement learning [16] were also investigated. Additionally, for laser injections, machine learning (ML) was used in the fast characterization method proposed in [8]. The authors generate a sensitivity curve and use neural networks (multilayer perceptron) to predict the target's behavior from the obtained data of the sensitivity curve. The authors also discuss the transferability issue and test their method on different samples of the same target.

In this work, we focus on the security evaluation process. We aim to find as many fault injection settings where the device does not behave as expected. At the same time, we do not consider exploiting the obtained faults with any specific attack. We start from the memetic algorithm for the LFI presented in [10] since it provides state-of-the-art results to the best of our knowledge. Then, we use the concept of prior knowledge to enhance the algorithm performance further and, consequently, find more faults. Our approach combines machine learning and a memetic algorithm where we use decision trees (DTs) to extract knowledge about the target's behavior and use it in the initialization phase of the MA.

Our main contributions are:

- We develop an approach to increase the number of desired outcomes in the initial population (i.e., parameters that lead to faults), which helps the memetic algorithm find more faults with less tested parameters. The improvement happens because the algorithm can distinguish between desired and other outcomes from the first iteration and learn from already found parameter sets. At the same time, the process is more efficient than running a memetic algorithm for more iterations as it can easily get stuck in specific regions of search space. Our results show that we can improve the efficiency of the search process by $\approx 60\%$.
- Our approach allows us to tackle the transferability issue. Changing the setup or the target requires repeating exploration of the search space to find faults. However, if the same target type is used, and we only change different samples of the same target while keeping (almost) the same bench setup, an intuitive assumption is that the resulting parameters should be (mostly) transferable. Indeed, minor differences in hardware introduced during the production or preparation of the target for conducting the laser injection (mechanical thinning of the backside silicon substrate) could be negligible for the LFI transferability. We test this assumption using our approach on different samples of the same target and slightly modified bench setups. We observe that prior knowledge is transferable and helps characterize the target more efficiently.
- As the decision tree outputs rules, we provide the initial results on the explainability of fault injection target characterization.

3.2. BACKGROUND

3.2.1. DECISION TREE (DT)

The decision tree is a supervised machine learning technique for classification and regression problems. More specifically, it is a tree-based technique in which any

path starting at the root node separates data based on specific criteria in the internal nodes (also called decision nodes) until the outcome in the leaf node is reached. An example of a decision tree is given in Figure 3.1, where the mentioned nodes are distinguished. We consider a classification problem with two features and three target classes, and this example demonstrates how the conditions can be expressed for categorical and continuous numeric variables. Decision trees are easier

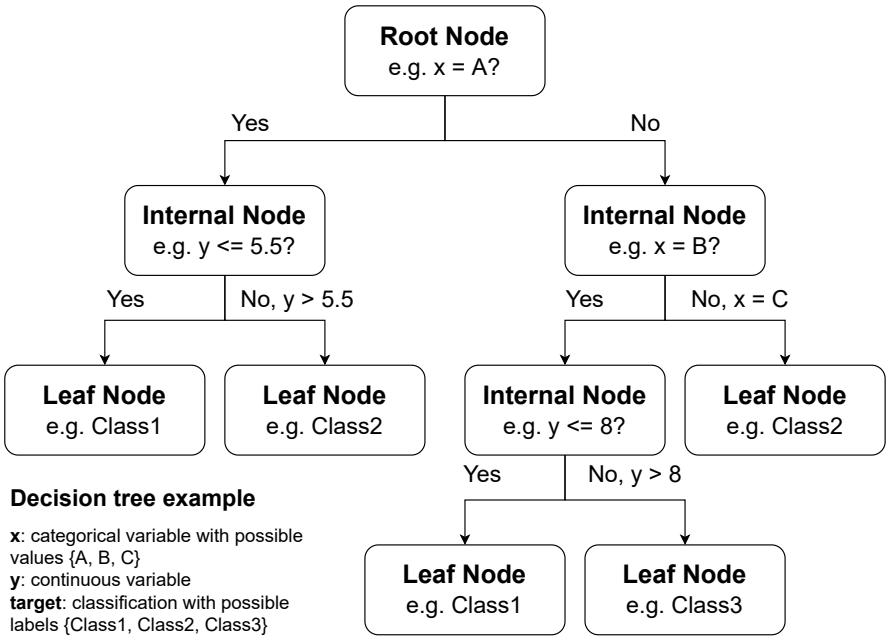


Figure 3.1.: Decision tree example. A possible structure of a binary decision tree is shown with different node types - root node, internal node, and leaf node. There are examples of conditions for both categorical variable *x* and continuous numeric variable *y*. The example has three classes visible in the leaf nodes.

to understand than, for example, Artificial Neural Networks (ANNs) because of their tree-like structure that mimics the human decision-making process and the rules generated from these trees. The rules are often structured as *if-then* statements.

Decision tree learning is done by finding the best set of conditions based on the features' values in the training data to split the datasets into subsets of instances corresponding to one (dominant) target outcome (class/label). There are many algorithms proposed on how trees can be constructed. Several surveys [17–19] discuss different decision tree algorithms, such as Iterative Dichotomies 3 (ID3) [20], successor of ID3 - C4.5 [21], an extension of C4.5 algorithm - C5.0 algorithm [22], Classification And Regression Tree (CART) [23], CHi-squared

Automatic Interaction Detector (CHAID) [24], and Quick, Unbiased and Efficient Statistical Tree (QUEST) [25]. Of the mentioned algorithms, the most popular and successful are C4.5, C5.0, and CART. However, C5.0 is not publicly available. Between C4.5 and CART, based on the performance, there is no superior one, so we use CART as the implementation is easier to integrate. We first describe the general procedure and mention the differences between various proposed learning algorithms.

The decision tree learning algorithm starts with a collection of samples described by a specific number of input features. Each instance has a particular outcome (target): a numeric value for regression or a class/label for classification. Since we consider a classification problem where we predict fault classes based on the laser fault injection parameters, we further explain the decision tree learning process considering a classification problem. We aim to have pure subsets of samples in the decision tree, where pure means that all the instances in the subset belong to one specific target class. The learning algorithm selects the conditions by evaluating the resulting subsets. The idea is to lower the impurity of the subsets with each split until the subsets become pure. The internal nodes represent the conditions over the feature values, and each branch from that node is either one possible value of that feature or a subset of those values. A few examples of different types of conditions are visible in Figure 3.1. The branches lead to other internal nodes with different conditions over the input features or leaf nodes. Leaf nodes represent the target classes or a probability distribution over the target classes. Passing the internal node conditions and reaching the leaf node, the data sample gets classified as the specific class indicated by the leaf node. The goal for classification trees is to arrive at different classes with the least number of splits and the lowest misclassification error rates. However, there are trade-offs to be considered to avoid overfitting to training data that causes the model to generalize poorly.

The difference in the many proposed learning algorithms is how they measure which split is better and which feature and its corresponding value should be used to separate the dataset in the best way. Another difference is how many branches can exist from a condition: only two (binary) or multiple partitions. For multiple partitions, a good example is a categorical variable. If a categorical variable has only three possible values, the condition could have three branches, each corresponding to one of the variable's values. Splitting the dataset is done when the subset at a node is pure or splitting no longer improves the prediction performance. Additionally, the learning algorithm can stop if, for example, the tree reaches a defined maximum depth or a minimal number of samples for the split.

CART Learning Algorithm

The CART algorithm supports classification and regression problems. Decision trees created by the CART algorithm are binary trees, meaning that the node branches in only two subtrees. For numeric feature A, the internal node criteria are defined as $A \leq h, A > h$, where the threshold h is found by sorting the values of feature A and then choosing the split between successive values that are best depending on the criterion utilized. The midpoint between two consecutive values is selected in CART as the threshold. For categorical values, the conditions are all subsets from

the possible set of categorical values. Different algorithms use different measures to select the condition in the internal nodes. CART originally proposed a Gini index for the splitting measure, also known as Gini impurity. The Gini index measures how often a randomly chosen sample from the set would be incorrectly classified. If Gini is 0, it expresses the purity of classification, i.e., all elements belong to a specified class. If Gini is 1, it indicates the random distribution of elements across various classes, and 0.5 shows an equal distribution of samples for all classes. On the other hand, Information Gain is calculated by subtracting the weighted entropy of each child node from the parent node, where entropy is a measure of impurity or randomness in the data points. Since the process continues until each subset contains samples of the same class or the splits no longer offer improvement, the resulting tree can often be very large and complex. Additionally, the generalization ability of such a tree beyond the training data might be poor. Thus, the CART includes pruning of the trees. The idea is to remove subtrees that do not contribute to the classification accuracy of unseen data. The pruning in CART is done from the bottom of the tree, examining each subtree. If replacing the subtree with a leaf or its most frequently used branch leads to a lower prediction error rate, the tree is pruned accordingly. We use the CART algorithm implemented in the Python package *scikit-learn* [26]. The *scikit-learn* implementation of the CART algorithm uses the total sample weighted impurity of the terminal nodes instead of estimating the prediction error rate. The impurity of a node depends on the criterion used. This way, there is no need for a test dataset for estimating the misclassification rate.

3.2.2. MEMETIC ALGORITHM (MA)

A memetic algorithm consists of a population-based search and a local search for some individuals [27]. The population-based search, in our case, is a genetic algorithm (GA), where the population is a set of solutions for the optimization problem. For each specific optimization problem, there is a particular solution representation. Once the solution representation is defined, the algorithm generates the initial population using an initialization procedure. Usually, the initialization procedure is random sampling. Then, the algorithm uses the GA operators. First, the individuals from the population get selected by a selection operator. The selection determines which solutions produce offspring. The fitness function samples the population to allocate the parent solutions. Fitness is a relative measure of how good solution A is compared to the rest of the current population or a different solution B. Usually, solutions with better fitness are selected as parents, reasoning that these solutions have "better genes". After selecting the parent solutions, the parents' genes are recombined in the crossover operator. Created offspring can then be modified using the mutation operator. The mutation operator introduces new variations in the solutions and thus the population. Mutation can be done by randomly changing some of the genes in the solutions or replacing an offspring with a completely new solution. A commonly used mechanism is elitism, where one or more best solutions from the current generation are placed directly into the next population. This mechanism ensures that the optimal solution will not be lost in the following generations once it is found. Before starting a new iteration, a portion of solutions

is selected for the local improvements. A good option for the local search is the Hooke-Jeeves algorithm, an optimization algorithm that does not require derivatives of the objective function [28]. This algorithm in the context of FI is explained in [10].

3.3. PROPOSED METHOD

Our proposed method combines the explained decision trees and the current state-of-the-art memetic algorithm. The MA we use is an existing implementation described in [10]. The representation of the laser fault injection solution is an array of numbers indicating laser fault injection parameters - *x*, *y*, *trigger delay*, *laser pulse width*, and *intensity*. The user sets the bounds for these parameters that can be used in the algorithm. The algorithm starts with the initial population, created by an initialization method. In [10], the authors explored different initialization methods. As no specific one achieved significantly better results than the usual random sampling, we keep it as the initialization method for MA to compare with the newly proposed approach. The difference in the new approach is in the initialization method. We propose to use decision tree rules for initializing the population. We perform a campaign with laser fault injections on a specific target to obtain the data. We can then analyze the acquired data to determine what parameters caused *fail* responses vs. the other responses. We utilize the decision tree algorithm to learn when we obtain each fault class depending on the parameters. That is a classification problem, as our labels are categorical values *pass*, *mute*, *fail*, and *changing*. These are the names of fault classes that correspond to a specific device response, and more details can be found in Section 3.4.1. The input features are the five LFI parameters. The trained model can be used to predict the behavior of the target on parameter combinations that were not tested. Additionally, we can use it to analyze what parameters are more relevant for achieving *fail* responses if there are such. We, however, use the model to improve performance when conducting another campaign on a different integrated circuit (IC). Specifically, we test the approach on transferability between different samples of the same target.

Since the decision trees can be easily translated to *if-then* rules, we can acquire rules for the *fail* fault class. We select a trained model and traverse the decision tree to extract paths where the leaf nodes are *fail* classes. Since the paths contain conditions on the different parameters, we get intervals for the parameters that lead to *fail* classes based on the model. We denote the combination of the intervals for parameters from one path in the tree as a rule. After we obtain the rules for the *fail* class, we apply them to initialize the memetic algorithm's population. We use only a maximum of 25 rules with the highest number of classified *fail* examples as our initial exploration gave good results with this number. The limit can be further investigated for an even better choice. To create one solution for the population, we first select one rule if there are many. The probability of selecting a rule is proportional to the number of *fail* examples classified by the rule. Since the rule defines the parameters' intervals, we select the parameter value uniformly at random from the defined intervals. Each solution can be created from a different rule, and the whole population is built in the same manner.

One could argue that this might reduce the exploration ability of the memetic algorithm. However, since the MA operators (crossover and mutation) are not modified to use the prior knowledge, the algorithm can still explore outside initial solutions and exploit the "head start".

The rest of the memetic algorithm in the proposed methods remains the same as in [10], except for the crossover operator. Instead of using the average crossover, we use a uniform crossover. With uniform crossover, values for the child solution are taken randomly from the two selected parents, and the probability is equal for both parents' parameter values. Compared to the average crossover that calculates the mean for all parents' parameters and leads to one specific child every time, the uniform crossover can create more different children with the same parents. That leads to more possibilities which can lead to better results sooner.

The evaluation includes performing the laser shots and acquiring the devices' responses. The responses are categorized into fault classes - *pass*, *mute*, and *fail*. Each parameter set is tested several times, so if multiple fault classes appear with the same solution, the class for that parameter combination is *changing*. The MA does not work directly with these fault classes. We use a fitness value, which is given to each fault class. The fitness values for the *pass*, *mute*, and *fail*, taken from [14], are 2, 5, and 10, respectively. The values display the preference of the fault classes, where the *fail* class, being the desired one, has the highest fitness value. As in [10], the fitness of the *changing* class is calculated as $\frac{f_P \cdot N_P + f_M \cdot N_M + f_F \cdot N_F}{N_P + N_M + N_F}$, where f_P, f_M, and f_F represent the fitness values for fault classes *pass*, *mute*, and *fail*, respectively. N_P, N_M, and N_F represent the number of times the *pass*, *mute*, and *fail* class occurred out of the number of measurements for a specific parameter set. The sum of N_P, N_M, and N_F is the number of measurements per parameter set.

The number of solutions taken for local search can vary, and, based on the improvements during it, the number of tested parameters can be different in each iteration. Additionally, MA keeps the tested solutions in a list to avoid repeating the measurements. Thus, if the algorithm creates the same solution already tested, we do not execute the laser injection again but return the previous result. That explains the variations in the number of tested parameters we display in different tables throughout the paper.

3.4. EXPERIMENTAL SETUP

3.4.1. TARGETS

We use products from STMicroelectronics. Due to confidentiality reasons, we cannot disclose the details of the targets and the utilized laser bench. Information about the laser bench might inform possible malicious attackers on what kind of bench they could use to attack the products. Utilized integrated circuits (ICs) are constructed with 40nm technology, and all went through a mechanical thinning process before the experiments as part of the preparation for conducting laser injections. Along with differences introduced during the hardware production, thinning of the backside silicon substrate can also affect differences in device sensitivity to laser injections. However, as mentioned, the assumption is that these differences are negligible. The

code running on the target device is a test program where data words are loaded from the non-volatile memory (NVM) into a register. Its implementation is in the C programming language displayed in Pseudocode 3.1. The *trigger_event* function is a monitored event that is used to inject faults at the desired time - on loading a data word into a register (marked with a comment in Pseudocode 3.1). After the fault injection, the register is read, and there is a fault if the register value has changed (fault class *fail*). If there is no response from the device, we categorize this as a fault class *mute*, and if the laser injection does not modify the data, the fault class is *pass*.

Pseudocode 3.1: Pseudocode of the program running on the target devices.

```
...
trigger_event()
load_register() // injection here
read_register()
...
```

We optimize the following five parameters - *x*, *y*, *delay*, *laser pulse width*, and *intensity*. These parameters are often used in literature and practice during a security evaluation. We use a subset of the available values for each of the five parameters, defined according to the known layout. While we cannot share the parameter intervals as they are specific to the product and laser bench, we note that there are 305017650 possible combinations of the parameter values. Thus, search optimization is highly relevant as the exhaustive search with the utilized subset of possible values would take around 529 days if we consider that one laser shot takes ≈ 0.15 seconds. Additionally, since we perform the laser shot several times with the same parameters, this would increase the necessary time to perform the exhaustive search.

3.4.2. EXPERIMENTAL PROCESS

We use three samples of the same target in our experiments, referring to them as IC1, IC2, and IC3. IC1 is an integrated circuit for obtaining the decision trees' training and test data. One training dataset is obtained from the memetic algorithm with random sampling for initialization and another from a random search. The random search is defined to test 50000 different examples. We train different decision tree models on the memetic algorithm (MA) and random search (RS) data. To test the models, we use prediction performance, and to calculate it, we need test data. Test data is again obtained from the IC1 using a random search with 5920 examples and a search we refer to as the fast grid search (FGS). The FGS uses the same bounds for x and y as other algorithms, but the laser settings are fixed. The values of the laser settings are defined based on the most often values for *fail* class from the initial experiments on IC1 using a MA with random initialization. We also defined the number of parameter sets for the random search based on the same experiments where we tested on average 5917.4 different parameter combinations. Later, we apply the memetic algorithm with the DT models, trained on data from IC1, for campaigns on IC2 and IC3. We investigate if we can use the knowledge

from IC1 on IC2 and IC3 to improve the memetic algorithms' performance and test how well the obtained knowledge transfers between different samples. Additionally, we make changes to the bench setup while changing the ICs. The laser focus is less sharp for experiments on IC2 but was again improved for experiments on IC3. Additionally, for the experiments on IC3, we change the bench setup to lose less power from the laser source to the target. These changes have the most effect on the laser intensity parameter.

3.4.3. MEMETIC ALGORITHM HYPERPARAMETERS

For the memetic algorithm, we use a maximum number of iterations of 100 as the stopping criterion since our experiments indicate that this is sufficient to reach convergence. We perform five measurements with the same parameter combination, which means we conduct five laser shots per parameter combination. This way, we can obtain different responses categorized as a *changing* fault class. We use a population of size 100, with an *elite_size* of 10. Thus, the ten best solutions from the population are transferred to another generation without changes. Lastly, the mutation probability is 0.05. These hyperparameters stay the same in our experiments and are decided based on the preliminary investigations and information from previous work [10].

3.4.4. DECISION TREE HYPERPARAMETERS

The *scikit-learn* implementation of CART has many DT hyperparameters, but we only used some to obtain models for comparison as many are not independent. The hyperparameters we chose to test and the specific values used are in Table 3.1. First,

Table 3.1.: Decision tree hyperparameters with values used in our experiments. All combinations of listed values are tested. For the numerical hyperparameters, the interval is inclusive, and the step size is mentioned next to the interval.

Hyperparameter	Values
criterion	{'gini', 'entropy'}
splitter	{'best', 'random'}
min_samples_split	[2, 42], step = 4
ccp_alpha	[0, 0.020], step = 0.005
class_weight	{None, 'balanced'}
balance data	{True, False}

the *criterion* hyperparameter is the function to measure the quality of the split. The first option is 'gini' for Gini impurity, and the other is 'entropy' for Information Gain. Next is the *splitter*, a strategy to choose the split at each node. Options are either the 'best' or the 'random' split. Then, *min_samples_split* that indicates the minimum number of samples required to split an internal node that by default is 2. *ccp_alpha* is a hyperparameter used for minimal cost-complexity pruning, but by default, no pruning is performed, and the value is 0. The intervals for the *min_samples_split* and *ccp_alpha* are defined based on the empirical study [29, 30],

where the authors used a package *rpart* written in R programming language[1] [31] for implementation of the CART algorithm. From their results, most values selected by the optimization techniques for the pruning hyperparameter were in the range 0 to 0.02. For the minimum number of instances necessary for a split to be attempted, most of the values were below 40. We note that the authors also considered the minimum number of samples in a leaf and the maximum depth of any node in the tree called *min_samples_leaf* and *max_depth* in *scikit-learn* implementation, respectively. However, we do not experiment with these two hyperparameters. The *min_samples_leaf* was mostly below 10, and the number of models with a certain *max_depth* value was increasing with a larger *max_depth* value. Thus, we keep the default value of 1 for the minimum number of samples in a leaf and enable the tree to grow until all leaves are pure or contain less than *min_samples_split* (default for *max_depth*). These two hyperparameters can regulate overfitting, but the *min_samples_split* and *ccp_alpha* for pruning do the same. Additionally, the *scikit-learn* has a slightly different pruning method, so we kept the *ccp_alpha* hyperparameter. Since there is an imbalance in the training set because there is usually a majority of *pass* classes compared to all others, we included the *class_weight* hyperparameter. The 'balanced' mode adjusts weights inversely proportional to class frequencies in the input data, and by default, with None, all classes have the weight one. We also included an option to balance the training dataset, which is not part of the *scikit-learn* implementation, but we added it to Table 3.1. The method for balancing the dataset finds the class with the least number of samples and then randomly samples the exact number of samples for all other classes (random undersampling). Other hyperparameters from the *scikit-learn* implementation that are not mentioned in the table are kept at default values. We train 880 decision tree models based on the described hyperparameter combinations. Each model is trained ten times because of the randomness in the algorithm.

3.5. EXPERIMENTAL RESULTS

3.5.1. COMPARISON OF DIFFERENT ICS WITH A FAST GRID AND RANDOM SEARCH

We execute a fast grid search where the bounds for x and y are the same as in other experiments, but we use a large step to test the whole area with a grid search rather fast. Laser settings are fixed based on the memetic algorithm results conducted on IC1. From the memetic algorithm, we analyzed the results and found the most often values for the *delay*, *pulse width*, and *intensity* with *fail* class and used them for FGS. We conducted the same search with identical parameter combinations on all three ICs, and the results are provided in Table 3.2. Similarly, we conduct a random search on all three ICs with the same bounds for all the parameters. We perform only one random search on each IC, and the results are shown in Table 3.3. From both experiments with fast grid and random search, we get most *fails* on IC3. Recall that we have changed the bench setup slightly between experiments with each target

[1] https://cran.r-project.org/web/packages/rpart/index.html

Table 3.2.: Fast grid search with 12350 tested parameters on different ICs.

Fast Grid Search (1 run)	**IC1**	**IC2**	**IC3**
Tested combinations	12350	12350	12350
fail	9 (0.07%)	0 (0%)	33 (0.27%)
changing	87 (0.7%)	14 (0.11%)	154 (1.25%)
mute	43 (0.35%)	0 (0%)	126 (1.02%)
pass	12211 (98.87%)	12336 (99.89%)	12037 (97.47%)

sample, as described before. Considering the changes on the bench, the differences

Table 3.3.: Random search with 5920 tested parameters on different ICs.

Random Search (1 run)	**IC1**	**IC2**	**IC3**
Tested combinations	5920	5920	5920
fail	7 (0.12%)	3 (0.05%)	19 (0.32%)
changing	74 (1.25%)	27 (0.46%)	66 (1.11%)
mute	43 (0.73%)	12 (0.2%)	99 (1.67%)
pass	5796 (97.91%)	5878 (99.29%)	5736 (96.89%)

between the fault class distributions seem reasonable. With a worse focus on IC2 compared to IC1, we have fewer discovered *fails*, while with the improved setup on IC3, we have ≈ 3.7 times more *fails* with fast grid search and ≈ 2.7 times more *fails* with the random search.

3.5.2. TRAINING AND TESTING DATASETS

In our approach with decision trees, we first need a training dataset. The training dataset, in our case, is the data from IC1. We create two datasets. First, we have a dataset acquired with a random search. The difference from the random search shown in Table 3.3 is that we execute the random search to test 50000 unique five-parameter combinations instead of 5920. The other training dataset comes from the experiments using the memetic algorithm with random initialization. The memetic algorithm is executed ten times to obtain ten independent results (runs). In total, with the memetic algorithm, we obtain 61107 unique five-parameter combinations. The described training datasets can be seen in Table 3.4, showing the fault class distributions for both datasets. In both datasets, the number of instances for the *fail* fault class is low, but in total, with MA, we have ≈ 3600 *fail* examples, and with random search, only 58. Thus, with the random search, the *pass* class is dominant as 98.53% of examples belong to the *pass* class while, with MA, 82.08% belong to the *pass* class. This might cause the classifier to ignore the classes with few instances, as predicting only the dominant class can still give high prediction accuracy. However, we include the MA data as another training dataset as there are more examples of *fail* class. Note that our results confirm observations from [10] that the memetic algorithm works better than random search.

We must define a way to evaluate the models trained on the datasets. The idea is to use the prediction performance of the decision tree models to evaluate how well these models would perform and improve the memetic algorithm when used in the

Table 3.4.: Training data on IC1: memetic algorithm (MA) with random initialization and random search (RS). The numbers present an average value over ten runs for MA and one run for RS.

Training data	**MA with Random Initialization** (10 runs)	**Random Search** (1 run)
Tested combinations	6 150.5	50 000
fail	366.5 (6.12%)	58 (0.12%)
changing	548.6 (8.92%)	451 (0.9%)
mute	182.6 (2.89%)	226 (0.45%)
pass	5 052.8 (82.08%)	49 265 (98.53%)

initialization. We need test data different from the training data (unseen examples) for predictions. In our case, we are interested in how well the model trained on IC1 predicts the fault classes on other ICs. However, since we want to use the proposed approach on other ICs without acquiring their campaign data, we also use IC1 for the test dataset. The test datasets are results from the fast grid search (FGS) and random search (RS) conducted on IC1, presented in Tables 3.2 and 3.3, respectively. Therefore, this random search is another campaign with the same algorithm (random sampling) as the training data (Table 3.4) but with fewer tested combinations. We can see that the distribution of fault classes in the experiments with the random search for training and test are similar. Thus, we expect the models trained on random search data to perform well on the random search test data. On the other hand, since the data from FGS is very different from both training datasets, it might be more challenging for these models to predict correctly on the FGS test dataset.

3.5.3. TRAINING THE MODELS AND THEIR PREDICTION PERFORMANCE

After we prepare the training and test datasets, we train 880 different decision tree models by applying different hyperparameters of the decision tree. In Table 3.1, we report values for all the hyperparameters we tested. Each combination of the hyperparameters is tested ten times, and then we calculate the average of the metrics from the predictions on test datasets to assess the performance. Additionally, we train the DT models separately on random search (RS) data and memetic algorithm (MA) data. We do not consider fast grid search for training as it only has data for one combination of laser settings, and in total, there are only 12 350 data points for learning.

As mentioned, the *pass* class is the dominant class based on the number of examples per class. For this reason, we investigate different classification metrics prevalent in machine learning. Usually, *accuracy* is used, but with imbalanced data such as ours, one could use *recall*, *precision*, or *f1_score*. In multi-class classification, the metrics *precision*, *recall*, and *f1_score* are calculated for each class. However, in our work, we focus on the *fail* responses, so we only consider the *precision*, *recall*, and *f1_score* for the *fail* class, ignoring the rest. *Accuracy* is the fraction of the total samples that were correctly classified. Since we want the model to learn when *fails* happens, *accuracy* might not be a good choice because of a low number of samples for all classes except *pass*. *Precision* is the number of samples correctly predicted as

a class for which the metric is calculated from all samples predicted as that class, including those belonging to some other class. With high *precision*, the rules from the DT model might be specific, e.g., defining one combination of parameters that leads to a *fail* class. However, considering our application of the rules, where we switch between different ICs, we can allow some samples to be classified as *fail* even if they are not *fails*. For example, if the same parameter set with a *fail* fault class on one IC does not lead to a *fail* class on another IC, a parameter set with minor differences may lead to a *fail*. The *recall* is the fraction of class samples for which the metric is calculated that were correctly predicted as that class. Thus, *recall* tells how many true *fails* in the test data were predicted as *fails*. We would like to get this metric quite high for our case, but the model may predict everything as a *fail* class to accomplish this, which again would not be useful. Thus, there is *f1_score*, calculated as a harmonic mean of *precision* and *recall*, to find a balance between the two metrics.

To get an idea about the models' predictions, we select the best model for each metric and compare the original distribution of the test data and the predicted distributions of the classes. In Table 3.5, we show the original distribution of the random search data set on IC1 in the first column of the table. Similarly, we show the class distribution based on the DT models' predictions. As mentioned, we take the best models based on each metric. Thus, the rest of the columns show distributions from predictions of the best model based on the *accuracy*, *precision*, *recall*, and *f1_score*, respectively. The metrics are highlighted in the header to specify the metric used to select the best model.

Table 3.5.: Comparing prediction metrics. The first column is the original test data from IC1 (random search). The following columns correspond to predictions from models trained on MA data on the given test data from IC1. The predictions come from the best models based on the highlighted metric. The numbers represent the distributions of fault classes.

	Random Search test data IC1 True distribution	**Accuracy: 0.9796** Precision: 0 Recall: 0 f1_score: 0	Accuracy: 0.9774 **Precision: 0.3333** Recall: 0.1429 f1_score: 0.1999	Accuracy: 0.1873 Precision: 0.0018 **Recall: 0.8** f1_score: 0.0036	Accuracy: 0.9535 Precision: 0.1765 Recall: 0.4286 **f1_score: 0.25**
fail	0.12%	0.05%	0.05%	100%	0.29%
changing	1.25%	0.56%	0.19%	0%	2.4%
mute	0.73%	0.25%	0%	0%	1.72%
pass	97.91%	99.14%	99.76%	0%	95.59%

We can see that with *accuracy*, the model preferred to predict a *pass* fault class, so the percentage of predicted *fails* is lower than in the original dataset. With *precision*, even fewer data points are predicted as any other class except for *pass*. Then, with the *recall* of 0.8, the model predicts all classes as *fail*, which is not the desired behavior. Lastly, the best model based on *f1_score* generalizes the best. While other models in this table predict too little or too many *fail* classes, this model finds a balance. We notice this model predicts more *fails* than there are, but this can be the desired effect for our analysis. Indeed, if the area for the *fail* class is small and specific, the *fails* on another IC might not be at those exact points. However, they

can be very close. Thus, if the model correctly allocates intervals for *fails* class and makes them slightly larger, it gives us more chances to find *fail* responses on another IC with those intervals. Therefore, we choose to use the *f1_score* metric for selecting the model to use its rules for initializing the population of the memetic algorithm.

There are two test datasets - fast grid search (FGS) and random search (RS), so we test on both and average the metric values. We show these average values in Table 3.6 for the best models based on *f1_score* trained on memetic algorithm (MA) data and random search (RS) data, followed by other models used in our experiments. This table also shows the decision tree hyperparameters that define the model. Hyperparameters values in table correspond to splitting *criterion*, *splitter*, *min_samples_split*, *class_weight*, *ccp_alpha*, and flag for balanced data, in that order. The two best models differ in only the *min_samples_split*. The *min_samples_split* is a

Table 3.6.: All models used in the experiments with their prediction metrics and hyperparameters. Metric values are average from results on random search (RS) and fast grid search (FGS) on IC1. The best models are selected based on the highlighted *f1_score* metric. Model parameters are *criterion*, *splitter*, *min_samples_split*, *class_weight*, *ccp_alpha*, and a flag for balance data, in that order.

Average tested on RS and FGS data	*accuracy*	*precision*	*recall*	***f1_score***	**Model parameters**
Best model from MA data	0.9529	0.1227	0.3254	**0.1776**	gini, best, **30**, balanced, 0.0, False
Best model from RS data	0.9630	0.1476	0.3254	**0.1999**	gini, best, **10**, balanced, 0.0, False
Model from MA data with lower *f1_score* 1	0.9831	0.1666	0.0714	**0.0999**	entropy, best, 10, None, 0.01, False
Model from MA data with lower *f1_score* 2	0.9833	0.1666	0.0556	**0.0833**	gini, best, 30, None, 0.0 False
Second-best model trained on RS data	0.9534	0.1238	0.3254	**0.1756**	gini, best, 14, balanced, 0.0, False

parameter that helps prevent overfitting in decision trees. If we allow this parameter to be very low (minimum is 2), the model can overfit to training data. However, if we increase the *min_samples_split*, the chances of overfitting are lower. Since in the MA training set, we have more examples of the *fail* class than with RS data, it is reasonable that the model trained on RS data requires a smaller *min_samples_split*. Smaller *min_samples_split* forces the model to learn when *fail* classes occur by finding those specific cases. On the other hand, if we include pruning or increase the *min_samples_split*, the model can predict only the *pass* class and, in most cases, it will be correct since the majority of examples in training and test data are indeed *pass* classes.

3.5.4. EXPERIMENTS ON DIFFERENT ICS

Once we obtained training and test data on IC1 and trained the decision tree models, we can change to other samples of the same target: IC2 and IC3. First, we ran experiments on IC2, starting with the already presented random search and then the memetic algorithm with random initialization. The target's behavior and the setup used for the injection can influence the results. When using the same target and setup and only changing different samples, the assumption is that the results

should be similar, with minor variations. The variations can come, for example, from unplanned production differences in hardware, unintended chip alignment on the setup, and silicon thickness variations. However, these should be negligible. In our case, we use different samples of the same target, but we also change the focus of the laser spot on the chip. The focus while running experiments on IC2 was worse than for experiments on IC1. The focus directly influences the laser parameter *intensity*. The results from the random search were already given in Table 3.3 but are here presented in Table 3.7 with the memetic algorithm with random initialization to allow easier comparison. We compare the results on IC2 with those on IC1, and, as seen in Table 3.3, with RS, we found less than half of what we found on IC1 for all classes except *pass*. Accordingly, the number of found *pass* classes increased. However, with the MA on IC1 (Table 3.4), only for the *changing* class, we had around two times more examples than with MA on IC2 (Table 3.7). For other classes, while still slightly fewer examples were found on IC2, the results are comparable. Thus, the guided search in MA contributes to the algorithm being more transferable as it adapts to the behavior of the target and changes in the setup.

Table 3.7.: Experiments on IC2: memetic algorithm (MA) with random initialization and random search.

	MA with Random Initialization (10 runs)	**Random Search** (1 run)
Total combinations	6389.6	5920
fail	304.7 (5.03%)	3 (0.05%)
changing	250 (3.88%)	27 (0.46%)
mute	147.9 (2.28%)	12 (0.2%)
pass	5687 (88.82%)	5878 (99.29%)

Finally, we run the newly proposed approach as described in Section 3.3. With the new method, we should have more parameter sets with a *fail* response in the initial population and therefore provide the memetic algorithm with a "head start" which can be exploited to obtain more *fails* in a similar number of tested parameters.

We first run the experiment with the best model trained on MA data, and then we test a model with a lower *f1_score*. Indeed, the question is whether *f1_score* is a good metric to consider when selecting the model for our application - initialization of the memetic algorithm's population on a different sample of the same target. Thus, we ran a model with a lower *f1_score* than the best one, but not the worst because that is a model with an *f1_score* of 0. If the *f1_score* is 0, the model does not perform well even for the predictions on IC1, which is used for training, so we expect it to not perform well in our case. Therefore, we selected the first model trained on MA data with an *f1_score* lower than 0.1. The selected model has an *f1_score* of 0.0999, and we use it to test if the lower *f1_score* leads to fewer *fails* found by the memetic algorithm using the model in the initialization. The numbers from these experiments can be seen in Table 3.8 in the second and third columns. From the headers, we distinguish models based on their *f1_score*. We see that both models trained on MA data improved the performance of the MA algorithm compared to MA with random initialization. MA with random initialization had 5.03% of *fails*, while MA with models has 16.86% and 12.48% for models with *f1_score* 0.1776 and

0.0999, respectively. Notice the performance of the MA is worse with the model with a lower *f1_score*. However, it is still better than a MA with random initialization.

Then, we also test with the best model trained on the RS data. The results are visible in Table 3.8. This model has a higher *f1_score* (0.1999) than the best model trained on MA data (0.1776), but it found the least *fails* from all the other models, even the model with an *f1_score* of 0.0999. We use data from a random search for training and calculating the metrics, which might be why a model with a higher *f1_score* trained on RS data does not find more *fails* when used in MA. It could be that the model created a precise interval for when the *fails* in RS data occurred, or it generated random intervals without learning what parameter values lead to *fail* responses. Thus, we ran the algorithm with a second-best model trained on RS data that has *f1_score* of 0.1756, which is close to the *f1_score* of the best model trained on MA data. The algorithm has found the most *fail* classes between the four tested decision tree models. We can see that the number of tested parameter sets (combinations) also increases, which could cause the difference in the found *fails* compared to the results with the model trained on MA data with *f1_score* of 0.1776. Thus, we conclude that the performance of the models trained on MA and RS data with a comparable *f1_score* is similar in the distribution of fault classes when used in the MA initialization.

Table 3.8.: Experiments on IC2: fault class distribution for the memetic algorithm (MA) with the random initialization, followed by MA with a decision tree (DT) rules used in initialization. The models are distinguished by their *f1_score*, and the hyperparameters for each are in Table 3.6. The header also states which data the model is trained on - memetic algorithm (MA) or random search (RS) data.

Mean (10 runs)	**MA with Random Initialization**	**MA with DT rules** f1: 0.1776 MA data	**MA with DT rules** f1: 0.0999 MA data	**MA with DT rules** f1: 0.1999 RS data	**MA with DT rules** f1: 0.1756 RS data
Tested combinations	6 389.6	5 059.3	5 284.2	5 581.3	5 507.9
fail	304.7 (5.03%)	848.4 (16.86%)	676.5 (12.48%)	633 (11.38%)	1 072.7 (19.39%)
changing	250.0 (3.88%)	234.6 (4.61%)	216.9 (4.11%)	160.2 (2.89%)	198.7 (3.6%)
mute	147.9 (2.28%)	37.3 (0.74%)	81.7 (1.53%)	16 (0.29%)	33 (0.58%)
pass	5 687.0 (88.82%)	3 939 (77.79%)	4 309.1 (81.51%)	4 772.1 (85.44%)	4 203.5 (76.43%)

We also compare the performance of the models based on the number of *fail* responses in the initial population and when the first *fail* is found. This can help understand how the model improves the initial population and impacts the algorithm's overall performance based on this information. The best model trained on the MA with *f1_score* 0.1776 finds the most parameter sets with *fail* response in the initial population - on average 18.2/100 compared to 6.3/100, 2.7/100, or 4.4/100, for the other three models. Both models trained on MA data had a higher number of *fail* classes in the first population than models trained on RS data. That

Table 3.9.: The first *fails* with all the DT models tested on IC2.

Population size = 100	**MA with Random Initialization**	**MA with DT rules** f1: 0.1776 MA data	**MA with DT rules** f1: 0.0999 MA data	**MA with DT rules** f1: 0.1999 RS data	**MA with DT rules** f1: 0.1756 RS data
First *fail* (0-indexed)	53, 3335, 2578, 377, 1381, 1381, 1847, 3071, 1223, 5681 **(avg 2092.7)**	41, 36, 38, 41, 31, 34, 50, 36, 57, 36 **(avg 40.0)**	43, 46, 25, 39, 66, 58, 8, 23, 46, 82 **(avg 43.6)**	33, 203, 47, 84, 52, 59, 55, 53, 56, 46 **(avg 68.8)**	56, 38, 46, 28, 32, 60, 71, 53, 73, 59 **(avg 51.6)**
Number of *fails* in the first population	1, 0, 0, 0, 0, 0, 0, 0, 0, 0 **(avg 0.1)**	14, 34, 23, 18, 15, 17, 16, 15, 14, 16 **(avg 18.2)**	5, 6, 7, 8, 7, 3, 8, 8, 10, 1 **(avg 6.3)**	2, 0, 3, 4, 3, 1, 6, 3, 4, 1 **(avg 2.7)**	6, 4, 5, 4, 6, 6, 3, 5, 3, 2 **(avg 4.4)**

might be because we have more parameter set examples for the *fail* response with MA data, which helps the model learn. The model trained on RS data with *f1_score* of 0.1756 on average had only 4.4/100 *fail* responses in the initial population. Still, in combination with MA, it found the most *fail* responses. Thus, we see that the rest of the algorithm can influence the overall performance of the memetic algorithm as it affects the exploration. Initially, we might focus on only one area found by the model, but crossover and mutation can find *fails* in other regions on another IC. Additionally, we noticed more tested parameters when the model in the initial phase found fewer *fail* responses. MA with DT rules finds the first *fail* response earlier in the algorithm compared to MA with the random initialization, and the initial population has more *fail* examples in the initial population. Therefore, the models improve the initial population. However, considering the overall performance, the algorithm with the most *fails* in the initial population did not find the most *fails*. Still, MA with all tested models improved the performance of the MA with random initialization by finding at least two times more *fail* classes.

We further test the models on another sample of the same target, IC3, where the bench was slightly modified as described in Section 3.4.2. We already discussed the results of the fast grid and random search compared to other ICs in Section 3.5.1. With IC3, we found more *fails*, even when the same points were tested (fast grid search). From the experiments on IC2, the memetic algorithm benefits from using DT models in the initialization. Thus, we do not test the memetic with random initialization on IC3. We test the best model trained on MA data with *f1_score* of 0.1776 and the second-best model trained on RS data with *f1_score* of 0.1756 as it was better than the model with 0.1999 *f1_score*. We also test another model trained on MA data with a lower *f1_score* of 0.0833. The results from the random search and the mentioned three models are in Table 3.10. The algorithm finds at least two times more *fail* responses on IC3 than IC2 with the models, which is in line with experiments with random search and fast grid search because of the changes on the bench. Again the second-best model trained on RS data in combination with MA found more *fails* compared to the best model trained on MA data with *f1_score* of 0.1776. However, in the case of IC3, it is interesting that the model with a worse *f1_score* of 0.0833 that was trained on MA data performed better than the other two models. The difference in the number of tested parameter combinations in these experiments does not explain that improvement. Thus, we again compare

Table 3.10.: Experiments on IC3: fault class distribution for a random search, followed by the memetic algorithm (MA) with decision tree (DT) rules used in initialization. The models are distinguished by their *f1_score*, and the hyperparameters for each are in Table 3.6. The header also states which data the model is trained on - memetic algorithm (MA) or random search (RS) data.

Mean (10 runs)	**Random Search** (1 run)	**MA with DT rules** f1: 0.1776 MA data	**MA with DT rules** f1: 0.0833 MA data	**MA with DT rules** f1: 0.1756 RS data
Tested combinations	5 920	5 262.7	5 495.1	5 554.5
fail	19 (0.32%)	1 662.8 (31.53%)	2 688.8 (48.85%)	2 325.7 (41.83%)
changing	66 (1.11%)	221.9 (4.23%)	210.1 (3.81%)	153.4 (2.76%)
mute	99 (1.67%)	126 (2.37%)	74.1 (1.35%)	101.5 (1.82%)
pass	5 736 (96.89%)	3 252 (61.86%)	2 522.1 (45.98%)	2 973.9 (53.58%)

the number of *fails* found in the initial population and discuss possible reasons. Compared to results on IC2, we see that both with IC2 and IC3, the model with the *f1_score* of 0.1776 found the most *fails* in the initial population, on average 33.5 examples. In both cases, the second model based on the number of *fails* found in the initial population is the model with a worse *f1_score* trained on MA data. For IC2, this was the model with *f1_score* of 0.0999 and 0.0833 for IC3. Then, it is the second-best model trained on RS data with *f1_score* of 0.1756 with 12.9 *fails* in the initial population. Considering the algorithm's overall performance, again, the model with the most *fail* examples in the initial population did not, in the end, find the most *fails*. Nonetheless, experiments on IC2 and IC3 show that the MA with rules improves the performance of random search by obtaining two orders of magnitude more *fails* and ≈ 60% more *fails* than the MA with random initialization.

We again notice that the number of tested parameters increases as the number of *fail* examples in the initial population decreases. Thus, we confirm that there is a need to balance the number of *fail* examples in the initial population and that the rest of the memetic algorithm improves the overall performance. This also raises the question of whether the prediction metric alone is the best metric for selecting a model for this specific use case. While all the models improve overall performance and the number of *fails* in the initial population compared to random search and MA with the random initialization, we might need to consider some other aspects of the models to improve the selection of the model. Since we do not use the models strictly for prediction purposes, possibly, instead of using a prediction metric *f1_score*, we need to analyze the models' size or some information about the rules. Combining different aspects of the model and its rules in one metric could give more reliable expectations of the models' overall performance for our use case. We leave this as future work.

We use machine learning (decision trees) to provide a good initial population that will, in turn, help the memetic algorithm to find many *fail* responses. It stands

Table 3.11.: The first *fails* with all the DT models tested on IC3.

Population size = 100	**MA with DT rules** f1: 0.1776 MA data	**MA with DT rules** f1: 0.0833 MA data	**MA with DT rules** f1: 0.1756 RS data
First *fail* (0-indexed)	0, 1, 1, 0, 0, 1, 0, 1, 1, 1 **(avg 0.6)**	4, 5, 2, 20, 7, 2, 6, 9, 2, 2 **(avg 5.9)**	37, 8, 13, 10, 40, 7, 44, 2, 6, 7 **(avg 17.4)**
Number of *fails* in the first population	40, 36, 30, 37, 50, 32, 35, 32, 30, 13 **(avg 33.5)**	33, 24, 33, 25, 28, 31, 24, 24, 15, 22 **(avg 25.9)**	9, 17, 14, 7, 21, 10, 8, 18, 12, 13 **(avg 12.9)**

to ask if it is possible to compare our approach with the deep learning (multilayer perceptron) approach followed by Wu et al. [8]. We consider those two approaches orthogonal as Wu et al. found a representative set of responses to predict future ones for the same sample of the target device. That way, the authors obtained the complete characterization from a small number of fault injections. This is only an estimate but still gives a great insight into the device's behavior. In our case, we use the model for the prediction on other samples of the same targets for transferability issues between targets. Additionally, we use machine learning to provide an initial population to guide the optimization process. Consequently, we use machine learning in different phases of the target characterization. Still, we believe it could be possible to use our approach to provide an initial population, which will then be used as the training set for a deep learning classifier. Doing this could make the target characterization even more powerful and efficient. We leave this as possible future work.

3.5.5. OBTAINED RULES FOR INITIALIZATION

In Table 3.12, we show the number of rules produced for the *fail* class by each of the utilized models. Additionally, we display the numbers for possible parameter combinations by a specific rule from the model's set of rules - minimum, median, mean, and maximum, and the total number of possible combinations for all the rules created by the model. It is essential to understand that there could be possible overlaps between the rules, which would lower the total number of unique combinations than those reported in the table. Lastly, we show the number of rules in which not all parameters were used to specify areas that the model predicts as the *fail* class. As mentioned, we use a maximum of 25 rules with the highest number of examples classified as a *fail* class by each rule for initialization. Thus, in the table, for the two models with more rules for *fail* than 25, we also show information specifically for the used 25 rules (separated by | symbol).

While the table only shows information about numbers considering the rules for the *fail* class, the number of rules for classifying all classes for each of the models is 1646, 14, 1614, 901, and 782, following the order in Table 3.12. We see that the model with pruning (*f1_score* of 0.0999) is the smallest, with only 14 rules, compared to other models without pruning. Additionally, the models trained on MA data have more rules concerning all classes than those trained on RS data, except for the mentioned model with pruning. The same is visible for the number of rules,

Table 3.12.: Information about the rules, specifically for *fail* class, found by the models used in our experiments.

Total possible combinations = 305 017 650	**MA with DT rules** f1: 0.1776 MA data	**MA with DT rules** f1: 0.0999 MA data	**MA with DT rules** f1: 0.0833 MA data	**MA with DT rules** f1: 0.1999 RS data	**MA with DT rules** f1: 0.1756 RS data
Number of rules for *fail*	359 | 25	1	205 | 25	24	22
Combinations per rule: Minimum	12 | 60	93 960	4 | 42	2 400	2 400
Median	360 | 135	93 960	96 | 110	25 875	43 310
Mean	2 093.82 | 491.24	93 960	690.6 | 169.28	26 772.25	41 237.27
Maximum	38 808 | 7 436	93 960	33 320 | 624	65 250	87 500
Combinations from all rules	751 682 | 12 281	93 960	141 574 | 4 232	642 534	907 220
Rules **not** defining all parameters	105/359 (29%) | 11/25 (44%)	1/1 (100%)	71/205 (34%) | 11/25 (44%)	18/24 (75%)	13/22 (59%)

specifically for the *fail* class. Thus, the models trained on MA without pruning are more complex and larger than those trained on RS data. Additionally, the percentage of rules for the *fail* class is larger for models trained on MA data than RS data, which might be because there are more examples for the *fail* class in the MA training set.

The rule from the model with *f1_score* of 0.0999 (with pruning) considers a path in the decision tree where the conditions for the parameter x were

$$\{x > x1, x > x2, x \leq x3, \ and \ x \leq x4\},$$
$$where \ \{x1 < x2, \ and \ x3 > x4\}.$$

We then obtain the bounds $x \in \langle x2, x4]$ from this path. For y parameter, the path only had requirements $\{y > y1, y \leq y2\}$. Thus, the bounds are set to $y \in \langle y1, y2]$. For $pulse\ width$, in the path, there was only a condition stating $pw > pw1$, so the rule bounds include the user-defined upper bound for the $pulse\ width$ and the mentioned $pw1$ as the lower bound. The $delay$ and $intensity$ are not in the path, and the user-defined bounds are used in the initialization. With this rule, there are 93 960 unique possible combinations for the initial population compared to user-defined bounds that provide 305 017 650 combinations. If the exhaustive search is done for only the rule bounds, it will last around 4 hours, compared to 529 days for the exhaustive search with user-defined bounds considering the laser shot lasts for ≈ 0.15 seconds.

While other models have more rules, each specific rule covers less search space. For example, a model trained on MA data has rules with as little as 4 possible combinations to a maximum of 33 320 unique combinations. Considering only the utilized rules, the number of combinations per rule goes from 42 to 624. We also show the number of possible combinations from all the rules, but this is not a unique number of possibilities because we do not consider the overlaps between them. Still, the rules from models trained on RS data have at least 50 times more combinations than the utilized 25 rules from models trained on MA data. The number of rules used in the initialization is similar for all models except for the one with pruning that had only one rule. Combining this information with the number

of *fails* in the initial population, models trained on MA data seem to have better rules for transferability. Models trained on MA data on both IC2 and IC3 found more *fails* in the initial population. There are more examples for the *fail* class in MA training data than RS data that could lead to the model creating better rules. Additionally, the number of possibilities per rule from models with MA data is lower than with RS data. That makes the rule more specific, defining smaller intervals where the *fail* class can be expected. However, not too small as having only four combinations because we could miss a *fail* class with a slightly different intensity or location since we change the IC. On the other hand, the lower number of *fails* in the initial population from models trained on RS data can come from large intervals in the rules. For example, the minimum number of combinations in one of the rules is 2400. We could miss the *fails* and select parameter sets leading to less interesting classes with that many possible combinations. We also present the number of rules where not all parameters included a reduced interval, but the user-defined intervals were used. Usually, the laser parameters were not reduced, while the location coordinates were specified in all the rules. Except for the model with pruning (*f1_score* of 0.0999), the models trained on MA data had a higher percentage of those rules specifying bounds for all parameters than models trained on RS data. That again suggests that the rules from models trained on MA data define smaller areas.

3.6. CONCLUSIONS AND FUTURE WORK

Previous work [10] showed that the memetic algorithm finds more interesting LFI parameter combinations than a random search that leads to possibly exploitable device responses. This work further improves the memetic algorithm approach by using decision tree models for cases where different samples of the same target are tested, or some minor changes are introduced on the bench. Such decision tree models store the knowledge in a tree structure from which *if-then* rules can be extracted. Thus, we extract rules for the device's interesting *fail* responses. The rules consist of intervals for the LFI parameters, and they can indicate areas where there were most *fail* responses. The approach uses the knowledge obtained from a campaign on one IC in the initialization phase of the memetic algorithm conducted on another IC. We only consider different samples of the same target and minor changes on the bench setup. Considering the number of found *fail* responses, we can see that the performance has significantly improved in the conducted experiments. More precisely, we obtain two orders of magnitude more *fail* responses than random search and $\approx 60\%$ more *fail* responses than the previous state-of-the-art (memetic algorithm with random initialization).

We show this is possible with the simple version of decision tree learning. In future work, we could consider decision tree ensembles, like the random forest, to further improve the models. Still, we note that the possible improved performance of the models would come with higher computational complexity and more difficult interpretability of the rules due to having many decision trees in the random forest. This work is limited to experimenting with different samples of the same target. The trained models correspond to a specific target and utilized bench and cannot

directly be used for other transferability issues, such as changing the bench entirely or using a different target. However, we believe the idea behind the approach can be extended to propose a similar algorithm or different training for those cases. We noticed that all of the models we tested improved the performance of the memetic algorithm. While in this work, we use *f1_score*, a popular prediction metric, to distinguish the best models to apply for the memetic algorithm, it might be beneficial to consider other aspects of the model. Since we do not use the model specifically for predictions, another metric might provide a more reliable way of selecting a model for the initialization phase of the memetic algorithm.

REFERENCES

[1] M. Krček, T. Ordas, D. Fronte, and S. Picek. “The more you know: Improving laser fault injection with prior knowledge”. In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 18–29.

[2] P. C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO ’96. Springer. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 104–113. ISBN: 3540615121.

[3] P. C. Kocher, J. Jaffe, and B. Jun. “Differential Power Analysis”. In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. Ed. by M. Wiener. CRYPTO ’99. Springer. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 388–397. ISBN: 3540663479.

[4] J.-J. Quisquater and D. Samyde. “ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards”. In: *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*. E-SMART ’01. Springer. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 200–210. ISBN: 3540426108. URL: http://dl.acm.org/citation.cfm?id=646803.705980.

[5] S. P. Skorobogatov and R. J. Anderson. “Optical fault induction attacks”. In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2002, pp. 2–12.

[6] J.-M. Dutertre, V. Beroulle, P. Candelier, S. De Castro, L.-B. Faber, M.-L. Flottes, P. Gendrier, D. Hely, R. Leveugle, P. Maistri, *et al.* “Laser fault injection at the CMOS 28 nm technology node: an analysis of the fault model”. In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2018, pp. 1–6.

[7] E. Biham and A. Shamir. “Differential fault analysis of secret key cryptosystems”. In: *Annual international cryptology conference*. Springer. 1997, pp. 513–525.

[8] L. Wu, G. Ribera, N. Beringuier-Boher, and S. Picek. “A fast characterization method for semi-invasive fault injection attacks”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2020, pp. 146–170. ISBN: 978-3-030-40185-6.

[9] I. Polian, M. Gay, T. Paxian, M. Sauer, and B. Becker. “Automatic construction of fault attacks on cryptographic hardware implementations”. In: *Automated Methods in Cryptographic Fault Analysis*. Springer, 2019, pp. 151–170.

[10] M. Krček, D. Fronte, and S. Picek. “On the Importance of Initial Solutions Selection in Fault Injection”. In: *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. 2021, pp. 1–12. DOI: 10.1109/FDTC53659.2021.00011.

[11] R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub. “Glitch it if you can: parameter search strategies for successful fault injection”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 236–252.

[12] S. Picek, L. Batina, D. Jakobović, and R. B. Carpi. “Evolving genetic algorithms for fault injection attacks”. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2014, pp. 1106–1111.

[13] S. Picek, L. Batina, P. Buzing, and D. Jakobovic. “Fault Injection with a new flavor: Memetic Algorithms make a difference”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2015, pp. 159–173.

[14] A. Maldini, N. Samwel, S. Picek, and L. Batina. “Genetic algorithm-based electromagnetic fault injection”. In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2018, pp. 35–42.

[15] V. Werner, L. Maingault, and M.-L. Potet. “Fast Calibration of Fault Injection Equipment with Hyperparameter Optimization Techniques”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2021, pp. 121–138.

[16] M. Moradi, B. J. Oakes, M. Saraoglu, A. Morozov, K. Janschek, and J. Denil. “Exploring fault parameter space using reinforcement learning-based fault injection”. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2020, pp. 102–109.

[17] M. Brijain, R. Patel, M. Kushik, and K. Rana. *A survey on decision tree algorithm for classification*. 2014.

[18] B. Charbuty and A. Abdulazeez. “Classification based on decision tree algorithm for machine learning”. In: *Journal of Applied Science and Technology Trends* 2.01 (2021), pp. 20–28.

[19] R. Kumar and R. Verma. “Classification algorithms for data mining: A survey”. In: *International Journal of Innovations in Engineering and Technology (IJIET)* 1.2 (2012), pp. 7–14.

[20] J. R. Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[21] J. R. Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[22] J. R. Quinlan. “See5/C5. 0”. In: *http://www. rulequest. com/* (1999).

[23] L. Breiman, J. Friedman, R. Olshen, and C. Stone. “Classification and regression trees. Wadsworth Int”. In: *Group* 37.15 (1984), pp. 237–251.

[24] G. V. Kass. “An exploratory technique for investigating large quantities of categorical data”. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29.2 (1980), pp. 119–127.

[25] W.-Y. Loh and Y.-S. Shih. “Split selection methods for classification trees”. In: *Statistica sinica* (1997), pp. 815–840.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[27] P. Moscato. “On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms”. In: *Caltech Concurrent Computation Program* (Oct. 2000).

[28] R. Hooke and T. A. Jeeves. ““ Direct Search” Solution of Numerical and Statistical Problems”. In: *J. ACM* 8 (1961), pp. 212–229.

[29] R. G. Mantovani, T. Horváth, R. Cerri, S. B. Junior, J. Vanschoren, and A. C. P. d. L. F. de Carvalho. “An empirical study on hyperparameter tuning of decision trees”. In: *arXiv preprint arXiv:1812.02207* (2018).

[30] R. G. Mantovani, T. Horváth, R. Cerri, J. Vanschoren, and A. C. de Carvalho. “Hyper-parameter tuning of a decision tree induction algorithm”. In: *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE. 2016, pp. 37–42.

[31] T. Therneau, B. Atkinson, and B. Ripley. “rpart: Recursive partitioning and regression trees”. In: *R package version* 4 (2015), pp. 1–9.

4

DIVERSITY ALGORITHMS FOR LASER FAULT INJECTION

Before third-party evaluation and certification, manufacturers often conduct internal security evaluations on secure hardware devices, including fault injection (FI). Within this process, FI aims to identify parameter combinations that reveal device vulnerabilities. The impracticality of conducting an exhaustive search over FI parameters has prompted the development of advanced and guided algorithms. However, these proposed methods often focus on a specific, critical region, which is beneficial for attack scenarios requiring a single optimal FI parameter combination.

In this work, we introduce two novel metrics that align better with the goal of identifying multiple optima. These metrics consider the number of unique vulnerable locations and clusters (regions). Furthermore, we present two methods promoting diversity in tested parameter combinations - Grid Memetic Algorithm (GridMA) and Evolution Strategy (ES). Our findings reveal that these diversity methods, though identifying fewer vulnerabilities overall than the Memetic Algorithm (MA), still outperform Random Search (RS), identifying at least $\approx 8\times$ *more vulnerabilities. Using our novel metrics, we observe that the number of distinct vulnerable locations is similar across all three evolutionary algorithms, with* $\approx 30\%$ *increase over RS. Importantly, ES and GridMA prove superior in discovering multiple vulnerable regions, with ES identifying* $\approx 55\%$ *more clusters than the worst-performing MA.*

This chapter has been published as M. Krček and T. Ordas. "Diversity Algorithms for Laser Fault Injection". In: *Applied Cryptography and Network Security Workshops*. Ed. by M. Andreoni. Cham: Springer Nature Switzerland, 2024, pp. 121–138. ISBN: 978-3-031-61486-6.

4.1. INTRODUCTION

Small embedded devices frequently employ cryptographic algorithms to provide security. According to Kerckhoff's principle, it is expected that security is intact when the secret key is unknown, even if all the other information about the cryptographic system is public. Consequently, these algorithms are often mathematically secure, rendering brute-force attacks impractical. Regardless, implementation attacks, such as side-channel attacks (SCA) and fault injection (FI) attacks, can potentially lead to a successful security breach of such cryptographic systems. Side-channel attacks are passive, with the attacker measuring the time [2], power consumption [3], or other side-channel data emanating from the target device. Given a correlation between the processed data and measured side-channel information, the attacker can obtain secret information. On the other hand, fault injection attacks are active, where the attacker purposely interacts with the device, inducing errors during the execution of the underlying algorithm. Specifically, the attack can use external sources, such as electromagnetic radiation [4], lasers [5], temperature [6], and voltage glitching [7], to manipulate data in memory, skip instructions, or alter instructions themselves. These implementation attacks are commonly used in security evaluations and are consequently extensively investigated [8, 9]. The objective is to establish an enhanced and automated evaluation process that surpasses the current standard in terms of efficiency. The new algorithms should excel in uncovering more potential vulnerabilities while making more efficient use of available resources or possibly even reducing the required resources.

We focus on laser fault injections (LFI), as introduced by Skorobogatov et al. [5]. The issue with laser injection (and other types of fault injections) comes from the injection parameters determined by equipment. With the laser, we have to define the location of the laser shot on the targeted hardware device (x and y coordinates), the distance from the microscope lens, which is commonly used with lasers, and, lastly, we also have the laser settings, such as laser *intensity*, *delay*, and *pulse width*. Additionally, lasers can have pulses that demand several more parameters to define. Another critical component of successful injections is the trigger on when to perform the injection. In security evaluation, the worst scenario is often considered, where it is assumed that we have open access to the targeted device, and the trigger can be placed at any point in the execution. Obviously, there are *many parameters* we should consider. Additionally, the *possible values and combinations* of those parameters increase to the extent that exhaustive search is not feasible for security evaluation or attack.

In the attack, the adversary aims to find the parameters that lead to exploitable fault injection effects. These desired effects also depend on the method for the attack, where some of the popular attacks are differential fault analysis (DFA) [10], statistical fault attack (SFA) [11], and statistical ineffective fault attacks (SIFA) [12]. Each attack can require different characteristics of the FI effects. Still, some commonly desired and possibly exploitable faults include causing the device to skip instructions or change values in memory [8]. This work does not address identifying exploitable faults but focuses on a scenario within the internal security evaluation. During the security evaluation, a target characterization is performed, striving to

uncover all vulnerabilities that can later be categorized based on their level of critical exploitability. While executing an exhaustive search would ensure that all possible vulnerabilities are observed, this process is not feasible as there are many products to be evaluated, and the search is impractical even for a single target. The aim is then adjusted to find as many vulnerabilities as possible within reasonable time and resources. Therefore, the FI parameter search is a process that should observe many vulnerabilities and provide high confidence that little to no vulnerabilities are overlooked. Instead of an exhaustive search, the location on the target is often searched in a grid-like manner using the same laser settings. Defined laser settings could come from previous experience, which might be misleading if the target or the bench is entirely new [13]. If more options for laser settings are tested over the whole target area, this process becomes time-consuming, so alternatively, a random search is applied. However, both methods could omit parameter sets that lead to faults. In grid search, while the location is relatively thoroughly inspected, fixing the laser settings can contribute to overlooking many vulnerabilities. By including different laser settings, the search converges to an exhaustive search where the security analysts aim to reduce the search space based on previous knowledge, but the execution time for these algorithms could still be measured in weeks. On the other hand, random search is unreliable as different runs can lead to very different observations, which causes misrepresentation of the target's security level. Therefore, there is an incentive to improve the process of exploring the FI parameter search space more efficiently in an automated way.

Evolutionary algorithms (EAs) were explored for laser fault injection [14], voltage glitching [15–17], and electromagnetic fault injection [18, 19] since laser fault injections are not the only type of injections suffering from the previously described issues. From the machine learning domain, hyperparameter optimization techniques [20], reinforcement learning [21] and Generative Adversarial Networks (GANs) [22] were also investigated. Additionally, the prediction ability of machine learning methods was explored for portability issues in the FI parameter search [23] and estimating the full target characterization [13].

The issue with aforementioned search algorithms, like evolutionary algorithms and tuning techniques, lies in their tendency to converge on a single vulnerable area as they are designed to obtain a single optimal solution. Previous works show a significant increase in the observed faults clustered in one sensitive region [17, 18]. While that can be highly effective for attackers, we focus on security evaluation, where identifying multiple vulnerable regions is deemed a more favorable outcome. To better assess algorithm success in parameter search concerning the security evaluation goals, we propose to use the number of unique locations (x-y) and clusters with faults as additional metrics. We investigate the performance of several algorithms: random search, memetic algorithm, and two novel algorithms not explored before in the FI context - Grid Memetic Algorithm (GridMA) and Evolution Strategy (ES). The new algorithms are introduced as they promote the diversity of the parameter combinations. This diversity aims to achieve a more diverse search, uncovering distant vulnerabilities and identifying multiple optima instead of a single sensitive region. Experiments are performed with laser fault injections but should be

suitable for other fault injection types.

Our main contributions are:

- We propose two methods that promote diversity among the tested FI parameter combinations. Promoted diversity ensures fewer vulnerabilities are overlooked, and multiple optima are uncovered during the search.
- We investigate other aspects of the algorithm performance for the FI parameter search, such as unique locations and clusters.
- The results show that the evolutionary algorithms find ≈ 30% more unique vulnerable locations than random search.
- The GridMA and ES algorithms found around 41% and 55% more vulnerable clusters than the worst-performing MA in this aspect, respectively. Thus, the diversity algorithms help determine more vulnerable regions.

4.2. Preliminaries

4.2.1. Random Search (RS)

Random Search (RS) is a widely used optimization method when exhaustive search is impractical. In the context of FI parameter search, it explores a predefined search space by randomly selecting parameter values and assessing their performance, with each value having an equal probability of selection. We ensure that only unique parameter combinations are considered, eliminating duplicates.

4.2.2. Memetic Algorithm (MA)

The memetic algorithm (MA) enhances the genetic algorithm (GA) by incorporating local search [24]. We apply the local search at the end of each GA iteration, constituting the first generation of memetic algorithms. This specific method has been successfully utilized in previous research [14, 18]. The flow of the MA is depicted in Figure 4.1. MA is a population-based optimization technique, operating on a set of individuals, each representing a potential solution to a specific optimization problem. The algorithm begins by generating an initial population using an *initialization method*, where a random sampling approach is often used. The algorithm then uses a problem-specific *fitness function* to evaluate each solution's performance. After evaluation, the genetic operators, including selection, crossover, and mutation, that drive the learning process are performed. The *selector operator* identifies solutions from the current population for reproduction. Usually, the best-performing solutions are favored as they are more likely to yield improved solutions. The selected solutions, called parent solutions, undergo the *crossover operator*, which combines their traits to create one or more offspring solutions. The new solutions (offspring) undergo the *mutation operator*, which introduces random variations into the new solutions. The mutation probability is commonly kept low, preventing the algorithm from acting like random sampling. The process generates a new population that continues into another algorithm iteration. To ensure the best-performing solutions are not lost, *elitism* is employed. Elitism explicitly preserves one or more of the best solutions from the current population for the next

generation. Lastly, some solutions are selected for further improvement using the *local search*. In this work, we use the memetic algorithm introduced in [14], where the algorithm incorporates the Hooke-Jeeves as local search [25]. The algorithm runs until a predefined *termination condition* is satisfied. These termination conditions commonly consider the number of iterations or evaluations for ending the execution.

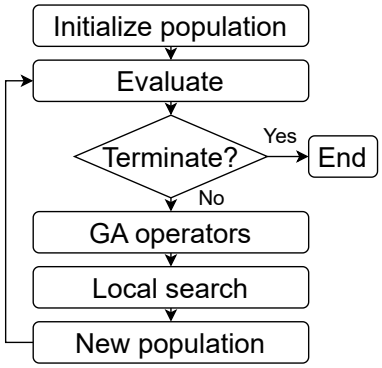


Figure 4.1.: Flow of the Memetic Algorithm.

4.2.3. CLUSTERING METHOD

In the analysis, we use a clustering method called Mean Shift [26], an unsupervised clustering algorithm designed to identify clusters in a continuous distribution of data points. It is a centroid-based algorithm that updates candidates for centroids by computing the mean of points within a specific region (referred to as the *bandwidth*). Subsequently, these candidates are filtered in a post-processing stage to eliminate nearly identical centroids, forming the final set of centroids. We opted for the Mean Shift clustering algorithm because, unlike some other popular methods such as K-Means [27], it does not require users to predefine the number of clusters. While several different algorithms share this characteristic, we chose Mean Shift due to its simplicity as a centroid-based algorithm with only one hyperparameter. We believe it will provide satisfactory results for our analysis. However, we do not claim Mean Shift as the superior clustering algorithm. Note that the considered algorithms were those provided by a Python package called *scikit-learn* [28] to enable quick implementation and usage, as clustering is not the main topic of this work.

4.3. RELATED WORK

Carpi et al. [15] investigated various search strategies for voltage glitch parameters, specifically the glitch shape and timing, for a successful FI attack. They explored glitch voltage and length with Monte Carlo (random), FastBoxing, Adaptive Zoom&Bound, and a Genetic Algorithm (GA). The genetic algorithm, without fine-tuning, required more measurements than the superior Adaptive Zoom&Bound algorithm. Picek et al. [16] extended the GA for the same voltage glitch parameters

by employing a specialized crossover operator and selection mechanism, finding more faults than random search. Later, Picek et al. [17] introduced a memetic algorithm considering three voltage glitching parameters, namely glitch length, voltage, and offset. The authors mentioned the impracticality of specific algorithms used in previous work [15] due to increased dimensionality, excluding them from comparison. Their objective was to efficiently identify favorable parameters within minimal time, seeking both successful parameter combinations and regions with consistent behavioral outcomes. Maldini et al. [18] increased the parameter search space by optimizing five parameters for Electromagnetic Fault Injection (EMFI) using a memetic algorithm. Krček et al. [14] demonstrated the effectiveness of a similar memetic algorithm for laser fault injection (LFI). These studies showcased the efficacy of the memetic algorithm across various FI types. Werner et al. [20] employed two hyperparameter optimization techniques from machine learning to enhance the parameter search for voltage glitching. They proposed a two-stage optimization strategy to reduce the dimensionality of the parameter space, similar to Carpi et al. [15]. Rais-Ali et al. [19] compared three different methods for EMFI, with the GA consistently outperforming the others in identifying areas of interest. The authors emphasize that, from an attacker's perspective, the goal is to identify a single exploitable fault using a specific FI parameter set. However, in the evaluation context, the objective is to ensure device security without excessive time investment, requiring a high-dimensional search to avoid overlooking potential parameter combinations. In [17–19], the authors evaluated performance based on the number of observed vulnerabilities, considering the notion of distinct regions and faults. We introduce diversity methods within the evolutionary approach to improve the algorithms' ability to discover more distinct regions with vulnerabilities. Related work on voltage glitching parameter search typically involved optimization of two or three parameters, while EMFI and LFI examined five. We optimize the same five parameters for LFI, performing a high-dimensional parameter search.

Wu et al. [13] focused on laser settings' impact on a specific building block. The authors noted that the complete characterization took over a week to execute. This underscores the need for faster and more efficient algorithms in the field. However, their work differs from ours, as our research investigates fault injection considering five distinct parameters, intending to uncover vulnerabilities across various building blocks within an integrated circuit (IC). Krček et al. [23] explored the transferability of results to different samples of the same target using decision tree models, falling outside the scope of this work for comparison as this work focuses on improving the parameter search without prior knowledge. Lastly, Moradi et al. [21] and Sedaghatbaf et al. [22] applied reinforcement learning and Generative Adversarial Networks (GANs), respectively, for efficiently exploring the fault injection space in simulations for adaptive cruise control systems in autonomous vehicles domain. These techniques could potentially extend to fault injection on hardware devices, aligning with our work.

Comparing all methods from the mentioned related work is time-consuming and complex. Hence, we leave this task to future work, recognizing the importance of unifying and evaluating these advanced methods to determine the state-of-

the-art approach for parameter search in the scope of security evaluation, target characterization, and FI attacks. In this study, we compare new algorithms with random search and memetic algorithm, previously employed for high-dimensional parameter search on EMFI and LFI.

4.4. DIVERSITY ALGORITHMS

This section explains the newly proposed diversity algorithms that should help identify multiple vulnerable regions within the FI parameter search.

4.4.1. GRID MEMETIC ALGORITHM (GRIDMA)

We propose a novel approach, named the Grid Memetic Algorithm algorithm, that involves partitioning the target area for exploration into a grid and running the previously explained memetic algorithm within each grid region. The primary objective of the GridMA approach is to ensure attention (time and evaluations) of the algorithm to all target regions, mitigating the risk of overlooking vulnerabilities in specific (x-y) locations. For instance, if the target area is divided into a 3×3 grid, resulting in nine distinct regions, GridMA executes the MA independently in each region during a single run. As the search space size within each grid region decreased, we reduced the MA hyperparameters, specifically the population and elite sizes. GridMA represents a minor adaptation to the established MA. Nevertheless, it is a valuable initial step in evaluating diversity algorithms, precisely when we aim to obtain multiple vulnerable target regions.

4.4.2. EVOLUTION STRATEGY (ES)

Evolution Strategy, like genetic algorithms, belongs to the class of evolutionary algorithms inspired by the principles of natural evolution [29]. The initial version of ES consisted of a single-parent solution from which one offspring was produced through a mutation-like procedure. The superior solution between the parent and offspring is preserved, and it resumes the same iterative process until it fulfills specific termination criteria. These termination conditions align with those described for MA in Section 4.2.2, consisting of attributes such as the number of iterations, evaluations, or acquiring a specified fitness level. Over time, ES has evolved, and in its more general form, it adopts the notation of $(\mu^{+}_{,}\lambda)$-ES. For instance, the original version can be denoted as $(1+1)$-ES, suggesting the presence of only one parent and one offspring in the process. Thus, μ represents the number of parents, and λ indicates the number of offspring. Additionally, a μ/ρ notation can be used for parents, where μ denotes existing parents, and ρ indicates the number of parents selected for producing offspring. Typically, ρ is less than or equal to μ, meaning that a subset of the best individuals is chosen for reproduction. In the notation, we use symbols + or , to indicate whether the solutions selected for the following generation are derived from both parents and offspring $(\mu+\lambda)$ or the parents (μ) are discarded, and only the offspring (λ), regardless of their fitness, continue to the next generation.

In the context of the described notation, we employ the $(\mu+\lambda)$-ES. The original, $(1+1)$-ES, using one single parent and an offspring, still converges to one optimal solution. Thus, to achieve diversity and reduce the risk of focusing on a single optimal solution, we set $\mu > 1$, effectively creating a population of size μ as employed in MA. The initial set of solutions is distributed across different locations, and in each iteration, new offspring are generated from each of these parent solutions as we set $\lambda > 1$. Consequently, ES maintains a population of diverse solutions that evolve through iterations. This iterative process may lead to finding distinct solution clusters representing local optima. Thus, by employing ES, we expect to decrease the chances of overlooking vulnerabilities within the target area and observe more distinct solutions with optimal fitness.

4.5. Experimental Setup

4.5.1. Target

In collaboration with STMicroelectronics, we utilize their products for our experiments. Due to confidentiality reasons, we cannot disclose the details of the targets and the utilized laser bench. The target for our experiments is an IC constructed with 40nm technology. Since we use lasers for fault injection, mechanical thinning, a standard procedure, was part of the preparation for the experiments. During security evaluation, test programs can be deployed on the targeted products. The program running on our target device is a test program where data words are loaded into a register from the non-volatile memory (NVM). This test program can commonly be a part of the functionalities occurring within different algorithms on these devices. The target has no security countermeasures as the purpose here is not an attack breaking the device's security and countermeasures. Additionally, this provides the worst-case scenario. The implementation is done in the C programming language, and the pseudocode is displayed in Pseudocode 4.1. The pseudocode shows calls to three functions, where the first function is the `trigger_event`. The trigger event is a monitored event used to trigger the laser shot to inject faults at the desired time. In this case, the injection is aimed during the execution of the following function. That function loads the data from the NVM into a register. Lastly, we read the register and compare the value with the expected data. There is a fault if the register value has changed (fault class *fail*). On the other hand, if the injection was unsuccessful and the data is unmodified, equal to the expected value, then we give this response a fault class *pass*. Lastly, if there is no response from the device due to a time-out error or reset, we categorize this as a fault class *mute*. Note that the IC was reset to the initial state after each injection to provide a clean condition for each injection.

Pseudocode 4.1: Pseudocode of the program running on the target device.

```
...
trigger_event()
load_register() // injection here
read_register()
```

...

The FI parameter search is done on the following five parameters - *x*, *y*, *delay*, *laser pulse width*, and *intensity*. These parameters are commonly used in literature and practice during a security evaluation [18, 23]. We use a subset of the available values for each of the five parameters, defined according to the known layout and target cartography. Step sizes are defined based on the minimum possible step according to the utilized bench equipment, and the target area size includes different building blocks of the IC. The intervals are kept the same for all experiments. While we cannot share the parameter intervals as they are specific to the product and laser bench, we note that there are 370772710 possible combinations of the parameter values. The exhaustive search with the defined subset of possible values will take around 643 days if we consider that one laser shot takes ≈ 0.15 seconds.

While we focus on a single target in this study, the parameter search algorithms we introduce are versatile and applicable across various targets, bench configurations, and FI types. On average, the relative performance of these algorithms is expected to remain similar across mentioned scenarios. The obtained target responses guide these algorithms. Therefore, regardless of the selected target and setup, they strive to identify optimal solutions within the current setup and measured responses. The extent of improvements is limited by the finite number of detectable vulnerabilities associated with a specific target and bench setup.

4.5.2. ALGORITHM DETAILS

In all our experiments, we specified a maximum limit of 6000 evaluations of unique FI parameter combinations as a termination condition. Since we perform injection five times with the same parameter combination, we allow 30000 laser shots. The number of evaluations is a practical upper bound on the algorithm's execution time. Previous work [23] indicates that a similar evaluation count leads to successful convergence, thus further justifying its selection.

In our approach, as we perform five measurements with the same parameter combination, we can acquire distinct fault class responses given the same parameters. Thus, there is a slight variation in our fault classification compared to related work. In our results, we present classes so that if there is even a single *fail* response within the measurements, we consider it a critical outcome and label it under *fail comb.* notation, signifying a *fail combination*. This approach aggregates all *fail* occurrences, disregarding the specific combinations that led to them. A more fine-grained categorization might be beneficial if we consider a specific attack, as parameter combinations with consistent outcomes might be more suitable for attacks. However, since we are in a security evaluation scenario, any occurrence of a *fail* response is considered critical. Other classes we include are those with *mute* response, a combination of *mute* and *pass* referred to as *mute_pass*, and lastly, there is a *pass* class where only *pass* class occurred in five measurements. The fitness function for all algorithms is calculated as

$$fitness = \frac{f_P \cdot N_P + f_M \cdot N_M + f_F \cdot N_F}{N_P + N_M + N_F},$$

where f_P, f_M, and f_F correspond to the fitness values assigned to the fault classes *pass*, *mute*, and *fail*, respectively. Similarly, N_P, N_M, and N_F represent the frequency of these classes occurrences within the number of measurements for a specific parameter combination. The sum of N_P, N_M, and N_F constitutes the total number of measurements per parameter combination. This fitness function definition follows the previous works [14, 23]. In our case, the fitness values for f_P, f_M, and f_F are 1, 2, 10, respectively. These values differ slightly from prior works, as we choose to create a more pronounced distinction in fitness value between each fault class. This design decision emphasizes the significance of any *fail* combination by assigning it a significantly higher fitness value. Before evaluating the entire population, we conduct a sorting operation using a greedy approach that considers the Manhattan distance between different locations of the FI parameter combinations within the population as described in [14].

MA HYPERPARAMETERS.

We employ a population of size 100, with an *elite_size* of 10. The initialization method employs a random sampling strategy while preventing duplicates. For selection, we implement the roulette wheel method. We use uniform crossover and a uniform mutation with a mutation probability of 0.05. Lastly, the Hooke-Jeeves algorithm is applied for local search. Note that the hyperparameters in our experiments remain the same and have been taken based on information from previous work [14].

GRIDMA HYPERPARAMETERS.

We dedicated additional experiments to exploring the hyperparameters of the GridMA algorithm, as it is a newly proposed method. We performed a minor hyperparameter search focused on the grid size, the population size, and the elite size of the MA. This section outlines the hyperparameters for the final version of the GridMA, whose results are shown in Section 4.6. The MA instances running in each grid have the same hyperparameters as described in Section 4.5.2, except for the mentioned hyperparameters that we were able to decrease due to the reduced scope of exploration within each grid region. Accordingly, the population size is set at 30 individuals, with an $elite\_size$ of 5. We divide the area in 4×4 grid, effectively conducting a total of 16 MA algorithms during a single run of GridMA. Since we maintain the total number of evaluations at 6000 parameter combinations, each grid region is limited to evaluating only 375 FI parameter combinations.

ES HYPERPARAMETERS.

Evolution Strategy is a new approach, so we explored several hyperparameters. Specifically, we assessed the algorithm's performance concerning the number of parents and offspring and the mutation probability. While we quickly obtained reported results, further fine-tuning may improve performance. The reported results are derived from ES employing 40 parent solutions and 5 offspring with the initial generation of parents established through random sampling. This algorithm uses the mutation operator as the sole source of introducing solution modifications, so

a higher mutation probability will be necessary. We observed that the mutation probability of 0.4, much higher than used with MA, produces the best results without converging to a purely random search approach. The mutation probability applies to each specific dimension within the parameter combinations. For example, with 40% probability, mutation will occur from uniformly distributed values of the given parameter. Uniform mutation encourages more substantial modification, allowing more 'jumps', particularly beneficial in the context of FI parameter search, as there are more non-vulnerable areas than vulnerable ones. In Section 4.6.4, we explore several modifications to the ES algorithm, including the Gaussian mutation approach, which is more commonly utilized to ensure a higher probability of local changes.

4.6. Experimental Results

This section presents results from applying the described algorithms to the same IC and laser bench. We aim to identify more locations with a *fail* outcome and uncover multiple vulnerable regions. To achieve this, we avoid restricting our search to a $2D$ location exploration, as it could overlook numerous parameter combinations due to the need for fixed laser settings. To effectively assess and identify algorithms that perform well for our objective, we compare them not only based on the observed $5D$ parameter combinations with a *fail* outcome but also on the number of unique locations ($2D$) and clusters. We executed each algorithm five times and reported the average results to ensure statistically relevant observations.

4.6.1. Number of Unique Parameter Combinations (5D)

We initially assess the number of unique parameter combinations with *fail* outcomes, where we use percentages from total tested combinations as in previous work for a more straightforward comparison. The results, shown in Table 4.1, reveal a comparable increase in *fail* responses between random search (RS) and memetic algorithm (MA) to the reported results in [14, 18, 23]. The MA identified $\approx 55.6\times$ more FI parameter combinations leading to *fail* response than RS. In contrast, the two new methods, which provide greater diversity in the population of the FI parameters, obtained a lower percentage of *fail* responses compared to the MA. Compared to RS, we still find $\approx 12.4\times$ more *fails* with GridMA, and $\approx 7.8\times$ more with the ES. This decrease in the percentage arises from GridMA's exploration of areas where vulnerabilities might not exist. The ES, which relies solely on mutation, introduces more randomness than the MA, leading to a decrease in the number of identified vulnerabilities. Moreover, each parent evolved independently, resulting in more dispersed parameters and less exploitation of sensitive locations. These features should enhance our current objective but are shown to impact this metric negatively. Considering only the number of unique $5D$ parameter combinations tested, MA outperforms the other tested algorithms. However, the evolutionary approaches with diversity still offer advantages and should be preferred over random search.

Table 4.1.: The average percentage of observed fault classes from all tested parameter combinations (6000) using four different algorithms on the same IC. The average is calculated over five runs.

	RS	MA	GridMA	ES
fail comb.	0.61%	33.84%	7.54%	4.77%
mute	1.23%	3.23%	4.44%	4.53%
mute_pass	0.79%	1.21%	2.24%	2.83%
pass	97.36%	61.72%	85.79%	87.88%

4.6.2. NUMBER OF UNIQUE LOCATIONS (2D)

In this work, we explore other metrics that could be used to evaluate the performance of different parameter search algorithms employed for fault injection. As we explain, MA converges commonly to one region sensitive to the utilized FI type and exploits it, leading to many observed FI parameter combinations with *fail* outcome. These parameter combinations come from a cluster of close *x*-*y* locations that can be detected visually (see Figure 1b in [18] and Figure 2 in [17]). In security evaluation, there should be a certain confidence that not many vulnerabilities are missed during the assessment of the IC. Also, we aim to find multiple regions with vulnerabilities, so we explore algorithms that promote diversity as it should help produce vulnerabilities distant in the utilized 5D space. More importantly, we want distant solutions when looking at the observed vulnerabilities' location (*x*-*y*). Thus, in Table 4.2, we report the number of unique parameter combinations with different fault classes and the number of unique locations per fault class from those parameter combinations. The table has two columns per algorithm, with the first showing the numbers from all the tested parameter combinations and the second showing the number of unique *x*-*y* locations. We also calculate what we refer to as location coverage, dividing the number of unique locations (2D) by the number of total tested unique 5D parameter combinations. This number shows the ratio of covered area within the tested parameter combinations. The numbers are rather small if we look at the absolute possible locations instead of relative to the tested parameters. To put it into perspective, from all possible combinations ($\approx$ 370 million), we only test 0.00162% with 6000 combinations. Unique tested locations from all possible locations (2D) per algorithm are 4.27%, 1.67%, 2.02%, and 3.07% for RS, MA, GridMA, and ES, respectively. We see an increase in the absolute location coverage between different evolutionary approaches, but RS has the best result. In the table, we report the relative location coverage as it provides an easier comparison. The relation between the algorithms is the same when we compare the absolute and relative location coverage. The results show that RS has the best coverage with 97.95% as the algorithm has no guidance. The worst location coverage is with MA (38.24%), supporting the motivation for this work. GridMA and ES improve coverage with 46.36% for GridMA and 70.29% for ES. Location coverage can serve as a measure of the algorithm's confidence in not overlooking vulnerable areas.

Comparing the unique locations with *fail* response between MA, GridMA, and ES, we see that the algorithms find a similar number of unique locations with *fail* - around 48, which is around 30% more than with RS (36.4). While similar in the number of unique locations with *fail* outcome, GridMA found the most unique locations on average with higher location coverage than MA. This improvement over MA is not as significant as the difference in performance between the evolutionary approaches and RS. Still, it shows the potential of diversity algorithms for security evaluation as they provide better coverage and thus confidence in identified vulnerabilities while delivering a similar improvement over RS in the number of unique, vulnerable locations.

Table 4.2.: The average number of unique parameter combinations and *x*-*y* locations per fault class, and in total for all four algorithms. The average is calculated over five runs.

	RS		MA		GridMA		ES	
Nb. comb. | Nb. loc.	6000	5877.2	6000	2294.6	6000	2781.4	6000	4217.4
Location coverage	0.9795		0.3824		0.4636		0.7029	
fail comb.	37	36.4	2030.2	48.4	452	**49.2**	285.6	47
mute	74	73	194	60.4	266.2	70.4	271.6	93.2
mute_pass	47.4	47	72.8	45.8	134.6	61.4	169.8	67.6
pass	5841.6	5723.4	3703	2218.8	5147.2	2704.8	5273	4088.6

4.6.3. NUMBER OF LOCATION CLUSTERS

Finding distant, vulnerable locations is considered more valuable as the smaller regions could further be explored with an exhaustive search on a significantly reduced search space [19]. Thus, we compare the algorithms based on the number of observed location clusters with a specific fault class. We calculate the number of clusters using the Mean Shift clustering algorithm. The bandwidth hyperparameter for the Mean Shift algorithm defines the window/region from which the mean is calculated. We executed the clustering with different bandwidth values, precisely $0.1, 0.2, 0.3, 0.4$. With a bandwidth of 0.3, the region was large enough to categorize all the *x*-*y* points as one cluster for all fault classes. Table 4.3 shows the number of clusters averaged over five runs with bandwidth set to 0.1. Using the same bandwidth ensures the number of clusters is comparable as the same window size is considered. Note that if the number of clusters is more significant, the vulnerabilities are observed in more distant and distinct locations on the target, which is the desired objective. The results show that GridMA finds the most clusters with *fail* outcome, implying that the observed locations are more distant than other algorithms. GridMA and ES obtain a similar number of clusters on average, closely followed by RS. MA, on the other hand, clearly shows a smaller number of clusters observed. These results further emphasize the benefits of diversity methods for security evaluation and finding multiple regions sensitive to the utilized FI type. We checked the clustering model's predictions visually for several cases to ensure that classified clusters are meaningful. While some more distinct locations were still

Table 4.3.: The number of clusters based on Mean Shift clustering algorithm over the unique *x*-*y* locations per fault class. The bandwidth size is 0.1. The number of clusters is averaged over five runs.

	RS	MA	GridMA	ES
fail comb.	7.8	5.8	**8.2**	8
mute	11.4	8.6	9.6	10.6
mute_pass	9.6	8.4	10.8	10.2
pass	41.8	37	34.2	33.8

clustered together using this bandwidth, the predicted clusters seemed reasonable. Moreover, we use the same bandwidth to ensure comparable results, as relative correlation is essential.

4.6.4. FURTHER EXPLORING THE EVOLUTION STRATEGY ALGORITHM

The results show that evolutionary algorithms perform better than random search when considering the number of unique FI parameter combinations and unique locations with *fail* response. Considering the number of clusters, RS was better than MA, but the diversity algorithms were better overall. Thus, while the performance was not significantly improved using the diversity algorithms considering these metrics, the observed minor improvements show promising results. Therefore, we deem it necessary to explore these algorithms more within the scope of future work. In this section, we explore several ES versions to obtain enhanced performance.

We test the ES algorithm with a more common Gaussian mutation, which uses the Gaussian distribution to set the probabilities of each of the parameter values getting selected. The mutation probability will then be used as a standard deviation σ parameter, while the parameter's current value will be the mean μ. This mutation makes local changes more likely, while the more distant significant changes have a low probability of occurring, but not zero. We refer to this version of ES as *ES gauss*. Another modification we test is the initialization method, where we use a grid approach to set the parents of ES in distinct regions over the target area, considering only the location parameters. This way, the location parameters within the initial population are well-distributed, and the evolution should have a better chance of observing more distant and distinct regions with *fail* response. This version of ES is named *ES grid*. We then combine both modifications into a third version of ES referred to as *ES grid gauss*. Lastly, we execute a GridES, similar to GridMA, where we run ES within each grid cell over the target area. The grid is split in the same manner as for GridMA. We ran the GridES with the *ES grid* version as it was the best considering the number of clusters, and it performed similarly to the best ES versions considering the other two metrics. Note that the reported results from the new ES versions are mean values from three runs, while the previous experiments ran five times. From the results in Table 4.4, considering **the number of unique FI parameter combinations** with *fail* response, the initial ES

version performs the best on average, followed by the *ES grid* version. The versions with Gaussian mutation perform more closely to the results observed with random search. However, applying grid initialization for the version with Gaussian mutation did help increase the number of observed vulnerabilities. Still, using uniform mutation proved better within these experiments. Similar to *ES gauss* and *ES grid gauss*, GridES obtained a similar number of faults as RS. Considering **the number of**

Table 4.4.: The average percentage of observed fault classes from all tested parameter combinations (6000) using five different versions of ES algorithm on the same IC. The average is calculated over three runs.

	ES gauss	ES grid	ES grid gauss	GridES grid
fail comb.	0.61%	4.19%	0.82%	0.82%
mute	1.83%	6.31%	2.02%	1.68%
mute_pass	1.17%	2.85%	1.08%	0.94%
pass	96.39%	86.66%	96.08%	96.57%

unique locations with *fail* response, versions *ES grid* and *ES grid gauss* were better than the initial ES version, as seen in Table 4.5. On average, the number of unique locations is now closer to the best GridMA algorithm, and it remains in the scope of previously observed improvements over RS using any of the evolutionary approaches. Considering this metric, *ES gauss* and GridES perform similarly to RS. Gaussian mutation increases the location coverage to the same level as RS, as evident from the results with the *ES gauss* and *ES grid gauss*. Finally, we consider **the number of**

Table 4.5.: The number of unique parameter combinations and *x*-*y* locations per fault class, and in total for all four algorithms. The average is calculated over three runs.

	ES gauss		ES grid		ES grid gauss		GridES grid	
Nb. comb. | Nb. loc.	6000	5869	6000	4235.5	6000	5857.3	6000	4322
Location coverage	0.9782		0.7059		0.9762		0.7203	
fail comb.	36.3	36	251	48	49	**48.7**	48.7	35.7
mute	110	107.3	378.5	124.5	121.3	121	101	78
mute_pass	70	69.7	171	86.5	64.7	64.7	56.3	47
pass	5783.7	5666	5199.5	4075	5765	5632	5794	4216.3

clusters with *fail* response in Table 4.6, and the *ES grid* version found 9 clusters on average, while the GridMA, had 8.2 clusters which was the previous best result. We also note that all the ES versions observed more clusters than the initial version, and GridES had the same number on average. Thus, we improved the initial ES, with the crucial modification being the grid initialization. Gaussian mutation provided more randomness in the location parameters, which led to enhanced location coverage but less vulnerable parameter combinations and locations. However, interestingly, all ES versions provided more clusters than RS and MA, demonstrating the potential of

Table 4.6.: The number of clusters based on the Mean Shift clustering algorithm over the unique *x*-*y* locations per fault class. The bandwidth size is 0.1. The number of clusters is averaged over three runs.

	ES gauss	ES grid	ES grid gauss	GridES grid
fail comb.	8.3	**9**	8.3	8
mute	11.3	10	9.6	10.3
mute_pass	9	12	10	9.3
pass	32.3	31.5	27.6	40

diversity methods.

4.7. CONCLUSIONS AND FUTURE WORK

Previous works show the benefits of algorithms such as memetic algorithm in finding more FI parameter combinations with vulnerabilities compared to commonly used random search. However, the observed results commonly come from a single sensitive region, and during security evaluation, we do not want to neglect possibly exploitable vulnerabilities. Thus, we propose diversity algorithms that promote diversity in the population of evolutionary algorithms and test the GridMA and Evolution Strategy and its variations. While we evaluate algorithms considering the number of unique FI parameter combinations as in related work, we additionally assess algorithm success based on the number of unique locations (*x*-*y*) and clusters with faults as two additional metrics that better align with the objective of finding multiple vulnerable regions. MA performs best only when the number of faults is concerned. However, GridMA and ES with grid initialization and Gaussian mutation (*ES grid gauss*) found more unique locations with faults. Nonetheless, all evolutionary algorithms, including MA, found around 30% more unique locations with *fail* responses than RS, performing similarly. Considering the number of clusters, MA performed the worst, while ES with grid initialization (*ES grid*) had the most clusters, followed by the GridMA algorithm and other ES versions. This work shows that the diversity approach helps find more distant locations with the desired outcome. However, the improvements are less significant than the difference between evolutionary algorithms and RS regarding the number of FI parameter combinations. Thus, while this work showcases the potential enhancement using the diversity approaches, future work could consider (μ, λ)-ES and more advanced diversity algorithms to provide more significant improvements.

REFERENCES

[1] M. Krček and T. Ordas. “Diversity Algorithms for Laser Fault Injection”. In: *Applied Cryptography and Network Security Workshops*. Ed. by M. Andreoni. Cham: Springer Nature Switzerland, 2024, pp. 121–138. ISBN: 978-3-031-61486-6.

[2] P. C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO ’96. Springer. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 104–113. ISBN: 3540615121.

[3] P. C. Kocher, J. Jaffe, and B. Jun. “Differential Power Analysis”. In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. Ed. by M. Wiener. CRYPTO ’99. Springer. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 388–397. ISBN: 3540663479.

[4] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz. “Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller”. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE. 2013, pp. 77–88.

[5] S. P. Skorobogatov and R. J. Anderson. “Optical fault induction attacks”. In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2002, pp. 2–12.

[6] M. Hutter and J.-M. Schmidt. “The temperature side channel and heating fault attacks”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 219–235.

[7] C. H. Kim and J.-J. Quisquater. “Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures”. In: *IFIP International Workshop on Information Security Theory and Practices*. Springer. 2007, pp. 215–228.

[8] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. “Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures”. In: *Proceedings of the IEEE* 100.11 (2012), pp. 3056–3076.

[9] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina. “Sok: Deep learning-based physical side-channel analysis”. In: *ACM Computing Surveys* 55.11 (2023), pp. 1–35.

[10] E. Biham and A. Shamir. *Differential Fault Analysis of Secret Key Cryptosystems*. 1997.

[11] T. Fuhr, E. Jaulmes, V. Lomné, and A. Thillard. “Fault Attacks on AES with Faulty Ciphertexts Only”. In: *Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. FDTC ’13. USA: IEEE Computer Society, 2013, pp. 108–118. ISBN: 9780769550596. DOI: 10.1109/FDTC.2013.18. URL: https://doi.org/10.1109/FDTC.2013.18.

[12] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas. “SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, Issue 3 (2018), pp. 547–572. DOI: 10.13154/tches.v2018.i3.547-572. URL: https://tches.iacr.org/index.php/TCHES/article/view/7286.

[13] L. Wu, G. Ribera, N. Beringuier-Boher, and S. Picek. “A fast characterization method for semi-invasive fault injection attacks”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2020, pp. 146–170. ISBN: 978-3-030-40185-6.

[14] M. Krček, D. Fronte, and S. Picek. “On the Importance of Initial Solutions Selection in Fault Injection”. In: *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. 2021, pp. 1–12. DOI: 10.1109/FDTC53659.2021.00011.

[15] R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub. “Glitch it if you can: parameter search strategies for successful fault injection”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 236–252.

[16] S. Picek, L. Batina, D. Jakobović, and R. B. Carpi. “Evolving genetic algorithms for fault injection attacks”. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2014, pp. 1106–1111.

[17] S. Picek, L. Batina, P. Buzing, and D. Jakobovic. “Fault Injection with a new flavor: Memetic Algorithms make a difference”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2015, pp. 159–173.

[18] A. Maldini, N. Samwel, S. Picek, and L. Batina. “Genetic algorithm-based electromagnetic fault injection”. In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2018, pp. 35–42.

[19] I. Rais-Ali, A. Bouvet, and S. Guilley. “Quantifying the Speed-Up Offered by Genetic Algorithms during Fault Injection Cartographies”. In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 61–72.

[20] V. Werner, L. Maingault, and M.-L. Potet. “Fast Calibration of Fault Injection Equipment with Hyperparameter Optimization Techniques”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2021, pp. 121–138.

[21] M. Moradi, B. J. Oakes, M. Saraoglu, A. Morozov, K. Janschek, and J. Denil. “Exploring fault parameter space using reinforcement learning-based fault injection”. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2020, pp. 102–109.

[22] A. Sedaghatbaf, M. Moradi, J. Almasizadeh, B. Sangchoolie, B. Van Acker, and J. Denil. “DELFASE: A Deep Learning Method for Fault Space Exploration”. In: *2022 18th European Dependable Computing Conference (EDCC)*. IEEE. 2022, pp. 57–64.

[23] M. Krček, T. Ordas, D. Fronte, and S. Picek. “The more you know: Improving laser fault injection with prior knowledge”. In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 18–29.

[24] P. Moscato. “On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms”. In: *Caltech Concurrent Computation Program* (Oct. 2000).

[25] R. Hooke and T. A. Jeeves. ““ Direct Search” Solution of Numerical and Statistical Problems”. In: *J. ACM* 8 (1961), pp. 212–229.

[26] D. Comaniciu and P. Meer. “Mean shift: A robust approach toward feature space analysis”. In: *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002), pp. 603–619.

[27] J. MacQueen *et al.* “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. Oakland, CA, USA. 1967, pp. 281–297.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[29] H.-G. Beyer and H.-P. Schwefel. “Evolution strategies–a comprehensive introduction”. In: *Natural computing* 1 (2002), pp. 3–52.

II

Deep Learning Side-channel Analysis

5

DEEP LEARNING ON SIDE-CHANNEL ANALYSIS

This chapter provides an overview of recent applications of deep learning to profiled side-channel analysis (SCA). The advent of deep neural networks (mainly multiple layer perceptrons and convolutional neural networks) as a learning algorithm for profiled SCA opened several new directions and possibilities to explore the occurrence of side-channel leakages from different categories of systems. This is particularly important for designers when verifying to what extent an adversary can extract sensitive information when possessing state-of-the-art attack methods. Deep learning is a fast-evolving technology that provides several advantages in profiled SCA, and we summarize the main directions and results obtained by the research community.

This chapter has been published as M. Krček, H. Li, S. Paguada, U. Rioja, L. Wu, G. Perin, and Ł. Chmielewski. "Deep learning on side-channel analysis". In: *Security and Artificial Intelligence: A Crossdisciplinary Approach*. Springer, 2022, pp. 48–71.

5.1. INTRODUCTION

Side-channel attacks (SCA) are a well-known and powerful class of implementation attacks against different types of systems, such as cryptographic implementations, processors, communication systems, and, more recently, machine learning models. What makes these attacks powerful is the fact that they use unintended leakage of information conveyed from different sources: power consumption, electromagnetic emanations, time, temperature, acoustic, photonic emission, etc. An adversary uses specialized equipment to monitor some of those side-channel leakages to extract secret information. For example, the amount of time needed to process a specific secret byte in a computer might be different from the amount of time needed to process other possible values for this byte. The monitoring of a side-channel can lead an adversary to recover secret information from time measurements.

In this chapter, we focus on the predominantly used form of side-channel attacks that are based on power consumption and electromagnetic analysis to extract secret cryptographic keys. Embedded devices make use of cryptographic primitives to protect the processing and storing of sensitive information. However, the cryptographic algorithmic implementations (e.g., AES, 3DES, RSA, etc.) in software and hardware also need protection on their private keys. Side-channel attacks can extract those keys from unintended information leakage if the device is not properly protected. Differential power analysis (DPA [2]) and correlation power analysis (CPA [3]) appeared in 1999 and 2004, respectively, as powerful statistical methods to recover private keys. These attacks, classified as non-profiled attacks, assume that an adversary can query multiple encryption (resp. decryption) executions from a target crypto operation by controlling, at least, the plaintext (resp. ciphertext). All the executions represent a set of side-channel measurements. Usually, an adversary splits the target key in chunks (divide-and-conquer strategy), and for each possible value of a key chunk, he or she creates a list of predicted labels based on the crypto algorithm (e.g., when attacking an AES implementation, an adversary predicts what are the values of S-box output in the first round based on possible byte key guesses). As a final act, the adversary performs a differential or correlation analysis between side-channel measurements and all possible sets of predicted labels. The key guess associated with the highest difference-of-means or correlation value is assumed to be the correct key chunk value.

In 2002, Chari et al. [4] proposed a different category of SCA known as Template or Profiled Attacks. Profiled SCA is a specific class of side-channel analysis that is conducted in two phases: profiling and attack phase. To profile a crypto implementation, the adversary collects a set of side-channel measurements where the key of the target cryptographic execution is known and may vary from measurement to measurement. Thus, the adversary creates statistical models (commonly called templates) that can describe the leakage and noise of the device under control. In the second phase, a separate set of side-channel measurements is collected from another and (usually) identical device running an unknown key. The attack (also known as the matching phase) applies the learned templates to this second device, which can indicate the most likely key values ordered according to their probabilities.

After the aforementioned publications, several countermeasures to protect crypto

implementations were proposed and applied by the security industry. Randomization of sensitive information, boolean masking schemes, and noise addition are among the most common forms of protection against side-channel attacks. However, research on SCA is constantly discovering new capabilities of side-channel attacks when adopting more advanced statistical techniques. The main goal is to investigate how far an adversary can go when using the most advanced and realistic attack techniques. In this sense, results are important for developers to know what kind of protections their crypto designs need depending on their applications and risk. Recently, publications considering deep neural network approaches demonstrated the ability to break protected crypto implementations [5–7]. This is the main reason why deep learning is the predominant form of profiled side-channel attacks nowadays. Indeed, deep neural networks perform extremely well on a wide variety of learning tasks. Observe that in the profiled SCA setting, the profiling and attack phases are similar to the learning and prediction steps of a deep neural network-based supervised classification task. The deep learning field is continuously evolving and impacting side-channel attacks in general, even beyond profiled SCA. In this chapter, we summarize results from recent publications where non-profiled SCA attacks also leverage powerful deep neural networks.

This chapter's focus is on deep learning-based SCA. We start by providing background knowledge on deep neural networks and profiled SCA. Section 5.3 provides an overview of the state-of-the-art in the application of deep neural networks to profiled SCA. In Section 5.4, we analyze the advantages of using deep learning in the context of profiled side-channel analysis. The main idea of Section 5.4 is to highlight why deep neural networks can be seen as powerful alternatives to classical profiled attacks such as template attacks and traditional machine learning, which have been considered the most powerful class of SCA for many years. Section 5.5 brings an important discussion about the correct interpretation of metrics for deep learning-based profiled side-channel analysis. As reported in recent publications, the usage of well-known supervised classification metrics (e.g., accuracy, loss, recall, etc.) can be meaningless in the context of SCA when evaluating protected targets. In Section 5.5, we describe solutions for such an important problem.

Another important aspect of training deep neural networks is the hyperparameter tuning. In Section 5.6, we describe this problem in the context of profiled SCA. As we will see, there are still no published efficient solutions for this problem for SCA, and we describe possible alternatives. Section 5.7 describes different applications of deep learning to SCA. Finally, Section 5.8 concludes the chapter while giving an overview of what we believe to be the most important perspectives and directions for future work.

5.2. BACKGROUND

In this section, we explain the notation we use throughout the chapter and deep learning-based profiled SCA.

5.2.1. NOTATIONS

Throughout this chapter, we use $\mathscr{X} = \{X, Y\}$ to denote a dataset composed of feature array X and label vector Y. The feature array $X = \{x_{i,f}\}$ defines a set of N side-channel traces, where i indicates the trace index and f indicates the feature index (a sample) inside a trace. In the label vector $Y = \{y_i\}$, each element y_i indicates the label associated with a side-channel trace i. The terms $\mathscr{X}_{train}$, $\mathscr{X}_{val}$, and $\mathscr{X}_{test}$ denote the training, validation, and test sets, respectively. The terms profiling traces and training traces are used interchangeably throughout the chapter.

The term k refers to a single key byte belonging to the full encryption or decryption key $\mathscr{K}$ with dimension $|\mathscr{K}|$. An input plaintext byte used as input to the encryption or decryption operation i (executed in order to obtain a side-channel trace) is referred to as pk_i. The plaintext byte pk_i belongs to the full input plaintext PK_i. Let also $f(pk_i, k)$ denote the function that returns the label associated with one execution of a cryptographic operation.

Let also $p_{i,j} = P[y = j|X = x_i]$, where $p_{i,j} \in \mathscr{P}$, denote the probability that a side-channel trace x_i contains the label j. In this chapter, bold letters and bold acronyms, e.g., **a**, denote vectors of dimension 2^b, where b is the bit-length of the target intermediate value in a cryptographic execution.

5

5.2.2. PROFILED SCA AND DEEP LEARNING

The deep learning-based profiled side-channel attack requires a training set of size N for the learning or profiling phase. Ideally, the training set should be composed of side-channel traces where each trace is measured with random input data (ciphertext or plaintext) pk and random key k. To create a labeled dataset for SCA, also referred to as the training set $\mathscr{X}_{train}$, it is important to first choose a leakage model that better describes the physical side-channel leakage present in the underlying measurements. Commonly selected leakage models against symmetric crypto implementations (e.g., AES, DES) are Hamming weight (HW), Hamming distance (HD), identity (ID), or bit-level models. In this case, the leakage is modeled for an intermediate value represented by a single byte or a bit. This intermediate value in an encryption or decryption operation depends on a key byte k and input (i.e., plaintext or ciphertext) pk.

The number of possible classes to label a dataset is directly derived from the selected leakage function $f(pk_i, k)$. As an example, the target intermediate value of an AES implementation is usually a byte in the S-box state in the first or last encryption/decryption round. In this case, HW or HD models define nine classes for the datasets. In case the ID model is used, the dataset is defined for 256 classes. Attacks on a single intermediate bit define only two possible classes. The bit-level leakage model is also a common situation when attacking public-key implementations (e.g., RSA, ECC). In this last case, the attacker has specific knowledge about the target implementation, such as scalar multiplication or modular exponentiation methods.

For training a deep neural network, the labeled set is split into training, $\mathscr{X}_{train}$, and validation, $\mathscr{X}_{val}$, sets. As we will discuss in Section 5.5, classic validation metrics (e.g., accuracy, loss, recall, precision, etc.) obtained during training may

not indicate the leakage detection performance of a neural network, especially for protected targets. Therefore, to have a meaningful validation metric for SCA, the best possible scenario is to compute guessing entropy or success rate during training. For that, it is crucial to have a validation set $\mathscr{X}_{val}$ where the key $\mathscr{K}$ is fixed for the full validation set. If $\mathscr{K}$ is random for each trace in $\mathscr{X}_{val}$, then a different efficient validation metric must be found.

Understanding deep learning metrics in the context of SCA is essential background knowledge for training efficient models. Conventional deep learning metrics usually are accuracy and loss (or error). *Accuracy* indicates the ratio between correctly predicted data and the total number of predictions. *Loss* is based on a *loss function* selection (the most common form of loss function for profiled SCA is *cross-entropy*), and it indicates the overall error for the evaluated set. These metrics are monitored during the training phase and can indicate different phases that can occur while the parameters (weights and biases) are being updated by stochastic (or adaptive) gradient descent methods.

Common metrics in SCA are success rate and guessing entropy [8]. In profiled SCA, these metrics are not aimed only at predicting correct labels, as is the case with machine learning metrics, but also to reveal the secret key. In particular, let us assume that given Q amount of traces in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathscr{K}|}]$ in decreasing order of probability with $|\mathscr{K}|$ being the size of the keyspace. So, g_1 is the most likely and $g_{|\mathscr{K}|}$ the least likely key candidate. The first-order success rate is defined as the average empirical probability that g_1 is equal to the secret key k^*. The guessing entropy is the average position of k^* in $\mathbf{g}$. When predicting an attack set $\mathscr{X}_{test}$ with a fixed key k^*, we obtain a prediction array $\mathscr{P} = \{p_{i,j}\}$. This array has the number of rows equivalent to Q, and the number of columns is equivalent to the number of possible classes. For all key candidates, we compute the probability that k is the correct key k^* in $\mathscr{X}_{test}$ as follows:

$$P[k = k^*] = \sum_{i=0}^{N-1} log(p_{i,j}) \tag{5.1}$$

where $p_{i,j}$ is the j-th class probability (or prediction) of the neural network for side-channel trace i. The class index j is determined according to a leakage model function $j = f(pk_i, k)$.

In profiled attacks, the main goal during the training phase is to reach a generalization performance so that the trained deep neural network can obtain a low guessing entropy (resp. a high success rate) after predicting Q attack traces. The generalization phase usually happens after the model starts fitting the side-channel leakages (after *underfitting*) and before this same model starts to degrade its performance, i.e., when it usually reaches an *overfitting* phase. Fitting refers to how well the model approximates the unknown underlying mapping function given the input and output variables. When overfitting occurs, the neural network can fit the training set with very high accuracy and small errors, but it cannot fit the validation or test sets. Ideally, we should always train a neural network until it achieves the maximum quality in terms of generalization concerning the validation set. If this

happens, we should be able to assess whether the model is in the generalization phase or not. This seems to be an easy task, but there are quite some difficulties in the interpretation of metrics to identify what phase the model belongs to while the training evolves. An interesting observation would be the detection of the boundaries between two of the phases above. These boundaries may be detected with the observation of conventional metrics (loss, accuracy, recall, or precision) using validation data. In Section 5.5, we discuss potential solutions to identify the generalization phase during training in the SCA context.

5.3. Recent results in deep learning-based profiled side-channel attacks

This section provides an overview of recently published results on deep learning-based SCA. Various types of deep neural networks have been used, and we summarize what is state-of-the-art regarding the selection of neural network topologies. The information contained in this section is a summary of recent results that mainly target cryptographic primitives based on AES [9], RSA [10], and ECC [11, 12].

5.3.1. From machine learning to deep learning in SCA

Machine learning techniques are quite successful in a lot of different fields, such as image classification [13] or speech recognition [14]. Due to the shared similarity between profiling SCA and supervised machine learning, researchers started experimenting with machine learning techniques in profiled SCA. They utilized standard machine learning techniques, such as Support Vector Machines (SVMs) and Random Forests [15], neural networks [16], and, more recently, deep learning [5, 17–19].

Deep learning is a class of machine learning where the learning algorithm (i.e., a deep neural network) extracts higher-level features from the raw input. Usually, in the case of deep neural networks, it is considered that the neural network has multiple layers. Therefore, multi-layer perceptrons (MLP) with more than one hidden layer are deep neural networks. Some other examples of deep learning techniques include deep belief networks (DBN), recurrent neural networks (RNN), convolutional neural networks (CNN), and residual neural networks.

The first publication to explore deep learning-based profiling SCA results is from Maghrebi et al. [17]. In this case, the authors applied MLP with one hidden layer (not considered a deep neural network), a Stacked Auto-Encoder with three hidden layers, and also introduced the application of CNNs for SCA. The authors also applied a specific RNN architecture called a long short-term memory (LSTM) network with two layers of 26 LSTM units and a Random Forest algorithm. Except for the Random Forest algorithm and the MLP the authors used, other algorithms are considered deep learning techniques. The authors also suggested hyperparameters tuning for SCA with a genetic algorithm, as stated in the appendix of their paper. Since then, there has been a variety of different architectures with different hyperparameter settings. We provide an overview of the deep learning techniques applied to SCA

and discuss state-of-the-art results.

5.3.2. DEEP LEARNING TECHNIQUES IN SCA

After the appearance of the first publication with deep learning results for SCA, researchers started to investigate the benefits of well-known learning techniques, such as regularization, visualization, hyperparameters optimization, and model interpretation, to improve the attack performance.

Regularization techniques are used to avoid overfitting during the training phase. Examples of regularization techniques are data augmentation, noise addition, weight decay, dropout layers [20], and early stopping. In [5], the authors presented the first results with data augmentation techniques for CNNs to bypass desynchronization in side-channel measurements. The adopted techniques are based on random trace shifting and trace warping during the training phase. This way, the trained CNNs can generalize to side-channel measurements where the leaking samples appear in random time locations due to desynchronizations caused by measurement setup or countermeasures. Results achieved for protected AES implementations demonstrate the benefits of these well-known regularization techniques. Kim et al. [7] explored how additional noise can be used as a regularization for preventing overfitting, and they also present their CNN architecture. Here, the authors compare the performance of their CNN architecture to the CNN architecture introduced in [18] paper. Also, the authors explore more datasets, and their neural network is larger in the number of layers, indicating the benefits of more convolution layers for leakage detection.

The selection of hyperparameters for deep neural networks is also explored in some of the publications. In [18], the authors provide several results for different CNN and MLP configurations and also introduce the open ASCAD dataset to serve as a basis for further work on the SCA. The authors also tested Self-Normalizing Neural Networks (SNN), but the performance compared to the MLP showed no significant improvement. Considering how to develop an adequate CNN architecture for SCA, the authors chose to test some state-of-the-art CNN architectures from the image recognition field, such as VGG-16 [21], ResNet-50 [22], and Inception-v3 [23]. From the initial architecture, the authors performed tuning of the hyperparameters, using guessing entropy as the metric to define the model. Another relevant paper to mention is where the authors present a methodology for creating neural network architectures for SCA application [19]. The paper introduces several CNN architectures, each fine-tuned for certain characteristics of utilized datasets, showing how the initial CNN architecture can be light-weighted by searching for appropriate hyperparameters.

Visualization techniques are also explored for profiled SCA. These techniques indicate the main features in input data that the trained model considers most important for its classification decisions. For instance, Masure et al. [24] provide results with gradient visualization. In [25], the authors compare the performance of different visualization techniques. These results are discussed in more detail in Section 5.4.5.

In the line of model interpretability, the work presented in [26] provides the

first results with the Singular Vector Canonical Correlation Analysis (SVCCA) tool to interpret what neural networks learn while training on different side-channel datasets. All the aforementioned deep learning techniques provide different perspectives for profiled SCA.

In the next section, we explore the main advantages of deep learning in comparison to classical profiled attacks, such as template attacks or machine learning.

5.4. ADVANTAGES OF DEEP LEARNING FOR PROFILED SIDE-CHANNEL ANALYSIS

In this section, we analyze the advantages of deep learning for profiled SCA compared to classic techniques such as template attacks and traditional machine learning. First, in Subsection 5.4.1, we discuss the fact that deep learning does not require preprocessing or feature selection. Subsection 5.4.2 describes results that indicate convolutional neural networks are less sensitive to trace desynchronizations. Subsequently, Section 5.4.3 discusses results indicating that deep neural networks can learn high-order leakages. Then, we explain how deep learning can take advantage of the domain knowledge in Subsection 5.4.4. Finally, Subsection 5.4.5 describes various attribution methods (or visualization techniques) for leakage detection.

5.4.1. SIDE-CHANNEL ANALYSIS WITHOUT PREPROCESSING

Template attacks (TA) [4] are commonly considered one of the most powerful side-channel attacks from an information-theoretic point of view. However, for template attacks to be successful in practice, one needs to choose some special samples as the interesting points in actual side-channel traces [27]. These points are usually referred to as *points-of-interest (POIs)*.

Much research has been done on choosing POIs that lead to the most successful TAs (see [28]). This choosing process is often called *POIs selection*, and many different approaches have been introduced over the years. However, it is unknown whether these approaches to choosing interesting points will lead to the best classification performance of TAs. For example, it is hard to quantify whether all useful points for TA have been chosen. In general, we do not know which approach is the best for all possible devices and implementations, as the proposed techniques are only validated experimentally, and no universally optimal solution has been presented. Moreover, significant processing of the samples, including, most notably, alignment, is necessary before both POIs selection and TAs are run.

Related to the POI selection is the problem of feature extraction from the traces. The goal of the extraction is to find the most leaking components in the traces and filter out the noisy components. A significant amount of effort has been put into this research direction. For example, Principal Component Analysis was used to extract features and use them from TA [29]. Feature selection is also necessary for machine learning techniques like SVM, random forests, or decision trees.

The significant problem with the POIs selection and feature extraction techniques is that they are not an integral part of TA and introduce additional complexity. In

comparison, the techniques based on deep learning suffer to a much lesser extent from these problems. In particular, feature extraction is a part of the problem that deep learning aims to solve. The input layer of a deep neural network directly receives the side-channel trace interval corresponding to the interval of interest. It is expected that during training, the backpropagation algorithm (often using stochastic or adaptive gradient descent) will learn network parameters (e.g., weights and biases) in a way that only some of the input features will be relevant in the neural network classification decisions.

As a result, an adversary does not need to identify leaking samples (or features) beforehand, as this process is automatically done by the backpropagation algorithm. Of course, the neural network selects input features representing points of interest as long as they can fit the leakage contained in the side-channel measurements. During the training process, it is possible that the neural network overfits the training data, and poor generalization is provided. In this case, the neural network will mostly make a decision based on different input features for each classified side-channel trace, meaning that automated POI selection was not successfully made. The most recommended method to address overfitting problems is regularization.

5.4.2. BYPASSING DESYNCHRONIZATION

Well-synchronized traces can significantly improve the correlation between the intermediate data and the trace values. Therefore, the alignment of the traces is an essential step to enhance the efficiency of the side-channel analysis. Static alignment is the most commonly used approach to align the traces. Usually, an attacker should select a distinguishable trigger/pattern from the traces so that the following part can be aligned using the selected part as a reference. There are two limitations to this approach. First, the selected trigger/pattern should be distinctive so that it will not be obfuscated with other patterns and lead to misalignment. Second, when countermeasures such as random delay interrupts are implemented, the selected trigger should be sufficiently close to POIs to minimize the countermeasure effect. From a practical point of view, a good reference that meets both limitations is not always easy to find. Although the other optimized alignment methods, such as Elastic Alignment [30] and Rapid Alignment Method [31], could be candidates to reduce the effect of the countermeasures, the rising of the preprocessing cost makes the traces synchronization a challenging task.

Deep learning provides alternative approaches to bypass desynchronization, which can be realized by either trace preprocessing or a direct attack on the misaligned traces. For traces preprocessing, autoencoder (AE) [32], an unsupervised-learning model well-known for the ability for feature extraction [33, 34], could be applied for denoising purpose and lead to an improvement of attack efficiency in both non-profiling [35] and profiling side-channel analysis [36]. Specifically, to train an AE for denoising purposes, the input and output are represented by noisy-clean trace pairs. A well-trained denoising AE could keep the most representative information (i.e., trace leakage) in the latent space while neglecting the less important features, such as random noise. Once the denoising AE is trained, much cleaner traces can be recovered by feeding noisy traces to the input of the AE. Back to desynchronization,

it can also be considered as a type of noise that introduces variation in the time domain. As stated in the paper [36], by training the denoising AE with a limited amount of traces, the reconstructed traces can lead to a good performance comparable to the original one.

In terms of deep learning-based side-channel analysis, several works have presented their effectiveness compared to classical methods [18, 37, 38]. There are two reasons in general: 1) deep learning-based attacks do not require critical preprocessing (i.e., realignment), as it is covered in the learning phase; 2) deep learning-based attacks automatically reduce the dimension of the traces by combining the raw features non-linearly with the interconnection of neurons and transferring them to the low-dimensional representations. Thanks to these two characteristics, the effect of misalignment can be reduced during network training. Among different deep learning models, convolutional neural networks (CNNs), thanks to their spatial invariance property, were demonstrated to be the most preferred architecture in coping with desynchronization and the random delay countermeasure [18]. To further enhance the capability of CNN in handling the desynchronization, two possible techniques could be applied: first, exploit the leakage in traces' frequency representation by transferring the traces with short-time Fourier transform [39]; second, applying data argumentation, such as artificially adding random delays, clock jitters [5] or Gaussian noise [40], as it could help increase the diversity of the training sets and balance the class distribution of the profiling data.

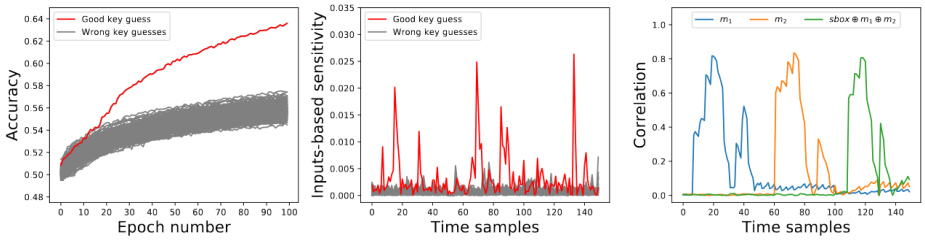
In summary, compared with conventional profiling attack methods, deep learning architectures are more resilient to variation in the time domain. Together with their attacking performance compared with classical profiling attack methods, deep learning becomes a preferable method for profiled side-channel analysis.

5.4.3. DEEP NEURAL NETWORKS CAN LEARN SECOND-ORDER LEAKAGES

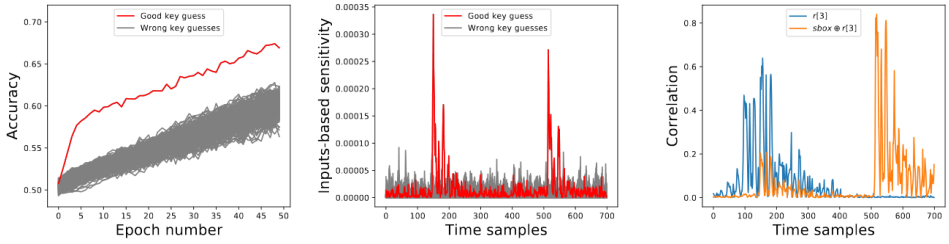
Side-channel attacks exploit the dependency between the power consumption of a cryptographic device and the intermediate values of the implemented cryptographic algorithm. Side-channel countermeasures try to obfuscate the aforementioned data dependency. In particular, masking countermeasures achieve it by randomizing the key-dependent intermediate values of the algorithm. The goal is to make the power consumption of the device independent of the intermediate values. This is achieved with the Boolean operations between random values (masks) and intermediate data. At the end of encryption/decryption execution, the masks are removed from the result [41–43].

Conversely, with the inclusion of deep learning techniques in the side-channel analysis field, masked cryptographic implementations have a new threat: neural networks have shown their capability to break masked cryptographic implementations. Up to now, publication results only demonstrated the capacity to break protected software implementations in the profiled setting. Benjamin Timon in [44] proposed a deep learning solution for non-profiled attacks against masked AES implementations. Even if the attack does not require profiling on an identical device, it still requires the training of a deep neural network for each key candidate, drastically increasing the attack complexity. The results provided in [44]

demonstrate the efficiency of their method against ASCAD dataset [18] and custom ChipWhisperer implementations. However, it is difficult to assume that this method will not be restricted by its high complexity limitations when applied to different devices. Nevertheless, it was demonstrated with input-based sensitivity that deep neural networks can fit high-order leakages. Figure 5.1 provides results from [44] demonstrating that deep neural networks fit high-order leakages when the training set is labeled based on the correct key candidate.



(a) Results on ChipWhisperer AES implementation.



(b) Results on ASCAD AES implementation.

Figure 5.1.: Results from [44] (Figure 12) illustrating the ability of deep neural networks to fit high-order side-channel leakages.

Masure et al. [24] used loss function-based input activation gradients to demonstrate that convolutional neural networks can learn second-order leakages, as indicated in Figure 5.2. Input activation gradients based on loss function indicate the samples where the loss function is more sensitive. As Figure 5.2 indicates, the input activation gradients are higher for the samples representing the processing of mask values.

Although there are no formal explanations for the fact that deep neural networks are able to fit second-order leakages, they actually combine multiple samples in order to make their decisions. As stated in Section 5.4.1, deep neural networks can learn high-order representations from data, and it is expected that these complex learning systems would be able to learn high-order side-channel leakages.

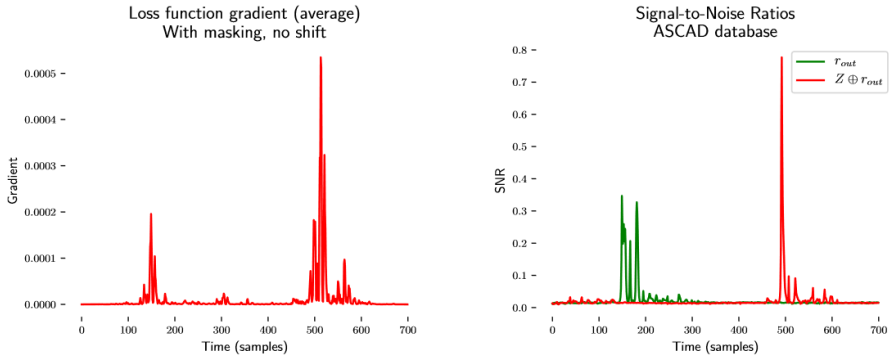


Figure 5.2.: Results from [24]. Input activation gradients indicate that CNNs can fit second-order leakages.

5

5.4.4. TAKE ADVANTAGE OF THE DOMAIN KNOWLEDGE

Domain knowledge (DK) [45] assumes that the information domain can be used as part of the dataset to improve the generality (to different datasets) and robustness (towards noise interference) of classifiers. The usage of DK neurons was first introduced to the side-channel domain by Hettwer et al. [46] in 2018. The authors provided the plaintexts as additional information into the neural network to learn the leakage regarding the secret key directly. Specifically, by concatenating the traces' latent representation (a dense layer of CNN) with the one-hot-encoded plaintexts at the byte level, better results can be obtained when performing attacks. Figure 5.3 illustrates this procedure: the features were extracted from the input by the convolution layers, which are then combined with DK neurons containing the plaintext information to enhance the classification performance. Following this, researchers found that combining the DK neurons (bit-encoded plaintexts) with the denoising autoencoder could reduce the effect of the masking countermeasure [35]. Indeed, merging domain-specific information with extracted features of the convolution layers enables the network to converge to different statistics at the decision level [47]. In other words, leakage traces are generated conditionally on the provided plaintext and the secret key. Although the correlation between plaintext and secret key may not exist, the extra information could still be helpful in extracting more meaningful features. By applying DK that may not be limited to plaintext (i.e., ciphertexts), we see the potential of DK in performing more powerful attacks.

5.4.5. VISUALIZATION TECHNIQUES TO IDENTIFY INPUT LEAKAGE

As we have already mentioned, profiling attacks are classification problems, and in this type of problem, it is always meaningful to have a method to visually interpret how the learning model is using the features to conduct the classification. Visualization techniques are very useful for manufacturers when evaluating the

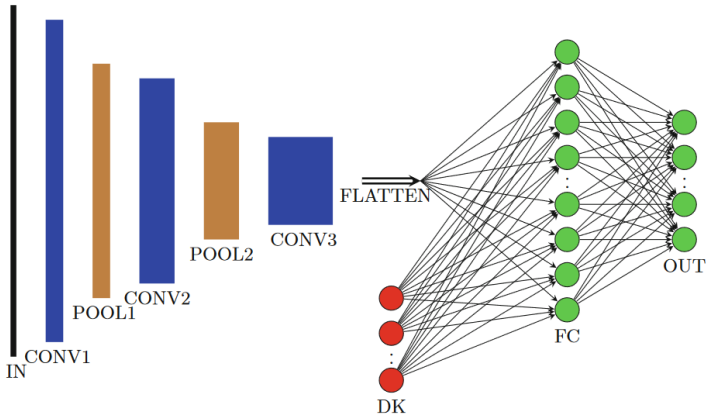


Figure 5.3.: Image from [46]. A convolutional neural network with domain knowledge in a fully connected layer.

side-channel security of their design. This is a useful fact to take advantage of. Recall that an important aspect of evaluating a supposedly secure device is pointing out where the leakage is being generated. This way, it is possible to propose modifications and recommendations to limit adversarial possibilities. Knowing where the leakage is generated becomes crucial to improving security and eliminating the main flaws in crypto implementation.

In general terms, visualization is conducted by analyzing what input features (given by a neuron in the input deep neural network layer as a one-to-one mapping) have more influence in classification during training. After the activation of the neurons, the learning algorithm back-propagates the error to update the weights in those neurons until they reach the first layer. In [24], the authors proposed the visualization of input activation gradients as a technique to characterize the automated selection of points of interest by deep neural networks. The result is a vector of gradients computed by the backpropagation algorithm as the derivative of the loss function with respect to the input activation. Gradient visualization is the technique that computes the value of the derivatives in a neural network regarding the input trace, which is then used to point out what feature needs to be modified the least to affect the loss function the most. In [44], the authors proposed the same solution, defined as sensitivity analysis, and they used it in the context of non-profiled side-channel analysis where input activation gradients are used as distinguishers. The input activation gradient method has already been used as a tool to show that neural networks can actually fit high-order leakages (see Section 5.4.3).

Another technique employed for this purpose is Layer-wise Relevance Propagation (LRP). LRP propagates the classification score through the network until the first layer and then conducts the one-to-one mapping [48]. The work in [49] applied a method gradient visualization to show how MLP could be used for the Leakage Assessment Methodology (see Section 5.7). In [25], authors used the LRP technique

to identify which samples of the power trace have a greater influence on the learning process. They compared three attribution methods (Saliency Maps [50], LRP, and Occlusion [51]) on three different datasets, showing how LRP is the most suitable for finding POIs. Finally, one of the most recent approaches is to use heatmaps (or feature maps) to interpret the impact of filters (also known as convolutional matrices or kernels in the context of CNNs) in order to adapt the neural network model according to how the features are selected [19].

5.5. METRICS FOR DEEP LEARNING-BASED PROFILED SCA

Supervised classification tasks require correct metrics to determine the performance of the algorithm as well as to measure its learning capacity. Accuracy, precision, recall, AOC-ROC, and log-loss are commonly used metrics in supervised classification tasks for many application domains. Computer vision and natural language processing are among applications where the classification accuracy must be very high to solve the underlying problem. Each element in a test set is classified separately. Thus, we are only interested in the generalization capacity of the trained algorithm as long as it provides a high classification accuracy. In particular, deep neural networks have performed extremely well over a wide variety of supervised classification tasks.

When a profiled side-channel attack makes use of a deep neural network as the learning algorithm, we are interested in verifying how effective these highly complex learning systems are in order to learn side-channel leakages and retrieve the cryptographic secret by classifying side-channel traces according to a leakage model. If we strictly consider a side-channel attack on an AES implementation (where the attacker trains a profiling model to attack each separate key byte), a profiled attack is theoretically able to recover the correct target key byte with a single side-channel measurement. This scenario would be possible if the identity leakage model of S-box output in the first encryption round is defined as the leakage function. However, real side-channel measurements are noisy, and several (from hundreds to thousands of) traces are required to conclude attack capability. This is done by summing up the output class probabilities for each classified side-channel trace, in which the selected output probability value for each trace corresponds to the class associated with the guessed key. For each key guess candidate, we compute a probability that key guess k is correct, $P[k = k^*]$, according to Equation (5.1). Therefore, in a situation where a deep neural network is trained in a way that it can fit the existing leakages, classifying multiple traces and combining their predicted class probabilities is necessary to estimate the success rate or guessing entropy for a certain amount of traces.

Sometimes, the number of classified side-channel traces in a test or attack phase needs to be very large to reach a consistent conclusion about the model generalization. In such a scenario, the output class probability for the true label (or class) is not always the highest value in the output layer. As a consequence, the overall test accuracy, precision, or recall will stay close to random guessing. In the same way, the cross-entropy loss function for the test set also does not inform

enough about the model generalization. This brings us to an important question: what metric should a deep learning-based SCA consider? The authors of [52] demonstrated that conventional machine learning metrics are not very informative for the side-channel analysis domain, concluding that the best metrics are guessing entropy and success rate. In [53], the authors proposed a didactic analysis of output class probabilities obtained by attacking protected AES targets. They once more provided evidence that guessing entropy and success rate become the main metrics for deep learning-based profiled SCA. Based on that, the analysis can be improved in many different directions. Analyzing the guessing entropy during the processing of training epochs can lead to identifying an efficient early-stopping metric. As a result, the neural network will be regularized for side-channel analysis. This is similar to what has been recently proposed in [54].

Therefore, to improve side-channel leakage detection with deep learning, side-channel metrics at a validation level still offer the best alternatives. The selection of correct metrics in deep learning profiled SCA is also important in hyperparameter tuning algorithms. As we discuss in the next section, these optimization algorithms converge according to a metric direction (minimum or maximum value), and if the selected metric is not consistent with SCA, the optimization algorithm may have convergence problems.

5.6. TUNING NEURAL NETWORK HYPERPARAMETERS FOR SCA

This section discusses the problem of selecting efficient hyperparameters for deep learning-based profiled SCA. The performance of deep learning-based profiled side-channel attacks depends greatly on the selection of hyperparameters for neural network topology. Different from other profiled attack methods, such as template attacks and machine learning-based attacks, deep neural networks have tenths of hyperparameters to be defined.

The definition of hyperparameters can be strictly related to the attacked dataset. Several aspects in the dataset may directly affect the selection of specific hyperparameters: countermeasures, noise levels, number of measurements, number of points in a side-channel measurement, or trace and appropriate leakage model. This already means that one of the main challenges in deep learning-based profiled SCA is the difficulty (if not impossible task) of finding a universal deep learning model that works well on a variety of datasets.

Hyperparameter search is a common task in all kinds of deep learning applications. For profiled side-channel attacks, the situation is not different. Usually, several combinations of hyperparameters are evaluated against a dataset, and the best possible combination that solves the problem (i.e., recovers the target key byte(s)) is assumed as an optimal model for the underlying task. Depending on the dataset features and target implementation details, manually finding efficient hyperparameters can be very hard or near impossible. Therefore, there are two main paths to solve this problem: by deeply understanding the role of specific hyperparameters and their effect on specific datasets or, more recommended,

by adopting an optimization algorithm to automatically find the best possible configuration given restricted computation time and resources.

Even if the second path is chosen, the analyst has to set hyperparameter ranges and possible options to be searched by the optimization algorithm. If the search range for a specific hyperparameter is completely wrong, it is very likely that the search or optimization algorithm will never identify an efficient combination of hyperparameters that solves the underlying problem. This can be a serious problem when evaluating the target with profiled SCA, as the analyst can draw wrong conclusions about the security of the device. If deep learning-based attacks cannot recover sensitive information from side-channel leakages, one of the main reasons can be the wrong definition of a deep neural network architecture.

If a dataset is too large (with a few million side-channel measurements) and strong countermeasures are present, we expect that a larger model is chosen. This can result in MLP or CNN with several hidden layers, similar to competitive architectures such as VGG-16. However, an appropriate definition of the size of the neural network (and consequently the number of trainable parameters) is insufficient to ensure its efficiency. A very large deep neural network can overfit even large datasets. This means that alternative solutions to restrict the overfitting to the training set are necessary to improve performance in terms of generalization. Widely adopted techniques to improve generalization are *regularization techniques*, and some of them can directly affect the selection of hyperparameters and their ranges in a search problem. Some hyperparameters are directly related to the way weights and biases are updated during network training. In [55], the authors investigate the impact of different widely used optimizers in profiled SCA. Their results indicate different performances concerning different amounts of profiling traces and neural network sizes. Moreover, they demonstrate that some optimizers (Adam and RMSprop) tend to provide fast convergence but have more chances to overfit the network. For other cases, Adagrad and Adadelta optimizers are appropriate for large network models and large datasets and tend to work better when large amounts of training epochs are considered while offering smaller chances to overfit. Although not investigated in [55], learning rate, batch size, and the number of epochs are also training hyperparameters that greatly influence the attack performance. The learning rate scheduler (by changing the learning rate during training) is an important artifact to reduce overfitting and stabilize the generalization even if a large number of epochs is considered.

A viable approach when manually tuning hyperparameters is their modification according to observed metrics. As discussed in Section 5.5, accuracy and loss functions are inconsistent metrics to evaluate attack performance (key recovery). However, these two metrics can still be used to analyze how fast a neural network overfits the training data. If this happens very early in the training phase while the guessing entropy or success rate indicates no key recovery, the neural network model is probably too large for the evaluated dataset. Alternative solutions are to increase the number of training or profiling traces, reduce the size of the network, or adopt regularization techniques (e.g., early stopping, data augmentation, noisy/batch normalization layers, regularization L1/L2, etc.). Other SCA options, such as leakage

models and the number of attack/validation traces, also directly influence the learning capacity of the model. For instance, to ease the computation cost, it is crucial to verify if searching for optimal hyperparameters or changing the amount of profiling traces is more efficient. The same principle may apply to the number of attack traces, as a trained model could require a large amount of traces to be able to select the correct key as the most likely one. Therefore, this trade-off between the number of profiling traces, the number of attack traces, and hyperparameters needs to be considered. In order to analyze the effect of each of the aforementioned aspects, the authors in [56] propose a new framework where they explore the number of traces and hyperparameter tuning experiments required in the profiling phase such that an attacker is still successful. One important benefit from [56] is the identification of what is the main attack component (hyperparameters, number of attack traces, or number of profiling traces) that affects mostly the performance of the profiled attack. With such a framework, an analyst can explore the minimum amount of (profiling and attack) traces if the number of hyperparameter combinations can be very large due to little limitations in time complexity.

Some hyperparameter selection techniques in the deep learning domain also apply to the specific SCA field, such as Grid and Random Search Optimization [18, 19, 38, 57, 58]. Regarding Grid Search Optimization, a step-wise search is defined by setting a range of values for specific hyperparameters. Specifically, to get the optimal parameter combination, the network training is implemented sequentially for every value in the grid. Thanks to its simplicity, Grid Search Optimization is the most widely (although not efficient) used strategy for hyperparameter tuning in SCA. For instance, in [18], the authors adopted Grid Search to experimentally show the process of choosing each hyperparameter for MLP and CNN and presented the impact of each hyperparameter on the performance of SCA. Besides, Grid Search is proved to be reliable in low dimensional spaces [58].

For Random Search Optimization, similar to its counterpart, a fixed range of values must be defined for each hyperparameter. Then, a set of hyperparameters is chosen randomly from their range as a combination, which is applied to the neural network for evaluation. The authors in [38] used Random Search to find the most optimized CNN model for the datasets by tuning 13 hyperparameters. Indeed, Random Search Optimization can be implemented automatically with high efficiency in selecting the optimal hyperparameters. However, the impact of each hyperparameter is not taken into consideration [18, 57].

Other optimization methods were also proved efficient in tuning the deep learning model. In [19], architecture hyperparameters were chosen by using the visualization techniques to understand how each hyperparameter impacts the efficiency of the CNN, and each optimizer hyperparameter was selected by Grid Search from a finite set of values. Moreover, Evolutionary algorithms, such as genetic algorithms and simulated annealing, could also provide better solutions as they implemented metric-based optimizations. Note that Bayesian optimization and Gaussian processes could provide optimal solutions when the training effort is very expensive, which is the case of profiled side-channel analysis. For that, a proper metric needs to be defined in order to correctly judge the performance of a deep neural network for

profiled attacks, as discussed in Section 5.5.

5.7. Different applications of Deep Learning to Side-channel Analysis

The feasibility of deep learning for side-channel analysis goes beyond the design and testing of new threats. In this section, we summarize different applications of deep neural networks on side-channel attacks that differ from using deep neural networks as a supervised, trained classifier.

Deep learning in leakage assessment

To the best of our knowledge, the work [49] was the first to present a version of leakage assessment methodology, where the mathematical foundation involves a function inferred by a learning algorithm. The architecture of the neural network model used could be categorized as a multi-layer perceptron shallow network. Considering its relation with the neural network, we include this work as a different application of deep learning for side-channel. The task of the learning algorithm is set for classification, where two types of classes are aimed to be distinguished. To do so, the acquisition procedure in the methodology remains the same, and the classes are defined regarding the combination of input data, i.e., random class and fixed class. Nevertheless, an extended version using more than two classes is also possible. The evaluator can compose different sets using different values of fixed data and group them by an identifier, creating more than two classes. Although experiments with more than two classes are not conducted in the original work, we could think that by doing so, a modification in the architecture might be required to deal with the overfitting that having more classes could originate.

Leakage detection also involves analyzing where the leak is located graphically. Having computed the statistical moments using even Welch's t-test or Pearson's χ^2-test, a plot pointing out where the spikes exceed a fixed threshold is depicted, showing what operation of the crypto-algorithm is being compromised. This is a particularly easy task for the statistical-based approaches. In the case of the learning algorithm, dealing with detecting the time sample where the leak happens is trickier. Authors dealt with this by using sensitivity analysis [59]. By using this measure, it is possible to backtrack the activated neurons until the most relevant time samples for the classification task appear. Using this measure, it is still not possible to appreciate a p-value that exceeds the threshold where the leakage happens, as is the case for non-neural network approaches. The only information it brings is the time samples where the leakage is located.

A deep neural network as a side-channel distinguisher

Timon in [44] presented the first non-profiled deep learning solution to attack protected AES implementations. The analysis leads to the conclusion that a trained deep neural network is used as a key distinguisher in a non-profiled setting. For that, the authors train an identical deep neural network architecture for each key

byte candidate. Separate training is then conducted based on training traces labeled according to the current key guess. In a divide-and-conquer strategy usually applied to AES, this analysis would require at least 256 training phases to recover a single key byte. The authors demonstrated that the complexity is not too high for some specific targets. However, besides the great contribution offered by this paper, the complexity can easily escalate beyond control if training a single model for a single key byte candidate requires too much time.

DEEP LEARNING AGAINST PUBLIC-KEY IMPLEMENTATIONS

State-of-the-art public-key implementations, such as RSA or ECC-based protocols, are nowadays protected with randomization techniques that make single trace attacks the only feasible side-channel solution. These attacks are commonly referred to as *horizontal attacks*. In [60], the authors proposed a supervised single trace deep learning attack on a real RSA target. Weissbart et al. in [61] applied CNNs to single trace EdDSA implementations based on Curve25519. These two reported applications of deep learning to public-key designs are supervised techniques and require the knowledge of random blinding and secret variables to label the traces. As an example of a realistic scenario application, the adversary would need access to the random number generator in a chip to be able to label the single traces for training purposes. The main difference from the application on symmetric crypto, where class probabilities from multiple traces are combined in a summation probability for each key candidate, classifying single traces resort again to conventional supervised classification metrics to analyze the attack results. In [62], authors proposed a combination of horizontal attack to deep learning techniques. Their proposed framework can break protected public-key implementations with CNNs by assuming that an adversary can provide initial labels (with a large number of errors) to a trace set after applying an unsupervised horizontal attack. In the end, the attack from [62] is kept unsupervised as no knowledge about private key bits is assumed for the whole framework application.

5.8. CONCLUSIONS AND PERSPECTIVES

This chapter provides a general overview of the state of the art in deep learning-based profiled SCA. The main advantages of deep learning against tradicional methods were described. Important discussions about the correct usage of metrics in a deep learning-based attack are addressed, as are the main concepts involving hyperparamater tuning in SCA domain.

The research on deep learning techniques for side-channel analysis still leaves several open questions. The main challenge in profiled SCA is the ability of a trained model to generalize to different devices. In deep learning-based SCA, we suggest that this problem could be addressed with efficient regularization techniques that are able to understand the variations that need to be applied to training traces to improve generalization.

Selecting an efficient metric for deep learning-based profiled SCA still poses difficulties for security evaluators. As conventional deep learning metrics may be

meaningless in scenarios with SCA countermeasures, SCA metrics remain a solution. However, the calculation of guessing entropy or success rate during the training phase can face complexity drawbacks, as these calculations can involve thousands of attack traces. Therefore, interesting research could focus on defining efficient loss functions that are based on SCA paradigms.

Finally, one of the main difficulties found by the SCA community in this domain is to propose a non-profiled attack solution based on training a single deep neural network. Auto-encoders are usually suggested as unsupervised methods. However, their efficiency for non-profiled SCA is still an open research question.

REFERENCES

[1] M. Krček, H. Li, S. Paguada, U. Rioja, L. Wu, G. Perin, and Ł. Chmielewski. "Deep learning on side-channel analysis". In: *Security and Artificial Intelligence: A Crossdisciplinary Approach*. Springer, 2022, pp. 48–71.

[2] P. C. Kocher, J. Jaffe, and B. Jun. "Differential Power Analysis". In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. Ed. by M. Wiener. CRYPTO '99. Springer. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 388–397. ISBN: 3540663479.

[3] 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.

[4] 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. ISBN: 3-540-00409-2. DOI: 10.1007/3-540-36400-5_3. URL: https://doi.org/10.1007/3-540-36400-5%5C_3.

[5] 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*. Ed. by W. Fischer and N. Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 45–68. ISBN: 9783319667867. DOI: 10.1007/978-3-319-66787-4_3. URL: https://doi.org/10.1007/978-3-319-66787-4%5C_3.

[6] L. Wu and S. Picek. "Remove Some Noise: On Pre-processing of Side-channel Measurements with Autoencoders". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.4 (Aug. 2020), pp. 389–415. DOI: 10.13154/tches.v2020.i4.389-415. URL: https://tches.iacr.org/index.php/TCHES/article/view/8688.

[7] 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*. Cryptology ePrint Archive, Report 2018/1023. https://eprint.iacr.org/2018/1023. 2018.

[8] F. Standaert, T. 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*. Ed. by A. Joux. Vol. 5479. Lecture Notes in Computer Science. Springer, Apr. 2009, pp. 443–461. ISBN: 3642010008. DOI: 10.1007/978-3-642-01001-9_26. URL: https://doi.org/10.1007/978-3-642-01001-9%5C_26.

[9] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)*. 1st ed. Springer, 2002. ISBN: 3540425802.

[10] R. L. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-key Cryptosystems”. In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[11] V. S. Miller. “Use of Elliptic Curves in Cryptography”. In: *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85*. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1986, pp. 417–426. ISBN: 0-387-16463-4. URL: http://dl.acm.org/citation.cfm?id=18262.25413.

[12] N. Koblitz. “Elliptic Curve Cryptosystems”. In: *Mathematics of Computation* 48.177 (Jan. 1987), pp. 203–209. ISSN: 0025-5718.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[14] A. Graves, A. Mohamed, and G. Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2013, pp. 6645–6649.

[15] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F. Standaert. “Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis)”. In: *COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers*. Springer. 2015, pp. 20–33.

[16] Z. Martinasek, J. Hajny, and L. Malina. “Optimization of Power Analysis Using Neural Network”. In: *Smart Card Research and Advanced Applications*. Ed. by A. Francillon and P. Rohatgi. Springer. Cham: Springer International Publishing, 2014, pp. 94–107. ISBN: 978-3-319-08302-5.

[17] 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. Vol. 10076. Lecture Notes in Computer Science. Springer. Springer, Dec. 2016, pp. 3–26. ISBN: 978-3-319-49444-9. DOI: 10.1007/978-3-319-49445-6_1. URL: https://doi.org/10.1007/978-3-319-49445-6%5C_1.

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

[19] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli. "Methodology for Efficient CNN Architectures in Profiling Attacks". en. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* Volume 2020 (2019).

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[21] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (Sept. 2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[22] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. June 2016. DOI: 10.1109/CVPR.2016.308.

[24] L. Masure, C. Dumas, and E. Prouff. "Gradient visualization for general characterization in profiling attacks". In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2019, pp. 145–167.

[25] B. Hettwer, S. Gehrer, and T. Güneysu. "Deep Neural Network Attribution Methods for Leakage Analysis and Symmetric Key Recovery". In: *Selected Areas in Cryptography – SAC 2019*. Ed. by K. G. Paterson and D. Stebila. Springer. Cham: Springer International Publishing, 2020, pp. 645–666. ISBN: 978-3-030-38471-5.

[26] D. van der Valk, S. Picek, and S. Bhasin. *Kilroy was here: The First Step Towards Explainability of Neural Networks in Profiled Side-channel Analysis*. Cryptology ePrint Archive, Report 2019/1477. https://eprint.iacr.org/2019/1477. 2019. URL: https://eprint.iacr.org/2019/1477.

[27] O. Choudary and M. G. Kuhn. "Efficient Template Attacks". In: *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*. 2013, pp. 253–270.

[28] G. Fan, Y. Zhou, H. Zhang, and D. Feng. "How to Choose Interesting Points for Template Attacks More Effectively?" In: *Trusted Systems*. Ed. by M. Yung, L. Zhu, and Y. Yang. Cham: Springer International Publishing, 2015, pp. 168–183. ISBN: 978-3-319-27998-5.

[29] C. Archambeau, E. Peeters, F. .-. Standaert, and J. .-. Quisquater. “Template Attacks in Principal Subspaces”. In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by L. Goubin and M. Matsui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–14. ISBN: 978-3-540-46561-4.

[30] J. G. van Woudenberg, M. F. Witteman, and B. Bakker. “Improving differential power analysis by elastic alignment”. In: *Cryptographers' Track at the RSA Conference*. Springer. 2011, pp. 104–119.

[31] R. A. Muijrers, J. G. van Woudenberg, and L. Batina. “RAM: Rapid alignment method”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2011, pp. 266–282.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[33] J. An and S. Cho. “Variational autoencoder based anomaly detection using reconstruction probability”. In: *Special Lecture on IE* 2.1 (2015).

[34] L. Theis, W. Shi, A. Cunningham, and F. Huszár. “Lossy image compression with compressive autoencoders”. In: *arXiv preprint arXiv:1703.00395* (2017).

[35] D. Kwon, H. Kim, and S. Hong. “Improving non-profiled side-channel attacks using autoencoder based preprocessing”. In: *Cryptology ePrint Archive* (2020).

[36] L. Wu and S. Picek. “Remove some noise: On pre-processing of side-channel measurements with autoencoders”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), pp. 389–415.

[37] Z. Martinasek, L. Malina, and K. Trasy. “Profiling power analysis attack based on multi-layer perceptron network”. In: *Computational Problems in Science and Engineering*. Springer, 2015, pp. 317–339.

[38] 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: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by A. Chattopadhyay, C. Rebeiro, and Y. Yarom. Cham: Springer International Publishing, 2018, pp. 157–176. ISBN: 978-3-030-05072-6.

[39] G. Yang, H. Li, J. Ming, and Y. Zhou. “Convolutional Neural Network Based Side-Channel Attacks in Time-Frequency Representations”. In: *Smart Card Research and Advanced Applications*. Ed. by B. Bilgin and J.-B. Fischer. Springer. Cham: Springer International Publishing, 2019, pp. 1–17. ISBN: 978-3-030-15462-2.

[40] 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. URL: `https://eprint.iacr.org/2018/1023.pdf`.

[41] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 0387308571.

[42] 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*. Ed. by M. Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 398–412. ISBN: 978-3-540-48405-9.

[43] L. Goubin and J. Patarin. “DES and Differential Power Analysis (The "Duplication" Method).” In: Jan. 1999, pp. 158–172.

[44] B. Timon. “Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 107–131. DOI: 10.13154/tches.v2019.i2.107-131. URL: https://doi.org/10.13154/tches.v2019.i2.107-131.

[45] V. Mirchevska, M. Luštrek, and M. Gams. “Combining domain knowledge and machine learning for robust fall detection”. In: *Expert Systems* 31.2 (2014), pp. 163–175.

[46] B. Hettwer, S. Gehrer, and T. Güneysu. “Profiled Power Analysis Attacks Using Convolutional Neural Networks with Domain Knowledge”. In: *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*. Ed. by C. Cid and M. J. J. Jr. Vol. 11349. Lecture Notes in Computer Science. Springer, 2018, pp. 479–498. ISBN: 978-3-030-10969-1. DOI: 10.1007/978-3-030-10970-7_22. URL: https://doi.org/10.1007/978-3-030-10970-7%5C_22.

[47] D. Wang, K. Mao, and G.-W. Ng. “Convolutional neural networks and multimodal fusion for text aided image classification”. In: *2017 20th International Conference on Information Fusion (Fusion)*. IEEE. 2017, pp. 1–7.

[48] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation”. In: *PloS one* 10.7 (2015).

[49] F. Wegener, T. Moos, and A. Moradi. “DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations”. In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 505.

[50] K. Simonyan, A. Vedaldi, and A. Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *preprint* (Dec. 2013). ISSN: 00994480. DOI: 10.1080/00994480.2000.10748487. arXiv: 1312.6034. URL: http://arxiv.org/abs/1312.6034.

[51] M. Zeiler and R. Fergus. “Visualizing and Understanding Convolutional Neural Networks”. In: vol. 8689. PART 1. Nov. 2013. ISBN: 9783319105895. DOI: 10.1007/978-3-319-10590-1_53. arXiv: 1311.2901.

[52] 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 (Nov. 2018), pp. 209–237. DOI: 10.13154/tches.v2019.i1.209-237. URL: https://tches.iacr.org/index.php/TCHES/article/view/7339.

[53] G. Perin, L. Chmielewski, and S. Picek. “Strength in Numbers: Improving Generalization with Ensembles in Machine Learning-based Profiled Side-channel Analysis”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.4 (Aug. 2020), pp. 337–364. DOI: 10.13154/tches.v2020.i4.337-364. URL: https://tches.iacr.org/index.php/TCHES/article/view/8686.

[54] 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* 2020.3 (June 2020), pp. 73–96. DOI: 10.13154/tches.v2020.i3.73-96. URL: https://tches.iacr.org/index.php/TCHES/article/view/8583.

[55] G. Perin and S. Picek. “On the Influence of Optimizers in Deep Learning-Based Side-Channel Analysis”. In: *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*. Ed. by O. Dunkelman, M. J. J. Jr., and C. O’Flynn. Vol. 12804. Lecture Notes in Computer Science. Springer, 2020, pp. 615–636. DOI: 10.1007/978-3-030-81652-0_24. URL: https://doi.org/10.1007/978-3-030-81652-0%5C_24.

[56] S. Picek, A. Heuser, and S. Guilley. “Profiling Side-channel Analysis in the Restricted Attacker Framework”. In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 168. URL: https://eprint.iacr.org/2019/168.

[57] J. Bergstra, R. Bardenet, B. Kégl, and Y. Bengio. “Algorithms for Hyper-Parameter Optimization”. In: Dec. 2011.

[58] J. Bergstra and Y. Bengio. “Random Search for Hyper-Parameter Optimization”. In: *The Journal of Machine Learning Research* 13 (Mar. 2012), pp. 281–305.

[59] H. Shu and H. Zhu. “Sensitivity Analysis of Deep Neural Networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 4943–4950. ISSN: 2159-5399. DOI: 10.1609/aaai.v33i01.33014943. URL: http://dx.doi.org/10.1609/aaai.v33i01.33014943.

[60] M. Carbone, V. Conin, M.-A. Cornélie, F. Dassance, G. Dufresne, C. Dumas, E. Prouff, and A. Venelli. “Deep Learning to Evaluate Secure RSA Implementations”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.2 (Feb. 2019), pp. 132–161. DOI: 10.13154/tches.v2019.i2.132-161. URL: https://tches.iacr.org/index.php/TCHES/article/view/7388.

[61] L. Weissbart, S. Picek, and L. Batina. “One Trace Is All It Takes: Machine Learning-Based Side-Channel Attack on EdDSA”. In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by S. Bhasin, A. Mendelson, and M. Nandi. Cham: Springer International Publishing, 2019, pp. 86–105. ISBN: 978-3-030-35869-3.

[62] G. Perin, L. Chmielewski, L. Batina, and S. Picek. “Keep it Unsupervised: Horizontal Attacks Meet Deep Learning”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (Dec. 2020), pp. 343–372. DOI: 10.46586/tches.v2021.i1.343-372. URL: https://tches.iacr.org/index.php/TCHES/article/view/8737.

6

A Comparison of Weight Initializers in Deep Learning-based Side-channel Analysis

The usage of deep learning in profiled side-channel analysis requires a careful selection of neural network hyperparameters. In recent publications, different network architectures have been presented as efficient profiled methods against protected AES implementations. Indeed, completely different convolutional neural network models have presented similar performance against public side-channel traces databases. In this work, we analyze how weight initializers' choice influences deep neural networks' performance in the profiled side-channel analysis. Our results show that different weight initializers provide radically different behavior. We observe that even high-performing initializers can reach significantly different performance when conducting multiple training phases. Finally, we found that this hyperparameter is more dependent on the choice of dataset than other, commonly examined, hyperparameters. When evaluating the connections with other hyperparameters, the biggest connection is observed with activation functions.

This chapter has been published as H. Li, M. Krček, and G. Perin. "A Comparison of Weight Initializers in Deep Learning-Based Side-Channel Analysis". In: *Applied Cryptography and Network Security Workshops*. Ed. by J. Zhou, M. Conti, C. M. Ahmed, M. H. Au, L. Batina, Z. Li, J. Lin, E. Losiouk, B. Luo, S. Majumdar, W. Meng, M. Ochoa, S. Picek, G. Portokalidis, C. Wang, and K. Zhang. Cham: Springer International Publishing, 2020, pp. 126–143. ISBN: 978-3-030-61638-0.

6.1. Introduction

There has been rapid progress in profiled side-channel attacks (SCAs) based on machine learning techniques in recent years. These techniques proved to be very successful by outperforming some of the classical attacks [2, 3], like template attacks [4]. Around a decade ago, machine learning algorithms like SVM [5] and Random Forest [6, 7] represented the standard choice for machine learning-based SCA.

More recently, deep learning-based SCAs started when Maghrebi et al. demonstrated the strong performance of several neural network types, most notably, convolutional neural networks [8]. Despite many successes, there are still many difficulties (and unanswered questions) when training deep neural networks, especially those related to how to tune hyperparameters. This tuning phase can highly influence the model's performance, so it is important to properly address the issue and have a good strategy for selecting the hyperparameters. Hyperparameters are all those configuration variables external to the model, like the number of hidden layers in a neural network. The parameters are the configuration variables internal to the model and estimated from data (e.g., the weights in a neural network).

As there are many hyperparameters, and numerous possible combinations that can be explored, selecting proper hyperparameters can be a very time-consuming process. Researchers commonly approach this problem by selecting the hyperparameters they deem relevant and then conducting a grid search. While such an approach works well (as confirmed by successful attacks on various AES implementations), there are also potential drawbacks. Most notably, grid search skips many possible values while limiting the setup to only certain hyperparameters, completely disregards other hyperparameters' influence. In [9], the authors proposed a methodology to select hyperparameters that are related to the size (number of learnable parameters, i.e., weights and biases) of layers in CNNs. This includes the number of filters, kernel sizes, strides, and the number of neurons in fully-connected layers. In [10], the authors conducted an empirical evaluation for different hyperparameters for CNNs on the ASCAD database. Kim et al. investigated how adding noise to the input (thus, serving as regularization) improves the performance of profiled SCAs [11], which is a technique that can be used with any neural network architecture.

In this work, we focus on the weight initialization strategies for CNNs in SCA, and we explore its influence on the performance of the attacks. Thus, we investigate a hyperparameter, i.e., selecting different weight initializers directly responsible for weights parameter. Our experiments show that most of the weight initializers work well. More precisely, there is a decent selection of weight initializers one can use in deep learning-based SCA and expect good results. Next, our experiments show significant differences concerning key rank results, as within one guessing entropy experiment, it is common to obtain both perfect attack and attack that does not work at all. Interestingly, our results indicate that independent training phases result in significantly different guessing entropy performances. This means that it is not enough to consider only one training experiment, but one must conduct a proper statistical analysis for training and testing phases. We evaluate the evolution of weights and biases concerning the progress of epochs, and we observe most changes

in Convolutional and Batch Normalization layers. In contrast, the fully-connected layers (those responsible for classification) remain almost constant throughout the training phase. Finally, we examine the connection between weight initializers and other hyperparameters, and we determine that the biggest influence comes from the combination of activation functions and weight initializers. This indicates that future experiments should consider both hyperparameters.

6.2. BACKGROUND

6.2.1. SIDE-CHANNEL ANALYSIS

Side-channel analysis is a type of implementation attacks, where instead of attacking the algorithm itself, adversary attacks the physical device that implements the algorithm [12]. Profiled side-channel attacks are the most powerful type of side-channel attacks as they assume that the attacker has access to an identical copy of a device to build a profile. These attacks have two phases, namely, profiling and online attack. The profiling phase is a modeling problem, for which machine learning algorithms perform well. The online phase is the actual attack on a similar device to recover the secret information and is done using the profiling phase's model.

6.2.2. MACHINE LEARNING AND SIDE-CHANNEL ANALYSIS

Machine learning is a subset of artificial intelligence and is based on learning specific patterns from given data. Since this approach is data-driven, it does not require explicit instructions and rules. Therefore, such algorithms work well in modeling problems. Currently, neural networks are a prevalent machine learning technique in SCA, and in our experiments, we investigate deep learning. Deep learning represents methods based on artificial neural networks, and some of the deep learning architectures are multilayer perceptrons (MLPs), recurrent neural networks (RNNs), and convolution neural networks (CNNs). In our experiments, we concentrate on CNNs from [9] and [11]. We opt not to consider MLP as there are less "accepted" MLP architectures in the literature, and the number of hyperparameters is more limited, which makes it possible to include weight initialization in the hyperparameter tuning phase.

To understand weight initializers, we first explain neurons, the base building block of artificial neural networks. Neuron takes input values and calculates the weighted sum using the weight matrix. For a neural network to learn nonlinear functions and models, nonlinear activation functions are applied to the weighted sum. Output of one neuron is described with the equation $y = f(b + \sum_{i=1}^{n} x_i w_i)$, where, the input x is of size n, w are the weights, b the bias and f is the activation function. Bias is also a weight for an input x_0 with an assigned value of 1. The equation takes a form $y = f(\sum_{i=0}^{n} x_i w_i)$ where $x_0 = 1$ and $w_0 = b$. This calculation is done in all neurons of one layer, so we can describe it with matrices, where features of the input samples can be arranged as columns or rows. In Keras, the features are arranged as columns,

and in this setting the equation equals:

$$\mathbf{Y} = \mathbf{X} * \mathbf{W} + \mathbf{B}, \tag{6.1}$$

where **X** is the input, **W** is the weight, and **B** is the bias matrix. The weight matrix of a layer l is a matrix of dimension (size of layer $l-1$, size of layer l), while the bias matrix is (1, size of layer l), with the size of the layer being the number of neurons in the layer. Weight initializers are strategies for setting the initial values of a weight matrix for a neural network layer. Later, in the training phase during back-propagation, the weights in the weight matrix are adjusted with the selected optimization algorithm. Commonly used optimization algorithms are Stochastic Gradient Descent, RMSprop, and Adam [13], which we use in our experiments. Here, we explore different weight initialization strategies and how they impact the performance of deep learning-based SCA.

6.2.3. WEIGHT INITIALIZERS

As mentioned, weight initializers represent how the initial values of a neural network layer's weight matrix are set. It is believed that neural networks are very sensitive to the initial weights [14]. When the deep learning algorithm was first successfully proposed, it was common to initiate weights with Gaussian noise, setting the mean equal to zero, and the standard deviation to 0.01. This way of initializing weights was not enough to train deep neural networks because of problems, such as *vanishing gradients*, *exploding gradients*, or *dead neuron* [14, 15], which significantly hampered its development. In 2010, Glorot and Bengio [16] analyzed the problem systematically and proposed a formula to initialize weights depending on the number of input and output units (neurons). Glorot initializer works well in many cases and is still popular today. In 2015, He et al. [17] put forward that Glorot initializer does not work with well ReLU activation function, and extended the formula to meet ReLU based neural networks through only using the number of input units and increasing the scaling by $\sqrt{2}$. As more people have devoted themselves to the study of weight initialization, various methods have appeared. In general, these methods can be divided into two categories: Zeros and Ones initialization, and Random initialization.

Zeros and Ones Initialization. With all weights initialized to 0 (1), all weights are the same, and the activation in all neurons is also the same. That way, the loss function's derivative is the same for every weight in a weight matrix of a layer. When all weights have the same value, in all iterations, this makes hidden layers symmetric. Every neuron of the layer computes the same function, so the model behaves like a linear model.

Random Initialization. All weight matrix values are set to random numbers, usually from a normal or uniform distribution. As mentioned, issues with random initialization are *vanishing* and *exploding gradients*. In *vanishing gradients*, weight update is minor, which results in slower convergence, while in *exploding gradients*, large gradients can result in oscillation around the optimum.

For deep networks, heuristics can be used to initialize the weights depending on the nonlinear activation function. Heuristics set the normal distribution variance to k/n, where k is a constant value that depends on the activation function, and n is the number of input nodes to the weight tensor or both input and output nodes of the weight tensor. This is adjusted to a uniform distribution, which can be seen in the provided list of initializers from Keras library [18]. While these heuristics do not entirely solve the *exploding/vanishing gradients* issue, they help mitigate it to a great extent. Initializers with explained heuristics are LeCun, Glorot/Xavier, and He initializers.

Different weight initializers available [19] in Keras are listed below with fan_in being the number of input units in the weight tensor and fan_out the number of output units in the weight tensor.

- *Zeros*: initializes weights to 0.
- *Ones*: initializes weights to 1.
- *Constant*: initializes weights to given constant, default is 0.
- *RandomNormal*: initializes weights with normal distribution, $mean = 0$, $stddev = 0.05$.
- *RandomUniform*: initializes weights with uniform distribution, $minval = -0.05, maxval = 0.05$.
- *TruncatedNormal*: similar to *RandomNormal* except that values more than two standard deviations from the mean are discarded and redrawn.
- *VarianceScaling*: adapts scale to the shape of weights, default values are $scale = 1, mode =' fan\_in'$ and normal distribution.
- *Orthogonal*: random orthogonal matrix, default value of multiplicative factor to apply to the matrix is 1.
- *Identity*: identity matrix, multiplicative factor again 1.
- *lecun_uniform*: uniform distribution within [-limit, limit] where limit is $sqrt(3/fan\_in)$.
- *lecun_normal*: truncated normal distribution centered on 0 with $stddev = sqrt(1/fan\_in)$.
- *glorot_normal*: truncated normal distribution centered on 0 with $stddev = sqrt(2/(fan\_in + fan\_out))$.
- *glorot_uniform*: uniform distribution within [-limit, limit] where limit is $sqrt(6/(fan\_in + fan\_out))$.
- *he_normal*: truncated normal distribution centered on 0 with $stddev = sqrt(2/fan\_in)$.
- *he_uniform*: uniform distribution within [-limit, limit] where limit is $sqrt(6/fan\_in)$.

6.3. EXPERIMENTAL SETUP

Algorithms used for these experiments are taken from [11] and [9], where CNN hyperparameters were fine-tuned specifically for each dataset the authors used. We vary available weight initializers in our experiments to investigate the performance difference according to each weight initializer. All of the other hyperparameters are

taken directly from the mentioned works. We consider these two architectures as they represent top-performing architectures from related works. Additionally, they differ in size, which will enable us to evaluate the influence of weight initializers on architectures of different complexity.

We will refer to CNN architecture as the *Noise* architecture for [11], and the *Methodology* architecture for [9]. For each architecture, two leakage models are used: Identity (ID) model [9, 11] and Hamming weight (HW) model [20], in which there are 256 classes and nine classes respectively corresponding to the output of neural networks. In both architectures, hyperparameters are tuned with the ID model (as the original works consider only ID model), but we use the same hyperparameters for the HW model.

Kim et al. [11] used *glorot_uniform* weight initializer, and Zaid et al. [9] used *he_uniform* weight initializer. In the last layer, [9] does not set weight initializer to *he_uniform*, but instead, the default weight initializer is utilized, which is *glorot_uniform*. We are not aware of this implementation's motivation, so in our experiments, we vary weight initializers in all layers, including the last layer with a Softmax activation function. This change causes a difference between our results with *Methodology* architecture and ID leakage model compared to results presented in the work of Zaid et al. [9], as shown later in Section 6.4.

We are not running experiments with *Constant*, *VarianceScaling*, *Identity*, and *Orthogonal* initializers from all available Keras weight initializers. *Identity* and *Orthogonal* initializers are not actively used, and *Constant* and *VarianceScaling* correspond to *Zeros* and *lecun_normal*, respectively, when using default values. We simulate ten times with each initializer and average the results for comparison with other weight initializers.

We use the public source code provided on GitHub by Zaid et al. [9] in Keras with Tensorflow backend [18]. We consider three publicly available datasets that consist of side-channel measurements for the AES cipher for our experiments. Following, we shortly describe these datasets and then discuss the results for each dataset in detail.

DPA contest v4 (DPAv4) dataset[1] is obtained from a masked AES software implementation [21]. Knowing the masked values, this dataset is easily converted into an unprotected scenario. We attack the first round of S-box operation, and identify each trace with $Y^{(i)}(k^*) = Sbox[P_0^{(i)} \oplus k^*] \oplus M$ where $P_0^{(i)}$ is the first byte of the i-th plaintext and M is the known mask.

AES_RD dataset[2] is obtained from an implementation on an 8-bit AVR microcontroller with a random delay countermeasure [22]. This countermeasure shifts each trace following a random variable of 0 to $N^{[0]}$. The attack is on the first round S-box operation, as in DPAv4 dataset, where traces are labeled as $Y^{(i)}(k^*) = Sbox[P_0^{(i)} \oplus k^*]$.

[1] http://www.dpacontest.org/v4/42_traces.php
[2] https://github.com/ikizhvatov/randomdelays-traces

ASCAD dataset[3] is obtained from a masked AES-128 implementation on an 8-bit AVR microcontroller introduced in [23]. The leakage model is the first round S-box operation, such that $Y^{(i)}(k^*) = Sbox[P_3^{(i)} \oplus k^*]$. In contrast to the DPAv4 and AES_RD datasets, the third byte is exploited (as this is the first masked byte).

6.4. EXPERIMENTAL RESULTS

This section shows the results for different weight initializers. We explore 1) how weight initializers impact the performance of the utilized CNN architectures, 2) which one is the best for a specific dataset and architecture, and 3) whether there is the best weight initializer for all datasets. As explained in Section 6.3, we use 11 weight initializers available in Keras and execute experiments on commonly used DPAv4, AES_RD, and ASCAD datasets. For each dataset, we run four experiments: *Methodology* architecture with ID and HW model, and *Noise* architecture with ID and HW model.

Recall, with *Zeros* and *Ones* initialization, the model is no better than a linear model. In our experiments, we still choose to show the results with *Zeros* and *Ones* weight initialization to show that a linear model is not sufficient for considered problems. There, all results show that guessing entropy is either staying at random guessing or increasing with *Zeros* and *Ones* weight initialization. Consequently, when discussing the performance of weight initializers, we usually ignore the performance of *Zeros* and *Ones*, as they never converge.

A good initializer is the one where GE decreases, preferably to zero, in the least number of traces, and is more stable, as observed from results from multiple independent experiments. As such, those weight initializers where GE behaves similarly in multiple experiments, we consider more stable than when this is not true. To get the best weight initializer, we consider two additional metrics: speed and stability. We sort the averaged GE value of all weight initializers to evaluate their "speed", and compare the consistency in multiple experiments to obtain the "stability". The key rank range shows the "best" GE from 10 experiments to present the range from multiple performed attacks. The "best" GE is the one that reaches the lowest value, and if multiple GE results reach the same minimum, then the one that reaches that value with fewer traces is considered better, and we plot key rank range for that experiment. The range is taken from the 100 attacks that are executed for calculating the GE. Weights' evolution figures show weights for each layer, and the layers in the legend are ordered from first input layer to the last output layer of the neural network. We provide a Table 6.1 as an overview of all experiments and best initializers in each setup.

6.4.1. RESULTS FOR THE DPAV4 DATASET

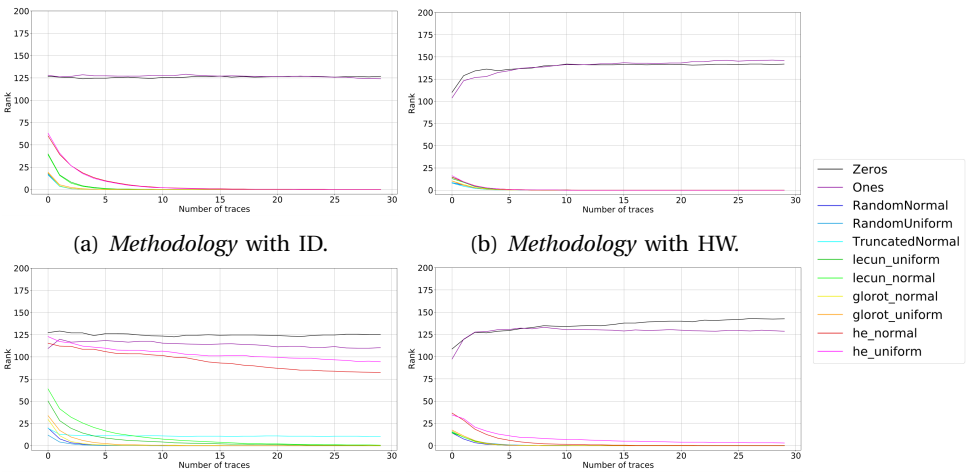
As in [9], we use 4 000 traces for the training set, 500 traces for the validation set, and 500 for attacking the device. Each trace has 4 000 features. The GE rankings

[3] https://github.com/ANSSI-FR/ASCAD

Table 6.1.: An overview of all experiments and best initializers in each setup.

Dataset	Architecture	Best initializer (ID/HW)
DPAv4	*Methodology*	*RandomUniform*
	Noise	*RandomUniform*/*RandomNormal*
AES_RD	*Methodology*	*he_normal*/*lecun_normal*
	Noise	*RandomUniform*
ASCAD	*Methodology*	*he_normal*
	Noise	*lecun_normal*

of the four experiments are shown in Figure 6.1. In the two experiments with the *Methodology* architecture (Figures 6.1a and 6.1b), most weight initializers perform similarly when weight initializer is varied, but *RandomUniform* is slightly faster in convergence and more stable with both leakage models. With the *Noise* architecture and ID leakage model (Figure 6.1c), the best weight initializer is *RandomUniform*, and with the HW model (Figure 6.1d), most weight initializers perform quite well, but we choose *RandomNormal* as the best one.

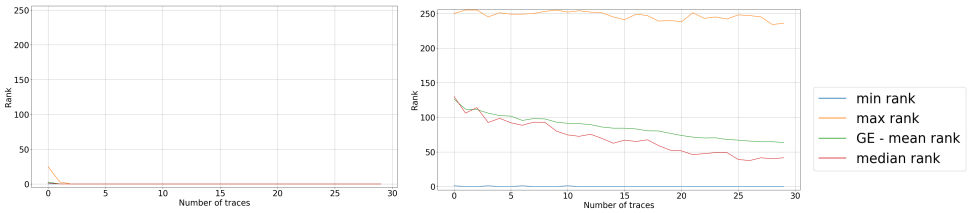


(a) *Methodology* with ID. (b) *Methodology* with HW.

(c) *Noise* with ID. (d) *Noise* with HW.

Figure 6.1.: Averaged GEs for all weight initializers with the DPAv4 dataset.

Figure 6.2 shows the key rank range for the best (Figure 6.2a) and the worst initializer (Figure 6.2b) with the *Noise* architecture and ID model for the DPAv4 dataset when ignoring the *Zeros* and *Ones*. While the GE is slowly converging with he_uniform initializer, in Figure 6.2b, we can see significant differences in the key rank results from multiple performed attacks within one guessing entropy experiment.



(a) *RandomUniform*. (b) *he_uniform*.

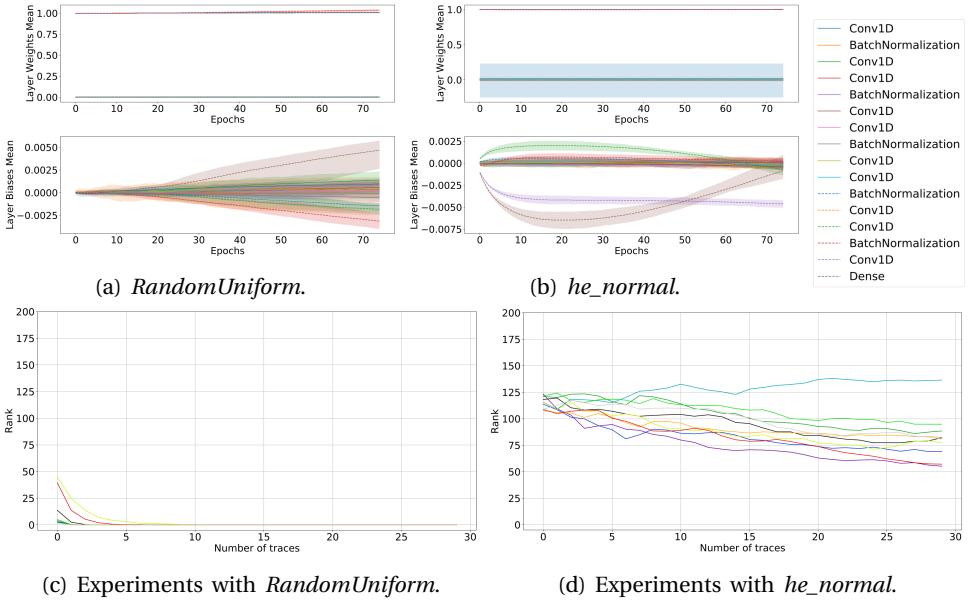
Figure 6.2.: The key rank range of *Noise* architecture with ID model for decreasing GE in DPAv4 dataset.

When looking at the weights' evolution, we observe the change of weights and biases in every neural network layer in every epoch and find that weights and biases change in Convolutional layers and Batch Normalization layers, and other layers such as dense layers do not exhibit much change. In the *Methodology* architecture, both weights, and biases change significantly, while in the *Noise* architecture, only biases change, and weights stay almost constant. According to the result, we can peek into the training processes of the two architectures. The iterative processes of the two architectures are radically different: in the *Methodology* architecture, both weights, and biases are trained, while in the *Noise* architecture, biases are the main training objects. This indicates that the *Noise* architecture is more "robust" as there is not much need for weight improvement to reach strong attack performance. More precisely, there seems to be more weight optima for the *Noise* architecture than for the *Methodology* architecture.

In the weights' evolution for the DPAv4 dataset, the random initializers without heuristics perform best for the *Methodology* ID setting and very similar to Glorot initializers. Weight initializers He and LeCun in this setting performed a bit worse, and their weights' evolution is also similar, but visually different from the weights' evolution of the other initializers. Similar weights' evolution is seen with the HW model.

For the *Noise* architecture, in Figures 6.3a and 6.3b, we show weights' evolution of the best and worst initializer, respectively. It seems as the *he_normal* (Figure 6.3b) could improve with more epochs and reach the performance of, at least, Glorot initializers. Additionally, we show corresponding experiments of the same initializer to show their stability in Figures 6.3c and 6.3d. Here, both are stable: *RandomUniform* is performing well, and *he_normal* consistently has a slow convergence. This is again visible through weights' evolution because the weights and biases' variance is not large. The performance of different weight initializers with both architectures and models on the DPAv4 dataset is quite similar, and most of the initializers reach GE of zero.

Lastly, we simulate experiments with the *Methodology* architecture and both leakage models to explore the influence of the weight initializer in the last fully-connected layer, similar to [9]. More precisely, we keep all hyperparameters of the two experiments except that the setting of the last layer in the neural network



(a) *RandomUniform*. (b) *he_normal*.

(c) Experiments with *RandomUniform*. (d) Experiments with *he_normal*.

Figure 6.3.: Weights' evolution and experiments with *Noise* ID setting on the DPAv4 dataset.

6

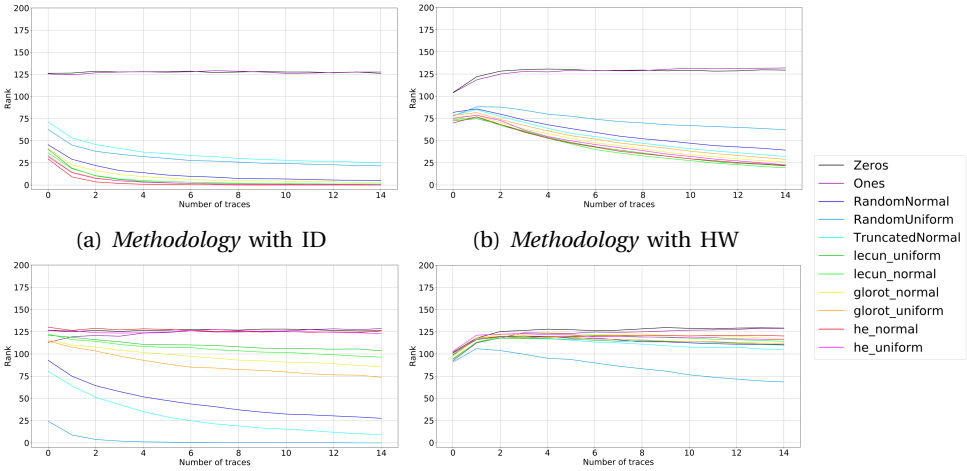
is the same as paper [9]. The results for the two experiments show that it has no impact on the outcome, and the performances of all the weight initializers in the ID and HW model are almost the same.

6.4.2. RESULTS FOR THE AES_RD DATASET

AES_RD dataset is a protected implementation, where adding random delays to the normal operation of AES makes it more difficult to conduct attack as features are misaligned. The dataset consists of 50 000 traces of 3 500 features each, where 20 000 traces are used for the training set, 5 000 for the validation, and 25 000 for the attack set. The GE rankings for the AES_RD dataset are illustrated in Figure 6.4. By observing all weight initializers' speed and stability, we get the best weight initializers in all scenarios: *he_normal*, *lecun_normal*, *RandomUniform*, and *RandomUniform*, respectively.

Like the DPAv4 dataset, weights and biases change mostly in Convolutional layers and Batch Normalization layers, but not in other layers. We can also see that in the *Methodology* architecture, both weight and bias change significantly, while in the *Noise* architecture, only biases change, and weights remain almost constant.

Figures 6.5a and 6.5b display the best and the worst initializer respectively in weights' evolution for the *Methodology* architecture on the AES_RD dataset. The difference in the initializers' performance stems from their stability because all reach GE equal to zero in several of ten simulations, which can be seen in Figures 6.5c

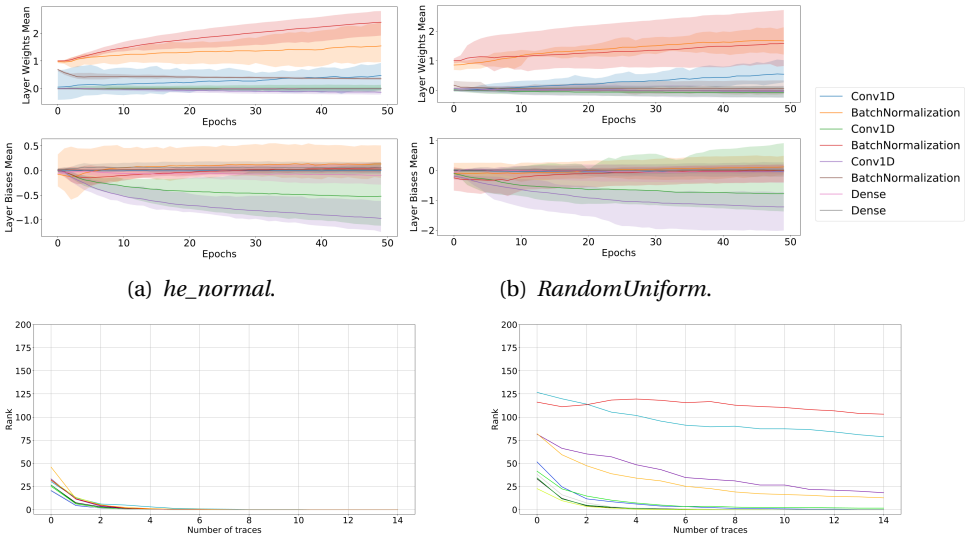


(a) *Methodology* with ID (b) *Methodology* with HW

(c) *Noise* with ID (d) *Noise* with HW

Figure 6.4.: Averaged GEs for all weight initializers with the AES_RD dataset.

and 6.5d. The stability of the weight initializer is also seen in weights' evolution. Since we show the mean of the weights and the range for the ten simulations: the more the weights' evolution varies, the more GE is also likely to vary.



(a) *he_normal*. (b) *RandomUniform*.

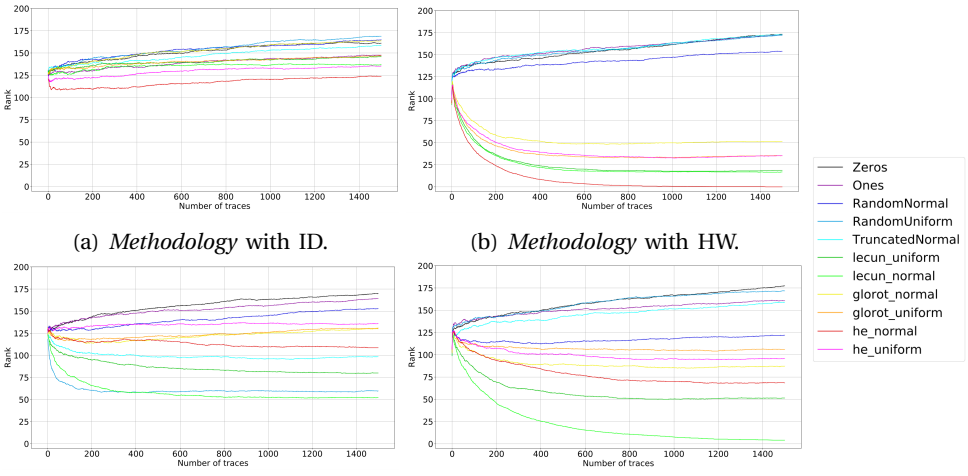
(c) Experiments with *he_normal*. (d) Experiments with *RandomUniform*.

Figure 6.5.: Weights' evolution and experiments with *Methodology* ID setting on the AES_RD dataset.

Finally, we investigate the weight initializer's influence in the last dense layer for the *Methodology* architecture. All hyperparameters are the same, except for the weight initializer in the last layer, which is set as default, according to the settings in paper [9]. The new results show that the change in the last layer also does not have a big effect on the initializer's stability, but it impacts the speed. With the HW model, the convergence for all weight initializers is slower. The best weight initializers for ID and HW model are *he_normal* and *lecun_normal*, respectively.

6.4.3. Results for the ASCAD Dataset

Next, we compare the performance of different weight initializers for the ASCAD dataset. We use the ASCAD dataset with 60 000 traces of 700 features without desynchronization. The dataset is divided into 45 000 training traces, 5 000 validation traces, and 10 000 attack traces. In Figure 6.6, we show the GE rankings. In the experiment with the *Methodology* ID setting (Figure 6.6a), increasing the number of attack traces leads to an increase of the GE for the correct key byte, even with *he_uniform*, which was used in paper [9] in all layers except for the last layer. By comparing the stability, we get that *he_normal* is the best one. We observe that the GE value of weight initializers with heuristics converges to zero with the HW model (Figure 6.6b). *he_normal* is the fastest one. In the setting with the *Noise* architecture (Figures 6.6c and 6.6d), the best weight initializers, *lecun_normal*, can be easily chosen by observing the speed.

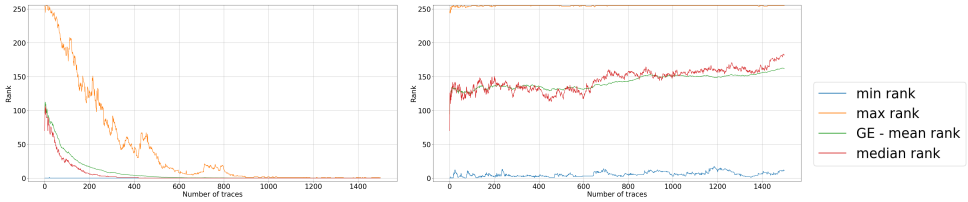


(a) *Methodology* with ID. (b) *Methodology* with HW.

(c) *Noise* with ID. (d) *Noise* with HW.

Figure 6.6.: Averaged GEs for all weight initializers with the ASCAD dataset.

Figure 6.7 shows the key rank range for *he_normal* initializer where GE reached zero (Figure 6.7a), and *RandomUniform* where GE increases with an increased number of traces (Figure 6.7b). Again, we see that even when the GE is increasing, some key rank results are showing perfect attacks.



(a) *he_normal*. (b) *RandomUniform*.

Figure 6.7.: The key rank range of *Methodology* architecture with HW model for decreasing and increasing GE in ASCAD dataset.

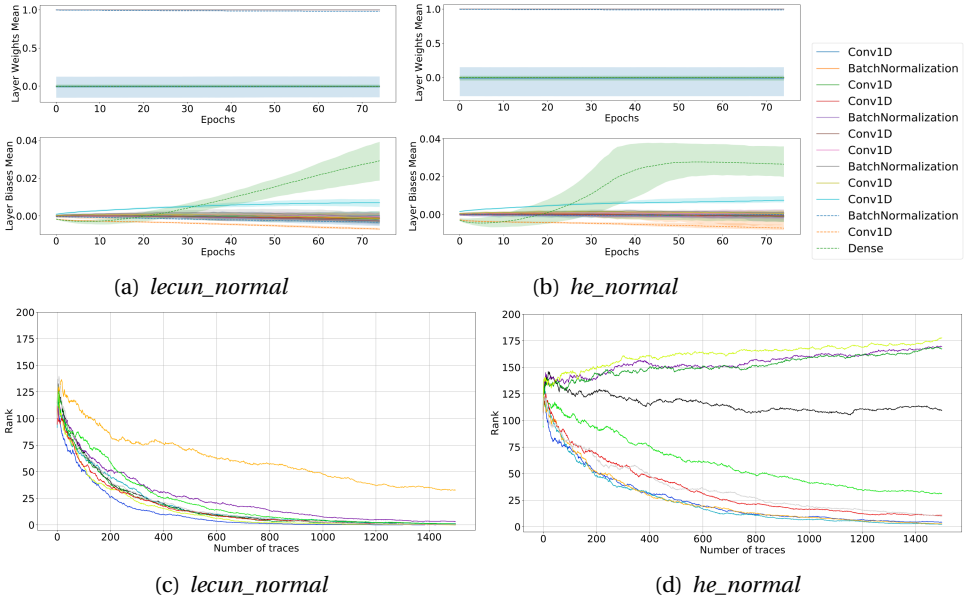
Next, we observe the weights and biases change of every layer throughout the epochs. Like the previous two datasets, weight and bias change mostly in Convolutional layers and Batch Normalization layers, but not in other layers. Once again, it can be seen that in the *Methodology* architecture, both weights and biases change significantly, while for the *Noise* architecture, only biases change and weights are almost constant.

In Figure 6.8, we show the weights' evolution of the best initializer (Figure 6.8a) and average performing one (Figure 6.8b). The corresponding experiments are shown in Figures 6.8c and 6.8d for the *Noise* architecture and the HW model. In these experiments, the worst initializer, *RandomUniform* (see Figure 6.6d), performed similarly to *Zeros* and *Ones*, as in every experiment, GE was increasing.

Finally, to explore the influence of weight initializers in the last layer, we run experiments with the *Methodology* architecture, using all the hyperparameters of the two experiments except the setting of the last layer in the neural network. Like [9], the weight initializer of the last layer is a default one. The new results show that weight initializer has a significant influence on the outcomes. In the experiments with the *Methodology* ID setting, the average GE values of all weight initializers (except *Zeros* and *Ones*) decrease, but there is a difference in the stability of the initializers. The best weight initializer is *he_normal*. With the *Noise* architecture, the average GE values of all weight initializers increase. The best weight initializer is *lecun_uniform*, since, for two out of ten simulations, GE converged to zero.

6.5. WEIGHT INITIALIZER INFLUENCE ON OTHER HYPERPARAMETERS

Based on the best weight initializers that we find to provide better performance for specific neural network architectures and datasets, we now analyze whether a weight initializer's performance depends on its combination with other hyperparameters or if a weight initializer method is connected to the dataset itself. In other words, we wish to understand if the selection of a weight initializers is optimal for a restricted group of hyperparameters or if it is more dependent on the nature of the side-channel traces, meaning that any small variations on hyperparameters would still lead to a successful attack in the majority of tests.



(a) *lecun_normal* (b) *he_normal*

(c) *lecun_normal* (d) *he_normal*

Figure 6.8.: Weights' evolution and experiments with *Noise* HW setting on the ASCAD dataset.

6

We select the *Methodology* convolutional neural network architecture used in the previous sections and make small variations in their hyperparameters to investigate the influence on the best found weight initializer. To do this analysis, we select ASCAD dataset. For this dataset and the *Methodology* CNN architecture, we find that *he_normal* weight initializer provide better results. Table 6.2 shows the ranges of hyperparameters that we vary in different CNN training phases. In total, we train 400 CNNs, and we use the HW leakage model.

Table 6.2.: Hyperparameter variations in the *Methodology* architecture.

Hyperparameters	**Original**	**Minimum**	**Maximum**	**Step**
Filters	4	4	8	1
Kernel Size	1	1	4	1
Neurons	10	5	15	1
Layers	2	2	3	1
Learning Rate	5e-3	1e-3	1e-2	1e-4
Mini-Batch	100	100	400	100
Activation function (all layers)	SELU	ReLU, Tanh, ELU, or SELU		

Figure 6.9 shows that *Tanh* is the only activation function that does not provide successful key recovery in any of the experiments. For the *ReLU*, *ELU* and *SELU* activation functions, the different trained CNNs architectures can return low GE.

Concerning the number of filters in the single convolution layer of this architecture, the usage of four filters tends to maximize the attack's success, as demonstrated in Figure 6.10. Increasing the filter size decreases the probability of the attack to be successful. Similar observations are visible in Figure 6.11 regarding kernel sizes. For kernel sizes 1 and 2, the probability of a successful attack is similar but almost twice higher than with kernel sizes 3 and 4.

Finally, we also observe that making small variations in the number of layers and neurons also does not provide too much effect on the final GE. As shown in Figures 6.12a and 6.12b, more layers, and more neurons tend to provide a subtle increase in the concentration of low GE values. These variations are insufficient to assume that the combination of architecture hyperparameters and weight initializer strictly depends on a specific number of layers and neurons.

We also do not observe a significant effect on the final GE results for different mini-batch sizes (from 100 to 400) and different learning rates (from 0.001 to 0.01). Therefore, this analysis's main conclusion is that the choice of a weight initializer for the *Methodology* CNN architecture (when using the ASCAD dataset with the Hamming weight model), depends mostly on the activation function rather than the rest of hyperparameters. However, for this scenario, a more precise conclusion would be to assume that for a specific dataset (and leakage model), there is an optimal combination of activation function and weight initializer. Weight initializers with heuristics are derived based on certain assumptions on the activation functions. For example, the *Glorot* initializer assumes that the activations are linear. This assumption is not valid for *ReLU* activation functions, so He et al. [17] derived a new initialization method, and it allowed their deep models to converge as opposed to the *Glorot* initialization method. Therefore, we see that weight initializers are closely related to activation functions, which supports our conclusion.

6.6. CONCLUSIONS AND FUTURE WORK

In this paper, we evaluate the influence of the weight initializer choice on the performance of CNNs in the profiled side-channel analysis. We consider 11 weight initializers, three datasets, two leakage models, and two CNN architectures. We evaluate the weight initializer performance by observing guessing entropy, the stability of results, and the evolution of weights through the training process.

Our results show that when the dataset is easy to attack, it is not important what weight initializer to use. Going toward more difficult datasets, we observe more influence stemming from this selection. Interestingly, we see that specific key rank experiments can behave extremely well or extremely badly from the guessing entropy results. What is more, we see significant differences in individual training processes, which means that weight initializers play a significant role in the training process, and it is necessary to run multiple training phases (and not only attacks to obtain guessing entropy). Next, most of the changes in weights happen in the

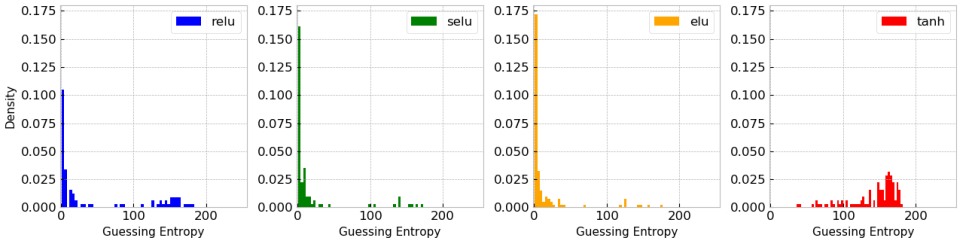


Figure 6.9.: Activation functions and guessing entropy.

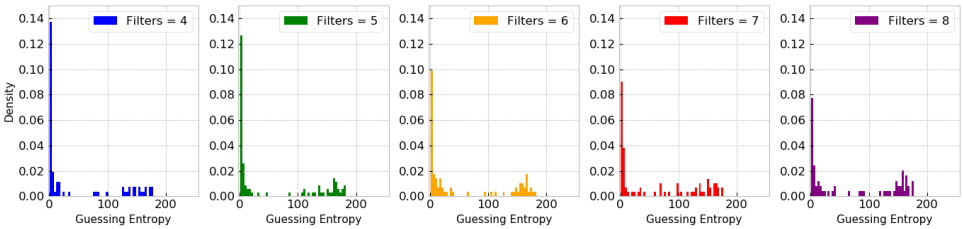


Figure 6.10.: Filters and guessing entropy.

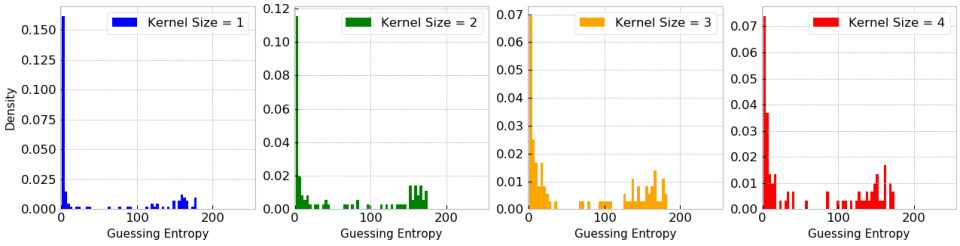
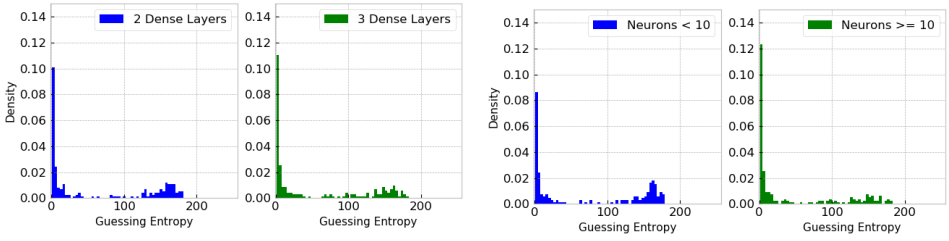


Figure 6.11.: Kernel sizes and guessing entropy.



(a) Layers and guessing entropy.

(b) Neurons and guessing entropy.

Figure 6.12.: Different layers and neurons variations and their relation to final GE results.

Convolutional and Batch Normalization layer, while we observe almost no change in weights in dense layers. Finally, we analyze the interconnection between weight initializers and other hyperparameters. Our results show a strong connection with activation functions and only marginal connection to other commonly explored hyperparameters. This is supported by the fact that the weight initializers with heuristics are designed based on certain properties of activation functions. However, more experiments could further support this observation. Mathematical explanations of weight initialization strategies were out of scope for this work, but this is an interesting and broad research topic that contributes to a deeper understanding of the deep learning models.

For future work, we see two particularly interesting directions. The first one is to explore the influence of weight initializers and activations functions. Indeed, our results indicate that changes in activation functions influence the results from different weight initializers significantly. The second direction is to explore the unsupervised pre-training setup. Results are showing that autoencoders can be used to assign weights to each layer in an unsupervised manner, which helps to guide the learning towards basins of attraction of minima that support better generalization from the training dataset [24].

REFERENCES

[1] H. Li, M. Krček, and G. Perin. “A Comparison of Weight Initializers in Deep Learning-Based Side-Channel Analysis”. In: *Applied Cryptography and Network Security Workshops*. Ed. by J. Zhou, M. Conti, C. M. Ahmed, M. H. Au, L. Batina, Z. Li, J. Lin, E. Losiouk, B. Luo, S. Majumdar, W. Meng, M. Ochoa, S. Picek, G. Portokalidis, C. Wang, and K. Zhang. Cham: Springer International Publishing, 2020, pp. 126–143. ISBN: 978-3-030-61638-0.

[2] L. Lerman, G. Bontempi, and O. Markowitch. “Power Analysis Attack: An Approach Based on Machine Learning”. In: *Int. J. Appl. Cryptol.* 3.2 (June 2014), pp. 97–115. ISSN: 1753-0563. DOI: 10.1504/IJACT.2014.062722. URL: http://dx.doi.org/10.1504/IJACT.2014.062722.

[3] 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*. Ed. by W. Fischer and N. Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 45–68. ISBN: 9783319667867. DOI: 10.1007/978-3-319-66787-4_3. URL: https://doi.org/10.1007/978-3-319-66787-4%5C_3.

[4] 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. ISBN: 3-540-00409-2. DOI: 10.1007/3-540-36400-5_3. URL: https://doi.org/10.1007/3-540-36400-5%5C_3.

[5] A. Heuser and M. Zohner. “Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines”. In: *COSADE*. Ed. by W. Schindler and S. A. Huss. Vol. 7275. LNCS. Springer, 2012, pp. 249–264. ISBN: 978-3-642-29911-7.

[6] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert. “Template Attacks vs. Machine Learning Revisited and the Curse of Dimensionality in Side-Channel Analysis”. In: *Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064*. COSADE 2015. Springer. Berlin, Germany: Springer-Verlag, 2015, pp. 20–33. ISBN: 9783319214757.

[7] 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 (Nov. 2018), pp. 209–237. DOI: 10.13154/tches.v2019.i1.209-237. URL: https://tches.iacr.org/index.php/TCHES/article/view/7339.

[8] 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. Vol. 10076. Lecture Notes in Computer Science. Springer. Springer, Dec. 2016, pp. 3–26. ISBN: 978-3-319-49444-9. DOI: 10.1007/978-3-319-49445-6_1. URL: https://doi.org/10.1007/978-3-319-49445-6%5C_1.

[9] 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 (Nov. 2019), pp. 1–36. DOI: 10.13154/tches.v2020.i1.1-36. URL: https://tches.iacr.org/index.php/TCHES/article/view/8391.

[10] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas. “Deep learning for side-channel analysis and introduction to ASCAD database”. In: *J. Cryptographic Engineering* 10.2 (2020), pp. 163–188. DOI: 10.1007/s13389-019-00220-8. URL: https://doi.org/10.1007/s13389-019-00220-8.

[11] 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*. Cryptology ePrint Archive, Report 2018/1023. https://eprint.iacr.org/2018/1023. 2018.

[12] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 0387308571.

[13] D. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014). arXiv: 1412.6980. URL: https://arxiv.org/pdf/1412.6980.pdf%20%7B%5C%%7D22%20entire%20document%20http://arxiv.org/abs/1412.6980.

[14] A. Y. Peng, Y. Sing Koh, P. Riddle, and B. Pfahringer. “Using Supervised Pretraining to Improve Generalization of Neural Networks on Binary Classification Problems”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim. Cham: Springer International Publishing, 2019, pp. 410–425. ISBN: 978-3-030-10925-7.

[15] S. Koturwar and S. Merchant. “Weight Initialization of Deep Neural Networks(DNNs) using Data Statistics”. In: *CoRR* abs/1710.10570 (2017). arXiv: 1710.10570. URL: http://arxiv.org/abs/1710.10570.

[16] Y. B. Xavier Glorot. “Understanding the difficulty of training deep feedforward neural networks”. In: *Journal of Machine Learning Research* 9 (2010), pp. 249–256. ISSN: 15324435.

[17] K. He, X. Zhang, S. Ren, and J. Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *IEEE International Conference on Computer Vision (ICCV 2015)* 1502 (Feb. 2015). DOI: 10.1109/ICCV.2015.123.

[18] F. Chollet *et al. Keras*. `https://keras.io`. 2015.

[19] Keras. *Layer weight initializers*. `https://keras.io/api/layers/initializers/`.

[20] 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: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by A. Chattopadhyay, C. Rebeiro, and Y. Yarom. Cham: Springer International Publishing, 2018, pp. 157–176. ISBN: 978-3-030-05072-6.

[21] S. Bhasin, N. Bruneau, J.-L. Danger, S. Guilley, and Z. Najm. “Analysis and Improvements of the DPA Contest v4 Implementation”. In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by R. S. Chakraborty, V. Matyas, and P. Schaumont. Springer. Cham: Springer International Publishing, 2014, pp. 201–218. ISBN: 978-3-319-12060-7.

[22] J.-S. Coron and I. Kizhvatov. *An Efficient Method for Random Delay Generation in Embedded Software*. Cryptology ePrint Archive, Report 2009/419. `https://eprint.iacr.org/2009/419`. 2009.

[23] E. Prouff, R. Strullu, R. Benadjila, E. Cagli, and C. Dumas. *Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database*. Cryptology ePrint Archive, Report 2018/053. `https://eprint.iacr.org/2018/053`. 2018.

[24] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. “Why Does Unsupervised Pre-Training Help Deep Learning?” In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 625–660. ISSN: 1532-4435.

7

AUTOENCODER-ENABLED MODEL PORTABILITY IN SIDE-CHANNEL ANALYSIS

Hyperparameter tuning represents one of the main challenges in deep learning-based profiling side-channel analysis. For each different side-channel dataset, the typical procedure to find a profiling model is applying hyperparameter tuning from scratch. The main reason is that side-channel measurements from various targets contain different underlying leakage distributions. Consequently, the same profiling model hyperparameters are usually not equally efficient for other targets. This paper considers autoencoders for dimensionality reduction to verify if encoded datasets from different targets enable the portability of profiling models and architectures. Successful portability reduces the hyperparameter tuning efforts as profiling model tuning is eliminated for the new dataset, and tuning autoencoders is simpler. We first search for the best autoencoder for each dataset and the best profiling model when the encoded dataset becomes the training set. Our results show no significant difference in tuning efforts using original and encoded traces, meaning that encoded data reliably represents the original data. Next, we verify how portable is the best profiling model among different datasets. Our results show that tuning autoencoders enables and improves portability while reducing the effort in hyperparameter search for profiling models. Lastly, we present a transfer learning case where dimensionality reduction might be necessary if the model is tuned for a dataset with fewer features than the new dataset. In this case, tuning of the profiling model is eliminated and training time reduced.

This chapter has been published as M. Krček and G. Perin. "Autoencoder-enabled model portability for reducing hyperparameter tuning efforts in side-channel analysis". In: *Journal of Cryptographic Engineering* (2023), pp. 1–23. DOI: 10.1007/s13389-023-00330-4.

7.1. INTRODUCTION

Hardware and software implementations of cryptographic algorithms may leak unintended and measurable side-channel information such as power consumption, electromagnetic emissions, and execution time. Although mathematically secure, these cryptographic implementations may become vulnerable to side-channel attacks (SCAs). SCA is an implementation attack mainly categorized into direct and two-stage attacks. Direct attacks, also known as non-profiled SCA, mainly consist of simple power analysis [2], differential power analysis [3], and correlation power analysis [4]. These attacks explore the statistical dependency between leaked side-channel information and secret cryptographic keys. Recovering the secret depends on running the attack over all possible key hypotheses through a divide-and-conquer strategy and selecting an efficient statistical distinguisher (e.g., Pearson correlation, difference-of-means, or mutual information). On the other hand, a two-stage or profiling SCA [5] can evaluate the security of a cryptographic implementation by assuming a stronger adversary. Profiling SCA assumes that a potential adversary has an open device (identical to the target one) that provides conditions to learn a profiling model by reprogramming the key and input data to the cryptographic algorithm. Depending on how much knowledge is assumed that the adversary possesses (e.g., source code and access to the secret randomness of the implementation), profiling SCA allows the deployment of worst-case (i.e., white-box) or black-box security assessment.

Countermeasures such as masking and hiding are often considered to mitigate SCA. For twenty years, Gaussian template attacks (GTA) [5] have proven to be theoretically the best option to test the worst-case security of SCA countermeasures [6]. Deep learning (DL) has been widely investigated as an alternative profiling SCA solution in the last few years. The results with real-world datasets have demonstrated that deep neural networks provide several practical advantages in comparison to GTA, such as skipping points-of-interest or feature selection from raw measurements [7, 8], relaxing assumptions about underlying leakage distribution, and being less sensitive to trace desynchronization [9–11]. However, together with large training times, the main open challenge for DL-based SCA is hyperparameter tuning. In [12], the authors suggested that hyperparameter tuning should be taken as one of the adversarial assumptions, together with the number of profiling and attack measurements. However, verifying the correctness and reliability of a DL-based profiling model concerning its hyperparameters is still difficult. Even considering advanced hyperparameter search algorithms [13, 14] cannot guarantee that the obtained best model delivers reliable security assessment.

Hyperparameter tuning is a trade-off between time effort and neural network performance, as there is no proven best way to tune the network in a reasonable time. According to [12], the maximum number of searched DL models should be considered when inferring the target's security. A profiling SCA process that is unbounded in the number of hyperparameter tuning models (or learnability capacity) would be able to deliver reliable security assessment[1]. However, as the

[1]Note that this conclusion only holds, at the moment, for first-order masking schemes with hiding countermeasures (e.g., desynchronization). For high-order masking schemes with or without shuffling

number of searched models is always limited in reality, one would like to optimize the model search process by reducing the hyperparameter tuning efforts by ensuring that a reliable and efficient DL model is always found and trained within available computation bounds. In other words, by applying DL-based profiling SCA, the security evaluator wants to ensure that a successful security assessment (i.e., the one that fails in recovering the secret) results from an SCA-secure implementation instead of a wrong profiling attack. One way to reduce hyperparameter search effort across different targets is to apply preprocessing techniques on raw side-channel measurements, such as points-of-interest selection or dimensionality reduction.

In this paper, we consider only dimensionality reduction because points-of-interest selection tends to be inefficient due to the presence of masking countermeasures in the evaluated datasets. We assume a black-box threat model, i.e., an adversary without access to secret masks during profiling and attack phases. We consider autoencoders for dimensionality reduction, which were already considered in several other applications in SCA [16–18], which we discuss in Section 7.2. Our primary goal is to verify whether efforts can be moved from tuning a profiling deep neural network model to tuning an autoencoder by reusing profiling modes across different datasets. Our main contributions are:

1. We experimentally confirm that the standard reconstruction error metric for autoencoders works well for SCA settings. Moreover, the data encoded with autoencoders stays relevant, where we show that tuning efforts on encoded data are similar to tuning on original traces.

2. We demonstrate that the portability of profiling model hyperparameters is possible. We apply the same best profiling model across different datasets encoded into the same dimension with the obtained best autoencoders. Thus, the same profiling model obtained for one dataset can be utilized for other datasets, which reduces the hyperparameter search effort.

3. We show through transfer learning that our best profiling model can be applied to different datasets, eliminating hyperparameter tuning of the profiling model and reducing training time.

The analysis provided in this paper contributes to making DL-based SCA more practical for security evaluations of cryptographic implementations when protected with the first-order Boolean masking schemes.

The paper is organized as follows. Related work is discussed in Section 7.2. We explain the necessary details on deep learning-based side-channel attacks, autoencoders, and transfer learning, followed by a description of utilized datasets in Section 7.3. Section 7.4 details our experimental setup, explaining the steps of our analysis and several use cases we consider. Experimental results are reported in Section 7.5. We conclude the paper in Section 7.6, shortly discussing possible future work.

and desynchronization, it is still an open question whether deep learning models can deliver reliable worst-case or non-worst-case security assessments (see, for instance, the discussion in [15], Section 6, for the non-worst-case assessments).

7.2. RELATED WORK

Several research papers address the portability issue between profiling and target devices of the same type. For instance, Choudary and Kuhn [19] employed different devices in template attacks (TAs) and utilized Fisher's Linear Discriminant Analysis and Principal Component Analysis to enhance TA performance. Another approach proposed in several papers involves using multiple devices to improve the attack performance [20, 21]. In Cao et al. [22], the authors suggest enhancing the pre-trained model on a profiling set with unlabeled traces from the target. They propose a loss function that consists of the standard classification task and minimizing the distribution discrepancy between data traces. In another work by Cao et al. [23], a method inspired by Generative Adversarial Networks (GANs) is presented, where an encoder replaces the generator to reduce the data discrepancy before the attack. Transfer learning is also proposed to transfer knowledge from the profiling device to the target device. Genevey-Metat et al. [24] investigate several scenarios that differ in the position and type of EM probe, side-channel information, and device samples belonging to the same family. All the aforementioned works discussed portability between profiling and target device samples, while our approach utilizes public data from different devices and acquisitions.

In Thapar et al. [25], the authors use transfer learning to accelerate the attack by reusing a model trained on a different target. However, they fixed their input size for model reuse. In contrast, we do not restrict the input size of the initial model we aim to reuse, as we can achieve effective dimension reduction and latent representation with autoencoders for our target device.

Autoencoders have been used to reduce noise and alignment issues in measurements, improving the performance of traditional and deep learning methods in non-profiling SCA [16]. Similarly, Wu and Picek [17] use autoencoders (denoising autoencoder) as a preprocessing method to remove the traces' noise and enable training with clean data in profiling attacks. Paguada et al. [18] utilize autoencoders to reduce the dimensionality of SCA traces, leading to lower complexity of the profiling phase, reduced computational time, and improved classification performance. Thus, autoencoders have predominantly been employed to enhance attack performance on the same dataset (target) by mitigating countermeasures and reducing the traces' complexity. Similarly, our work employs autoencoders to obtain a reduced latent representation of the given input (SCA trace). However, unlike the related works, our approach utilizes autoencoders to facilitate attacks on different datasets (targets).

7.3. BACKGROUND

7.3.1. DEEP LEARNING-BASED SIDE-CHANNEL ATTACKS

In deep learning-based side-channel attacks (DL-based SCA), the main goal is to train deep neural network parameters θ with training data $\mathscr{D}$ by minimizing a loss function $\mathscr{L}$. Each instance of training data $\mathscr{D}$ consists of a tuple $(\mathrm{x}_i, \mathrm{y}_i)$, where x_i is a one-dimensional vector representing the i-th side-channel measurement (or trace) in a dataset $\mathscr{D}$. The range of i is from 0 to the size of the dataset $|\mathscr{D}|$. The term y_i refers to the label (or class) associated to x_i.

Labeling a dataset requires the definition of a leakage model and a selection function. In SCA, the main leakage models are identity (ID), Hamming weight (HW), Hamming distance (HD), and bit-level models. The identity model refers to the direct value of an intermediate being processed by a cryptographic algorithm, while HW refers to the Hamming weight of such intermediate. The HD model returns the Hamming weight from the xor between two intermediate variables. Bit-level models usually consider the most or least significant bit from an intermediate variable. The intermediate variable is defined according to a key-dependent selection function that usually returns an intermediate byte from the cryptographic algorithm. For the case of AES encryption, this intermediate for the i-th trace $\mathtt{x}_i$ could be an $\mathtt{S\text{-}Box}$ output byte in the first encryption round, i.e., $\mathtt{y}_i = \mathtt{S\text{-}Box}(\mathtt{d}_j \oplus \mathtt{k}_j)$, where $\mathtt{d}_j$ and $\mathtt{k}_j$ are the j-th plaintext and key bytes ($j \in [0,15]$ for a key size of 128 bits), respectively.

From the training set $\mathscr{D}$, we select a subset $\mathscr{V}$ to validate the trained model. This model is later tested on a separate dataset $\mathscr{A}$ collected from the attacked device that we refer to as the attack set. Since the goal is to obtain the secret key from $\mathscr{A}$ (or a single byte of the key), we use guessing entropy (GE) [26] to assess the attack performance. The best possible neural network model is the one that requires minimal attack complexity, which is measured in terms of the minimum number of attack traces that are necessary to successfully recover the key [27].

To compute GE, we first predict the validation or attack set and obtain class probabilities $\mathtt{p}_{i,\mathtt{y}_i}$ for each trace i. As labels $\mathtt{y}_i$ are derived from a key-dependent selection function, we obtain the log-likelihood l_k of a certain key byte $\mathtt{k}_j \in [0,255]$:

$$l_k = \sum_{i=0}^{N_a-1} \log \mathtt{p}_{i,\mathtt{y}_i}, \tag{7.1}$$

where N_a is the number of traces in the predicted set. This process is then repeated for all possible key byte hypotheses. Each hypothesis will define different labels $\mathtt{y}_i$ for each trace. The key rank of the correct key $\mathtt{k}^*$ is obtained by sorting all l_k values and by returning the position of l_{k^*} associated with the correct key byte k^*. The GE of the correct key, $\mathtt{ge}^*$, is given by an empirical process in which we repeat the key rank process multiple times (each time with a different and randomly selected subset from the attack or validation set). We obtain an average log-likelihood or key guessing vector g and get the average position of the correct key $\mathtt{k}^*$ inside g. When $\mathtt{ge}^* = 1$, we say that the model successfully recovers the key with N_a attack traces. The minimum number of traces to retrieve the key is referred to as $N_{\mathtt{ge}^*=1}$.

Although the primary goal of training a deep neural network in the SCA context is to minimize $N_{\mathtt{ge}^*=1}$, the models in this paper are still trained with a categorical cross-entropy loss function. In [28], the authors showed that minimizing this loss function is aligned with minimizing $N_{\mathtt{ge}^*=1}$.

7.3.2. AUTOENCODERS (AES)

Autoencoder (AE) is a specific self-supervised neural network used for data compression, dimensionality reduction, generating new data, denoising, etc. The authors in [29] first used autoencoders for dimensionality reduction. Different

autoencoders, such as denoising or variational autoencoders, are described in [30, 31]. We use autoencoders for dimensionality reduction to learn, in an unsupervised manner, an informative smaller representation of the data. We consider deep autoencoders since they are often better than shallow or linear counterparts. While variational autoencoders are very popular, they are more helpful in generating new data, which is different from our goal here.

Autoencoders usually have an encoder and decoder part. The encoder takes the original input and learns a function that encodes the data into a representation given by a latent space. In dimensionality reduction, the input dimension is reduced in latent space. That middle layer is known as the "bottleneck layer", as it holds the data's compressed representation. Later, we use the decoder function to reconstruct the original input from the encoded data. Both encoder and decoder are neural networks, commonly symmetrical, having the same type and number of layers with the same layer sizes.

The objective function of the autoencoder is minimizing the difference between input and output by preserving the relevant information. The compressed data is evaluated by the decoder's ability to reconstruct the original input from the compressed data, so the common metric is Mean Squared Error (MSE). The output for autoencoders is the input itself, so MSE is calculated with

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\mathrm{x}_i - \hat{\mathrm{x}}_i)^2, \tag{7.2}$$

where x_i is the original observation and $\hat{\mathrm{x}}_i$ its reconstruction, while m is the number of inputs (samples). In SCA, x_i is the side-channel trace with n features, for which the distance from $\hat{\mathrm{x}}_i$ is again MSE. Therefore, we do not use labels as in profiling models and do not need to use any leakage model. In this work, we search for the best autoencoders, following the information from [31] for defining the hyperparameter tuning space.

7.3.3. TRANSFER LEARNING

Transfer learning (TL) in machine learning focuses on transferring knowledge across domains and aims to leverage knowledge from a related domain to improve learning in a new task (target domain). The success of transfer learning depends on many factors, such as the relevance between the source and target domains and the learner's (model's) capacity to find transferable and valuable knowledge across the two domains. Transfer learning can be categorized based on the feature space between the two domains and the availability of the labels. More information on categorizations of TL is found in surveys, e.g., [32–34].

Our case belongs to inductive transfer learning, where we have labels for both the source and target domains (different intermediate values belonging to a specific dataset). We aim to achieve high performance in the target task. There are many approaches to transfer learning, and they depend on what we aim to transfer. In our case, we use parameter-based TL to transfer knowledge at the model/parameter level. We use models trained on one dataset and use them for different datasets. Our

main objective is to obtain accurate predictions in the target domain for the new task. Specifically, we train the model to learn the correct key k^* of another dataset. We do it with parameter sharing so that we have a neural network for the source task, and we share (freeze) most of the layers and fine-tune the last few layers to obtain a network that works for the targeted task. We keep the first layers since the first layers in deep neural networks appear not to be specific to particular datasets or tasks [35].

7.3.4. DATASETS

We describe three datasets that are used in our experiments. For all datasets, we use 5 000 traces for validation and another 5 000 traces as the attack set in both profiling attacks and autoencoders. We use 3 000 traces randomly chosen from that 5 000 in each key rank calculation to calculate GE.

DPACONTEST V4.2

DPAcontest v4.2 dataset (here referred as `DPAv4.2`)[2] is the second implementation available in the DPAcontest v4 [36]. It is an improved version implemented in software on an 8-bit Atmel ATMega-163 smart card and corrects several leaks identified in its previous generation. This dataset represents the power consumption of the first AES encryption round, and the AES implementation is protected with Rotate Shift countermeasure. The dataset contains a total of 80 000 traces, and each of them contains 1 704 402 sample points. In our experiments, we trim the dataset to the interval representing the processing of the 13-th `S-box` byte, resulting in 2 000 samples per trace. The first interval ranges from sample 305 000 to 315 000 from original measurements. We apply the resampling process with a resampling window of 10 and step of 5, resulting in 2 000 samples per measurement. We use 70 000 traces for training (which contains 14 different keys).

ASCAD

ASCAD dataset[3] with a fixed key (`ASCADf`), along with ASCAD dataset with a random key (`ASCADr`), consists of measurements from masked AES on the 8-bit ATMega8515 MCU target without any specific hiding countermeasures activated on the target [37]. For `ASCADf` dataset, the key is fixed for all measurements. We have 50 000 training traces with 700 features per trace. `ASCADr` dataset corresponds to the second campaign with the same target and setup as in `ASCADf`. However, in this setting, the key is variable for 66% of the measurements. We use 200 000 training traces with 1 400 features per trace.

[2] `https://www.dpacontest.org/v4/42_doc.php`
[3] `https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1`

7.4. EXPERIMENTAL SETUP

In this section, we provide details about our experimental setup. The process starts with a hyperparameter search to find the best autoencoders for different datasets. Before that, we verify that the MSE metric is appropriate as it keeps the side-channel leakage in the reconstructed traces. Then we verify if searching for profiling neural network models remains similar when we train the models with the encoded datasets. We compare the attack performance of profiling models trained with encoded and original datasets. That is necessary to validate that encoded data stays relevant without worsening tuning efforts. Next, we reused profiling models' hyperparameters across multiple datasets as it was shown that tuning encoded data is equal to tuning original datasets. We consider portability from encoded data to other encoded data and from original to encoded data. The first case enables universal models where all datasets are represented in a similar latent space. The second case addresses the portability of architecture between different feature spaces. Finally, we explore transfer learning advantages utilizing autoencoders and profiling models. To summarize, we apply the following steps:

1. Search for the best latent space size for all datasets based on two datasets.
2. Search for the best autoencoders with the lowest Mean Squared Error (MSE) by setting the best found latent space size.
3. Compare the performance of profiling models when trained with original and encoded traces of the datasets.
4. Investigate the portability of best profiling model hyperparameters trained with an *encoded dataset* to other *encoded datasets*. All datasets are encoded into the same latent dimension by using best-found autoencoders.
5. Investigate the portability of the best profiling model hyperparameters trained with an *original dataset* to other *encoded datasets*. The concept is used when a new dataset has more features than the original dataset. The new dataset is encoded into the same dimension as the original dataset using best-found autoencoders.
6. Investigate transfer learning of best profiling model trained with an *original dataset* to other *encoded datasets* encoded into the same dimension using the best autoencoders.

The overall structure of our experimental setup and the corresponding steps are shown in Figure 7.1. Additionally, the source code is publicly available[4].

7.4.1. AUTOENCODER ARCHITECTURES

We consider the following CNN and MLP autoencoder structures:

- `ae_cnn`: autoencoders with convolution layers.

[4]The code is available at `https://github.com/marinakrcek/AutoEncodersDLSCA`.

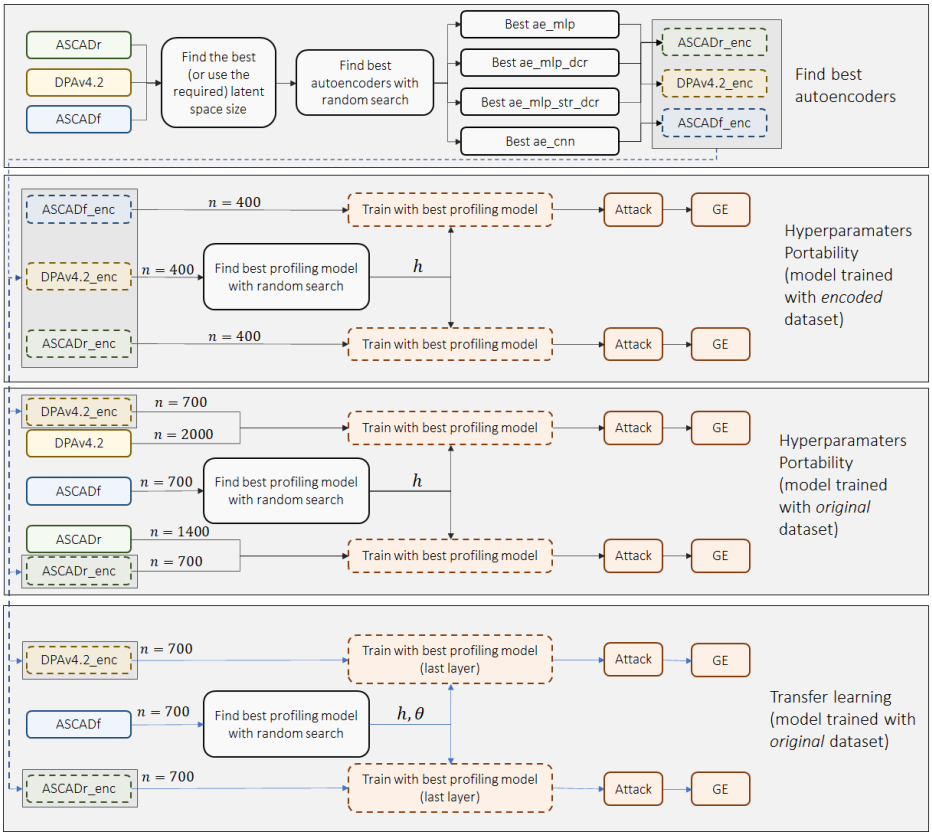


Figure 7.1.: Experimental setup. The term n refers to the number of features in datasets. We denote h as the set of architecture hyperparameters and θ as trainable parameters (weights and biases) in portability cases.

- `ae_mlp`: autoencoders given by symmetric encoder and decoder blocks, in which all layers have the same number of neurons. Latent size can be smaller, equal, or larger than the number of neurons in previous layers.
- `ae_mlp_dcr`: autoencoder with decreasing number of neurons in subsequent layers (with possible repetition). We do not ensure that the layer before the latent space is strictly larger (or equal) to the latent dimension.
- `ae_mlp_str_dcr`: autoencoder with decreasing number of neurons in subsequent layers in the encoder. Here, *_str_dcr* stands for strictly decreasing. The latent size is smaller than the number of neurons in the previous layer. However, the cases where we still use the same number of neurons in layers before the latent layer are possible.

We have several options for MLP autoencoders, while the usual, most common

choice is `ae_mlp_str_dcr`. A decreasing number of neurons in the encoder and symmetrical decoder are commonly chosen because, intuitively, decreasing the number of neurons forces generalization and seems useful for dimensionality reduction. The real benefit of this structure is possibly lower computation costs compared to alternatives. However, as in classification, other options can be explored. Thus, we test the possibilities mentioned above where the number of neurons is not consistently decreasing, and the latent size is not strictly following the decreasing pattern.

In described autoencoder types, the encoder and decoder with MLP structure are always symmetrical, which means that the number of layers is the same in both encoder and decoder blocks. Also, the layer sizes are symmetrically decreasing in the encoder while increasing in the decoder. While common, this is again not strictly defined and can be explored. Intuition again says it makes the most sense for the decoder to follow a reverse structure from its encoder counterpart, but other possibilities can be similarly capable of good performance. For this setting, we keep the traditional symmetrical design.

CNN autoencoder uses similar convolutional blocks to those reported in [31]. Specifically, we use a convolutional layer followed by a pooling layer in the encoder. While they specified Max pooling, we allow both Max and Average pooling in hyperparameter selection. For the decoder, we use upsampling followed by a standard convolutional layer. Since there are more options, the convolutional autoencoder (ConvAE) structure is more complex to define than the MLP autoencoder. We observe in the literature versions of ConvAE increasing and decreasing the number of filters while kernel size and pooling size remain the same. In some cases, kernel sizes were changing. Thus, there is no specific best way to structure the ConvAE. In our case, we increase the number of filters in the encoder because the kernel size and pooling reduce the number of features, sometimes to only one. Thus, having more filters in those deeper layers ensures that after flattening, we have more than one neuron before the last fully-connected layer. We increase the number of filters per layer following the expression $nb\_filters \cdot 2^i$, where i is $the\ order\ of\ the\ layer + 1$. In the decoder, with the combination of upsampling and standard convolutional layer, upsampling increases the number of features, while kernel size again decreases it. Thus, we keep the same expression for increasing the number of filters. Both the encoder and decoder end with a flattened layer followed by a fully-connected layer with the number of neurons equal to the latent size in the encoder and input size in the decoder.

In this work, we tested different structures of MLP autoencoders. At the same time, more analysis should be done for the CNN autoencoder, as the described structure is one of many possibilities. We leave this exploration on CNN structures for future work.

7.4.2. AUTOENCODER METRIC ANALYSIS

Autoencoders for dimensionality reduction imply finding a reduced representation of input data through a latent space. To assess the quality of the reduction and obtained latent representation, the most common error metric is Mean Squared

Error (MSE):

$$MSE = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\mathtt{x}_{ij} - \hat{\mathtt{x}}_{ij}\right)^2, \tag{7.3}$$

where $\mathtt{x}_{ij}$ is the j-th feature value of i-th original side-channel observation and $\hat{\mathtt{x}}_{ij}$ its reconstruction. m is the number of traces (inputs), and n is the number of features in the side-channel trace. Minimizing the MSE leads to a good reconstruction of the original input. To verify whether minimizing MSE is meaningful for SCA traces, we quantify if the leakage is still preserved in the reconstructed traces by calculating the Signal-to-Noise Ratio (SNR). SNR is computed as a leakage assessment of side-channel measurements according to a pre-selected intermediate variable. Since evaluated datasets in this paper were collected from first-order masking AES implementations, first-order intermediate values (such as $\mathtt{S\text{-}Box}(\mathtt{d}_j \oplus \mathtt{k}_j)$) show no significant leakages. Thus, we compute SNR to verify the occurrence of leakages for the masked S-Box output intermediate values[5], i.e., $v = \mathtt{S\text{-}Box}(\mathtt{d}_j \oplus \mathtt{k}_j) \oplus \mathtt{m}_i$, where $\mathtt{m}_i$ is the mask of the i-th trace. For that, we compute the mean and variance side-channel traces for a group of traces represented by a specific intermediate variable $v \in [0, 255]$:

$$\mu_v = \frac{1}{N_v}\sum_{i=0}^{n_v-1}\mathtt{x}_i^v \tag{7.4}$$

$$\sigma_v^2 = \frac{1}{N_v - 1}\sum_{i=0}^{N_v-1}\left(\mathtt{x}_i^v - \mu_v\right)^2, \tag{7.5}$$

where N_v is the number of side-channel traces represented or labeled with intermediate variable v. Next, we obtain the mean vector from all 256 variance vectors σ_v^2:

$$\mu_\sigma = \frac{1}{256}\sum_{v=0}^{255}\sigma_v^2 \tag{7.6}$$

and the variance of mean vectors μ_v:

$$\sigma_\mu = \frac{1}{255}\sum_{v=0}^{255}\left(\mu_v - \mu_\sigma\right)^2. \tag{7.7}$$

Finally, SNR is given by:

$$SNR = \frac{\sigma_\mu}{\mu_\sigma}. \tag{7.8}$$

The SNR from Eq. (7.8) results in a vector with the same length as side-channel traces. We compute this vector for original and reconstructed traces. Then, we take the maximum SNR peak obtained with original traces and subtract it from the value on that exact location in the SNR obtained from reconstructed traces. In the result figures, we refer to this as `SNR diff`.

[5] Although our profiling attacks in later sections are all executed in a black-box manner, here we assume the knowledge of the masks only to assess if MSE metric is consistent.

7.5. EXPERIMENTAL RESULTS

7.5.1. AUTOENCODERS SEARCH

In this section, we deploy a random search to find the best latent space size for autoencoders based on experiments with two datasets. After defining the best latent space size, we deploy a random search to find the best autoencoder architecture for MLP and CNN-based structures. We also obtain the best autoencoder types. Datasets are then encoded with these best autoencoders. Finally, we deploy another random search to find the best profiling model trained with the encoded datasets. We include the third dataset later for portability experiments while also evaluating how well the decisions made on two other datasets for latent size and autoencoder type apply to new datasets.

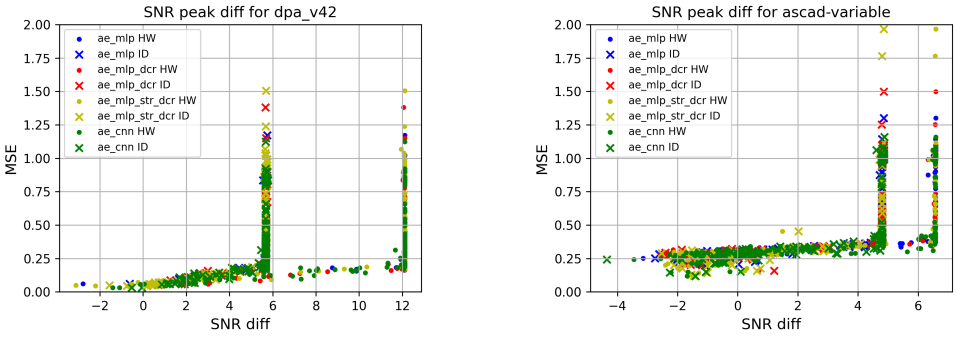
ASSESSING MSE METRIC WITH SNR

We conduct a random search on autoencoders using MSE as the loss function for achieving good reconstruction from the latent space. In these experiments, we consider SNR to verify that minimizing MSE is a meaningful objective when tuning autoencoder hyperparameters. Here, we are not searching for the best latent space size, so we fix the latent dimension to 100 features in all cases to evaluate the MSE metric.

Hyperparameter search space for all autoencoder types are listed in Tables B.1 and B.2 in Appendix B.1. We randomly search for 20 models with each of the four autoencoder types and calculate the SNR difference, as described before, between SNR vectors obtained from original and reconstructed traces. The analysis is conducted for the Hamming weight and identity leakage models with $v = \texttt{S-Box}(\texttt{d}_j \oplus \texttt{k}_j) \oplus \texttt{m}_i$ as the intermediate variable to compute SNR.

Results are shown in Figure 7.2 for the `DPAv4.2` and `ASCADr` datasets. The x-axis in Figures 7.2a and 7.2b shows the maximum peak SNR value difference between the original and reconstructed traces. The corresponding MSE value for each autoencoder is on the y-axis. These figures show that MSE increases as the SNR peak difference increases regardless of the autoencoder type and leakage models in SNR calculations. The vertical lines occur when the reconstructed traces result in insignificant SNR peak values, indicating that side-channel leakages concerning $v = \texttt{S-Box}(\texttt{d}_j \oplus \texttt{k}_j) \oplus \texttt{m}_i$ are not preserved in the reconstructed traces. Negative SNR difference values on the x-axis indicate that the reconstructed trace has a higher SNR value than the original trace on the same sample point, which means that the corresponding autoencoder is preserving and even amplifying the occurrence of side-channel leakages concerning v. However, MSE is not created to lead the autoencoder (AE) to amplify any such SNR peak, as it gets minimized by correct reconstruction of the trace without amplifications.

Next, we consider Pearson correlation coefficient ρ to test whether there is a positive (linear) correlation between MSE and SNR differences. Indeed, from the results in Table 7.1, we see a high correlation until the vertical lines (maximum difference in the SNR values). Specifically, for both datasets, the correlation is stronger for MSE below 0.5. For the `DPAv4.2` dataset, the maximum correlation is



(a) `DPAv4.2` dataset. (b) `ASCADr` dataset.

Figure 7.2.: Relation between MSE and SNR difference.

for MSE values below 0.25, and in the case of `ASCADr`, below 0.5. Therefore, we conclude that minimizing MSE is a meaningful objective error function to optimize autoencoder models for the given datasets.

Table 7.1.: Pearson correlation coefficient ρ and p-value for testing non-correlation. LM stands for leakage model.

Dataset	MSE	LM	ρ	p-value
`DPAv4.2`	>0	HW	0.12	2.32e-03
		ID	0.12	2.67e-03
	<0.5	HW	0.76	8.85e-68
		ID	0.76	1.62e-65
	<0.375	HW	0.83	1.03e-80
		ID	0.83	4.35e-78
	<0.25	HW	**0.93**	**2.04e-78**
		ID	**0.93**	**1.32e-80**
`ASCADr`	>0	HW	0.03	4.84e-01
		ID	0.03	5.01e-01
	<0.5	HW	**0.87**	**3.92e-112**
		ID	**0.86**	**6.58e-106**
	<0.375	HW	0.68	1.05e-34
		ID	0.70	3.10e-37
	<0.25	HW	-0.10	5.16e-01
		ID	0.01	9.37e-01

SEARCHING FOR THE BEST LATENT SPACE SIZE

In this section, we use random search to compare different latent space sizes. The latent sizes we consider are 20, 40, 50, 100, 200, 250, 400, and 500 for all autoencoder types except that for the `ae_mlp_str_dcr`, we do not use the latent size 500 as we also limit the search to 400 neurons per layer (see Table B.1). By choosing

these latent sizes, we ensure that the bottleneck layer in the autoencoder is always smaller than the input layer (which contains the same number of units as the input side-channel trace dimension). The datasets evaluated in this section contain 2 000 features (`DPAv4.2`) and 1 400 features (`ASCADr`), which is significantly larger than the chosen latent space sizes given by the bottleneck layer. Hyperparameter search space (Tables B.1 and B.2) is the same as in the metric analysis provided in the previous section. The hyperparameters for the autoencoders are chosen at random, and we train 20 autoencoder models per latent size, dataset, and autoencoder type combination. The total number of autoencoder combinations in this search is 62.

The main idea here is to verify if a specific latent space size tends to provide the lowest MSE among the searched ones regardless of the dataset and autoencoder (AE) type. Considering our two datasets and four autoencoder types, we have eight cases, each testing eight or seven latent sizes. Autoencoder type `ae_mlp_str_dcr` does not use latent size 500, which leads to trying seven latent sizes instead of eight. We apply the following procedure to obtain the best latent size:

1. For each of these 62 combinations, we extract the autoencoder (out of 20) with the lowest MSE for that dataset, autoencoder type, and latent size.

2. For each autoencoder type and dataset combination, we rank the latent sizes based on the best (lowest) MSE. The latent size for the model with the lowest MSE gets the rank 1 being the best one.

3. For each of the latent space sizes, we average these eight ranks coming from the dataset-autoencoder (AE) type combination.

Table 7.2 shows the average ranks of the latent sizes. The left side of the table is for AE types except for `ae_mlp_str_dcr` as that one must have a decreasing structure, and with the latent size of 500, we cannot achieve that as we limited our number of neurons to a maximum of 400. On the right side of Table 7.2, we order all latent sizes according to the average rank, except for 500, and include the results from `ae_mlp_str_dcr`.

Following, we use the Friedman test [38] across all autoencoder types and the two datasets (`ASCADr` and `DPAv4.2`). This test determines whether there is a statistically significant difference between the means of three or more groups in which the same subjects appear in each group. In our case, groups are based on latent sizes, and subjects are dataset-AE type combinations. The comparison is based on the lowest MSE obtained. Friedman test calculates test statistic Q using the ranks from the samples in the groups. Q value has to be greater than the critical value of Q for a selected significance level α to reject the null hypothesis. Commonly, significance level α of 0.05 works well [39]. We determine the critical value from the Chi-Square distribution table with $k-1$ degrees of freedom where k is the number of groups and selected significance level α. The p-value is the probability of obtaining test results at least as extreme as a result observed under the assumption that the null hypothesis is correct. The null hypothesis for the Friedman test is that the mean of the groups is the same. A very small p-value means such an extreme observed

outcome would be very unlikely under the null hypothesis. The null hypothesis can be rejected if the p-value is below α.

We report the Friedman test results at the bottom row in Table 7.2. Since the p-value is below 0.05, we conclude that the difference between the mean values of the groups (latent sizes) is statistically significant. Additionally, the test statistic Q on the left part is greater than the critical value of 14.07 for the degree of freedom 7. On the right side, the test statistic is greater than the critical value 12.59 for $\alpha = 0.05$ and degree of freedom 6. We perform the Nemenyi post-hoc test to determine which groups have different means. The results are shown in Appendix B.2 in Tables B.5 and B.6 corresponding to the cases without and with `ae_mlp_str_dcr` model. The values in the tables are p-values where if the value is below 0.05, the two groups (column-row combination) have statistically significantly different means. The lowest MSE values for models with latent sizes 400 and 200 differ significantly from models with lower latent sizes (20, 40, and 50). For latent sizes 100 and 250, the difference is significant compared only to latent size 20 in the case with the `ae_mlp_str_dcr` model. Thus, we select latent sizes 200 and 400 to find the best autoencoders using a random search.

Table 7.2.: Average ranks for each latent space size.

Latent space size	w/o `ae_mlp_str_dcr`	Latent space size	with `ae_mlp_str_dcr`
400	1.5	400	1.375
200	1.83	200	2
500	3.67	250	3.375
100	4	100	3.5
250	4.3	50	5.25
50	6.33	40	5.75
40	6.67	20	6.75
20	7.67		
Q: 35.17, p-value: 1.04e-5		Q: 40.66, p-value: 3.38e-7	

SELECTING THE BEST AUTOENCODERS

After we found the best latent size for the `ASCADr` and `DPAv4.2` datasets, we randomly search for additional 80 autoencoder models to obtain a total of 100 models for latent space sizes of 200 and 400. The hyperparameter search space for each autoencoder type stays the same as in the search for the best latent size. From these 100 autoencoder models, we select the best autoencoder for each dataset. Table 7.3 shows the MSE of the best autoencoder for each of the given latent sizes (200 or 400) and autoencoder types per dataset. We see that for `ASCADr`, latent size 400 always results in a lower MSE. For `DPAv4.2`, `ae_mlp` and `ae_mlp_dcr` had better results with 200 features in latent space. However, we can conclude that the best

autoencoder types are `ae_cnn` and `ae_mlp_str_dcr`, with the lowest MSE in both datasets obtained using latent size 400.

Table 7.3.: Best autoencoders. The rows are sorted based on the MSE, so the latent size order in rows is not fixed.

Dataset	AE type	Latent size	MSE
DPAv4.2	ae_mlp	200	0.053842735
		400	0.068908036
	ae_mlp_dcr	200	0.053061113
		400	0.071744457
	ae_mlp_str_dcr	400	0.033211511
		200	0.042860519
	ae_cnn	400	**0.026164241**
		200	0.057221718
ASCADr	ae_mlp	400	0.177198691
		200	0.198677342
	ae_mlp_dcr	400	0.133906147
		200	0.194515368
	ae_mlp_str_dcr	400	0.121792312
		200	0.196267218
	ae_cnn	400	**0.118710345**
		200	0.209023259

Since we initially only allow up to 400 neurons per layer, with a latent space size of 400, the autoencoder `ae_mlp_str_dcr` type could not create a bottleneck architecture with decreasing number of neurons in consecutive layers of the encoder. Thus, we repeated the random search for another 100 models for this autoencoder type by allowing layers with 500 and 600 neurons. Table 7.4 shows the minimum, mean, median, and maximum MSE found in 100 models for the two datasets. Note that this table shows MSE results when the autoencoder contains layers with 400 neurons and MSE results when layers can include 400, 500, and 600 neurons.

Table 7.4.: Autoencoder `ae_mlp_str_dcr` with latent space size of 400. Values are calculated on MSE from a random search of 100 different models, and the number of neurons represents the allowed values from the hyperparameter search space.

Dataset	Nb. neurons	min	mean	median	max
DPAv4.2	400	**0.03321**	**0.5295**	**0.3889**	**3.68**
	400, 500, 600	0.03373	0.8548	0.4529	27.67
ASCADr	400	0.12179	**0.6028**	**0.4701**	**2.01**
	400, 500, 600	**0.12107**	0.7988	0.4832	7.71

For `DPAv4.2` dataset, an autoencoder with up to 400 neurons per layer results in a lower MSE than when we allow 400, 500, and 600 neurons per layer. The hyperparameters of the best models for both cases are in Table 7.5. Note that this

architecture has one hidden layer with 400 neurons between the input layer and the layer with the specified latent size, and the decoder is symmetrical. When allowing 500 and 600 neurons in the random search, the best-found autoencoder has an architecture with two hidden layers with 400 neurons in the encoder and decoder. They also differ in batch size, activation function, learning rate, and weight initialization, but the optimizer is the same. The best model in the second case is not using a larger number of neurons in the first layers.

The autoencoder for the ASCADr dataset has a lower minimal MSE when the number of neurons includes 500 and 600 neurons in the random search. However, the ability to use a larger number of neurons in layers closer to the input layer was not utilized. In both cases for ASCADr dataset, the best model has the same architecture: one hidden layer with 400 neurons for the encoder and decoder and a bottleneck layer with 400 neurons. They differ in activation function and weight initialization, while batch size, learning rate, and optimizer are the same. As the best autoencoders have the same architecture (layers and neurons) for both datasets, the slight difference in the performance comes from the other hyperparameters.

In general, the results in Table 7.4 indicate that a random search only including the option of 400 neurons per layer delivers better MSE values than when we allow more neurons per layer. Mean, median, and maximum MSE are always lower when only 400 neurons are permitted. These results are confirmed for both datasets.

Table 7.5.: MLP autoencoders for DPAv4.2 and ASCADr with latent space size of 400 with different allowed numbers of neurons in layers.

Nb. neurons	Hyperparameters	MSE
	DPAv4.2	
400	ae_mlp_dpav42_best: architecture: [400], batch_size: 200, activation: tanh, learning_rate: 0.0001, weight_init: he_normal, optimizer: Adam	0.03321
400, 500, 600	architecture: [400, 400], batch_size: 100, activation: elu, learning_rate: 0.001, weight_init: glorot_normal, optimizer: Adam	0.03373
	ASCADr	
400	ae_mlp_ascadr_best: architecture: [400], batch_size: 100, activation: elu, learning_rate: 0.0001, weight_init: random_uniform, optimizer: RMSprop	0.12179
400, 500, 600	architecture: [400], batch_size: 100, activation: selu, learning_rate: 0.0001, weight_init: random_normal, optimizer: RMSprop	0.12107

Based on these results, the best autoencoder we use in further experiments is the MLP autoencoder for the DPAv4.2 dataset with an MSE of 0.03321. For the ASCADr dataset, we use the MLP autoencoder with an MSE of 0.12179. The autoencoder with the CNN structure achieves even better MSE - with 0.026 for DPAv4.2, and 0.1187 for ASCADr. The hyperparameters for the best ae_cnn autoencoders are in Table 7.6. We use these four autoencoders in the further experiments, which are denoted ae_mlp_dpav42_best, ae_mlp_ascadr_best, ae_cnn_dpav42_best, and ae_cnn_ascadr_best.

Table 7.6.: Best ae_cnn autoencoders for DPAv4.2 and ASCADr with latent space size of 400.

Dataset	Hyperparameters	MSE
DPAv4.2	ae_cnn_dpav42_best: batch_size: 200, filters: 16, kernel_size: 10, strides: 5, pool_size: 2, pool_strides: 2, pooling_type: Avg, conv_layers: 1, activation: tanh, learning_rate: 0.0001, weight_init: random_normal, optimizer: Adam	0.02616
ASCADr	ae_cnn_ascadr_best: batch_size: 200, filters: 16, kernel_size: 20, strides: 5, pool_size: 2, pool_strides: 2, pooling_type: Avg, conv_layers: 1, activation: tanh, learning_rate: 0.001, weight_init: random_uniform, optimizer: Adam	0.11871

7.5.2. ARE ENCODED DATASETS AS GOOD AS ORIGINAL DATASETS?

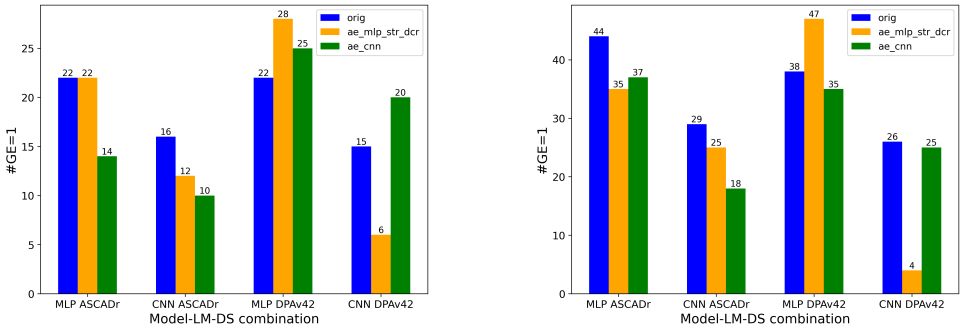
After defining the best ae_mlp_str_dcr and ae_cnn autoencoder structures for DPAv4.2 and ASCADr datasets, we investigate if the encoded datasets can keep relevant leakage information when they are considered as training and attack datasets.

We run a random search to find different MLP and CNN profiling models, and we compare the search performance using original and encoded traces obtained from the best autoencoders listed in Tables 7.5 and 7.6. The hyperparameter search space for the profiling models is shown in Table B.3. Training, validation, and attack sets are labeled with S-box$(d_2 \oplus k_2)$ (third S-box output byte in first AES encryption round[6]) for ASCADr and S-box$(d_{12} \oplus k_{12})$ (13-th S-box output byte in the first AES encryption round) for DPAv4.2. We consider the Hamming weight (HW) and identity (ID) leakage models. We search for 100 models for each combination of the leakage model and profiling model type (MLP-ID, MLP-HW, CNN-ID, and CNN-HW). We measure how many out of the random 100 models reach $\mathrm{ge}^* = 1$ for a given number of validation traces. With that information, we compare if we can more easily obtain a good model using original or encoded traces within the same hyperparameter search space. If we can get a similar amount of models out of 100 that reach $\mathrm{ge}^* = 1$ with original and encoded traces, it means that encoded traces preserve enough information and can be used for training profiling models in SCA. Results for both datasets are shown in Figure 7.3.

From the results, we see that out of 100 models, for the ASCADr dataset, only in the case with MLP and the ID leakage model we obtained the same number of models with $\mathrm{ge}^* = 1$. Looking at the number of traces $N_{\mathrm{ge}^*=1}$, using original traces on average 1 462.8 traces are necessary, while for the dataset encoded with ae_mlp_ascadr_best, we need on average $N_{\mathrm{ge}^*=1} = 1205.9$ traces. For other attack setups, using original traces led to more models with $\mathrm{ge}^* = 1$. However, using encoded data was not much worse.

On the other hand, for DPAv4.2, in three out of four attack settings, we obtained more models with $\mathrm{ge}^* = 1$ when using traces encoded with ae_mlp_dpav42_best or ae_cnn_dpav42_best. Those cases are the MLP profiling model with both leakage models and the CNN profiling model with the ID leakage model. The result with the CNN profiling model and HW leakage model is again close in performance for

[6]Note that we start counting from byte index 0.



(a) ID leakage model.

(b) HW leakage model.

Figure 7.3.: Results on tuning effort using original and encoded traces. We compare the number of models reaching $ge^* = 1$ out of 100 trained models with different hyperparameters selected using random search.

encoded data with `ae_cnn_dpav42_best` and original traces.

We use the Friedman test on the eight scenarios presented in Figure 7.3. We want to see if there is a statistical difference between training on encoded and original traces. We obtain a test statistic of 2.7742 and a p-value of 0.2498. Thus, there is no significantly better setup based on the number of models reaching $ge^* = 1$ out of 100 runs. The initial hypothesis for Friedman is that there is no statistically significant difference in the mean of these numbers. Since the p-value, in this case, is not below 0.05, all the setups lead to similar performance. To conclude, using original traces is not statistically significantly better than using encoded data, meaning that encoded data preserves relevant features that can be used in a profiling attack.

7.5.3. THE PORTABILITY OF PROFILING MODELS

In this section, we verify the efficiency of the best profiling model (obtained in the previous section) when found through hyperparameter tuning with one dataset but concerning different datasets. We include a third dataset to show portability from one to two other datasets. This way, we can answer the following question: *can we move effort from profiling model tuning into autoencoder tuning to reuse the same profiling model across multiple encoded side-channel datasets?*

PORTABILITY OF ENCODED-DATA TRAINED PROFILING MODEL TO DIFFERENT ENCODED DATASETS

We start by verifying the portability of a best-found profiling model trained with an encoded dataset concerning other encoded datasets. That is possible because we encode all datasets into the same encoding dimension, i.e., all encoded datasets contain an equal number of features. We take the best MLP and CNN profiling models trained with encoded `DPAv4.2` dataset for both leakage models. Their attack performance is shown in Tables 7.7 and 7.8, while their hyperparameters

are presented in Tables B.7 and B.8 in Appendix B.3, respectively. We test the performance of those architectures on the encoded ASCADr and ASCADf datasets.

Table 7.7.: Best MLP and CNN profiling models obtained for the encoded DPAv4.2 dataset when encoded with the best-found ae_mlp_dpav42_best.

Model	LM	ge*	$N_{ge^*=1}$
MLP	ID	1	3
	HW	1	29
CNN	ID	1	65
	HW	1	819

Table 7.8.: Best MLP and CNN profiling models obtained for the encoded DPAv4.2 dataset when encoded with the best-found ae_cnn_dpav42_best.

Model	LM	ge*	$N_{ge^*=1}$
MLP	ID	1	2
	HW	1	16
CNN	ID	1	3
	HW	1	18

For ASCADr, we already have the best autoencoders with the latent size of 400 (see Tables 7.5 and 7.6 for hyperparameters and MSE). We additionally train autoencoders ae_mlp_str_dcr and ae_cnn for the ASCADf dataset to encode it to 400 features per trace as well. The hyperparameters range to find the best ae_cnn autoencoder with the random search are the same as considered for the DPAv4.2 and ASCADr datasets (see Table B.2). To find the best ae_mlp_str_dcr autoencoder for the ASCADf dataset, we again consider the random search settings shown in Table B.1. However, we allow the number of neurons per layer for a latent size of 400 to be [400, 500, 600, 700]. The hyperparameters for best autoencoders ae_mlp_ascadf_best and ae_cnn_ascadf_best for ASCADf with latent size 400 are reported in Table 7.9.

Table 7.10 shows the attack performance of the best CNN and MLP profiling architectures on DPAv4.2 from Tables 7.7 and 7.8 when trained with the encoded ASCADr and ASCADf datasets. The results in this table indicate the number of times (out of 100) that the profiling model reaches $ge^* = 1$ for each scenario (leakage model, profiling model type, and autoencoder type). Before the training, we performed standardization on the encoded datasets. Standardization is a typical preprocessing method before training in the SCA and other domains. However, later we also test without standardization to observe the effects.

Table 7.9.: Best autoencoders for ASCADf with latent space size of 400.

AE type	Hyperparameters	MSE
	ASCADf	
ae_mlp_str_dcr	ae_mlp_ascadf_best: architecture: [400, 700], batch_size: 200, activation: tanh, learning_rate: 0.0001, weight_init: random_normal, optimizer: Adam	0.01245
ae_cnn	ae_cnn_ascadf_best batch_size: 200, filters: 16, kernel_size: 20, strides: 10, pool_size: 4, pool_strides: 2, pooling_type: Avg, conv_layers: 1, activation: tanh, learning_rate: 0.0001, weight_init: he_uniform, optimizer: Adam	0.01380

The results with encoded ASCADr indicate superior performance compared to results obtained with the encoded ASCADf. Performance with the encoded ASCADr for MLP with the identity leakage model when this architecture was found with ae_cnn_dpav42_best-encoded DPAv4.2 dataset is slightly worse with finding 19 and 51 models out of 100 reaching $ge^* = 1$. As mentioned, the results with encoded ASCADf are not good, specifically for cases with ASCADf encoded with ae_mlp_ascadf_best. The poor performance might come from the fact that the features in encoded data do not share comparable features despite the equal latent size. We observe that the architecture of that autoencoder is different from the architectures for the other two datasets. To improve these results for encoded ASCADf, and since for ASCADf we allowed more than 400 neurons per layer (which was not the case for other datasets), we again train 100 ae_mlp_str_dcr autoencoders for ASCADf but with only 400 neurons per layer and latent size 400. This way, the autoencoder architecture will be more similar to autoencoders of other datasets. The resulting features also become more comparable, which could improve the performance. The hyperparameters of the best autoencoder for this case are in Table 7.11.

The results using the ae_mlp_ascadf_best-encoded ASCADf from the described search are in Table 7.12. Here, we see an improvement, which indicates our hypothesis on the similarity of latent representations with DPAv4.2 and ASCADr could be true. Accordingly, this is a crucial remark to consider if universal models are to be considered. As the feature space is more similar, the portability becomes easier. Despite the MSE being slightly worse than before, the attack performance is better since the representations are more comparable.

After improving ae_mlp_ascadf_best, we also test best MLP and CNN profiling architectures from Tables 7.7 and 7.8 without data standardization. The results are in Table 7.13 and show that for ASCADr, we have similar successful behavior in comparison to results from Table 7.10 when data standardization was done. For encoded ASCADf, we compare results with Table 7.12 for ae_mlp_ascadf_best as that autoencoder was used for encoding as it was shown to be better. Additionally, we compare it with Table 7.10 for ae_cnn_ascadf_best. Results with and without standardization for ASCADf are also similar.

Our analysis demonstrates that reusing profiling models trained on an encoded dataset is possible. That reduces hyperparameter tuning efforts when considering new encoded datasets, where the effort is moved to tuning the autoencoder.

Table 7.10.: Portability results with best MLP and CNN models obtained with the encoded DPAv4.2 datasets (from ae_mlp_dpav42_best and ae_cnn_dpav42_best). Datasets ASCADr and ASCADf are encoded with their respective best autoencoders. In these results, before training, we use standardization. The training is done 100 times, and the reported number is the number of times we reach $ge^* = 1$.

Model LM		encoded DPAv4.2 with ae_mlp_dpav42_best		encoded DPAv4.2 with ae_cnn_dpav42_best	
		ae_cnn _*_best	ae_mlp _*_best	ae_cnn _*_best	ae_mlp _*_best
encoded ASCADr					
MLP	ID	100	100	19	51
	HW	97	97	89	73
CNN	ID	79	65	99	100
	HW	98	100	100	99
encoded ASCADf					
MLP	ID	35	24	17	0
	HW	0	0	10	5
CNN	ID	97	5	30	0
	HW	16	0	66	1

7

Table 7.11.: Best ae_mlp_str_dcr autoencoder for ASCADf with latent space size of 400, allowing only 400 neurons per layer.

Hyperparameters	MSE
ae_mlp_ascadf_best: architecture: [400], batch_size: 100, activation: selu, learning_rate: 0.0001, weight_init: he_normal, optimizer: RMSprop	0.01345

Table 7.12.: Results with using encoded ASCADf from ae_mlp_ascadf_best with only 400 neurons per layer. We use standardization of the encoded dataset when training the profiling model 100 times. The number represents the number of times we reach $ge^* = 1$.

Model LM		Best model for DPAv4.2 encoded with	
		ae_mlp_dpav42_best	ae_cnn_dpav42_best
MLP	ID	69	37
	HW	0	2
CNN	ID	100	100
	HW	51	17

Table 7.13.: Portability results with best MLP and CNN models obtained with encoded DPAv4.2 datasets (from ae_mlp_dpav42_best and ae_cnn_dpav42_best). Datasets ASCADr and ASCADf are encoded with their respective best autoencoders. We use encoded data directly, without standardization. The training is done 100 times, and the reported number is the number of times we reach $ge^* = 1$.

Model LM		encoded DPAv4.2 with ae_mlp_dpav42_best		encoded DPAv4.2 with ae_cnn_dpav42_best	
		ae_cnn _*_best	ae_mlp _*_best	ae_cnn _*_best	ae_mlp _*_best
encoded ASCADr					
MLP	ID	75	100	0	73
	HW	98	93	100	87
CNN	ID	87	83	100	100
	HW	97	100	100	99
encoded ASCADf					
MLP	ID	0	83	100	11
	HW	12	3	0	2
CNN	ID	98	100	43	100
	HW	29	85	1	86

Additionally, universal profiling architecture is then something we can consider on autoencoder-encoded data. Moreover, tuning autoencoders is easier as optimization of MSE is more straightforward.

PORTABILITY OF ORIGINAL-DATA TRAINED PROFILING MODEL TO DIFFERENT ORIGINAL AND ENCODED DATASETS

In this section, we test the portability of a best-found profiling model architecture (from random search) when it is trained on an original (i.e., not encoded) dataset. For that, we consider ASCADf, which contains 700 features. We made this choice because ASCADf has fewer features than ASCADr and DPAv4.2, which contain 1 400 and 2 000 features, respectively, in their original versions. In this case, to reuse that architecture, we need to decrease the number of features of other datasets to the size of the data used in training. However, since the input layer is a dedicated first layer in neural networks, to reuse the architecture, we can also replace that first layer. In that case, we can keep the original number of features of the new datasets. Our goal is to verify if the best-found profiling architecture with ASCADf also provides good attack performance when trained with the encoded and original ASCADr and DPAv4.2 datasets. Therefore, we have three cases per dataset - using original and encoded data with two different AE types.

Since this time we have to encode ASCADr and DPAv4.2 into 700 features, we

again run a hyperparameter search to find the best autoencoders, which are reported in Table 7.14 with their corresponding MSE values. The hyperparameter search spaces are shown in Tables B.2 and B.4. Table 7.15 shows the results with the best

Table 7.14.: Autoencoders for DPAv4.2 and ASCADr with latent space size of 700.

AE type	Hyperparameters	MSE
	DPAv4.2	
ae_mlp_str_dcr	ae_mlp_dpav42_best_700: architecture: [1400], batch_size: 100, activation: tanh, learning_rate: 0.0001, weight_init: he_normal, optimizer: Adam	0.017
ae_cnn	ae_cnn_dpav42_best_700: batch_size: 200, filters: 16, kernel_size: 10, strides: 5, pool_size: 4, pool_strides: 2, pooling_type: Avg, conv_layers: 1, activation: elu, learning_rate: 0.001, weight_init: random_uniform, optimizer: Adam	0.013
	ASCADr	
ae_mlp_str_dcr	ae_mlp_ascadr_best_700: architecture: [900], batch_size: 200, activation: selu, learning_rate: 1e-05, weight_init: random_uniform, optimizer: RMSprop	0.052
ae_cnn	ae_cnn_ascadr_best_700: batch_size: 400, filters: 8, kernel_size: 20, strides: 5, pool_size: 2, pool_strides: 4, pooling_type: Avg, conv_layers: 1, activation: elu, learning_rate: 0.001, weight_init: he_uniform, optimizer: RMSprop	0.141

MLP and CNN profiling architectures found for the original ASCADf dataset. We ran a random search for 100 models using hyperparameter search space from Table B.3. Hyperparameters for the models with results presented in Table 7.15 can be found in Table B.9.

Table 7.15.: Best profiling models for ASCADf.

Model	LM	ge*	$N_{ge^*=1}$
MLP	ID	1	151
	HW	1	1476
CNN	ID	1	265
	HW	1	1734

Since we reuse only the architecture and not the trained parameters (weights and biases), we modify the input layer to use the original ASCADr and DPAv4.2 datasets that have more features than the original ASCADf. This way, we take the best architectures from Table 7.15 and train them with the original ASCADr and DPAv4.2 datasets as well as with their encoded versions by using the best-found autoencoders listed in Table 7.14. For each dataset, profiling model architecture, and leakage model, we run 100 trainings and compare the number of times the model reaches $ge^* = 1$. The analysis is also done with and without data standardization.

The results in Table 7.16 show that best-found architecture provides good performance even if we use directly original traces from the DPAv4.2 and ASCADr datasets. However, with DPAv4.2, the best-found CNN architectures are less successful. For the encoded DPAv4.2 dataset, results are better than original traces

as it leads to either similar performance or often better. With the dataset encoded with ae_cnn_dpav42_best_700, we got better results without standardization, and for the encoded dataset from ae_mlp_dpav42_best_700, it was better using standardization.

Using original traces was already very successful for the ASCADr dataset, so using encoded data is less valuable, but still shows good performance when the dataset is encoded with ae_cnn_ascadr_best_700, especially with standardization. On the other hand, using ae_mlp_ascadr_best_700 encoded data usually resulted in worse outcomes. Considering the standardization of encoded data, we see that it was slightly beneficial to use standardization for data encoded with both ae_mlp_ascadr_best_700 and ae_cnn_ascadr_best_700 encoded cases. Statistically, however, we cannot claim that it is always necessary to use standardization. On the other hand, based on our results, models trained with encoded data perform similarly or better in most experiments than those trained with original data. In Table 7.16, the cases where the performance is worse are marked in red color. Thus, we conclude that using encoded data to reuse the profiling attack architecture trained with other datasets' original traces can be done despite different feature spaces. Moreover, encoded data is beneficial when the performance with the original data is unsuccessful. Hyperparameter tuning for new datasets can be significantly reduced in that way. Again, tuning is more straightforward for autoencoders by minimizing MSE and does not require the typical attack phase in classification with GE calculations.

Table 7.16.: Results with DPAv4.2 and ASCADr using attack architecture trained on the ASCADf dataset. The numbers represent the number of times we reach a GE of 1 when the training is done 100 times.

Model	LM	orig.	w/o stand. ae_cnn _*_best _700	w/o stand. ae_mlp _*_best _700	with stand. ae_cnn _*_best _700	with stand. ae_mlp _*_best _700
DPAv4.2						
MLP	ID	48	100	100	100	100
	HW	100	100	100	100	100
CNN	ID	0	48	1	0	100
	HW	4	96	77	100	95
ASCADr						
MLP	ID	25	9	0	75	66
	HW	100	100	99	100	100
CNN	ID	95	100	0	100	0
	HW	99	100	4	98	0

7.5.4. TRANSFER LEARNING WITH PROFILING MODELS TO DIFFERENT ENCODED DATASETS

We also test the benefit of autoencoders in the context of transfer learning. Again, we have the same best profiling models for the `ASCADf` dataset (Table 7.15), and we retrain the last layer to obtain the secret key byte for the new dataset. In this case, the input must be the same size since we also use the trained parameters (weights and biases). Therefore, we encoded the `ASCADr` and `DPAv4.2` datasets for transfer learning to the profiling model input size. Additionally, we again test with and without standardization of the encoded data. Since the training is faster as we train only one layer, we have a setting where we train one by one epoch, calculating the GE after each epoch and stopping when we reach a $ge^* = 1$. The maximum number of epochs is 100. Another setting is running training for a given number of epochs, which is 100, as in all our experiments.

Table 7.17.: Results for transfer learning with datasets encoded with **ae_cnn_*_best_700**. The table shares the number of epochs that the model was trained for as well as the minimum GE with a corresponding number of traces (NT).

Model	LM	w/o stand.			with stand.		
		epochs	GE	NT	epochs	GE	NT
encoded `DPAv4.2`							
MLP	ID	74	1	2878	1	23.9	2998
		100	80.8	2809	100	38.1	2998
CNN	ID	53	1	2769	82	34	2832
		100	1	309	100	57.95	2938
MLP	HW	9	1	2738	43	53.65	2743
		100	1	71	100	77.6	2724
CNN	HW	14	1	2861	37	1	2557
		100	1	812	100	1	852
encoded `ASCADr`							
MLP	ID	5	1	1226	24	1	2581
		100	1	459	100	25.95	80
CNN	ID	10	1	1702	11	1	2286
		100	1	356	100	1	382
MLP	HW	4	1	2641	4	1	2508
		100	1	1621	100	1	1643
CNN	HW	12	1	2738	9	1	2157
		100	1	1636	100	1	1486

In the results shown in Table 7.17, when datasets were encoded using the **ae_cnn_*_best_700** autoencoder, we see that in all cases, we reached $ge^* = 1$. Often

the necessary number of epochs is small. However, when using standardization of encoded data, we see that with DPAv4.2, we reach $ge^* = 1$ in fewer cases. Standardization for encoded ASCADr did not have much influence.

Table 7.18 shows the attack results when datasets are encoded with **ae_mlp_*_best_700**. Here, we see that the performance is a bit worse. However, we can still reach $ge^* = 1$ in some cases without standardization. For DPAv4.2, profiling models using the identity leakage model did not reach $ge^* = 1$. Using CNN, we see that it got close to one with $ge^* = 1.65$ and $ge^* = 1.15$, so we believe this can be corrected using, e.g., more epochs. Thus, we increased the number of epochs to 150 and got a $ge^* = 1$ within $N_{ge^*=1} = 854$ traces. Similarly, we select other specific cases that did not reach $ge^* = 1$, and we experiment with the number of epochs and training two instead of one last layer in the model to verify if with those modifications we can obtain better performance. Since using the standardization primarily led to worse results, we only experimented without standardization, changing the number of epochs and the number of layers we train.

Table 7.18.: Results for transfer learning with datasets encoded with **ae_mlp_*_best_700**. The table shares the number of epochs the model was trained for as well as the minimum GE with a corresponding number of traces (NT).

Model	LM	w/o stand.			with stand.		
		epochs	GE	NT	epochs	GE	NT
encoded DPAv4.2							
MLP	ID	41	67.8	2791	53	28.35	2700
		100	115.75	2827	100	77.9	17
CNN	ID	100	1.65	2935	59	44.2	2931
		100	1.15	2635	100	86.4	2915
MLP	HW	12	1	2603	1	72.1	249
		100	3.8	2501	100	136.7	0
CNN	HW	15	1	1990	19	1	2711
		100	1	747	100	1	1741
encoded ASCADr							
MLP	ID	57	18.5	14	56	21.85	1074
		100	1.95	2799	100	28.95	2973
CNN	ID	96	1.8	2992	101	6.65	2873
		100	2.55	2969	100	21.15	2826
MLP	HW	11	1	1931	12	46.1	2090
		100	7.95	2989	100	48.75	2993
CNN	HW	25	1	2430	84	1.2	2925
		100	1.55	2976	100	6.3	2894

Specifically, for the MLP and HW combination with DPAv4.2, we reach $ge^* = 1$ in 12 epochs when checking the GE after every epoch. Training for 100 epochs at once, GE gets worse (3.8). Thus, we tested with only 50 epochs and reached $N_{ge^*=1} = 1993$. A combination of MLP and the ID leakage model is the worst, with minimal GE being 67.8 in 41 epochs and 115.75 after 100 epochs. Therefore, we tested multiple modifications. We tested with training two last layers in the model, again with a different number of epochs - 100, 150, and 200. The lowest GE are in cases with 150 epochs training one layer where we reach $ge^* = 98.8$, and training two last layers with 200 and 150 epochs reaching $ge^* = 104.7$ and $ge^* = 99.55$, respectively. Similarly, we do this for the ASCADr dataset. In the case of MLP and the ID leakage model, again, we tested all cases as with DPAv4.2, and the improvement happens only with training the two last layers with 200 epochs getting $ge^* = 1.25$. In combination with CNN and ID, the minimal GE we get is 2.55 after 100 epochs, and when we train epoch by epoch, the minimum GE is 1.8 in epoch number 96. The results indicate that the model has the capacity to learn the new dataset. We tried adding more epochs, 150 and 200, but we did not reach better results (GE was 2.5 and 4.5, respectively). Also, with only 50 epochs, we get worse results with minimal GE of 35.45. We reached $ge^* = 1.3$ by training the two last layers with 150 epochs. While not investigated, it seems that perhaps using early stopping could help in this case. Early stopping could prevent GE from increasing after a certain number of epochs. The last combination we tested is the MLP and HW, where we reach a GE of 1 when training epoch by epoch. Using 50 epochs gets us to $ge^* = 1.05$, and using 150 epochs results in $ge^* = 1.2$. However, we already showed that we could get $ge^* = 1$ training epoch by epoch. Thus, the model can learn, and early stopping could help get $ge^* = 1$.

In most cases, we could get GE close to 1. In many cases, we also see capacity in the model to learn the new dataset where early stopping could be beneficial as GE seems to deteriorate after some epochs. On the other hand, training modifications did not help reach $ge^* = 1$ for the MLP and the ID leakage model for the DPAv4.2 dataset. Possibly, the autoencoder requires improvements, but we also see that the results for this case specifically were better with standardization. Including standardization with training modifications might help. Additionally, from the setup with training epoch by epoch, the minimum GE is around 50 epochs and gets larger as we train for the entire 100 epochs. Thus, early stopping might also be beneficial in this case, along with other training alternatives.

Here, we see that results using data encoded with ae_cnn_*_best_700 are better than those encoded with ae_mlp_*_best_700 with and without standardization. In both cases, standardization made performance worse, so with transfer learning, we could opt not to use standardization, at least when the reused model is trained on the original dataset and now used for encoded data. However, more exploration of this can be done as the sample might be small. Additionally, if we use transfer learning from a model trained on encoded data and then used for new encoded data, this conclusion about standardization may not be valid. Still, our experiments show significant benefits of transfer learning where tuning the profiling model for a new dataset was eliminated and training time reduced. That holds while the data

is also in different feature spaces as the model is trained on original data, then transferred for encoded data of other datasets.

7.6. CONCLUSIONS AND FUTURE WORK

In this work, we proposed autoencoders to decrease the hyperparameter tuning effort of profiling models for new datasets. Hyperparameter tuning for profiling models in SCA is a necessary but time-consuming task, and additionally, those efforts are needed for each specific dataset. Thus, we propose reusing profiling models to reduce the efforts for each new dataset by using autoencoders. The commonly used metric for autoencoders is MSE, which we showed to be positively correlated with the SNR difference between the original and reconstructed trace. Tuning autoencoders is more effortless as the MSE metric is relevant to the goal of reconstruction. On the contrary, with the classification of intermediate values, we need to perform GE calculations to validate the performance of the profiling model. Since those calculations are computationally expensive, they are not done during training, contrary to MSE computation. Therefore, AEs are easier to tune and train than profiling models of SCA.

Concerning the observed results, we show that the hyperparameter tuning was not significantly better with original traces, which means that encoded data does keep relevant information for the attacks. We consider three portability cases enabled with autoencoders.

- **Reusing profiling architecture trained on one encoded dataset for other encoded datasets:** This approach comes close to finding a universal profiling model, where all the datasets get encoded to the same feature size using autoencoders and then attacked with the same attack architecture. The results show good performance over encoded datasets. One important note is that the autoencoders with similar architecture (number of layers and neurons) lead to better attack performances. In that way, the features in encoded data are more alike, which boosts the performance of the same profiling architecture across different datasets.

- **Reusing profiling architecture trained on one dataset's original traces for attacking other datasets with more features:** Here, we use autoencoders to decrease the number of features of the new datasets to the feature size of the dataset used to train the model. Training with encoded data was better or similar in performance to training with original traces. Thus, if original traces do not lead to good performance, we can consider using autoencoders to encode data to fewer features to achieve better performance with lower tuning efforts than finding a new profiling model.

- **Reusing profiling model trained on one dataset's original traces for attacking other datasets with more features:** We utilize autoencoders to allow using transfer learning between different datasets. Dimensionality reduction is necessary as we keep the trainable parameters of the model. The results show great performance with data encoded with the `ae_cnn` architecture type

without standardization. In other cases, we also reach $ge^* = 1$ with a bit longer training or training more layers. The benefit of transfer learning enabled by autoencoders is that we eliminate the hyperparameter tuning of the profiling model and significantly reduce training time for the new dataset.

In future work, CNN autoencoder types need to be more thoroughly investigated as they are more powerful considering feature extraction than MLPs. On the other hand, we should study what is represented in the latent space of autoencoders for SCA traces. We can compare autoencoders as feature processing tools with classical approaches, such as principal component analysis (PCA). Instead of running a DL-based SCA attack on encoded data, performing classical SCA on AE-encoded data would be interesting.

REFERENCES

[1] M. Krček and G. Perin. “Autoencoder-enabled model portability for reducing hyperparameter tuning efforts in side-channel analysis”. In: *Journal of Cryptographic Engineering* (2023), pp. 1–23. DOI: 10.1007/s13389-023-00330-4.

[2] P. C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO ’96. Springer. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 104–113. ISBN: 3540615121.

[3] P. C. Kocher, J. Jaffe, and B. Jun. “Differential Power Analysis”. In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. Ed. by M. Wiener. CRYPTO ’99. Springer. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 388–397. ISBN: 3540663479.

[4] E. Brier, C. Clavier, and F. Olivier. “Correlation Power Analysis with a Leakage Model”. In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*. Ed. by M. Joye and J. Quisquater. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2. URL: https://doi.org/10.1007/978-3-540-28632-5%5C_2.

[5] 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. ISBN: 3-540-00409-2. DOI: 10.1007/3-540-36400-5_3. URL: https://doi.org/10.1007/3-540-36400-5%5C_3.

[6] O. Bronchain. “Worst-case side-channel security: from evaluation of countermeasures to new designs”. PhD thesis. Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2022. URL: https://hdl.handle.net/2078.1/258155.

[7] G. Perin, L. Wu, and S. Picek. “Exploring Feature Selection Scenarios for Deep Learning-based Side-channel Analysis”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.4 (Aug. 2022), pp. 828–861. DOI: 10.46586/tches.v2022.i4.828-861. URL: https://doi.org/10.46586/tches.v2022.i4.828-861.

[8] X. Lu, C. Zhang, P. Cao, D. Gu, and H. Lu. “Pay Attention to Raw Traces: A Deep Learning Architecture for End-to-End Profiling Attacks”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (July 2021), pp. 235–274. DOI: 10.46586/tches.v2021.i3.235-274. URL: https://doi.org/10.46586/tches.v2021.i3.235-274.

[9] 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*. Ed. by W. Fischer and N. Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 45–68. ISBN: 9783319667867. DOI: 10.1007/978-3-319-66787-4_3. URL: https://doi.org/10.1007/978-3-319-66787-4%5C_3.

[10] Y. Won, X. Hou, D. Jap, J. Breier, and S. Bhasin. “Back to the Basics: Seamless Integration of Side-Channel Pre-Processing in Deep Neural Networks”. In: *IEEE Trans. Inf. Forensics Secur.* 16 (2021), pp. 3215–3227. DOI: 10.1109/TIFS.2021.3076928. URL: https://doi.org/10.1109/TIFS.2021.3076928.

[11] Y. Zhou and F. Standaert. “Deep learning mitigates but does not annihilate the need of aligned traces and a generalized ResNet model for side-channel attacks”. In: *J. Cryptogr. Eng.* 10.1 (2020), pp. 85–95. DOI: 10.1007/s13389-019-00209-3. URL: https://doi.org/10.1007/s13389-019-00209-3.

[12] 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. https://eprint.iacr.org/2019/168. 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.

[13] L. Wu, G. Perin, and S. Picek. “I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis”. In: *IEEE Transactions on Emerging Topics in Computing* (2022), pp. 1–12. DOI: 10.1109/TETC.2022.3218372.

[14] J. Rijsdijk, L. Wu, G. Perin, and S. Picek. “Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (July 2021), pp. 677–707. DOI: 10.46586/tches.v2021.i3.677-707. URL: https://doi.org/10.46586/tches.v2021.i3.677-707.

[15] L. Masure, V. Cristiani, M. Lecomte, and F. Standaert. “Don’t Learn What You Already Know: Grey-Box Modeling for Profiling Side-Channel Analysis against Masking”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 493. URL: https://eprint.iacr.org/2022/493.

[16] D. Kwon, H. Kim, and S. Hong. “Improving non-profiled side-channel attacks using autoencoder based preprocessing”. In: *Cryptology ePrint Archive* (2020).

[17] L. Wu and S. Picek. “Remove Some Noise: On Pre-processing of Side-channel Measurements with Autoencoders”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.4 (Aug. 2020), pp. 389–415. DOI:

10.13154/tches.v2020.i4.389-415. URL: https://tches.iacr.org/index.php/TCHES/article/view/8688.

[18] S. Paguada, L. Batina, and I. Armendariz. “Toward practical autoencoder-based side-channel analysis evaluations”. In: *Computer Networks* 196 (2021), p. 108230.

[19] O. Choudary and M. G. Kuhn. “Template attacks on different devices”. In: *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*. Springer. 2014, pp. 179–198.

[20] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen. “X-DeepSCA: Cross-device deep learning side channel attack”. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. Vol. 1. New York, New York, USA: ACM Press, 2019, pp. 1–6. ISBN: 9781450367257. DOI: 10.1145/3316781.3317934. URL: http://doi.acm.org/10.1145/3316781.3317934%20http://dl.acm.org/citation.cfm?doid=3316781.3317934.

[21] S. Bhasin, A. Chattopadhyay, A. Heuser, D. Jap, S. Picek, and R. Ranjan. “Mind the portability: A warriors guide through realistic profiled side-channel analysis”. In: *NDSS 2020-Network and Distributed System Security Symposium*. 2020, pp. 1–14. URL: https://eprint.iacr.org/2019/661.pdf.

[22] P. Cao, C. Zhang, X. Lu, and D. Gu. “Cross-device profiled side-channel attack with unsupervised domain adaptation”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), pp. 27–56.

[23] P. Cao, H. Zhang, D. Gu, Y. Lu, and Y. Yuan. “AL-PA: cross-device profiled side-channel attack using adversarial learning”. In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 691–696.

[24] C. Genevey-Metat, B. Gérard, and A. Heuser. “On what to learn: Train or adapt a deeply learned profile?” In: *Cryptology ePrint Archive* (2020).

[25] 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).

[26] 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*. Ed. by A. Joux. Vol. 5479 LNCS. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9. DOI: 10.1007/978-3-642-01001-9_26.

[27] O. Bronchain, J. M. Hendrickx, C. Massart, A. Olshevsky, and F. Standaert. “Leakage Certification Revisited: Bounding Model Errors in Side-Channel Security Evaluations”. In: *IACR Cryptol. ePrint Arch.* (2019), p. 132. URL: https://eprint.iacr.org/2019/132.

[28] 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 (1 Nov. 2019), pp. 348–375. DOI: 10.13154/tches.v2020.i1.348-375. URL: https://tches.iacr.org/index.php/TCHES/article/view/8402.

[29] G. E. Hinton and R. R. Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[30] D. Bank, N. Koenigstein, and R. Giryes. “Autoencoders”. In: *arXiv preprint arXiv:2003.05991* (2020).

[31] K. Pawar and V. Z. Attar. “Assessment of autoencoder architectures for data representation”. In: *Deep Learning: Concepts and Architectures*. Springer, 2020, pp. 101–132.

[32] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. “A comprehensive survey on transfer learning”. In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[33] K. Weiss, T. M. Khoshgoftaar, and D. Wang. “A survey of transfer learning”. In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[34] S. J. Pan and Q. Yang. “A survey on transfer learning”. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[35] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems* 27 (2014).

[36] S. Bhasin, N. Bruneau, J.-L. Danger, S. Guilley, and Z. Najm. “Analysis and Improvements of the DPA Contest v4 Implementation”. In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by R. S. Chakraborty, V. Matyas, and P. Schaumont. Springer. Cham: Springer International Publishing, 2014, pp. 201–218. ISBN: 978-3-319-12060-7.

[37] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas. “Deep learning for side-channel analysis and introduction to ASCAD database”. In: *J. Cryptographic Engineering* 10.2 (2020), pp. 163–188. DOI: 10.1007/s13389-019-00220-8. URL: https://doi.org/10.1007/s13389-019-00220-8.

[38] M. Friedman. “The use of ranks to avoid the assumption of normality implicit in the analysis of variance”. In: *Journal of the american statistical association* 32.200 (1937), pp. 675–701.

[39] G. P. Quinn and M. J. Keough. *Experimental design and data analysis for biologists*. Cambridge university press, 2002.

8

Label Correlation in Deep Learning-based Side-channel Analysis

The efficiency of the profiling side-channel analysis can be significantly improved with machine learning techniques. Although powerful, a fundamental machine learning limitation of being data-hungry received little attention in the side-channel community. In practice, the maximum number of leakage traces that evaluators/attackers can obtain is constrained by the scheme requirements or the limited accessibility of the target. Even worse, various countermeasures in modern devices increase the conditions on the profiling size to break the target.

This work demonstrates a practical approach to dealing with the lack of profiling traces. Instead of learning from a one-hot encoded label, transferring the labels to their distribution can significantly speed up the convergence of guessing entropy. By studying the relationship between all possible key candidates, we propose a new metric, denoted Label Correlation (LC), to evaluate the generalization ability of the profiling model. We validate LC with two common use cases: early stopping and network architecture search, and the results indicate its superior performance.

This chapter has been published as 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* 18 (2023), pp. 3849–3861. DOI: 10.1109/TIFS.2023.3287728.

8.1. Introduction

Side-channel analysis (SCA) is recognized as one of the most powerful attack methods on the implementations of cryptographic algorithms. Commonly, such attacks are divided into direct attacks like Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [2], and two-stage (profiling) attacks like template attack [3], stochastic models [4], and machine learning-based attacks [5–7]. The profiling attacks impose additional requirements as they assume an 'open' device (or a copy of it). Still, the actual key recovery might need only a few measurements or, in some cases, a single trace [8, 9].

In recent years, machine learning-based attacks positioned themselves as a strong alternative for more 'classical' SCA [8, 10, 11], which has become a standard evaluation approach for security evaluation and certification. The success of such methods relies on a sufficient number of training traces so that a machine learning classifier can accurately map the relationship between the traces and corresponding labels (intermediate data). In the worst-case attack scenario, an attacker can obtain unlimited training traces from the clone device for profiling attacks. However, an easy-to-ignore fact, especially in SCA research, is that it is not easy to acquire a large number of attack traces, even for a white-box evaluation in a security lab. We rarely see research focus on reducing the number of profiling traces number. Indeed, such a restriction mainly comes from three factors: time constraints, countermeasures, and the device's life cycle. We argue the importance of developing techniques that effectively decrease the required number of profiling traces while keeping a similar attack performance.

- Time constraints: An evaluation's time budget dramatically limits the number of traces one can obtain. For instance, according to [12], measuring one million profiling traces for a software RSA implementation with a 1 024-bit key could take more than a week. Additionally, in post-analysis tasks such as trace realignment, noise filtering, and leakage assessment, an evaluator may not have enough budget to measure sufficient traces to break the target. Therefore, reducing the required number of profiling traces would save time and enhance the evaluator's attack capability.

- Security Countermeasures: Unlike most deep learning applications, the SCA training data are most likely 'protected' - the SCA countermeasures represent a standard/default setting for the modern smart card/SoC's implementations. These protection mechanisms further increase the difficulties in learning the trace-label relationship, thus increasing the demand for the number of measurements. From a security developers' point of view, an increasing number of side-channel measurements to break the target implementation means higher security assurance of their product. If we can effectively reduce the required number of profiling traces, such vulnerabilities will be considered again.

- Application-level Protections: For a black/grey box evaluation, the available traces can drop to hundreds or thousands due to the upper limit of program

counters such as Application Transaction Counter (ATC) or PIN Try Counter (PTC) [13], which is commonly insufficient when implementing an efficient profiling model. Building a profiling model with limited profiling traces would significantly increase the capability of exploiting the potential vulnerabilities protected by life-cycle-related counters.

There are limited evaluation metrics optimized for SCA. Evaluation metrics are essential in the training process: by actively monitoring the metric value, one can easily interpret the learning process, e.g., underfitting or overfitting. However, accuracy, a commonly used metric for deep learning, is less indicative for SCA in multi-trace attack scenarios [8]. The reasons can be explained from two aspects. First, side-channel leakages are more difficult to classify due to the noise/countermeasures in the traces. Second, accuracy does not represent the success of an attack well, as we commonly need to consider continuous attacks that are better evaluated with metrics capturing this continuity. Guessing entropy and success rate are the commonly used metrics for SCA. Unfortunately, using such evaluation metrics would significantly increase the training time due to their computation complexity. Moreover, guessing entropy $only$ evaluates the rank of the correct key. Although effective, we argue that it can be less indicative as the internal relationships with other (wrong) key candidates are not considered. More discussion is available in Section 8.5.2.

We put the above concerns forward as the motivations for this work. First, to reduce the required number of training traces, we transfer the one-hot encoded labels to their Gaussian distribution centering on the corresponding labels motivated by [14]. The proposed learning scheme is illustrated in Figure 8.1. A one-hot encoded label that belongs to class 4 has been transferred to the distributed label with the value of the fourth index with the highest probability. Based on our experiment, regardless of the used leakage model and deep learning architecture, if using our learning scheme, the profiling traces can be reduced more than five times compared with the number of profiling traces used in the literature.

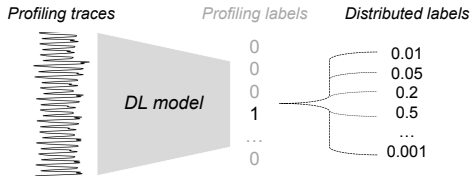


Figure 8.1.: Learning with distributed labels.

One essential assumption of the distributed label is that the label closer to the correct label is more likely to be selected. Under the same assumption, we propose key distribution to measure the geometric distance between the most likely key (not necessarily the correct key) and all the other keys. From this method and guessing entropy estimation, we propose a novel profiling model fitting metric - Label Correlation (LC) that calculates the correlation between key distribution and the key guessing vector of all key guesses. As demonstrated with experiments

on publicly available datasets, the proposed metric can indicate the generalization ability of a profiling model and thus serve as a reliable evaluation metric of early stopping and network architecture search. LC is more indicative than conventional metrics, such as validation (cross-entropy) loss, as it directly links with the attack performance. On the other hand, compared with GE, LC requires less computation effort as it does not rely on the averaging of multiple realizations of key ranks [15]. Thus, it can be a good metric to monitor the model training.

To summarize, the main contributions of this paper are:

1. We introduce a new efficient training scheme for profiling SCA when the number of profiling traces is limited. The attack performance is improved by learning from the distributed labels compared to conventional one-hot encoded labels.
2. We propose a novel method to calculate the distance between the target key and other keys called key distribution (KD).
3. Based on the guessing entropy, we introduce a new metric called Label Correlation (LC) that can effectively estimate how well the profiling model fits the data. To that end, we show that the proposed metric is reliable in reflecting the generality of the profiling model. We demonstrate two use cases for potential implementors: early stopping and network architecture search in various testing conditions. The results show that the LC metric performs better than other commonly used metrics.

We provide comprehensive experimental results on publicly available datasets to validate our claims. We also consider two commonly used deep learning architectures in SCA: multilayer perceptron and convolutional neural networks. The source code is available in the GitHub: `https://github.com/AISyLab/Label-distribution-and-correlation`.

The paper is organized as follows. Section 8.2 provides information about profiling SCA, commonly used evaluation metrics, and datasets used in this paper. The related works are discussed in Section 8.3, followed by the proposal of the label distribution learning and novel SCA evaluation metric LC in Sections 8.4 and 8.5. Section 8.6 validates the proposed methods experimentally with different datasets, attack models, and leakage models. Finally, Section 8.7 concludes this paper and proposes possible future research directions. In Appendix C.1, we provide details about the neural network architectures we used.

8

8.2. BACKGROUND

8.2.1. NOTATION

We use calligraphic letters like $\mathcal{X}$ to denote sets and the corresponding upper-case letters X to denote random variables and random vectors $\mathbf{X}$ over $\mathcal{X}$. The corresponding lower-case letters x and $\mathbf{x}$ denote realizations of X and $\mathbf{X}$, respectively. We use a sans serif font for functions (e.g., f).

k represents a key byte candidate that takes its value from the keyspace $\mathcal{K}$. k^* is the correct key byte, and k^{ref} is the key byte assumed by an attacker to be correct

(as the attacker does not know the correct key).[1]

A dataset $\mathbf{T}$ is defined as a collection of traces $\mathbf{t}_i$, where each trace $\mathbf{t}_i$ is associated with a key-related label (or the key itself) y_i. A complete set of labels with c classes is denoted by $\mathcal{Y} = \{y_1, y_2, \cdots, y_c\}$. The number of profiling traces equals N, the number of validation traces equals V, and the number of attack traces equals Q. Finally, $\boldsymbol{\theta}$ denotes the vector of parameters to be learned in a profiling model.

8.2.2. PROFILING SIDE-CHANNEL ANALYSIS

As depicted in Figure 8.2, profiling side-channel attacks consist of two phases:

1. **Learning or profiling phase.** The profiling phase consists of building a profiling model $\mathsf{f}_M^{\boldsymbol{\theta}}$ to map the inputs (side-channel measurements) to the outputs (classes as obtained by evaluating the leakage model on the sensitive operation) on a set of N profiling traces. $\mathsf{f}_M^{\boldsymbol{\theta}}$ represents the profiling model trained for a given leakage model M and the set of learning parameters $\boldsymbol{\theta}$. This phase aims to fit the parameters of a function that maps the side-channel traces to the labels in the best way (minimizing the error function). It is common to use the validation set of size V to know when to stop the learning process.
2. **Test or attack phase.** The attack phase consists of obtaining label predictions for the traces from a different dataset of size Q to test the model. The trained model processes each attack trace and produces the attack's output as a vector of probabilities $\mathbf{p}_{i,j} \in \mathscr{K}$, where each index is the probability that a trace t_i is associated with the leakage value j. We can estimate the attack performance from this matrix of probabilities (as we have multiple vectors - one for each attack trace).

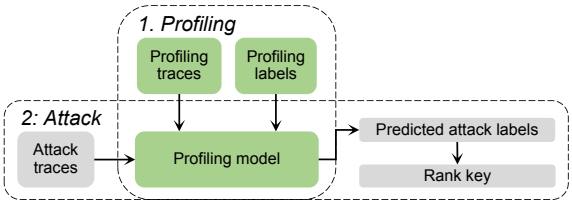


Figure 8.2.: Profiling side-channel analysis.

We consider two common profiling approaches:

- **Template Attack.** Template attack (TA) uses Bayes' theorem to obtain predictions, dealing with multivariate probability distributions as the leakage over consecutive time samples is not independent [3]. In the state-of-the-art, template attack relies mostly on a normal (Gaussian) distribution.
- **Deep Learning-based SCA (DL-SCA).** We consider supervised machine learning and the classification task as the side-channel attack's goal. Supervised learning

[1] Note that the subkey candidates can have any number of bits that are being guessed and while here we assume the AES cipher scenario, the concept is algorithm-independent.

deals with the task of learning a mapping f from a set of input variables from $\mathcal{X}$ to the set of output variables Y ($\mathsf{f}_M^{\boldsymbol{\theta}} : \mathcal{X} \rightarrow \mathcal{Y}$). For SCA, the profiling phase aims to learn the parameters $\boldsymbol{\theta}$, minimizing the empirical risk represented by a loss function on a dataset. In the attack phase, the goal is to predict the classes (more precisely, the probabilities that a certain class would be predicted) based on the previously unseen set of traces and the trained model $\mathsf{f}_M^{\boldsymbol{\theta}}$.

8.2.3. EVALUATING THE ATTACK PERFORMANCE

An attack's output is the logarithmic sum of all Q probability vectors of single model predictions, where each index is associated with one key hypothesis. Sorting this vector by decreasing probabilities leads to a key guessing vector with increasing confidence predicted by a profiling model. The key rank denotes the position of the correct key. Then, one can use metrics such as guessing entropy to estimate the attacker's performance [16].

Definition 2. ***Key guessing vector.*** *The key guessing vector* $\mathbf{g}$ *is the vector of probabilities for all key candidates from the output of the profiling model's predictions:*

$$\mathbf{g} = sort\left(\sum_{i}^{Q} log\ \mathbf{P_r}(\mathbf{t}_i; \mathsf{f}_M^{\boldsymbol{\theta}})\right), \tag{8.1}$$

where $\mathbf{P_r}(\mathbf{t}_i; \mathsf{f}_M^{\boldsymbol{\theta}})$ *is the prediction vector from the profiling model* f_M^{θ} *on a trace* $\mathbf{t}_i$. *sort is the function sorting array elements in order of decreasing values of their probabilities. Since the labels are key-related, the cumulative probabilities of labels can be easily mapped to their corresponding keys. From* $\mathbf{g}$*, the index of* g *represents the likelihood of the corresponding key candidate being the correct key candidate.* g_0 *and* $g_{|\mathcal{K}|-1}$ *are the first (best) and last (worst) element of* $\mathbf{g}$*, respectively.*

Definition 3. ***Key rank.*** *In a known-key setting, the key rank is the number of (most likely) keys an attacker needs to brute force until recovering the correct key. Among various key enumeration techniques [17], one of the more popular methods is to try every key given its probability after generating a key guessing vector. In this scenario, the key rank is the position of the correct key in the guessing vector.*

Definition 4. ***Guessing entropy.*** *The guessing entropy* [2] *represents the averaged rank of the correct key* k^* *in the key guessing vector* $\mathbf{g}$*:*

$$GE = \boldsymbol{E}\big(rank_{k^*}(\mathbf{g})\big), \tag{8.2}$$

where $rank_k(\mathbf{g}) \in \{0, \ldots, |\mathcal{K}| - 1\}$. $\boldsymbol{E}$ *is the average of multiple realizations of key rank, which is commonly performed by attacking with a profiling model multiple times with randomly selected attack traces.*

[2] As we attack only a single key byte, the proper term is partial guessing entropy. Nevertheless, we use the two terms interchangeably.

8.2.4. DATASETS

ASCAD DATASET

The ASCAD dataset is generated by taking measurements from an ATMega8515 running a masked AES-128 implementation and is proposed as a benchmark dataset for SCA [18]. Side-channel traces represent the AES encryption, where the commonly attacked trace interval represents the processing of the third byte in the S-box operation (S-box is fixed and publicly known for AES) taking place in the first round (the third byte is the first masked one). The operation is masked, and we assume no knowledge about masks in the profiling phase. There are two versions of this dataset:

1. ASCAD_f: The first version consists of 50 000 profiling traces and 10 000 attack traces, where each trace consists of 700 features (pre-selected window around the leaking spot). The profiling and attacking sets use the same fixed key, and we denote this dataset as ASCAD_f.
2. ASCAD_r: The second version of the ASCAD dataset contains 200 000 traces for profiling with random keys and random plaintexts and 100 000 for the attack phase, with a fixed key and random plaintexts. A window of 1 400 points of interest is extracted around the leaking spot. We denote this dataset as ASCAD_r.

For both datasets, different numbers of profiling and attack traces are used in our experiments (see Section 8.6 for details), and 5 000 traces are used for validation and attack. The datasets are provided at `https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1`.

CHES CTF DATASET

This dataset refers to the CHES Capture-the-flag (CTF) AES-128 measurements released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). The traces consist of masked AES-128 encryption running on a 32-bit STM microcontroller. In our experiments, we consider 45 000 traces for the training set, which contains a **fixed key**. The validation and attack sets consist of 5 000 traces. Each trace consists of 2 200 features. This dataset is available at `https://chesctf.riscure.com/2018/news`.

8.2.5. LEAKAGE MODELS

The leakage model simulates the hypothetical power consumption to process one byte (as we attack the AES cipher that is byte-oriented). Our work considers two commonly used leakage models: the Hamming Weight (HW) and Identity (ID). For the HW leakage model, the attacker assumes the leakage is proportional to the sensitive variable's Hamming weight. This leakage model results in nine classes for a single intermediate byte for the AES cipher. In terms of the ID leakage model, an attacker considers the leakage in the form of an intermediate value of the cipher. This leakage model results in 256 classes for a single intermediate byte for the AES cipher.

8.3. Related Works

In Chari *et al.*'s seminal work, the authors proposed the template attack (TA) and showed that it could break implementations secure against other forms of side-channel attacks [3]. This attack is the most powerful one from the information-theoretic point of view, but to reach its full potential, it requires an unbounded number of traces and noise following the Gaussian distribution [5]. Template attack is interesting as it is a generative technique, which means it will commonly overfit less as it allows the user to provide more information in the form of class conditionals.

While machine learning techniques have been widely used for several decades, the SCA community showed interest in such techniques only around a decade ago. In the beginning, the most attention was given to techniques like random forest [19], support vector machines [7, 20, 21], and multilayer perceptron [22] (commonly in the context of shallow learning as it had only a single hidden layer).

The rapid development of deep learning-based SCAs started in 2016 when Maghrebi *et al.* demonstrated the strong performance of several neural network types, most notably, convolutional neural networks [6].

In [18], an empirical evaluation for different hyperparameters is conducted for CNNs on the ASCAD database. In [11], the authors proposed a methodology to select hyperparameters related to the size (number of learnable parameters) of layers in CNNs. The methodology includes observations for the number of filters, kernel sizes, strides, and neurons in fully connected layers. Wouters *et al.* showed how to reach similar attack performance with data regularization and even smaller neural network architectures [23]. Perin *et al.* investigated deep learning model generalization and demonstrated how ensembles of random models could perform better than a single carefully tuned neural network model [24]. Wu *et al.* and Rijsdijk *et al.* explored different automatic hyperparameter tuning strategies, namely Bayesian optimization [25] and reinforcement learning [26] paradigms to find neural networks that perform well. While their approach requires a significant tuning effort (computational time), the authors improved state-of-the-art results. These works showed that deep learning models' good performance relies on an efficient selection of hyperparameters for specific datasets. If those hyperparameters are not selected properly, the attack will fail (or at least not work as well as possible).

It is intuitive that the number of measurements and input features also limits the performance of a profiling attack. Deep neural networks provide top-level performances in many domains when training data is sufficiently large. However, they could also perform excellently when the training data is reduced. In an effort to improve attack performance, already Choudary *et al.* investigated how adding noise to the input improves the performance of template attacks on different devices [27]. The same idea is applied to deep learning-based attacks, introduced by Kim *et al.* [8]. In the context of profiling side-channel attacks, Cagli *et al.* investigated how to create measurements that improve the attack performance synthetically [10]. Unlike the previous work where the authors developed a specialized data augmentation technique, Picek *et al.* showed that generic data augmentation techniques help in profiling SCA also [28]. Another work investigated whether limiting the number

of traces can be beneficial both from the experimental setup and performance sides [29].

From the input features perspective, Bursztein introduced the usage of raw traces for profiling in an invited talk at CHES 2018 [30]. Lu *et al.* also worked in this direction, showing that better attack performance is achieved but with significantly more complex neural networks (e.g., having around 50 layers) [31]. Perin *et al.* showed how simple re-sampling of raw traces could result in extremely powerful attacks (requiring only a single attack trace) while using simple neural networks with only a few hidden layers [9]. Finally, similarity learning was applied to pre-process the leakage traces and extract high-level features, leading to state-of-the-art attack performance with significantly reduced computation effort [32].

As already discussed, commonly, in machine learning, one estimates the behavior of a profiling model based on statistics of individual observations like accuracy, loss, or recall. Unfortunately, such metrics can be misleading in SCA, as one considers cumulative predictions. Picek *et al.* showed that standard machine learning metrics could suggest radically different performance than the SCA metrics [28]. Masure *et al.* connected the perceived information and negative log-likelihood, showing there can be common ground when using machine learning metrics in SCA [33]. Perin *et al.* discussed how mutual information could be a good metric to indicate when to stop the machine learning training process [34]. Finally, Rădulescu *et al.* compared efficient metrics (Massey's guessing entropy and empirical guessing entropy) in full-key recovery, which may help decide when to stop profiling [35].

8.4. LABEL DISTRIBUTION

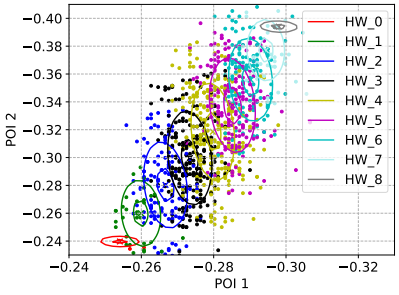
By asking *'how much does each label describe the instance?'*, Geng *et al.* first proposed Label Distribution Learning (LDL) by assigning a *description degree* (probability) of each possible label, leading to enhanced performance compared with hard (one-hot encoded) labels [14]. This method has been used in tasks such as age estimation [36] or personality recognition [37]. However, the application of LDL is restricted since one should have a reasonable estimation of the relation between labels, and such an estimation could be challenging in many tasks, e.g., image classification. Fortunately, SCA uses the leakage model to construct labels, which inherently leads to a clear relationship between labels. Indeed, two leakage traces with closer label (intermediate value) distance could be more similar. As a result, a combination of LDL and SCA could enhance attack performance.

Definition 5. ***Description degree.*** *The description degree $d_x^{y_i}$ represents the degree of a label y_i to describe an input x. From the machine learning perspective, $d_x^{y_i}$ can be considered as the probability of the label y_i being selected. If a complete set of labels $\mathscr{Y}$ can fully describe the given input, then:*

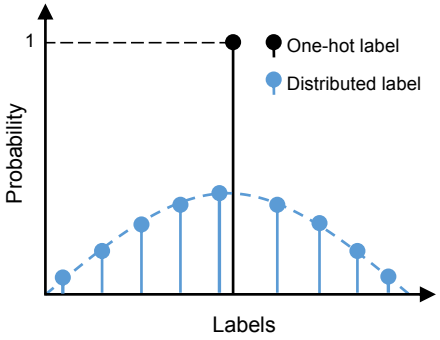
$$\sum_i d_x^{y_i} = 1, y_i \in \mathscr{Y}. \tag{8.3}$$

The conventional DL-based SCA represents a multi-class classification task that describes a measurement with a unique cluster/label. Using binary variables, the

label is one-hot encoded to train a deep learning model (see Figure 8.3b). In an ideal case, the label y_i perfectly represents the leaking features within a measurement (i.e., the correlation between labels and leaking features equals one). However, the presence of noise/countermeasure increases the description degree of other labels to the corresponding leakage traces. For illustration, Figure 8.3a shows the Probability Density Function (PDF), and point-of-interests distributions (POI1 and POI2) from 1 000 measurements[3]. The color of each point is attributed based on its cluster label. Using the HW leakage model, nine PDFs representing nine HW clusters are built during the profiling phase. Each PDF is represented by two ellipses representing 0.5 (low) and 0.9 (high) of the maximum probabilities. Note that PDF is the basis of template attack, used to present the leakages' distributions and making label predictions [27, 39].



(a) PDFs and POIs distribution for the correct key.



(b) Comparison between one-hot and distributed labels.

Figure 8.3.: PDFs and a demonstration of distributed labels.

8

From Figure 8.3a, each PDF can be separated. However, the overlap between each PDF cannot be ignored. Although the traces in the middle between two PDFs have deterministic (single) labels representing the targeted intermediate data, they are also geometrically close(r) to their neighboring clusters leakage-wise. Here, we denote the squared Euclidean distance between two label values as *label distance*. Indeed, one can observe a natural measure of description degree that associates the labels with the traces. An accurate description of these traces should involve the 'incorrect' labels. Since their similarity to each cluster is inversely correlated with their label distance, as demonstrated in Figure 8.3b, the one-hot label and the highest distributed label should be on the same abscissa; the distribution degree of other labels is assigned with reduced probability based on label distance. We denote this label representation as *distributed labels*. The description degree of each label is sorted based on their actual values. Distributed labels more precisely describe

[3]ChipWhisperer dataset [38] is used as it represents measurements obtained from a physical device, where two point-of-interests are selected based on the signal-to-noise ratio to represent the traces. Note that this dataset is not noiseless, but obtaining less noisy measurements without resorting to simulations is challenging.

the leakage features, thus helping relax the conditions on the required number of training traces to achieve a robust performance than training with one-hot encoded labels.

It is worth noting the links between template attacks and distributed labels. The multivariate normal distribution, parameterized by the mean vector μ and covariance matrix Σ, can be represented using the following equation

$$f(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{k}{2}} \cdot |\Sigma|^{\frac{1}{2}}} \cdot \exp^{-\frac{1}{2}(x-\mu)^T \cdot \Sigma^{-1} \cdot (x-\mu)}, \tag{8.4}$$

where x represents the random vector from the multivariate normal distribution with k dimensions. Indeed, both methods calculate the mean and variation of the variable to precisely describe the leakage features. The main difference is that the template attack directly characterizes the leakage features with μ and Σ, while our method focuses on enhancing leakage features' label representation.

Notice, our learning method fundamentally differs from linear regression attack (LRA) [40]. Although LRA would also lead to smooth labels by estimating weight parameters to each binary decomposition of the target value, these labels are constructed during the regression process. On the other hand, our method considers SCA as a classification task. The goal of using distributed labels is to reach more efficient classification.

One should notice the relation between label smoothing [41] and label distribution. As a regularization technique, label smoothing improves accuracy by computing cross-entropy not with the 'hard' (i.e., one-hot encoded) labels from the dataset but with a weighted mixture of all possible labels with the noise (i.e., uniform) distribution. Label distribution further preserves relations between different labels to describe leakage features. From the model training perspective, the model is less penalized by the loss value caused by the inconsistency between predictions and real labels (e.g., one-hot labels), thus speeding up the learning process. The performance benchmark between these two techniques can be found in Section 8.6.1. A natural choice to form distributed labels is a normal distribution. Indeed, the construction of distributed labels should align with the actual distribution of leakages. This paper assumes the leakage follows a Gaussian distribution commonly observed in practice. Moreover, Gaussian distribution inherently fits the distribution of the environmental noise, a main factor that increases the description degree of labels.

Definition 6. ***Label distribution learning.*** *Given a training set with trace-label pairs (x, y) sampled from $\mathbf{T}$, where $x \in \mathscr{X}$ and $y \in \mathscr{Y}$, the goal is to learn a function $\mathrm{f}_M^{\boldsymbol{\theta}}$, so that the predicted output $\hat{y}$, representing the probability of all possible labels given an input x, has a similar distribution to the distributed label $D(y)$:*

$$D(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{y-y'}{\sigma}\right)^2\right), y' \in \mathscr{Y}, \tag{8.5}$$

where $D(y)$ denotes the distributed label for the input label y; the variance is denoted by σ.[4]

[4] We assume the leakage follows a Gaussian distribution. If this assumption does not hold for the given leakage traces, the calculation of the distributed label should be adjusted accordingly.

An essential assumption of label distribution learning is that the label y should be pre-determined by an attacker. Then, the attacker can calculate the distributed label $D(y)$ with Eq. (8.5). Indeed, the only adjustable parameter σ depends on the data property (specifically, the dataset's noise). In Section 8.6, we systematically analyze the influence of σ with different datasets (including their noisy versions) and leakage models and then give suggestions on the value selection.

Next, to optimize the learning parameter θ, instead of using conventional loss functions such as categorical cross-entropy or mean squared error, following [14], Kullback-Leibler (KL) divergence is used as the loss function to measure the similarity between the predicted and ground truth distribution:

$$L = \sum_i D(y_i) \ln(\frac{D(y_i)}{\hat{y}_i}), \ y_i \in \mathscr{Y}, \tag{8.6}$$

where $\hat{y}_i$ and y_i denote the predicted probability for label i and the true label, respectively.

Stochastic gradient descent is used to minimize the loss function L. Once a network is trained, given a random input x with an unknown label from the attack dataset, the model f outputs a predicted label distribution $\hat{y}$. The predicted label is the one in $\hat{y}$ with the highest probability.

$$i^* = \underset{i}{\operatorname{argmax}}\, \hat{y}_i. \tag{8.7}$$

8.5. LABEL CORRELATION METRIC

8.5.1. KEY DISTRIBUTION

Following Eq. (8.5), the probability of a label y being selected as the correct label depends on its label distance to the true label y^*. Since these labels are key-related, we can also calculate the differences between key candidates, denoted as *key distribution* (KD), based on the label distance.

$$KD(k^{ref}, k) = \left\| \mathsf{f}(d, k^{ref}) - \mathsf{f}(d, k) \right\|^2, k \in \mathscr{K}. \tag{8.8}$$

where f is the leakage model function (described in Section 8.2.5) that returns the leakage value (labels) according to a key candidate k and data value d. Similar to Eq. (8.5), KD is based on the squared Euclidean distance between the leakage distribution of all key hypotheses $k \in \mathscr{K}$ and the reference key candidate k^{ref}. We form a KD vector sorted by the KD value for each k (so k^{ref} always ranks the first).[5] Note that when it is clear from the context, we use the notations $KD(k^{ref}, k)$ and KD interchangeably.

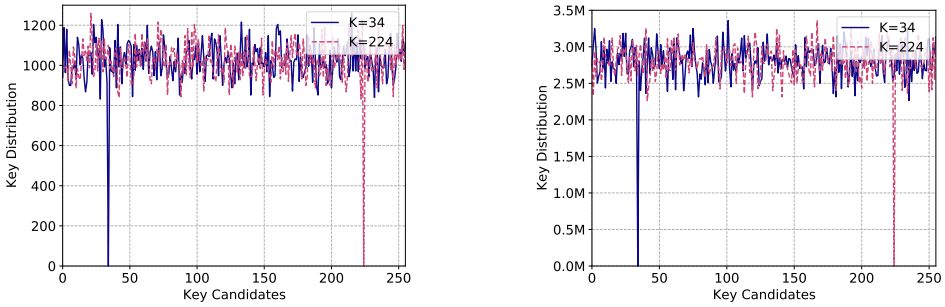
KD gives a unique distribution of all key candidates k based on their difference to the reference key k^{ref}. Therefore, k^{ref} determines the KD value for each key

[5] We also investigated the Manhattan distance and found the results to be in line but with smaller discriminate power. Besides, since KD is a list of labels associated with the given key that does not follow any known distribution, f-divergence functions (i.e., KL-divergence, Hellinger distance) are not considered.

candidate. Typically, k^{ref} has a distribution difference equal to zero with itself, and the lower the distribution difference, the more similar the key candidate is to the reference key. The reference key can be set to the k^* (correct key). When k^* is unknown (black-box), k^{ref} should be the most likely key.

From an attack perspective, for a model built in a successful profiling attack (the correct key k^* is the best guess), suppose KD is large between a specific key $k \in \mathcal{K}$ and k^*. Then, k will likely be ranked low (i.e., with guessing entropy close to $2^b - 1$) as it has a negligible probability of being selected. Consequently, KD can be considered an ideal key rank[6] metric indicating the best possible scenario where the correct key is maximally separated from all the other keys.

The KD definition can be extended to any leakage model, i.e., the Hamming distance or the Least Significant Bit. Figure 8.4 illustrates the summed KD (for all key candidates) with the HW and ID leakage models for the key candidates $k^{ref} = 34$ (correct third subkey for the ASCAD_r) and $k^{ref} = 224$ (correct third subkey for the ASCAD_f). Although all KD values except for $k^{ref} = k^*$ act as 'noisy' values, the similarity difference between k^{ref} and other keys can be observed. One should note that a precise estimation of KD relies on the chosen leakage model. An incorrect leakage model would not only degrade the attack performance, but KD's effectiveness will also drop. Since the publicly available datasets leak mostly in the HW leakage model, we calculate KD with the HW leakage model throughout the paper.



(a) HW leakage model.

(b) ID leakage model.

Figure 8.4.: Illustration for the Key Distribution for the HW/ID leakage models and correct keys 34 and 224.

8.5.2. LABEL CORRELATION - LC

Key distribution defines the distance between k^{ref} and other key candidates. Following this, we define a *profiling model fitting metric* by correlating KD with the predicted probability for all $k \in \mathcal{K}$, denoted as *Label Correlation* (LC), as a function

[6] Here, 'ideal' means the perfect fit between an attack model and the leakage. Under this circumstance, the resulting key rank is equivalent to KD as discussed in Section 8.5.2.

of KD and the key guessing vector **g** (defined in Definition 4):

$$LC = \mathsf{corr}(KD, \mathbf{g}). \tag{8.9}$$

Eq. (8.9) defines how well a profiling model fits the data concerning a key candidate k^{ref} for a chosen leakage model. The notation corr represents the Spearman correlation [42] that evaluates the monotonic relationship with two inputs. We also considered the Pearson correlation, but the results are less optimal due to the significant differences in key distribution between the correct key and other keys (Figure 8.4).

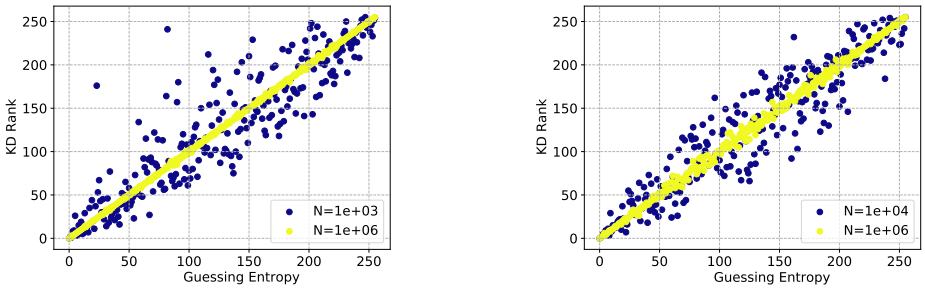
Following Eq. (8.9), if the profiling model outputs the correct key as the most likely key, one could expect a stronger correlation between KD and **g**. Conversely, if the profiling model fails to fit the data, the outputted random (but still) most likely key would lead to a low correlation between KD and **g**. As a demonstration, Figure 8.5 depicts the 'almost' perfectly fitted profiling model for the HW and ID leakage models. We use simulated measurements with strong HW and ID leakages and a controlled Gaussian noise level, normal-distributed with a variance of 0.01 around a mean of zero. The simulated traces have two features that hold the leakage, which is proportional to $HW(S_{box}(d \oplus k))$ and $S_{box}(d \oplus k)$, to simulate the ideal HW and ID leakages, respectively. The profiling set has plaintexts d and keys k chosen from a uniformly random distribution. The attack set's plaintexts are selected uniformly at random, while the attack key is the same for the whole dataset. We use the template attack and consider the increasing number of profiling traces N. In both figures, LC increases w.r.t. the number of profiling traces, reaching 0.999 and 0.998 for the HW and ID leakage models. The results confirm that the correlation between KD and **g** tends to increase with better (fitter) models (since we use template attack, better models are those that are trained with more traces).

8

Definition 7. ***Perfectly fitted profiling model.*** *A perfectly fitted profiling model reaches* $LC = 1$ *in the attack phase for any set of* Q *attack traces.*[7]

It is worth mentioning that KD can also be used to calculate the confusion covariance metric ($\mathbb{E}(KD)$) [43], a metric designed initially to measure the DPA resistance of S-boxes. A low expectation of KD indicates less distinctive intermediate data, which could lead to reduced data leakage. On top of that, the LC metric suggests that the variance of KD should also be low to secure the target. Indeed, a low KD variance indicates high similarity between different keys, which will lead to a less deterministic order of **g**. Since the LC value is more likely to be low in this case, one can expect more effort in obtaining the security assets via SCA. Another way of perturbing **g** is by introducing additional noise or countermeasures, a common implementation in modern products.

[7] We assume there are many possible ways to select Q traces from the available traces.



(a) HW leakage model. (b) ID leakage model.

Figure 8.5.: 'Perfectly' fitted profiling model with template attack, considering the HW and ID leakage models and simulated traces with an increasing number of profiling traces N. KD ranks (Y-axis) stands for a sorted KD.

8.6. EXPERIMENTAL RESULTS

8.6.1. PROFILING WITH DISTRIBUTED LABELS

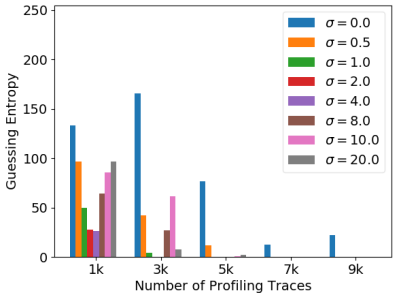
In Section 8.4, we argue that the distributed label enlarges the description degrees of labels to the leakage traces and can lead to more efficient learning even with fewer profiling examples. We validate this assumption with machine learning models by training the state-of-the-art CNNs [26] and MLPs [25] with a different number of profiling traces. The models' hyperparameters are listed in Appendix C.1. We use a diverse selection of DL architectures to ensure the generalization of the results. Note that we select MLP and CNNs due to their wide applications in SCA. Still, we expect other supervised learning methods to benefit from distributed labels thanks to their higher description degree of the leakage features. Besides, we tune the σ value of the distributed label to find the optimal value for different training settings. The distributed labels are pre-computed before the training starts. To obtain the most representative performance, the attack results of each training setting (σ and profiling traces number) are the median value from 20 independent training (and attacks) with random weight initialization following recommendations from [15].

Figures 8.6, 8.7, and 8.8 show the results for the ASCAD_f, ASCAD_r, and CHES_CTF datasets, respectively. The conventional training method (one-hot encoded label) is represented with $\sigma = 0.0$ (blue bar). We used the categorical cross-entropy (CCE) loss when training with the conventional method. CCE is a standard loss function for classification tasks widely used in DL-SCA. When learning from the label distribution, the KL divergence loss measures the distribution difference between the true and predicted label distributions.

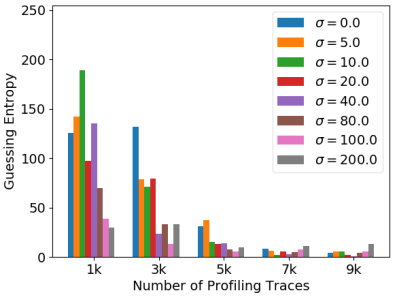
For the ASCAD_f dataset, as shown in Figure 8.6, by distributing HW-based labels, GE equal to zero can be reached with less than 3 000 profiling traces for both MLP and CNN within the given number of attack traces, which is more than ten times less than the number of the profiling traces commonly used in literature (50 000). At the same time, more than 10 000 profiling traces are insufficient when considering

the conventional training method ($\sigma = 0.0$). Using the ID leakage model, although one-hot encoded labels lead to better performance in some cases (discussed in later paragraphs), one can confirm the advantages of using the distributed label in low profiling settings.

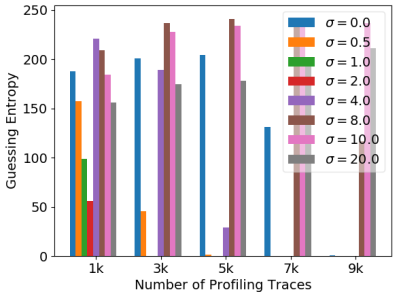
When looking at the influence of the label distribution variation σ (Figure 8.6), although different numbers of profiling traces, leakage models, and attack models are considered, the optimal settings show consistency: for the HW leakage model, σ ranges from 1 to 2 can lead to the best attack performance. This value increases to 20-80 for the ID leakage model. We have also tested the traces with Gaussian noise levels 2 and 4. While the optimal value of σ defined in the paper still holds for most settings, we expect the best σ to be larger since the leakage traces become more difficult to classify correctly.



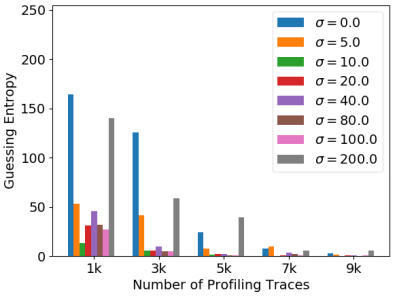
(a) MLP with the HW leakage model.



(b) MLP with the ID leakage model.



(c) CNN with the HW leakage model.



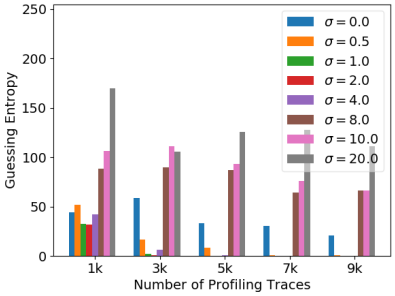
(d) CNN with the ID leakage model.

Figure 8.6.: Label distribution learning on the ASCAD_f dataset.

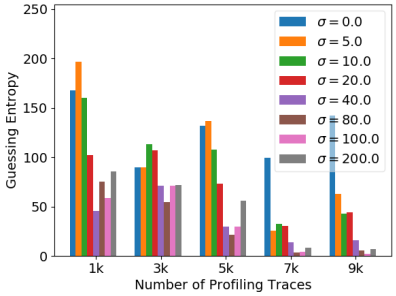
Although ASCAD_r is considered more difficult to break than ASCAD_f [25], as shown in Figure 8.7, the distributed label boosts the attack performance significantly. For the HW leakage model, around 6 000 profiling traces are sufficient for MLP and CNN models to reach GE of zero, which is around ten times less than the related works (≥50 000 profiling traces). For the ID leakage model, aligned with the attack on the ASCAD_f dataset, although none of the training settings can retrieve the secret

information with 5 000 attack traces, label distribution learning halves the GE value compared with its one-hot encoded counterpart, indicating a faster GE convergence with our learning scheme.

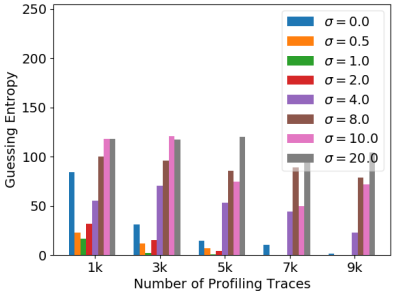
Finally, similar results can be obtained when attacking the CHES_CTF dataset. Since this dataset leaks limited ID leakage according to literature [25, 26], we attack with the HW leakage model only. With the MLP and CNN models, 5 000 profiling traces are needed to break the target, nine times less than the traces used in the literature (45 000 traces). It is important to note that the optimal σ setting shows similarity for all three tested datasets. From the experimental results on three datasets, good prior knowledge about the leakage model is necessary to construct a meaningful label distribution. Still, when attacking leakages from other devices, one could start with low σ and monitor the attack performance until it reaches optimal behavior.



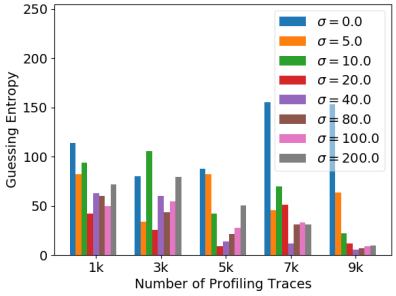
(a) MLP with the HW leakage model.



(b) MLP with the ID leakage model.



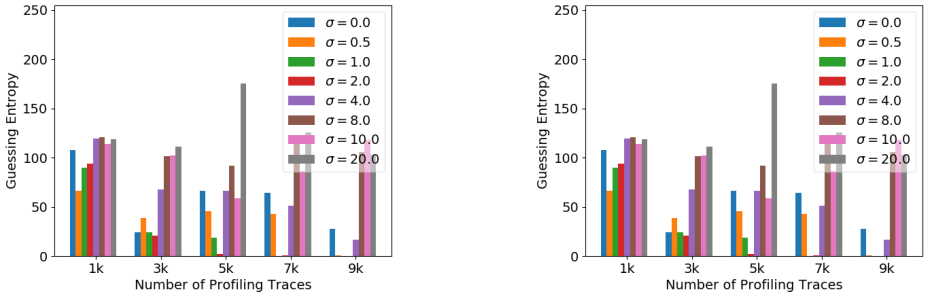
(c) CNN with the HW leakage model.



(d) CNN with the ID leakage model.

Figure 8.7.: Label distribution learning on the ASCAD_r dataset.

Indeed, there are various techniques available when the profiling traces are limited. To better illustrate the pros and cons of label distribution learning compared to these methods, we benchmark the attack performance of previously used state-of-the-art (SotA) MLPs and CNNs with multiple profiling settings. We consider several commonly-used techniques to counter the limitation of the training data.



(a) MLP with the HW leakage model.

(b) CNN with the HW leakage model.

Figure 8.8.: Label distribution learning on the CHES_CTF dataset.

- 10 000 profiling traces with and without techniques such as label distribution, label smoothing, L2 regularization, and dropout.
- 50 000 profiling traces, obtained directly or generated with Gaussian noise-based data augmentation.
- 100 000 profiling traces generated from 10 000 traces with Gaussian noise-based data augmentation.

The dropout rate and regularization factor are tuned to 5e-2 and 1e-4. For data augmentation, four augmentation levels (0.25, 0.5, 0.75, 1.0) are selected following [8], and the one with the best performance is presented in the benchmark. The label smoothing factor is set to be optimal based on the various search options.[8] Each profiling setting is tested with two label formats: one-hot encoded and distributed labels. For label distributed learning, σ is set to 1/2 and 40/80 for HW and ID leakage models. Recall that the number of attack traces is set to 10 000. The attack performance is evaluated by calculating the required number of attack traces to reach GE of zero, denoted as T_{GE0}. The results are the median T_{GE0} from 20 independently trained models. If an attack setting fails to reach GE zero with a given number of attack traces, the results are marked with "-".

The benchmark results are shown in Tables 8.1 and 8.2. The best results for each profiling setting are marked in **bold**. With limited (10 000) profiling traces, distributed labels bring a significant performance boost with various attack settings. The considered regularization techniques are helpful in some attack settings, improving significantly when combined with distributed labels. Similarly, data augmentation helps obtain better performance in some cases; the combination with distributed labels makes it even better. In practice, due to the limitation of controlled devices and time budget, attackers would likely use smaller networks, more regularization, and more data augmentation to run their attacks in lower-data settings. However, as shown in the table, label distribution is the best technique considering the additional efforts to tune hyperparameters and their performance.

Note that one-hot encoded labels often lead to comparable or superior results

[8]The possible label smoothing factors are 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 5, and 10.

Traces	Label	ASCAD_f	ASCAD_r	CHES_CTF
10 000	One-hot	-/-	-/-	-
	Smoothed	3 484/-	**-/-**	-
	Distributed	**1 618/4 964**	**3 623/4 892**	**2 337**
10 000	One-hot	-/-	-/-	3 728
(L2)	Distributed	-/-	-/-	**1 930**
10 000	One-hot	-/-	-/-	4 156
(Dropout)	Distributed	**2 264**/-	-/-	**2 493**
50 000	One-hot	**1 219/182**	970/**2 625**	**567**
	Distributed	1 421/3 530	**919**/-	905
50 000	One-hot	1 588/-	-/-	-
(Augmented)	Distributed	**1 095/4 728**	**2 784**/-	**2 735**
100 000	One-hot	**1 254**/-	-/-	-
(Augmented)	Distributed	1 447/**4 895**	**2 998**/-	**2 793**

Table 8.1.: Benchmark the attack performance (T_{GE0}) with SotA MLP. Attack results for the HW and ID leakage models are separated by '/'.

Traces	Label	ASCAD_f	ASCAD_r	CHES_CTF
10 000	One-hot	2940/-	-/-	-
	Smoothed	**2 994/-**	**-/-**	**-**
	Distributed	**1 252/4 050**	**1 939/3 753**	**2 182**
10 000	One-hot	2 217/**3 779**	-/-	-
(L2)	Distributed	**1 096**/-	**2 034/4 892**	**1 458**
10 000	One-hot	1 913/-	-/-	-
(Dropout)	Distributed	**1 338/4 219**	-/-	**1 868**
50 000	One-hot	**544/87**	650/**487**	455
	Distributed	779/-	**553**/3 684	**450**
50 000	One-hot	2 829/**4 061**	-/-	-
(Augmented)	Distributed	**1 201**/-	**2 190**/-	**1 724**
100 000	One-hot	2 278/**1 621**	-/-	-
(Augmented)	Distributed	**1 218**/-	**2 298**/-	**2 105**

Table 8.2.: Benchmark the attack performance (T_{GE0}) with SotA CNN. Attack results for the HW and ID leakage models are separated by '/'.

when training with 50 000 profiling traces. Indeed, one-hot encoded labels are more precise in discriminating the correct labels than distributed labels. On the other hand, increasing the number of profiling traces amplifies the side-effect of distributed labels, leading to high estimation variance and reduced predictive performance. Finally, although data augmentation could also generate more profiling traces, the difficulty of setting a proper augmentation level makes the generated traces less helpful in the profiling phase.

8.6.2. Use Cases of Label Correlation

In this section, we investigate the effectiveness of the LC metric for different use cases. Specifically, we consider network architecture search (NAS) and overfitting prevention as they significantly influence the attack performance with DL-based SCA. Indeed, adjusting the profiling model size will directly influence its learning capacity. On the other hand, a correctly set training epoch number could improve the model's fitness to the dataset. Since these two aspects rely on well-performing evaluation metrics [24, 26], we show the performance of LC in various settings and benchmark it with other standard metrics.

Early Stopping

As an evaluation metric, LC can be used as early stopping regularization or as an indicator of when to save the best model. For illustration, we evaluate state-of-the-art models by training with different training epochs ranging from 1 to 150 in steps of 10, representing the number of iterations to train a profiling model. Aligned with previous sections, the attack performance is assessed by T_{GE0}. Besides, four metrics, accuracy, loss, mutual information (MI) [34], and LC, are calculated per epoch with 5 000 validation traces.[9] One may argue that T_{GE0} can be used as an evaluation metric. However, T_{GE0} can only be calculated when GE equals zero. For a model that cannot break the target with a given number of attack traces, T_{GE0} is not indicative. Similarly, the key rank metric is only meaningful when GE is larger than zero: when the key rank stays zero, one cannot know if the model is still learning or starting overfitting. Since all selected models reached the key rank of zero quickly and never changed, we omit the key rank metric as it is less indicative in the training process.

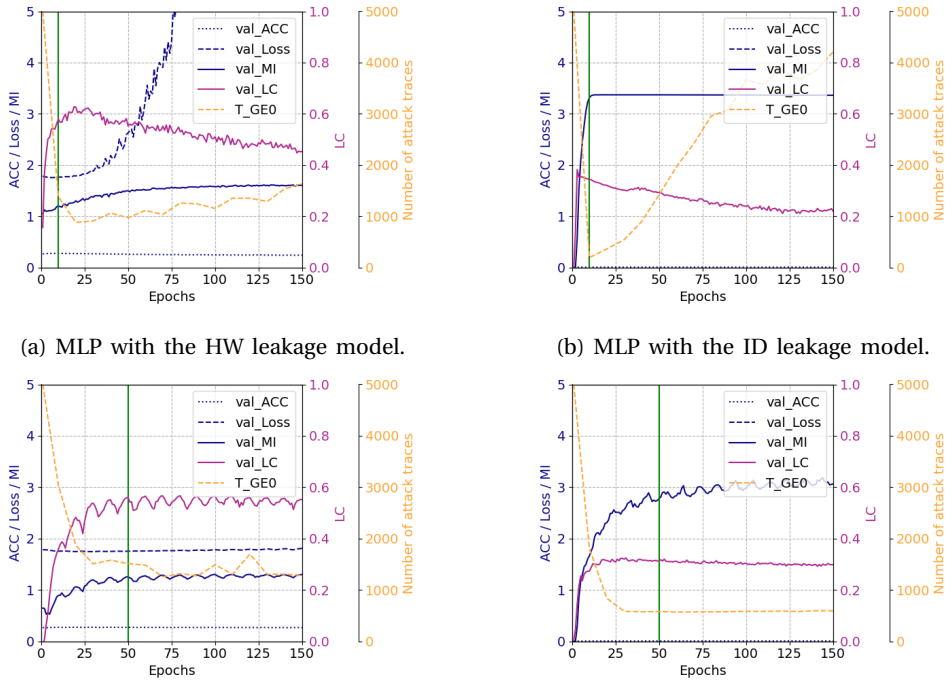
The results for three datasets and two leakage models are shown in Figures 8.9, 8.10, and 8.11. Since the metrics and T_{GE0} have different scales, multiple Y-axes are used to scale the results data. The optimal training epoch proposed in the literature is marked by green vertical lines (10 for MLPs and 50 for CNNs). Aligned with the previous section, all the presented results are the median from 20 independent pieces of training.

Regarding ASCAD_f, LC perfectly reflects the generalization variation of the profiling model with different training epochs when using T_{GE0} as a reference. From both figures, LC indicates the overfitting effect accurately, or even before the attack performance degrades. Indeed, LC evaluates the order of the key candidates, and the order closer to KD is more likely to be perturbed when overfitting starts. When the overfitting effect accumulates to a certain level, the "disorder" of the key candidate propagates to the correct key, finally captured by GE-related metrics. Due to LC's sensitivity, one can decide on optimal epochs with more patience (i.e., the number of epochs to wait before an early stop if there is no progress on the validation set) without suffering from overfitting.

Regarding other metrics, the MI metric is somewhat misleading as it keeps increasing (e.g., Figures 8.9a, 8.9d, and 8.10d) or does not change much (Figures 8.10b and 8.11b) even when T_{GE0} suggests performance degradation. The loss value is

[9] If GE is greater than zero, T_{GE0}=5 000.

only useful in limited cases (e.g., Figure 8.9a), which confirms the conclusion from Picek *et al.* [28] that it is commonly not considered a good evaluation metric for SCA, and accuracy remains mostly stable with different training epochs, indicating mediocre performance. Lastly, the literature's optimal training epochs are not optimal for Figures 8.9a and 8.9d. On the other hand, LC consistently indicates the epoch that achieves the best attack performance.



(a) MLP with the HW leakage model.

(b) MLP with the ID leakage model.

(c) CNN with the HW leakage model.

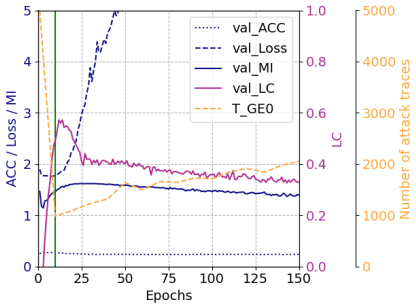
(d) CNN with the ID leakage model.

Figure 8.9.: Metrics performance on the ASCAD_f dataset.

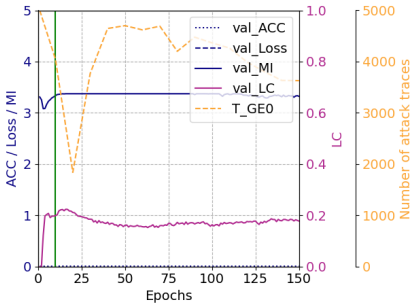
Attacks on ASCAD_r and CHES_CTF show consistent results with ASCAD_f. LC performs the best among all evaluated metrics, alarming the overfitting effect precisely. As an evaluation metric, LC combines the advantages of key rank and T_{GE0} with limited computation overhead, thus becoming a reliable metric for the applications such as early stopping.

NETWORK ARCHITECTURE SEARCH

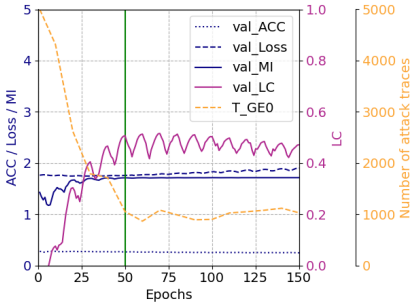
Network architecture search (NAS) is essential in DL-SCA. A smartly designed neural network can break the target and reduce the training complexity as well [11, 26]. To better illustrate the advantage of the LC metric, we use CNN listed in Table 8.3 with a tunable α parameter to control the size of the deep learning model. Specifically, α determines the number of filters in convolutional layers and neurons in the fully



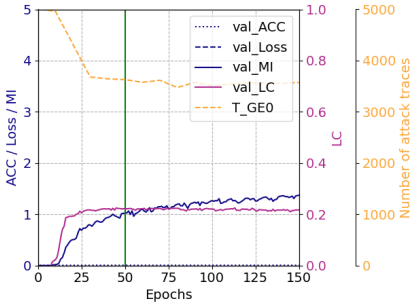
(a) MLP with the HW leakage model.



(b) MLP with the ID leakage model.

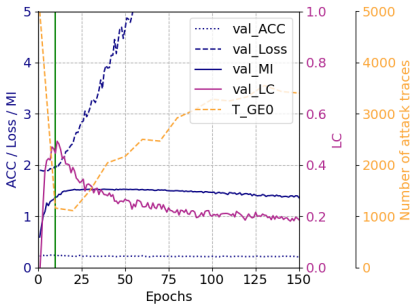


(c) CNN with the HW leakage model.

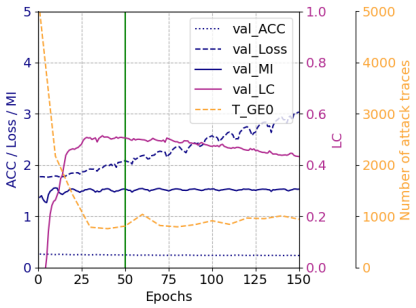


(d) CNN with the ID leakage model.

Figure 8.10.: Metrics performance on the ASCAD_r dataset.



(a) MLP with the HW leakage model.



(b) MLP with the ID leakage model.

Figure 8.11.: Metrics performance on the CHES_CTF dataset.

connected layers. We use α (range from 1 to 64) to estimate the complexity of a profiling model. Note, for the CNN_best from [18], α equals 64. The training epoch is set to be optimal (75) based on [18], represented by the green vertical line in the plot. This section presents the results for the ASCAD_f and ASCAD_r datasets only.

Since CHES_CTF produce similar results, we omit them from this section.

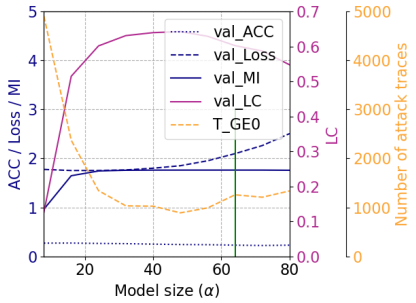
Table 8.3.: CNN architecture used for the attack.

Layer Types	Filter Size	# of Filters	Pooling Stride	# of Neurons
Conv block	11	a*1	2	-
Conv block	11	a*2	2	-
Conv block	11	a*4	2	-
Conv block	11	a*8	2	-
Flatten	-	-	-	-
Fully connected (2×)	-	-	-	a*64

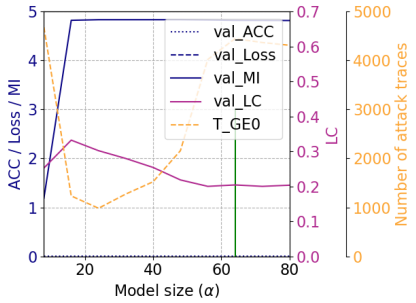
The results are shown in Figure 8.12. Aligned with the previous section, accuracy, loss, MI, and LC are used as evaluation metrics. As a reference, T_{GE0} represents the attack performance. Among the three considered metrics, LC best represents the attack performance. For instance, in Figure 8.12a, T_{GE0} reaches minimum when α equals around 50. Further increase of the profiling model size degrades the attack performance, meaning the fitness reduction for a dataset. The LC metric perfectly represents this tendency, as it reaches the maximum when α is around the same model size, then decreases gradually. Regarding other metrics, the validation accuracy has limited changes regardless of the variation of α. Validation loss, in contrast, is more indicative than its counterpart. However, it is challenging to judge when to stop the training. For instance, the loss value in Figure 8.12c suggests that the profiling should end after training with around 35 epochs, but the best performance is reached 15 epochs later. MI keeps on increasing with the HW leakage model. However, it does not correctly reflect the attack performance. Finally, the training epoch suggested in the literature is still sub-optimal when looking at the results (i.e., Figure 8.12b). Using LC as an evaluation metric can help monitor the attack performance in various settings.

In addition, we have also tested the influence of the noise on the considered metrics by adding Gaussian noise to the traces with incremental variations ranging from 0 to 10 in a step of 0.5. The results show that the LC metric can correctly and precisely reflect the negative influence introduced by the noise. Since the results align with the conclusions from the previous sections, the results are omitted.

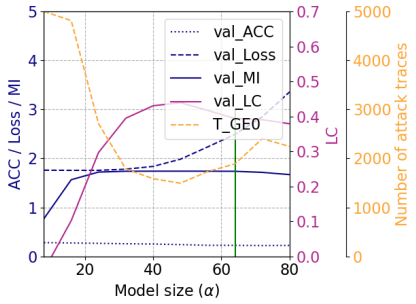
In conclusion, the LC metric reliably reflects the generality of the profiling model in various training conditions. Compared to other metrics, the evaluation of the keys' order helps in increasing the sensitivity of the LC metric in measuring the model's performance. Indeed, in almost all of the experimental results, LC is the first metric that indicates the overfitting effect. Additionally, due to its computation simplicity, we believe LC is an ideal candidate as an evaluation metric.



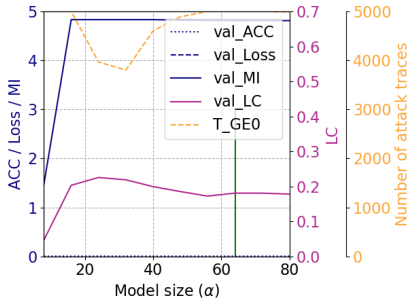
(a) ASCAD_f with the HW leakage model.



(b) ASCAD_f with the ID leakage model.



(c) ASCAD_r with the HW leakage model.



(d) ASCAD_r with the ID leakage model.

Figure 8.12.: Metrics performance with different model sizes.

8.7. Conclusions and Future Work

In the profiling side-channel analysis, one commonly uses intermediate data to form a one-hot encoded label for the profiling. Additionally, it is common to use guessing entropy to estimate the attack performance. This paper introduces distributed labels as a new learning approach that can effectively reduce the required number of profiling traces. Then, based on the relationship between each key candidate, we define the Key distribution (KD) metric and use it to form a novel LC metric. Our results show that the LC metric can be a reliable candidate for evaluating the generality of a model, which has been validated with two use cases: early stopping and network architecture search. Our findings are confirmed for several experiments considering various usage cases, attack methods, leakage models, and datasets.

In future work, we plan to extend the application of label distribution for high-order masked implementations. In terms of the LC metric, since the key distribution relies on the hypothetical distance between key candidates, the distance depends on the algorithm and hardware implementation. Following this, we plan to investigate if the method can be easily adapted to a new implementation or a new algorithm. Moreover, prior knowledge about the leakage model plays a significant role in the proposed label distribution, so we plan to explore LC in the context

of leakage assessment for the black-box devices without this knowledge. Finally, applying our results to the non-profiling SCA would be an exciting research direction.

ACKNOWLEDGMENTS

This work received funding in the framework of the NWA Cybersecurity Call with project name PROACT with project number NWA.1215.18.014, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). Additionally, this work was supported in part by the Netherlands Organization for Scientific Research NWO project DISTANT (CS.019).

REFERENCES

[1] 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* 18 (2023), pp. 3849–3861. DOI: 10.1109/TIFS.2023.3287728.

[2] 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. ISBN: 3-540-66347-9. DOI: 10.1007/3-540-48405-1_25. URL: https://doi.org/10.1007/3-540-48405-1%5C_25.

[3] 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. ISBN: 3-540-00409-2. DOI: 10.1007/3-540-36400-5_3. URL: https://doi.org/10.1007/3-540-36400-5%5C_3.

[4] W. Schindler, K. Lemke, and C. Paar. "A Stochastic Model for Differential Side Channel Cryptanalysis". In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by J. R. Rao and B. Sunar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 30–46. ISBN: 978-3-540-31940-5. DOI: 10.1007/11545262_3.

[5] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F. Standaert. "Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis)". In: *COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers*. Springer. 2015, pp. 20–33.

[6] 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. Vol. 10076. Lecture Notes in Computer Science. Springer. Springer, Dec. 2016, pp. 3–26. ISBN: 978-3-319-49444-9. DOI: 10.1007/978-3-319-49445-6_1. URL: https://doi.org/10.1007/978-3-319-49445-6%5C_1.

[7] 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.

[8] 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. URL: https://eprint.iacr.org/2018/1023.pdf.

[9] G. Perin, L. Wu, and S. Picek. “Exploring Feature Selection Scenarios for Deep Learning-based Side-channel Analysis”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.4 (Aug. 2022), pp. 828–861. DOI: 10.46586/tches.v2022.i4.828-861. URL: https://doi.org/10.46586/tches.v2022.i4.828-861.

[10] 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*. Ed. by W. Fischer and N. Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 45–68. ISBN: 9783319667867. DOI: 10.1007/978-3-319-66787-4_3. URL: https://doi.org/10.1007/978-3-319-66787-4%5C_3.

[11] 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 (Nov. 2019), pp. 1–36. DOI: 10.13154/tches.v2020.i1.1-36. URL: https://tches.iacr.org/index.php/TCHES/article/view/8391.

[12] Y. K. Kumar and R. M. Shafi. “An efficient and secure data storage in cloud computing using modified RSA public key cryptosystem”. In: *International Journal of Electrical and Computer Engineering* 10.1 (2020), p. 530.

[13] I. Card. *EMV Integrated Circuit Card Specifications for Payment Systems, Book 3 Application Specification*. https://www.emvco.com/wp-content/uploads/2017/04/EMV_v4.3_Book_3_Application_Specification_20120607062110791.pdf. Nov. 2011.

[14] X. Geng. “Label distribution learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.7 (2016), pp. 1734–1748.

[15] L. Wu, G. Perin, and S. Picek. “On the evaluation of deep learning-based side-channel analysis”. In: *Constructive Side-Channel Analysis and Secure Design: 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*. Vol. 13211. Springer. 2022, p. 49.

[16] 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*. Ed. by A. Joux. Vol. 5479 LNCS. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9. DOI: 10.1007/978-3-642-01001-9_26.

[17] R. Poussier, F.-X. Standaert, and V. Grosso. “Simple key enumeration (and rank estimation) using histograms: An integrated approach”. In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2016, pp. 61–81.

[18] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas. “Deep learning for side-channel analysis and introduction to ASCAD database”. In: *J. Cryptographic Engineering* 10.2 (2020), pp. 163–188. DOI: 10.1007/s13389-019-00220-8. URL: https://doi.org/10.1007/s13389-019-00220-8.

[19] L. Lerman, S. F. Medeiros, G. Bontempi, and O. Markowitch. “A Machine Learning Approach Against a Masked AES”. In: *CARDIS*. Lecture Notes in Computer Science. Berlin, Germany. Springer, Nov. 2013.

[20] G. Hospodar, B. Gierlichs, E. D. Mulder, I. Verbauwhede, and J. Vandewalle. “Machine learning in side-channel analysis: a first study”. In: *J. Cryptogr. Eng.* 1.4 (2011), pp. 293–302. DOI: 10.1007/s13389-011-0023-x. URL: https://doi.org/10.1007/s13389-011-0023-x.

[21] A. Heuser and M. Zohner. “Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines”. In: *COSADE*. Ed. by W. Schindler and S. A. Huss. Vol. 7275. LNCS. Springer, 2012, pp. 249–264. ISBN: 978-3-642-29911-7.

[22] R. Gilmore, N. Hanley, and M. O’Neill. “Neural network based attack on a masked implementation of AES”. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. May 2015, pp. 106–111. DOI: 10.1109/HST.2015.7140247.

[23] 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* 2020.3 (June 2020), pp. 147–168. DOI: 10.13154/tches.v2020.i3.147-168. URL: https://tches.iacr.org/index.php/TCHES/article/view/8586.

[24] G. Perin, L. Chmielewski, and S. Picek. “Strength in Numbers: Improving Generalization with Ensembles in Machine Learning-based Profiled Side-channel Analysis”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.4 (Aug. 2020), pp. 337–364. DOI: 10.13154/tches.v2020.i4.337-364. URL: https://tches.iacr.org/index.php/TCHES/article/view/8686.

[25] L. Wu, G. Perin, and S. Picek. “I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis”. In: *Cryptology ePrint Archive* (2020).

[26] J. Rijsdijk, L. Wu, G. Perin, and S. Picek. “Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (July 2021), pp. 677–707. DOI: 10.46586/tches.v2021.i3.677-707. URL: https://doi.org/10.46586/tches.v2021.i3.677-707.

[27] O. Choudary and M. G. Kuhn. “Template attacks on different devices”. In: *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*. Springer. 2014, pp. 179–198.

[28] 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 (Nov. 2018), pp. 209–237. DOI: 10.13154/tches.v2019.i1.209-237. URL: https://tches.iacr.org/index.php/TCHES/article/view/7339.

[29] 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. https://eprint.iacr.org/2019/168. 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.

[30] E. Bursztein. “Leveraging Deep-Learning to Perform SCA Attacks against AES Implementations”. In: Invited talk. 2018.

[31] X. Lu, C. Zhang, P. Cao, D. Gu, and H. Lu. “Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), pp. 235–274.

[32] L. Wu, G. Perin, and S. Picek. “The Best of Two Worlds: Deep Learning-assisted Template Attack”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.3 (June 2022), pp. 413–437. DOI: 10.46586/tches.v2022.i3.413-437. URL: https://tches.iacr.org/index.php/TCHES/article/view/9707.

[33] 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 (1 Nov. 2019), pp. 348–375. DOI: 10.13154/tches.v2020.i1.348-375. URL: https://tches.iacr.org/index.php/TCHES/article/view/8402.

[34] G. Perin, I. Buhan, and S. Picek. *Learning when to stop: a mutual information approach to fight overfitting in profiled side-channel analysis*. Cryptology ePrint Archive, Report 2020/058. https://eprint.iacr.org/2020/058. 2020.

[35] A. Rădulescu, P. G. Popescu, and M. O. Choudary. “GE vs GM: Efficient side-channel security evaluations on full cryptographic keys”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022), pp. 886–905.

[36] X. Geng, Q. Wang, and Y. Xia. “Facial age estimation by adaptive label distribution learning”. In: *2014 22nd International Conference on Pattern Recognition*. IEEE. 2014, pp. 4465–4470.

[37] D. Xue, Z. Hong, S. Guo, L. Gao, L. Wu, J. Zheng, and N. Zhao. “Personality recognition on social media with label distribution learning”. In: *IEEE access* 5 (2017), pp. 13478–13488.

[38] C. O’Flynn and Z. D. Chen. “Chipwhisperer: An open-source platform for hardware embedded security research”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2014, pp. 243–260.

[39] M. O. Choudary and M. G. Kuhn. “Efficient, portable template attacks”. In: *IEEE Transactions on Information Forensics and Security* 13.2 (2017), pp. 490–501.

[40] J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert. “Univariate side channel attacks and leakage modeling”. In: *Journal of Cryptographic Engineering* 1.2 (2011), pp. 123–144.

[41] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. “Rethinking the Inception Architecture for Computer Vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. June 2016. DOI: 10.1109/CVPR.2016.308.

[42] J. Hauke and T. Kossowski. “Comparison of values of Pearson’s and Spearman’s correlation coefficients on the same sets of data”. In: *Quaestiones geographicae* 30.2 (2011), p. 87.

[43] Y. Fei, Q. Luo, and A. A. Ding. “A statistical model for DPA with novel algorithmic confusion analysis”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2012, pp. 233–250.

III

Final Insights

9

Discussion

The need for secure systems, including cryptographic systems, is more critical than ever, given the increasing reliance on digital technology for everything from personal communication to national security. To ensure the security of these systems, manufacturers conduct security evaluations internally. Even more importantly, these products undergo security evaluations from accredited security laboratories as an independent evaluation of their security features, ensuring compliance with industry best practices, laws, regulations, and baseline requirements. These third-party evaluations enable product certification, fostering user trust.

This thesis's primary objective is to enhance this security evaluation process by providing insights into strategies that can lead to more effective and efficient evaluations. The focus is on implementation attacks, specifically fault injection and side-channel analysis, as these are commonly conducted during security evaluations of cryptographic systems [1]. Recent studies have demonstrated that integrating artificial intelligence methods into these processes significantly enhances the performance of the two implementation attacks. This thesis aims to advance AI-based implementation attacks further by exploring hyperparameter tuning of the AI methods, portability issues of these implementation attacks, and developing alternative metrics in AI applications. We aim to provide practical insights, solutions, and knowledge to support security analysts in performing more efficient security evaluations.

This chapter presents the key findings that address the research questions outlined in Section 1.5. We discuss the limitations of this work and conclude with a discussion on broader implications and future research directions.

9.1. Critical Overview of Key Findings

The research objective is to improve AI-based fault injection parameter search and deep learning side-channel analysis for security evaluations. To achieve this objective, we introduced three sub-questions, each addressed in dedicated chapters. Each sub-question targets one aspect of the practical execution of the AI-based implementation attacks: hyperparameter tuning of utilized AI methods, portability issues within implementation attacks, and evaluation metrics for AI-based

implementation attack frameworks. Through a targeted, divide-and-conquer approach, we aim to improve the efficiency of these implementation attacks.

9.1.1. HYPERPARAMETER TUNING

Proper hyperparameter values are crucial for maximizing the performance of AI methods. Hyperparameter tuning is an essential, but often complex and resource-intensive, part of deploying AI methods for any problem. For AIFI, the tuning of evolutionary methods has mostly relied on manual tuning based on guidelines from the evolutionary algorithms community, preliminary experiments, or employed grid search techniques [2, 3]. However, these methods can be inefficient, failing to explore the full potential of the hyperparameter search space, and may not be directly applicable across different domains. In DLSCA, the reliance on predefined architectures with limited hyperparameter variations initially led to suboptimal results due to variations in the targets and acquisition methods. While some studies have introduced methodologies for dataset-specific neural network tuning [4], they demonstrate poor generalizability across different datasets. Automated tuning offers a solution but presents its specific challenges. For AIFI, the physical execution required for tuning can be extremely time-consuming. On the other hand, although feasible for DLSCA, it demands significant resources and time, offering potentially better outcomes [5, 6]. Nonetheless, defining an effective hyperparameter search space is essential to avoid exhaustive exploration of non-beneficial hyperparameter combinations. To address these challenges, we deem it necessary to investigate specific hyperparameters to understand their influence on the performance of AI methods for the two implementation attacks. Therefore, our research focuses on the initialization methods hyperparameter for both implementation attacks. A sub-question is stated as *What impact do initialization methods have on the performance of AI methods in AIFI parameter search and DLSCA, and how can their selection be optimized?*

AI technique utilized for AIFI parameter search is a memetic algorithm that begins its optimization process with an initial population of solutions generated by initialization methods. The quality of these initial solutions and how well-distributed they are within the search space can significantly affect the speed of convergence and the likelihood of identifying optimal solutions. Chapter 2 analyzed four initialization methods, namely random sampling, the Taguchi method, and two versions of Latin Hypercube Sampling, aiming for a more evenly distributed initial population. Our experiments indicate that no single initialization method consistently outperforms the others across all scenarios. However, in specific contexts, like working with a smaller population or under time constraints, a well-distributed initial solution set offers advantages over random sampling. Notably, the performance gap between the least and most effective initialization methods decreased from around 5% to 2% as the population size increased from 36 to 128 solutions. Moreover, Latin Hypercube Sampling for multidimensional space demonstrated excellent stability with a smaller population size, while random sampling had the least reliable results across multiple runs. In conclusion, initializations offering well-distributed samples can improve performance when the algorithm is limited to a smaller population size, as the

distribution matters more due to the smaller number of samples. Without the population size limitation, random sampling given multiple runs will provide similar results as the other tested initialization methods.

This study was conducted using a single laser bench and a single target since execution is resource-intensive. Despite these limitations, the results support the broader application of evolutionary algorithms, where random sampling proves effective for larger population sizes without strict iteration or evaluation (time) constraints. This work is the first to investigate the benefits of a specific evolutionary algorithm for laser FI parameter search. Our results indicate that the memetic algorithm can find $\approx 30\times$ more vulnerabilities than random search, making it more efficient and confirming its generality further across different FI types. This extension is based on previous successful applications for voltage glitching and EMFI, indicating the potential generalizability of our findings about the initialization methods to other FI types, though experimental validation remains for future work. While this work demonstrates that random sampling works well for larger population sizes, the study does not provide precise thresholds for the population size. Further, only four specific initialization methods are considered, two of which follow the same principle but differ in adjustments for higher dimensions. Future investigations might explore more advanced methods to achieve well-distributed samples, especially considering the five-dimensional data space of our LFI parameter search.

Within the DLSCA framework, the choice of initialization method for neural network weights and biases is crucial for the efficiency of the training process. Proper initial values can significantly reduce the loss function's convergence time, which leads to decreased training time. In Chapter 6, we explored 11 initializers (initialization methods), evaluating their performance across three datasets, two leakage models, and two convolutional neural network architectures. Our findings suggest that initializers setting weights and biases to a constant fixed value can be excluded from the hyperparameter tuning process, as experiments confirm they cannot lead to successful attacks. Similar to our findings for AIFI, there is no universal weight initializer that consistently outperforms others across all scenarios. However, we observe that more complex datasets require a more careful initializer selection, while for simpler datasets, we may achieve satisfactory results with a broader range of weight initializers. Additionally, we investigate the influence of other hyperparameters on the performance (GE) in combination with the weight initializers. Our findings highlight the significance of carefully pairing the activation function with the weight initializer, as this combination can significantly enhance or decrease the performance. Given the observed variance in outcomes due to the stochastic nature of the training process, repeated training processes with identical initializers are required to ensure reliability in the obtained results.

The effectiveness of initializers is dataset-dependent, and while we utilize several public datasets, these might include only a partial scope of possible characteristics of side-channel measurements. Despite this limitation, our analysis did not observe a universally best weight initializer even within the utilized datasets, emphasizing the need for including this hyperparameter in the tuning process. This investigation

focused exclusively on CNNs, excluding multi-layer perceptron networks. This decision was based on the commonly superior capabilities of CNNs over MLPs, for which CNNs are also more broadly adopted, and the already time-consuming experimentation. Thus, this focus may limit our conclusions to CNNs. Further, the selection of CNN architectures, taken from related work that tuned them for specific datasets, might introduce a bias towards certain weight initializers. A more extensive range of diverse CNN architectures could potentially offer further insights. Regarding weight initializers, this study employs a wide range of options available within the TensorFlow framework, offering a good variety and confidence in the results.

This research offers valuable insights into the hyperparameter tuning of AI algorithms for AIFI parameter search and DLSCA, advancing beyond the traditional trial-and-error approach toward a more strategic optimization approach. It offers recommendations for improving the tuning process, emphasizing the role of initialization methods in both general and specific scenarios. In the context of DLSCA, these insights help define a more efficient search space for automated tuning that can significantly reduce computational resources and time requirements. Automated tuning for AIFI is currently not considered as it is highly time-consuming. However, the insights from this thesis can help reduce the search space for more efficient tuning using grid search or other methods. Moreover, this work highlights the importance of multiple training sessions for profiling models. This compensates for the variability in each training session, ensuring a more appropriate evaluation of the models' performance and capabilities. Other researchers and practitioners in this field, like security evaluators, can utilize these insights to refine their methodologies and approaches, leading to more robust and effective security evaluations. Furthermore, this work provides a foundation for future investigations into the impact of other hyperparameters on the efficacy of AI-based implementation attacks that can further enhance the current methods.

9.1.2. PORTABILITY

Portability issues within FI target characterization and SCA refer to challenges and concerns related to the reproducibility and adaptability of these attacks across different hardware platforms, software configurations, or environments. Specifically, in profiling SCA, we assume two clone (identical) devices, one used for developing the profiling model and the other, the target device, for performing the attack. We expect that the model generated from one device will effectively attack the other, assuming similar behavior. Moreover, various and numerous devices undergo security evaluation, so enhancing portability across devices and setups can significantly improve security evaluations, making them more time-efficient and broadly applicable. Therefore, portability can be divided into various complexity levels. The straightforward cases involve identical device samples, like clone devices in profiling SCA, where the hardware and the system running on the device are the same, with minor unintended production differences, as no two devices are perfectly identical. In this scenario, the FI bench or the SCA acquisition process is also considered identical using the same equipment. The most challenging scenario involves different targets (hardware and software) and measurement setups.

The described simplest portability scenario assumes identical attack outcomes and responses to fault injection across device samples, but practical experiences with both FI [7] and SCA [8–10] indicate likely discrepancies. Therefore, the second sub-question of this thesis addresses portability issues present within implementation attacks. Specifically, this thesis investigates how AI methods can mitigate portability issues in both AIFI parameter search and DLSCA to help make security evaluation more efficient. By deploying AI, this study aims to enhance the transferability and adaptability of security evaluations across different portability cases. The related sub-question is stated as: *How can AI methods help mitigate portability challenges in AIFI parameter search and DLSCA, enhancing the efficiency and applicability of security evaluations?*

In Chapter 3, we explore the portability case for AIFI parameter search concerning different samples of the same target with minor LFI bench setup modification on the focus impacting the laser intensity parameter. We conducted experiments using Fast Grid Search (FGS), Random Search (RS), and memetic algorithms with both random initialization and Decision Trees (DTs) to integrate the prior knowledge, the latter being a novel approach aimed at enhancing portability. Our experiments suggest that MAs, by adapting to the observed real-time behavior of the target under the FI bench setup, naturally mitigate portability challenges, demonstrating an advantage over the RS and FGS methods, similar to the adaptability of the technique proposed in [7]. However, incorporating prior knowledge further improves MA's performance, as the rules from the trained DT models provide a better initial population, leading to more successful identification of vulnerabilities. Specifically, the novel approach can find two orders of magnitude more vulnerabilities than RS and $\approx 60\%$ more than MA with random initialization, significantly improving portability scenarios. Additionally, we analyzed the *if-then* rules from the trained DT models. Our investigation indicates that models trained with data obtained from executing MA generated more precise rules, consequently identifying more vulnerabilities within the initial population compared to models trained with RS data. These findings offer practical advice for effectively implementing the proposed method in security evaluations.

This study considers one of the simpler forms of the portability issue, suggesting that the proposed solution might require adjustments to handle other, more complex portability cases effectively. The study was conducted using one set of samples of the same target with confidentiality restrictions, limiting the scope of our conclusions. Nevertheless, given the method's independence from specific target knowledge, we expect it should apply to different targets, bench setups, and FI types with minor adjustments. We observed that DT models consistently outperformed memetic algorithms with random initialization, highlighting the potential for further exploration into more effective metrics for DT model selection. While standard deep learning metrics were investigated in this research, designing a specific metric might enhance the selection process for this use case. In this work, we chose decision trees to obtain prior knowledge due to their simplicity and high explainability, although more advanced machine learning methods could potentially offer improved performance. However, such complexity introduces a compromise between predictive power and the transparency of the decision-making process,

calling for careful consideration between performance and explainability. Due to the confidentiality of the target, the depth of discussion regarding the generation and implications of the DT rules within this publication was constrained, underscoring possible future research focused on public targets.

As already mentioned, the portability issue of profiling models across different samples of the same target is also present within the SCA domain. However, relying on public datasets makes it harder to evaluate, and there are publications already discussing this issue. Portability issues become clearer with the diversity in hardware and software configurations, resulting in significant differences in the behavior of the target. Considering the publicly available datasets for SCA, which predominantly feature AES implementations on various hardware and include different acquisition techniques, presents an opportunity to explore a more complex portability case. In Chapter 7, we employ Autoencoders (AEs) for dimensionality reduction of the input traces to improve the portability of DL profiling models for successful attacks. This method potentially reduces the need for extensive hyperparameter retuning of profiling models by reusing the hyperparameters across different datasets. However, the tuning effort is not entirely eliminated, as the AE for the new datasets remains to be adjusted. We indicate that tuning autoencoders is relatively straightforward by demonstrating the positive correlation between the Mean Squared Error (MSE) metric, commonly used with AEs, and the Signal-to-Noise Ratio (SNR) difference between the original and reconstructed traces. While the MSE metric effectively guides the reconstruction task, the training metrics of DL models do not entirely correspond to the attack evaluation metrics in SCA, namely guessing entropy and success rate. Consequently, tuning AEs is more straightforward, leading to a more reliable and efficient tuning. Additionally, we demonstrate that hyperparameter tuning does not significantly differ when using the original traces compared to encoded traces, indicating that encoded data keeps relevant information for successful attacks. As we aim to advance security evaluations by enabling the reuse of tuned and even pre-trained models, omitting certain steps from the SCA workflow to reduce execution time, we explored three scenarios. Firstly, by encoding all datasets and employing a single profiling model architecture, successful attacks were achieved in 90% of cases, potentially facilitating the development of a universal profiling model across various datasets. Secondly, we assessed the portability of a profiling model with its hyperparameters between datasets with original and encoded traces. The results suggest that using original traces of the new dataset is feasible with the same architecture, but encoded traces can offer improved success rates in many instances. Third, we examined how a trained model could adapt to different datasets. Utilizing transfer learning in our approach, we achieve successful attacks (GE equal to 1) with only a few epochs, significantly reducing training time.

This chapter used three datasets, where two datasets were collected from the same target and acquisition, while the third dataset used a different bench and hardware of the target, but all ran an AES implementation. Despite the identical target and setup for the first two datasets, the portability challenges were present due to the measurements considering fixed and random AES keys between these datasets. Moreover, we employed a cross-validation approach by switching roles

of the utilized datasets between usage to acquire the profiling model and test its portability. However, it would be beneficial to further experiment with the approach to validate the generality, especially if other symmetric cryptographic algorithms could also be considered. Our study evaluated both CNN and MLP structures for autoencoders, finding MLP tuning more straightforward and explored more than CNN. Since CNNs were shown to be more powerful, further exploration is necessary to examine their potential in this context. The research did not extensively explore the characteristics of the latent space generated by autoencoders from side-channel traces. Investigating this could facilitate the creation of a universal latent space useful for general profiling models. Moreover, we did not compare the AE-encoded data against traditional feature processing techniques within the SCA domain. Such comparisons, alongside testing AE-encoded traces with classical SCA methods, like template attacks, represent promising directions for future research that could lead to further insight into feature and data processing.

This work improves the security evaluation by addressing portability challenges in implementation attacks, presenting solutions that utilize powerful AI techniques and practical guidance for their deployment. The proposed solutions significantly enhance the evaluation process for AIFI parameter search, allowing security evaluators to efficiently evaluate the security of product series by facilitating tests across different target samples. The thesis investigates more complex scenarios in the context of DLSCA to enhance security evaluations for different targets throughout their production and certification stages. By providing security evaluators with practical methods and suggestions for incorporating these advanced solutions, this work advances the current evaluation process and offers a foundation for further exploration in this field. For example, using the decision trees in AIFI parameter search demonstrates their predictive capabilities that can help benchmark parameter search algorithms by providing complete estimated target characterizations. These estimated target characterizations eliminate the need for FI equipment and provide researchers with realistic data, potentially leading to highly successful search algorithms beneficial to the industry. Additionally, the estimated characterizations can help optimize the AI methods for AIFI parameter search by enabling an automated hyperparameter tuning process, reducing time and resource requirements. Another area of interest, beneficial for secure system designers, would involve exploring the interpretability of DTs and the explainability of DT rules to gain insights into system vulnerabilities and defense mechanisms. Moreover, enabling portability in DLSCA offers the potential to create a universal profiling model that would work effectively across diverse datasets and systems once encoded into the same latent space. Such a model aims to standardize and optimize the security evaluation process for DLSCA, enabling greater collaboration between industry and academia. This study also offers key observations for developing a suitable autoencoder, essential for refining our methodology and possibly crucial in designing a universal profiling model.

9.1.3. EVALUATION METRICS

Proper evaluation metrics are necessary to develop valuable frameworks and benchmarks in any domain. The integration of AI methods for fault injection is relatively new, presenting many opportunities for improvement and reevaluation of the current framework. We identified several limitations in the existing evaluation metric for comparing algorithm performance in AIFI parameter search, such as promoting undesirable algorithm behaviors in specific cases. Within the already established DL-based SCA framework, it is known that there are discrepancies between the training metrics of the DL methods and the evaluation metrics of SCA [11]. Research has demonstrated that minimizing the commonly used categorical cross-entropy loss correlates with minimizing the standard attack metric, guessing entropy [12]. Nevertheless, bridging the gap between training and evaluation metrics could significantly enhance the training process, making the attack more effective. Developing a less resource-intensive metric than GE or SR could lead to improved attack performance. This thesis explores alternative metrics to evaluate AI methods applied to both implementation attacks more effectively. The sub-question is stated as *What alternative metrics can facilitate a more efficient AIFI parameter search and DLSCA model training, offering a comprehensive evaluation of AI methods for implementation attacks?*

In Chapter 4, we discuss the objectives of FI parameter search in more detail, proposing new metrics that better align with security evaluation goals. Instead of focusing only on the number of FI parameter combinations that find vulnerabilities, considering all five parameters, we evaluate performance based on discovering unique vulnerability locations and clusters. This approach prioritizes finding more distant vulnerabilities, ensuring a comprehensive security evaluation by reducing the risk of overlooking potential vulnerabilities. We introduce two novel diversity algorithms and several variations that enhance performance considering the new metrics. These algorithms, being population-based, promote diversity of the solutions in the population. Firstly, we present the Grid Memetic Algorithm (GridMA), which divides the target area into a grid-like structure, with each cell representing an area subjected to search with a single MA. This simple extension forces the algorithm to allocate attention to all regions of the target area in terms of time and evaluations. Another algorithm we propose is multi-parent ES, which uses only a mutation-like operator to evolve the parent solutions. By using the multi-parent version, we enable the discovery of multiple optima, aligning with our objective. We compare these algorithms using several metrics with RS and MA. Our results demonstrate that MA still performs best when considering the number of found vulnerabilities, a metric used in previous related work. While GridMA and ES performed relatively poorer in this regard, they still outperformed RS, finding at least $\approx 8\times$ more vulnerabilities. However, when evaluating based on the number of unique locations, the evolutionary algorithms utilized in this study found around 30% more vulnerable locations than RS, showcasing similar advantages but emphasizing the MA's convergence issue. Lastly, diversity algorithms demonstrate greater ability in discovering distinct vulnerable clusters, where MA performs least effectively.

The proposed metrics are better aligned for the security evaluation perspective,

which enables us to develop and discover better-suited algorithms for the FI parameter search considering that use case. While these metrics present an improvement, they are relatively simple, requiring further refinement and theoretical foundation. Moreover, one of the new metrics relies on a specific clustering method, for which the exploration of alternative methods was limited. Thus, further investigation could be beneficial. Due to the confidentiality of the utilized target, we do not visually present the clusters, which could have offered a more transparent illustration of vulnerable clusters' distribution. However, these metrics still provide a more objective evaluation method than visual inspections, highlighting the importance of formulating a mathematically robust metric aligned with this objective. Considering the proposed metrics, the enhancements by the new algorithms were minor, indicating their potential but emphasizing the need for more investigation. Exploring more advanced or customized methods may lead to more significant improvements.

Chapter 8 focuses on enhancing the DLSCA framework by refining the training process and introducing a metric to assess the generality of the trained model more effectively. Given the inherent connection between labels (intermediate values) in SCA due to the usage of leakage models, we adopt Label Distribution Learning (LDL) to enhance attack performance. This approach uses distributed labels, reducing the number of profiling traces needed for training the profiling model while maintaining robust attack performance. Our results indicate that employing distributed labels required around $10\times$ fewer profiling traces than commonly used $45\,000 - 50\,000$ traces. However, although standard one-hot encoded traces could achieve a low GE with $7\,000 - 10\,000$ profiling traces in some cases, using distributed labels decreased that number around $2 - 3$ times for a successful attack. We compare our approach to other techniques used in cases with limited profiling traces, such as L2 regularization, dropout, or data augmentation. We demonstrate that distributed labels outperform one-hot encoded labels even when these regularization techniques are utilized. Additionally, our results show that as the number of profiling traces is increased, using one-hot encoded labels demonstrates greater prediction precision than distributed labels due to high estimation variance. We also propose a novel metric, label correlation, which has been empirically validated as a reliable candidate for evaluating the generality of a model. Since the LC metric is less computationally expensive than GE or SR, it can be used during training on validation datasets. We validate the metric for preventing overfitting with early-stopping and a tuning process focused on neural network architecture, demonstrating its effectiveness in optimizing model development and evaluation within the DLSCA framework. Compared to several other metrics, such as accuracy, loss, and mutual information, LC proved to be the most accurate in indicating the overfitting effect. During architecture search, LC effectively represents attack performance across different profiling model sizes, reliably indicating the model's generality.

We successfully reduced the number of profiling traces required for DLSCA without compromising the attack's success by employing Gaussian-distributed labels instead of the traditional one-hot encoding. This method relies on the assumption that the labels closer to the correct label are more likely to be selected, measuring the

distance in squared Euclidean distance. This learning scheme also assumes that the leakage follows a Gaussian distribution, which, if not aligned with the actual leakage in the traces, the calculation of the distributed labels should be adjusted accordingly. The label correlation metric depends on the hypothetical distance between key candidates, which varies depending on the specific algorithm and hardware implementation. Further investigation is needed to determine the metric's adaptability to new implementations or algorithms. In exploring the LC metric for optimizing neural network architectures, we restricted our experiments to CNNs from a limited hyperparameter search space. There is potential for more investigation, where future work could evaluate LC metric's applicability across a broader range of neural networks. The proposed LC metric is limited to use in the validation step of the training process, as it is still more resource-intensive than utilized loss functions. While this research already provides an improved training process, replacing the loss function with an attack-related metric could significantly impact the DLSCA performance. However, this is a challenging issue as time complexity is crucial.

These chapters aim to improve core aspects of AI-based frameworks for implementation attacks. The research dedicated to AIFI highlights the importance of defining clear objectives for the FI parameter search. This precision facilitates clear communication of solutions and their purpose within the academic community and helps identify state-of-the-art solutions for various objectives. That further aids industry practitioners in employing the most relevant solutions for their use cases. Furthermore, this work opens new research directions, such as optimizing the fitness function for evolutionary algorithms. The fitness function, currently limited to categorical labels and predefined values, represents a crucial area for enhancement since it is the evolutionary approach's primary guidance element. This future work to enhance the fitness function for evolutionary methods mirrors the efforts done in DLSCA, where the emphasis is on improving training schemes and metrics instead of focusing on the attack performance metric itself. Given that most devices have a life cycle that dictates the termination of their functionality based on execution count or time, reducing the number of required profiling traces in security evaluations is necessary. Security analysts also have time and resource limitations, where the proposed training scheme with distributed labels could be helpful, allowing for a more efficient and effective evaluation process. Moreover, introducing the label correlation metric for monitoring model training progress on validation datasets represents a significant improvement, offering a method more closely aligned with actual attack performance metrics than traditional DL metrics like accuracy. The application of this metric requires further experimentation and validation across diverse scenarios for its broad adoption in the DLSCA framework.

9.2. LIMITATIONS

In the AIFI section, a single confidential laser bench setup and target were commonly used, limiting the details and information that could be shared in the publication. Although the proposed methods are employed without modifications or assumptions based on the used target and equipment, suggesting potential

applicability across various targets, benches, and FI types such as voltage glitching and EMFI, empirical validation across these diverse settings has not been done. Thus, conducting additional experiments in these contexts would provide more confidence in the solutions presented in this thesis. The confidentiality of the target also excluded the possibility of presenting visual representations of the target characterization. This limitation hinders a complete understanding of the target's characterization and behavior, as well as the effectiveness of the different algorithms in AIFI parameter search. Throughout this thesis, the focus has been on introducing relatively simple solutions. While these solutions promote greater accessibility and ease of implementation, exploring more advanced evolutionary algorithms designed to identify optimal and multiple solutions could offer significant advantages in the AIFI domain. Moreover, while decision trees were employed for their simplicity and explainability, particularly in generating *if-then* rules, the potential of more complex machine learning models to offer enhanced rules or predictions should be part of further investigations. Such exploration would also examine the trade-off between model explainability, resource consumption, and performance outcomes.

One of the major limitations in the study of DLSCA is the reliance on publicly available datasets, which, despite being considered standard benchmarks, are well-studied and may not fully challenge or demonstrate the capabilities of more advanced methodologies. Although our advanced methods have shown enhancements on these datasets, the effectiveness of even simpler methods on these datasets leads to concerns about their complexity. Extending research to include attacks on more secure systems and creating new, more realistic public datasets could significantly enhance future research outcomes. Additionally, most of the public datasets consider the AES cryptographic algorithm. Therefore, the generality of the proposed methods to cryptographic algorithms beyond AES, such as lightweight and post-quantum algorithms, remains to be investigated. The experimental scope was often time-constrained, requiring a balance between the comprehensiveness of the experiments, use case selection, and the available resources and time. Despite these constraints, the methods should generalize within the assumptions outlined in each dedicated chapter. However, these limitations suggest that further empirical work and theoretical investigation could offer more confidence in the findings, their applicability, and generalizability.

9.3. BROADER IMPLICATIONS AND FUTURE DIRECTIONS

Before this study, AI methods have already been applied to both implementation attacks in the focus of this thesis, with SCA having a well-defined DL-based SCA framework, which we have further explored. In contrast, the use of AI in FI is a more recent development, with AIFI framework starting to take shape from the literature as detailed in Section 1.4. Although AI algorithms have shown significant benefits for both FI parameter search and SCA, many questions remain, particularly for AIFI, where its application has not been extensively investigated yet, and for DLSCA, where specific challenges are discussed in more detail in Chapter 5. Therefore, this thesis addresses the main research question: *How can AI-based implementation*

attacks, specifically fault injection and side-channel analysis, be further enhanced for a more efficient security evaluation of cryptographic systems?

Our exploration of the main question includes addressing three critical aspects of AI-based implementation attacks. First, we consider the challenge of **hyperparameter tuning** for applied AI methods, recognizing it to be a highly time-consuming process crucial for the success of the AI methods. While AIFI parameter search and DLSCA use different AI methods, both share this issue. Investigating one specific hyperparameter, initialization methods, we enhance understanding of its influence and provide guidance and insights for security analysts aiming to improve the efficiency of their hyperparameter tuning process. Next, we address the need for **portability** in security evaluations, given the diverse range of devices requiring security evaluation. We propose leveraging prior data and findings to expedite evaluations on other devices utilizing different AI methods. We study the portability of FI parameter search results across different samples of the same target and SCA profiling models across different datasets (targets). Our proposed method enabled efficient portability in considered cases, helping understand similarities between targets and enabling future development of universal methods to help standardize the evaluation process. Lastly, we revisit and refine **evaluation metrics** within the AI-based frameworks of the considered implementation attacks. Redefining objectives and aligning evaluation metrics more closely with the considered goals of implementation attacks, we significantly improve the effectiveness of AI-based implementation attacks and, consequently, the overall process of security evaluations.

This thesis enhances AI-based implementation attacks by addressing three key aspects, promoting their broader adoption, and fostering standardization in security evaluation and the security of consumer products. For practitioners in the field of hardware security, the findings offer advanced tools and strategies for more efficient and adaptable security evaluations of cryptographic systems. As these attacks become more refined, they can be adopted and set the new standard for security evaluation and certification, significantly contributing to the overall security of various systems. While focusing on offensive security, this research showcases potential threats by attackers and provides insights into the products' flaws and vulnerabilities. Thus, focusing on research from the offensive perspective is crucial as it can force the development of more robust and secure designs and protection mechanisms as emerging threats need to be addressed. In that manner, this thesis contributes to the iterative process of identifying vulnerabilities and strengthening defenses, which leads to progressively stronger security systems.

Overall, the research presented in this thesis provides a significant contribution to the field of AI-based implementation attacks, security evaluation, and hardware security. The methodologies and findings from this thesis will impact future developments in these fields, emphasizing the increasing importance of AI in addressing complex security challenges. While this thesis's contributions are significant, the focus is on specific aspects of AI-based implementation attacks, namely, hyperparameter tuning, portability, and evaluation metrics. Though crucial and impactful within the scope of AI-based implementation attacks, these aspects represent only part of the considered frameworks. Therefore, having already

discussed incremental improvements to this work, we now outline potential broader directions for research and identify some unresolved questions in the domain. This can guide further exploration and innovation in addressing the complex field of hardware security and, more generally, cybersecurity.

AI in Target Alignment. The alignment of the target is a critical factor affecting the portability of FI results, particularly given the size and complexity of modern hardware architectures. Suboptimal alignment of bench equipment relative to the target device can significantly impact the attack's efficacy. To address this challenge, future research may investigate the integration of AI methods to automate and improve device alignment for fault injections. For example, employing deep learning techniques, proven successful in object detection tasks, researchers could explore automatic target alignment for LFI or EMFI benches using inputs from cameras commonly integrated into these setups. These advanced algorithms could identify and align corresponding to the target hardware components, which can help enhance the precision of the FI. This approach can optimize the setup process and reduce misalignment errors, improving the reliability and effectiveness of security evaluations with FI. This seems achievable due to successes of DL in image processing. However, the trade-off between the improvement and the cost incurred must be assessed to conclude if the AI-based alignment should be part of the security evaluation.

AI Methods for FI Analysis. While this thesis focuses on the AI application in the parameter search phase of FI attacks, future research could explore the utilization of AI techniques to analyze the findings from the FI parameter search. This includes developing AI models to automatically detect exploitable behavior caused by FI and demonstrating security flaws and their impact. Currently, fault models are often manually discovered for each cryptographic algorithm or system being attacked. A recent report demonstrates the use of reinforcement learning to automatically find fault models [13], which aligns with this research direction. Furthermore, future studies could focus on developing frameworks for automatically generating attack scenarios based on identified vulnerabilities, enabling security analysts to fully assess the impact of identified flaws on system security. This could involve creating automated testing environments capable of simulating diverse FI scenarios and evaluating their impact on system integrity and security.

Universal Profiling Model for DLSCA. Developing a universal profiling model for DLSCA could help standardize security evaluation and foster enhanced collaboration between industry and academic research. While recent studies, such as that by Bursztein et al. [14], proposed first efforts in this direction, there are still significant opportunities for enhancement and exploration. One approach could also focus on exploring trace representation and feature extraction to help investigate and define the limits of these potential universal models. Our work on portability using autoencoders offers a way to explore a common trace representation in the latent space that could lead to establishing a universal attack model.

Enhancing DL-based Non-profiling SCA. This thesis is focused on improving DL-based profiling SCA. However, non-profiling SCA do not require clone devices, which decreases the requirements for conducting these attacks, making them a more realistic threat from a practical point of view. Research on this topic already exists [15–17], but has to be further explored to reach the level of extensive research done on profiling SCA. The work presented in this thesis could potentially be used as a foundation for exploring and testing methodologies applicable to non-profiling SCA as well. Further development in this area could lead to a more detailed understanding and application of SCA techniques across various scenarios.

Integration with Emerging Technologies. Investigate how AI-based implementation attacks can be adapted and applied to emerging technologies and cryptographic systems, such as post-quantum algorithms and IoT devices. This includes exploring the unique vulnerabilities of these technologies and developing AI frameworks to evaluate and enhance their security. The frameworks utilized in this thesis are explored on symmetric cryptographic systems but provide a foundation for developing similar frameworks for these emerging systems. By studying these domains, novel insights and methodologies can be discovered, leading to a more robust security framework for new technologies and cryptographic algorithms.

Defensive Use of AI in Cryptographic Systems. While the thesis focuses on the offensive application of AI, we can explore how AI can be employed defensively. Future research could aim to develop AI-based anomaly detection systems that identify and mitigate FI and SCA attacks, enhancing the robustness of cryptographic devices. On the other hand, we can use the generative abilities of AI to develop lightweight defense methods given hardware and software constraints. For example, in [18], the authors use reinforcement learning to develop hiding countermeasure against SCA. Such innovative systems can improve the security of cryptographic devices against malicious threats.

Automated Attacks. Explore the development of AI systems to automate the entire attack chain, from finding vulnerabilities to exploiting them with FI. On the other hand, from collecting raw traces to successful attacks in SCA. This would represent a significant advancement in the field, offering an automated AI-based approach to security evaluation. To achieve this ambitious goal, multiple steps must be addressed. Firstly, a robust framework for finding vulnerabilities must be established, integrating advanced AI algorithms with high success for identifying weaknesses in systems. Findings from this work help with this objective. Bridging the gap between finding vulnerabilities and exploitation requires integrating intelligent decision-making capabilities as discussed already for future research in applying AI methods for FI analysis. For DLSCA, steps could include the acquisition process, preprocessing, and analyzing the raw traces effectively. After these steps, the process discussed in the thesis is performed, making our findings relevant for automating the next steps. Furthermore, comprehensive testing and validation frameworks must be defined to ensure the reliability and robustness of the automated attack

chain, including both simulated and real-world data. Achieving this objective requires interdisciplinary collaboration and innovation across several domains but can significantly advance security evaluation and implementation attacks in general.

AI-based Design Synthesis. Develop AI models that automatically generate cryptographic designs or security protocols based on specified requirements and constraints. Evolutionary algorithms, reinforcement learning, or Generative Adversarial Networks (GANs) can be utilized to improve design robustness against known attacks while optimizing and automating the design process. Cryptographic primitives are already being optimized using AI [19]. We can explore expanding these concepts to more complex tasks within secure design synthesis. By developing an automated design and attack cycle, AI methods can iteratively adapt secure designs and defense mechanisms to evolving threats, ensuring robustness against emerging threats. Moreover, AI can help simulate the iterative process of evolving and improving secure designs while simultaneously exploring novel attack methods, similar to the process of GANs. This approach enables anticipation and mitigation of potential future threats, facilitating the overall robustness of secure systems.

REFERENCES

[1] NIST. *FIPS 140-3 development*. 2017. URL: https://csrc.nist.gov/projects/fips-140-3-development.

[2] S. Picek, L. Batina, D. Jakobović, and R. B. Carpi. "Evolving genetic algorithms for fault injection attacks". In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2014, pp. 1106–1111.

[3] A. Maldini, N. Samwel, S. Picek, and L. Batina. "Genetic algorithm-based electromagnetic fault injection". In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2018, pp. 35–42.

[4] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli. "Methodology for Efficient CNN Architectures in Profiling Attacks". en. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* Volume 2020 (2019).

[5] L. Wu, G. Perin, and S. Picek. "I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis". In: *Cryptology ePrint Archive* (2020).

[6] J. Rijsdijk, L. Wu, G. Perin, and S. Picek. "Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.3 (July 2021), pp. 677–707. DOI: 10.46586/tches.v2021.i3.677-707. URL: https://doi.org/10.46586/tches.v2021.i3.677-707.

[7] L. Wu, G. Ribera, N. Beringuier-Boher, and S. Picek. "A fast characterization method for semi-invasive fault injection attacks". In: *Cryptographers' Track at the RSA Conference*. Springer. 2020, pp. 146–170. ISBN: 978-3-030-40185-6.

[8] O. Choudary and M. G. Kuhn. "Template attacks on different devices". In: *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*. Springer. 2014, pp. 179–198.

[9] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen. "X-DeepSCA: Cross-device deep learning side channel attack". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. Vol. 1. New York, New York, USA: ACM Press, 2019, pp. 1–6. ISBN: 9781450367257. DOI: 10.1145/3316781.3317934. URL: http://doi.acm.org/10.1145/3316781.3317934%20http://dl.acm.org/citation.cfm?doid=3316781.3317934.

[10] S. Bhasin, A. Chattopadhyay, A. Heuser, D. Jap, S. Picek, and R. Ranjan. "Mind the portability: A warriors guide through realistic profiled side-channel analysis". In: *NDSS 2020-Network and Distributed System Security Symposium*. 2020, pp. 1–14. URL: https://eprint.iacr.org/2019/661.pdf.

[11] 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 (Nov. 2018), pp. 209–237. DOI: 10.13154/tches.v2019.i1.209-237. URL: https://tches.iacr.org/index.php/TCHES/article/view/7339.

[12] 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 (1 Nov. 2019), pp. 348–375. DOI: 10.13154/tches.v2020.i1.348-375. URL: https://tches.iacr.org/index.php/TCHES/article/view/8402.

[13] H. Guo, S. Saha, S. Patnaik, V. Gohil, D. Mukhopadhyay, and J. J. Rajendran. "Vulnerability Assessment of Ciphers To Fault Attacks Using Reinforcement Learning". In: *Cryptology ePrint Archive* (2022).

[14] E. Bursztein, L. Invernizzi, K. Král, D. Moghimi, J.-M. Picod, and M. Zhang. "Generic attacks against cryptographic hardware through long-range deep learning". In: *arXiv preprint arXiv:2306.07249* (2023).

[15] B. Timon. "Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 107–131. DOI: 10.13154/tches.v2019.i2.107-131. URL: https://doi.org/10.13154/tches.v2019.i2.107-131.

[16] D. Kwon, H. Kim, and S. Hong. "Non-profiled deep learning-based side-channel preprocessing with autoencoders". In: *IEEE Access* 9 (2021), pp. 57692–57703.

[17] N.-T. Do, P.-C. Le, V.-P. Hoang, V.-S. Doan, H. G. Nguyen, and C.-K. Pham. "MO-DLSCA: Deep Learning Based Non-profiled Side Channel Analysis Using Multi-output Neural Networks". In: *2022 International Conference on Advanced Technologies for Communications (ATC)*. 2022, pp. 245–250. DOI: 10.1109/ATC55345.2022.9943024.

[18] J. Rijsdijk, L. Wu, and G. Perin. "Reinforcement learning-based design of side-channel countermeasures". In: *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2021, pp. 168–187.

[19] L. Mariot, D. Jakobovic, T. Bäck, and J. Hernandez-Castro. "Artificial intelligence for the design of symmetric cryptographic primitives". In: *Security and Artificial Intelligence: A Crossdisciplinary Approach*. Springer, 2022, pp. 3–24.

IV

Appendices

A

Appendix: On the Importance of Initial Solutions Selection in Fault Injection

A.1. Orthogonal Arrays used in the Experiments for the Taguchi Method

In Table A.1, we present the orthogonal array used in the experiments with a population size of 36. Table A.2 gives the orthogonal array used in the experiments with a population size of 128.

Table A.1.: The orthogonal array used in the experiments with a population size of 36. This is an array of strength two with the number of samples being 36, and the number of levels, for each factor, equals 3, 3, 3, 2, 2, respectively to the order of columns (LFI parameters).

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	1
0	1	1	0	1
0	1	1	0	1
0	1	1	1	0
0	1	1	1	0
0	2	2	1	0
0	2	2	1	1
0	2	2	1	1
0	2	2	1	1
1	0	1	1	0
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1
1	1	2	0	0
1	1	2	0	0
1	1	2	0	0
1	1	2	0	1
1	2	0	0	1
1	2	0	0	1
1	2	0	1	0
1	2	0	1	0
2	0	2	0	1
2	0	2	0	1
2	0	2	1	0
2	0	2	1	0
2	1	0	1	0
2	1	0	1	1
2	1	0	1	1
2	1	0	1	1
2	2	1	0	0
2	2	1	0	0
2	2	1	0	0
2	2	1	0	1

Table A.2.: The orthogonal array used in the experiments with a population size of 128. This is an array of strength two with the number of samples being 128, and the number of levels for every factor is two. In this representation, the first column represents the number of times the same combination repeats. There are eight different combinations of the factor levels, and each repeats 16 times.

16x	0	0	0	0	0
16x	0	0	0	1	1
16x	0	1	1	0	0
16x	0	1	1	1	1
16x	1	0	1	0	1
16x	1	0	1	1	0
16x	1	1	0	0	1
16x	1	1	0	1	0

B

Appendix: Autoencoder-enabled Model Portability in Side-channel Analysis

B.1. Hyperparameter Search Spaces

We execute a random search over hyperparameter search spaces for autoencoders and profiling models. This section reports hyperparameter search spaces for all of our experiments. Hyperparameter search space for MLP autoencoders in the initial experiments with metric analysis and best latent size search are in Table B.1. The differences are in the number of neurons per layer for the different types of autoencoders we use. Ta-

Table B.1.: Hyperparameter search space for autoencoder ae_mlp, ae_mlp_dcr and `ae_mlp_str_dcr` in the initial experiments for metric analysis and latent dimension search.

Hyperparameter	Values	
	ae_mlp	ae_mlp_(str_)dcr
Layers	[1, 2, 3, 4, 5, 6]	
Neurons	[20, 50, 100, 250]	[20, 40, 50, 100, 150, 200, 300, 400]
Batch size	[100, 200, 400]	
Activation	[tanh, elu, selu, sigmoid]	
Learning rate	[0.005, 0.001, 0.0001, 0.00001]	
Weight init	[random_uniform, he_uniform, glorot_uniform, random_normal, he_normal, glorot_normal]	
Optimizer	[Adam, RMSprop, SGD, Adagrad]	

ble B.2 shows search space for CNN autoencoders. The batch size, activation, learning rate, weight initialization, and optimizer are the same for all AE types.

Table B.2.: Hyperparameter search space for autoencoder ae_cnn.

Hyperparameter	Values
Conv. layers	[1, 2, 3, 4]
Filters	[4, 8, 16]
Kernel size	[10, 20]
Strides	[5, 10]
Pool size	[2, 4]
Pool strides	[2, 4]
Pooling type	[Avg, Max]
Batch size	[100, 200, 400]
Activation	[tanh, elu, selu, sigmoid]
Learning rate	[0.005, 0.001, 0.0001, 0.00001]
Weight init	[random_uniform, he_uniform, glorot_uniform, random_normal, he_normal, glorot_normal]
Optimizer	[Adam, RMSprop, SGD, Adagrad]

For profiling models, the hyperparameter search space for MLP and CNN is in Table B.3.

Lastly, we use autoencoders with latent size 700, so we report in Table B.4 the number of layers and neurons per layer we allow. Other hyperparameters stay the same as in Table B.1 and Table B.2.

B.2. STATISTICAL TESTS

Using a Friedman test, we identify that there is indeed a significant difference in the means of the groups. However, we need to find out which ones differ specifically. Thus, a post-hoc test is necessary. One such test is the Nemenyi test, and using Python packages, we obtain the results for different latent sizes in Tables B.5 and B.6. Latent dimensions are in rows and columns. The Nemeyi post-hoc test returns the p-values for each pairwise comparison of means. Using a significance level $\alpha = 0.05$, the pairwise latent sizes with a significant difference are **bolded**. Table B.5 shows the pairwise comparison for eight latent sizes because we exclude the results for `ae_mlp_str_dcr` type as with specified hyperparameter search it did not work for latent size 500. Table B.6 shows results including that AE type but excluding the latent size 500.

B.3. HYPERPARAMETERS VALUES OF MODELS FROM EXPERIMENTS

This section provides information on hyperparameter values for the models we use in our experiments. In Table B.7, we show hyperparameters of the best MLP and CNN

Table B.3.: Hyperparameter search space for profiling models. For CNN models, the convolutional layer is followed by BatchNormalization and then the pooling layer. The number of filters increases by being multiplied by the corresponding order of the layer.

Hyperparameter	MLP	CNN
FC layers	[1, 2, 3, 4, 5, 6]	[1, 2]
Neurons	[20, 40, 50, 100, 150, 200, 300, 400]	[20, 50, 100, 200]
Conv. layers	-	[1, 2, 3, 4, 5]
Filters	-	[4, 8, 12, 16]
Kernel size	-	[10, 20, 30, 40]
Strides	-	[5, 10, 15, 20]
Pool size	-	[2]
Pool strides	-	[2]
Pooling type	-	[Avg]
Batch size	[100, 200, 400]	
Activation	[elu, selu, relu]	
Learning rate	[0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001]	
Weight init	[random_uniform, he_uniform, glorot_uniform, random_normal, he_normal, glorot_normal]	
Optimizer	[Adam, RMSprop, SGD, Adagrad]	

Table B.4.: Hyperparameter search space for autoencoder `ae_mlp_str_dcr` with latent space size of 700. We exclude the batch size, activation, learning rate, weight initialization, and optimizer hyperparameters as they are already shown in Tables B.1 and B.2.

	`DPAv4.2`	`ASCADr`
Layers	[1, 2, 3, 4, 5, 6]	
Neurons	[700, 800, 900, 1000, 1200, 1400, 1600, 1800]	[700, 800, 900, 1000, 1100, 1200, 1300]

profiling models for the encoded `DPAv4.2` dataset when encoded with the best-found `ae_mlp_dpav42_best`. These models correspond to results in Table 7.7. Corresponding to Table 7.8, we provide hyperparameter values for the best MLP and CNN models for the encoded `DPAv4.2` dataset when encoded with the best-found `ae_cnn_dpav42_best` in Table B.8. Hyperparameter values of the best MLP and CNN models for `ASCADf` dataset are visible in Table B.9. The attack performance of those models is shown in Table 7.15.

Table B.5.: p-values of Nemenyi post-hoc test for without the `ae_mlp_str_dcr` model.

Latent sizes	20	40	50	100	200	250	400	500
20	1.0	0.9	0.9	0.158	**0.001**	0.263	**0.001**	0.088
40	0.9	1.0	0.9	0.552	**0.015**	0.693	**0.006**	0.403
50	0.9	0.9	1.0	0.693	**0.032**	0.834	**0.015**	0.552
100	0.158	0.552	0.693	1.0	0.763	0.9	0.623	0.9
200	**0.001**	**0.015**	**0.032**	0.763	1.0	0.623	0.9	0.9
250	0.263	0.693	0.834	0.9	0.623	1.0	0.481	0.9
400	**0.001**	**0.006**	**0.015**	0.623	0.9	0.481	1.0	0.763
500	0.088	0.403	0.552	0.9	0.9	0.9	0.763	1.0

Table B.6.: p-values of Nemenyi post-hoc test for with the `ae_mlp_str_dcr` model.

Latent sizes	20	40	50	100	200	250	400
20	1.0	0.0	0.783	**0.042**	**0.001**	**0.030**	**0.001**
40	0.9	1.0	0.9	0.364	**0.009**	0.296	**0.001**
50	0.783	0.9	1.0	0.647	**0.042**	0.579	**0.006**
100	**0.042**	0.364	0.647	1.0	0.783	0.9	0.438
200	**0.001**	**0.009**	**0.042**	0.783	1.0	0.851	0.9
250	**0.030**	0.296	0.579	0.9	0.851	1.0	0.511
400	**0.001**	**0.001**	**0.006**	0.438	0.9	0.511	1.0

Table B.7.: Best MLP and CNN profiling models obtained for the encoded `DPAv4.2` dataset when encoded with the best-found `ae_mlp_dpav42_best`.

Hyperparameters	MLP		CNN	
	ID	HW	ID	HW
FC layers	2	3	1	2
Neurons	100	40	50	20
Filters	-	-	16	12
Kernel size	-	-	20	40
Strides	-	-	5	15
Conv. layers	-	-	4	5
Batch size	200	200	100	100
Activation	selu	selu	selu	elu
Learning rate	0.005	0.0025	0.005	0.001
Weight init.	random uniform	glorot uniform	random normal	glorot uniform
Optimizer	Adam	RMSprop	Adam	RMSprop

Table B.8.: Best MLP and CNN profiling models obtained for the encoded `DPAv4.2` dataset when encoded with the best-found `ae_cnn_dpav42_best`.

Hyperparameters	MLP		CNN	
	ID	HW	ID	HW
FC layers	6	6	1	2
Neurons	200	50	200	200
Filters	-	-	12	12
Kernel size	-	-	30	40
Strides	-	-	5	15
Conv. layers	-	-	4	5
Batch size	200	200	400	400
Activation	selu	elu	selu	elu
Learning rate	0.0005	0.0025	0.0025	0.0005
Weight init.	random uniform	glorot normal	glorot normal	random normal
Optimizer	RMSprop	RMSprop	RMSprop	RMSprop

Table B.9.: Best profiling models for `ASCADf` dataset.

Hyperparameters	MLP		CNN	
	ID	HW	ID	HW
FC layers	4	4	2	1
Neurons	40	20	50	200
Filters	-	-	4	4
Kernel size	-	-	10	40
Strides	-	-	5	5
Conv. layers	-	-	1	4
Batch size	400	100	200	400
Activation	relu	elu	relu	selu
Learning rate	0.001	0.001	0.0001	0.005
Weight init.	random uniform	random normal	random uniform	glorot uniform
Optimizer	Adam	RMSprop	RMSprop	RMSprop

C

Appendix: Label Correlation in Deep Learning-based Side-channel Analysis

C.1. The State-of-the-art Models

The used state-of-the-art models are listed in Tables C.1 and C.2. All of the non-listed hyperparameter settings are aligned with the original papers [1, 2]. The convolution layer is denoted by C; averaging pooling layer is denoted by P. FLAT and FC denote the flatten layer and fully connected layer, respectively. Finally, SM denotes the output layer with the $softmax$ activation function.

Table C.1.: CNN architecture used for the attack [1].

Dataset	Leakage Model	Architectures	Learning Rate	Batch Size
ASCAD_f	HW	C(2,25,1), P(4,4), FLAT, FC(15, 10, 4), SM(9)	5e-3	50
	ID	C(128,25,1), P(25,25), FLAT, FC(20, 15), SM(256)	5e-3	50
ASCAD_r	HW	C(4,50,1), P(25,25), FLAT, FC(30, 30, 30), SM(9)	5e-3	128
	ID	C(128,3,1), P(75,75), FLAT, FC(30, 2), SM(256)	5e-3	128
CHES_CTF	HW	C(2,2,1), P(7,7), FLAT, FC(10), SM(9)	5e-3	128

Table C.2.: MLP architecture used for the attack [2].

Dataset	Leakage Model	Architectures	Learning Rate	Batch Size
ASCAD_f	HW	FC(496, 496, 136, 288, 552, 408, 232, 856), SM(9)	5e-4	32
	ID	FC(160, 160, 624, 776, 328, 968), SM(256)	1e-4	32
ASCAD_r	HW	FC(200, 200, 304, 832, 176, 872, 608, 512), SM(9)	5e-4	32
	ID	FC(256, 256, 296, 840, 280, 568, 672), SM(256)	5e-4	32
CHES_CTF	HW	FC(192, 192, 616, 248, 440), SM(9)	1e-3	32

ACKNOWLEDGEMENTS

This PhD has been one of the most rewarding adventures of my life, bringing me joy through research and the privilege of working alongside incredible people. I am deeply grateful to everyone who has been part of this unforgettable journey.

First and foremost, I want to express my immense gratitude to my promotor, Prof.dr.ir. Inald Lagendijk, and copromotor, Dr. Stjepan Picek. Inald, thank you for your guidance and insightful questions, which consistently challenged and refined my research. I deeply appreciate your support over the years, especially during the final stages of this thesis. Stjepan, thank you for entrusting me with this PhD opportunity and for your continuous encouragement and support. Your expertise, passion, and dedication to research have been truly inspiring, and it has been a privilege to work with you. I am grateful to have gotten to know you outside of work. Your fun and vibrant personality made exploring new places and surfing together incredibly enjoyable. Thank you for being an exceptional mentor and for your patience with my 'CroEnglish'.

I would also like to express my sincere thanks to the committee members, Prof.dr. Peter Bosman, Prof.dr.ir. Nele Mentens, Prof.dr.ir. Patrick Schaumont, Prof.dr. Georgios Smaragdakis, and Dr.ir. Richard Hendriks, for taking the time to review my thesis and participate in the defense ceremony.

This PhD was made possible through collaboration with STMicroelectronics (ST), so I extend my thanks to everyone involved in the project. Christophe Laurencin, thank you for your support, especially during the challenging times amidst organizational changes and the disruptions caused by the pandemic. Pierre-Yvan Liardet, thank you for your encouraging words and the valuable ideas you shared that helped shape the trajectory of my PhD research. Daniele Fronte, I deeply appreciate your guidance during the initial phase of integrating and running experiments at ST; your expertise ensured everything ran smoothly. Thank you for the insightful discussions that enhanced my work. Thomas Ordas, thank you for your feedback and support throughout the years. I enjoyed working with you, and I believe our effective communication was key to the success of this project, helping us balance the objectives of ST while producing publishable work for my PhD. I am especially thankful for your help in realizing my ambitious experimental schedules during my visits to ST. My further thanks go to Yanis Linge, Valère Dejardin, Christian Cornesse, Ibrahima Diop, Benoit Feix, Simon Landry, Davide Poggi, and other ST colleagues. Your engaging discussions and thought-provoking questions have greatly contributed to the progress of the GAIA project. Thank you all for your warm welcome and for introducing me to the wonderful world of *pétanque*.

I want to express my gratitude to my collaborators. Thank you, Domagoj Jakobović and Marko Đurasević, for putting in a good word for me with Stjepan, which paved the way for my PhD. Domagoj, I am grateful for your supervision during my projects at FER, which sparked my interest in research and pursuing a PhD. Marko, your advice and our

discussions at AVL were greatly appreciated and provided further encouragement on this path. I am very grateful for my close collaboration with Guilherme Perin and Lichao Wu. I learned so much from both of you, and your dedication and expertise were a constant source of motivation and inspiration. Lejla Batina, your advocacy for women in science is truly inspiring, and I am sincerely grateful for your kind words and advice. Luca Mariot and Łukasz Chmielewski, thank you for your invaluable input on several of my projects. I also extend my sincere thanks to Huimin Li, Léo Weissbart, Servio Paguada, Unai Rioja Sabando, Sengim Karayalcin, Ioana Savu, and Daan van der Valk, with whom I had the pleasure of collaborating. Mauro Conti, Antonino Nocera, Marco Arazzi, Stefanos Koffas, and Jing Xu, thank you for giving me the opportunity to explore a different research topic.

I had the enriching opportunity to visit NTU Singapore and collaborate with Shivam Bhasin, Dirmanto Jap, and Prasanna Ravi. Thank you all for your warm welcome. Our discussions and collaborative work were truly enjoyable, and I am grateful for the knowledge and insights I gained. I am inspired by the work your group produces and hope we can collaborate again in the future. Special thanks to you, Dirmanto, for being my guide and fellow explorer in Singapore! Trevor Yap, it was a pleasure meeting you during your visit to the Netherlands.

Furthermore, I extend my thanks to the entire CYS group at TU Delft. Sandra Wolff, thank you for brightening the often gloomy atmosphere of worried PhD students and for your assistance with the administrative aspects of the PhD. Georgios Smaragdakis, I appreciate the positive atmosphere you fostered in the group and your understanding and support during my final year. Sicco Verwer, thank you for serving on my Go/No-Go committee and offering valuable advice on academic life. I offer my thanks to Zekeriya Erkin, Kaitai Liang, Alexios Voulimeneas, Giovane Moura, Mark Luchs, Laurens Bliek, and Christian Hammerschmidt for all the enjoyable conversations. Gamze Tillem, Chibuike Ugwuoke, Harm Griffioen, and Vincent Ghiette, thank you for the warm welcome during my first year. Jelle and Daniël Vos, Clinton Cao, Stefano Cecconello, Florine Dekker, Bartosz Czaszyński, and Roland Kromes, I appreciate the engaging coffee break discussions, where the topics often seemed as complex as our research. Curious minds like yours never rest! Stefano, I am also grateful for your POS machine projects, which gave me a sense of contribution when mine were on hold. Lastly, I express my heartfelt thanks to Ruud de Jong, whose assistance played a significant role in conducting my research more efficiently and with less stress.

I want to thank the AISyLab members for creating a wonderful atmosphere during our meetings and outings. Abraham, Behrad, Praveen, Zakaria, Susanta, Vipul, Alexandros, Achilleas, Maikel, Jorai, Maurits, and Radinka—thank you for sharing your interesting research challenges and findings. A special thanks to Huimin, Jing, and Stefanos for being awesome office mates. Huimin and Jing, thank you for your kindness, our fun trips, and for normalizing post-lunch power naps. Working with you both was a pleasure. Stefanos, thank you for indulging me in *One Piece* conversations, even though I only started watching it at the age of 'too old'. I hope you have managed to clear out your browser tabs, and one day, I aspire to reach your level of work focus. Léo, thank you for your help with SCA, and I apologize for my disastrous attempt at teaching bachata! Gorka, thank you for sharing your enthusiasm for research with the group and your CV template. Azade and Luca, thank you for the unforgettable full-day bike ride from Delft to

Haarlem.

To everyone I met during conferences and work trips, thank you for making these experiences far more enjoyable than expected. Ngoc Khanh, thank you for pushing me out of my comfort zone—though I am still waiting for that rejection-inspired music you promised! Krijn, thank you for reviewing ChatGPT's Dutch translation of my thesis summary. Thanks also for being a great conference buddy, supervising my "second PhD", and for all the great book recommendations. Estuardo, thank you for being my bucket list partner. I look forward to hearing more of your wild stories. Yanis, thank you for supporting me during my first karaoke attempt. Monika, gossiping in our blend of Croatian and Macedonian was a blast. A huge thank you to Michiel, Łukasz, Roman, Julius, Servio, Lorenzo, Amber, Thomas W., Asmita, Tu, Alexandre, Charlotte, Konstantina, Shahram, Parisa, Valerie, Thomas, Michael, Jonathan, Gustavo and Lorenz—it was a pleasure getting to know you all.

I am deeply thankful to the friends who made life in a new city less lonely and truly exciting. Ozzy, thanks for being awesome. Our Berlin trip was a blast, and I cannot wait to hit the slopes with you again. Your advice on both career and life is always appreciated. Miray, thanks for enlightening me on the origins of Greek, or should I say, Turkish yogurt and for sharing your insights into human psychology. Azqa, thank you for your "encouragement" in solo traveling and all the answers to my numerous questions. I am grateful to both of you for the much-needed girls' nights filled with gossip, laughter, and great advice. Maja, thank you for following in my footsteps to the Netherlands—it was great having someone familiar nearby. Samuel, thanks for being one of the rare people (outside of work) who understood my PhD topic just from the keywords! I am grateful for the TU Delft game, which introduced me to Ivan P. and Dajana. I admire your passion for your work and always learn something new from you while having a great time. Dajana, thanks also for being my dancing buddy—it was the perfect escape during thesis writing.

I am deeply grateful for the shared software development background with my boyfriend, Ivan. My implementation would be in much worse shape without our discussions. Thank you not only for your willingness to understand and help but also for just being there to listen. Your support and encouragement are truly appreciated.

To my friends in Croatia, thank you for your support through video calls and visits. Thank you for keeping my sanity during the pandemic, enabled by the Croatian lack of lockdowns. My dearest friends, Antonija, Vedran, Vana, Gabriel, Andrea, Dario, Valentina, Amir, Anna, Nicolette, Renato, and Matija, I am also sorry for missing too many important life events over the years. I extend my thanks to a few more friends who helped enhance my leisure time so my research can be more focused—Mislav, Filip, Luka T., Ivan's friends, Matija's adventure crew, Renato's escape room buddies, choir friends and "various IT companies lounge".

Finally, my deepest gratitude goes to my parents, brother, and the entirety of my extended family. Without their support and the values they have instilled in me, I am unsure whether this thesis would have seen the light of day. I am forever grateful for your love, unwavering support, and encouragement. A special thank you to my cats, M and Misty, for their furry support whenever they had the chance.

CURRICULUM VITÆ

Marina KRČEK

Marina Krček, born in Zagreb, Croatia, on February 4, 1994, holds bachelor's and master's degrees in Computer Science from the University of Zagreb, earned in 2015 and 2017, respectively. Following her master's studies, Marina worked as a software development engineer at AVL-AST in Zagreb, contributing to the development and maintenance of software products for post-processing and analyzing automotive test data.

In November 2019, Marina started as a PhD student, joining the Cyber Security group at Delft University of Technology under the supervision of Prof. dr. ir. R.L. Lagendijk and Dr. Stjepan Picek. Her doctoral research, funded by STMicroelectronics under the GAIA project, focused on improving the fault injection parameter search using artificial intelligence, particularly evolutionary algorithms. Concurrently, she also worked on deep learning side-channel analysis. She presented her findings at several international workshops and conferences. Throughout her PhD, Marina was part of organization teams for several workshops and conferences and served as a reviewer for numerous journals and conferences. Additionally, she provided valuable support for 'System Security' and 'Algebra and Cryptography' courses at TU Delft and supervised master's and bachelor's students.

List of Publications

Published

11. M. Krček and T. Ordas. "Diversity Algorithms for Laser Fault Injection". In: *Applied Cryptography and Network Security Workshops*. Ed. by M. Andreoni. Cham: Springer Nature Switzerland, 2024, pp. 121–138. ISBN: 978-3-031-61486-6

10. I. Savu, M. Krček, G. Perin, L. Wu, and S. Picek. "The need for MORE: unsupervised side-channel analysis with single network training and multi-output regression". In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2024, pp. 113–132

9. 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* 18 (2023), pp. 3849–3861. DOI: 10.1109/TIFS.2023.3287728

8. M. Krček and G. Perin. "Autoencoder-enabled model portability for reducing hyperparameter tuning efforts in side-channel analysis". In: *Journal of Cryptographic Engineering* (2023), pp. 1–23. DOI: 10.1007/s13389-023-00330-4

7. C. Coello Coello, M. Durasevic, D. Jakobovic, M. Krček, L. Mariot, and S. Picek. "Modeling Strong Physically Unclonable Functions with Metaheuristics". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 2023, pp. 719–722

6. M. Krček. "What Do You See? Transforming Fault Injection Target Characterizations". In: *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2022, pp. 165–184

5. M. Krček, T. Ordas, D. Fronte, and S. Picek. "The more you know: Improving laser fault injection with prior knowledge". In: *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2022, pp. 18–29

4. M. Krček, H. Li, S. Paguada, U. Rioja, L. Wu, G. Perin, and Ł. Chmielewski. "Deep learning on side-channel analysis". In: *Security and Artificial Intelligence: A Crossdisciplinary Approach*. Springer, 2022, pp. 48–71

3. M. Krček, D. Fronte, and S. Picek. "On the importance of initial solutions selection in fault injection". In: *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2021, pp. 1–12

2. D. van der Valk, M. Krček, S. Picek, and S. Bhasin. "Learning from a big brother-mimicking neural networks in profiled side-channel analysis". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6

1. H. Li, M. Krček, and G. Perin. "A Comparison of Weight Initializers in Deep Learning-Based Side-Channel Analysis". In: *Applied Cryptography and Network Security Workshops*. Ed. by J. Zhou, M. Conti, C. M. Ahmed, M. H. Au, L. Batina, Z. Li, J. Lin, E. Losiouk, B. Luo, S. Majumdar, W. Meng, M. Ochoa, S. Picek, G. Portokalidis, C. Wang, and K. Zhang. Cham: Springer International Publishing, 2020, pp. 126–143. ISBN: 978-3-030-61638-0

UNDER REVIEW

3. M. Arazzi, M. Conti, S. Koffas, M. Krček, A. Nocera, S. Picek, and J. Xu. "BlindSage: Label Inference Attacks against Node-level Vertical Federated Graph Neural Networks". In: *arXiv preprint arXiv:2308.02465* (2023)

2. S. Karayalcin, M. Krček, L. Wu, S. Picek, and G. Perin. *It's a Kind of Magic: A Novel Conditional GAN Framework for Efficient Profiling Side-channel Analysis*. Cryptology ePrint Archive, Paper 2023/1108. 2023. URL: `https://eprint.iacr.org/2023/1108`

1. M. Krček, L. Wu, G. Perin, and S. Picek. *Shift-invariance Robustness of Convolutional Neural Networks in Side-channel Analysis*. Cryptology ePrint Archive, Paper 2023/1100. 2023. URL: `https://eprint.iacr.org/2023/1100`