BERGISCHE
UNIVERSITÄT
WUPPERTAL

Doctoral Dissertation

How Machine Learning Enables Automated Side-Channel Detection

Jan Peter Drees, M.Sc.

April 19, 2023

Submitted to the
School of Electrical, Information and Media Engineering
University of Wuppertal

for the degree of
Doktor-Ingenieur (Dr.-Ing.)

Jan Peter Drees
Place of birth: Coesfeld, Germany
Author's contact information:
jdrees@mailbox.org

Thesis Advisor:	**Prof. Dr.-Ing. Tibor Jager** University of Wuppertal, Wuppertal, Germany
Second Examiner:	**Prof. Dr.-Ing. Juraj Somorovsky** Paderborn University, Paderborn, Germany
Thesis submitted:	April 19, 2023
Thesis defense:	July 12, 2023
Last revision:	November 15, 2023

Acknowledgements

First, I would like to thank my advisor Tibor Jager for convincing me to pursue Cryptography by making me an offer I could not refuse. You supported me every step of the way and created a working environment for experimentation and growth. I would like to extend my deepest gratitude to my collaborator Pritha Gupta, without whom none of this would have been possible. In addition to teaching me the arcane magic that is machine learning, you supported me as a colleague and as a friend. I would further like to thank my other coauthors Juraj Somorovsky, Eyke Hüllermeier, Arunselvan Ramaswamy, Claudia Priesterjahn, and Alexander Konze. Additionally, I would like to thank Karlson Pfanschmidt, Alexander Tornede, Gabriel Zaid, Łukasz Chmielewski, Maikel Kerkhof, Guilherme Perin, and Stjepan Picek for their valuable and helpful suggestions on hardware side channels. My thanks also extend to Björn Haddenhorst and Vitalik Melnikov for their support with the side-channel detection theory. The help of the transport layer security (TLS) attacker community was also much appreciated, especially Robert Merget who helped create the TLS attacker Bleichenbacher client. Special thanks go to Anastasija Berlinblau, Dennis Funke, and Varun Nandkumar Golani, who helped by contributing experiments, code improvements, and datasets as part of their theses. Further, my colleagues at the ITSC group were instrumental in helping me navigate the maze of research and teaching, and as such I want to thank Jutta, Kai, Máté, Lin, David, Marloes, Pascal, Denis, Amin, Tobias, Raphael, Jonas, Gareth, Peter, Saqib, and Rafael. Experiments were performed on resources provided by the Paderborn Center for Parallel Computing (PC2).

Außerdem möchte ich meiner Familie und meinen Freunden danken, ohne die ich diese Dissertation nicht hätte schreiben können. Allen voran Aida, die meinen Fokus im entscheidenden Moment geschärft hat und die trotz der Anstrengungen für mich da war. Außerdem meinen Eltern Ute und Georg, die schon als Kind meine Neugier gefördert und meinen Wissensdurst gestillt haben. Ich danke ihnen und meiner ganzen Familie für ein allzeit offenes Ohr, auch wenn ich mal wieder nicht mit dem Reden aufhören konnte. Dazu danke ich denjenigen, die mich bei meiner Promotion unterstützt haben, unter anderem Helena, Jonas, Stefan und Tim, sowie allen anderen Flunkyballern. Zuletzt möchte ich allen danken, die mich auf meinem Weg begleitet haben und die ich hier nicht genannt habe, deren Unterstützung ich aber trotzdem nicht missen möchte.

Contents

1 Introduction

In a world where digital communication has become ubiquitous, protecting it against malicious actors is as important as ever. This is where modern cryptography shines, using authentication and encryption methods to protect data during storage and while being transmitted, from online banking credentials to company secrets and sensitive personal data. Using state-of-the-art cryptographic protocols like TLS, the vast majority of the communication online is already protected against eavesdropping during transport [Goo23]. The capabilities of current cryptographic protocols to resist these attacks are reflected in their rigorous mathematical security proofs, making it increasingly infeasible to break the protocol outright. However, all these proofs have to rely on at least some simplifying assumptions, which do not necessarily hold when deployed in actual implementations running on physical hardware [DPW11]. This means that an attacker can try to gather information via channels that are not considered in security proofs. This could be as simple as trying to recover a user's password by listening to the sound of the keystrokes of a user typing it in [ZZT09] or as complex as measuring the power consumption of a server's cryptography chip to recover the secret key used inside [CRR03]. All of these would be considered side-channel attacks, as information necessary for the attack is gathered not by reading the encrypted messages on the "main" channel but instead by some other unintended *side*-channel not considered in security proofs.

Remote side channels Side channels are dangerous because they can be used to attack implementations of cryptographic protocols which have been proven secure. One of these attacks is Bleichenbacher's attack [Ble98], which uses information gathered from side channels like error messages returned by a web server to break Rivest–Shamir–Adleman (RSA) encryption. It allows the decryption of data secured with the transport layer security (TLS) protocol and can be executed fully remotely over the internet. While a security proof for the RSA key exchange used in TLS exists [JKM18], any implementation needs to take care that it does not inadvertently leak the information necessary for Bleichenbacher's attack through a side channel. Many of the recent attacks on TLS used a side channel, circumventing its security guarantees [Avi+16; BSY18; Mer+21; MDK14].

Hardware side channels In contrast to remote side channels like the error messages used by Bleichenbacher [Ble98], local side channels target the hardware running the cryptographic algorithms. This is done either via fine-grained power

consumption measurement or by placing an electromagnetic probe within a few millimeters of the chip executing the software. This type of attack is usually not geared towards attacking consumer hardware but high-value targets like military equipment, hardware security modules used in servers, or smart cards. These devices are designed with the premise of adversaries gaining physical access to them, so recovering secret keys inside them needs to be infeasible even with local access. In this area, hardware side-channel attacks are a big threat, as only a handful of measurements of cryptographic operations on the targeted device can be sufficient to recover the secret keys [Pic+23].

Scanning for side channels Consequently, detecting and removing side channels in the various implementations of cryptographic protocols is a big priority. In the areas of remote side channels, this is usually done by scanning servers for their reaction to inputs chosen by IT security researchers. For example, Böck, Somorovsky, and Young [BSY18] conducted scans for Bleichenbacher side channels. Their input was deliberately designed to confront the implementation with unexpected data or behavior, exposing issues with error handling. They then manually analyse the reaction of the server for variations that would indicate possible side channels, which they found both in the form of TLS alert messages, as expected, but also in the TCP timeout and disconnect behavior of the servers.

The state of detection That these vulnerabilities were discovered 20 years after the initial publication of the attack by Bleichenbacher [Ble98] highlights the challenges faced by IT companies in avoiding side channels. While the testing of functional requirements is the de-facto standard in the software development process, e.g., automated regression testing in continuous integration (CI) pipelines, the testing for security issues can be more difficult to achieve. The manual, time-intensive, and expensive investigation, usually done by outside IT security experts, necessary to check cryptographic software for side channels, appears thoroughly out of date in comparison. That does not mean that automation is impossible: After Böck, Somorovsky, and Young [BSY18] had discovered which new inputs triggered the side channel in TLS servers, and they knew which parts of the server behavior to watch out for, they were able to provide a script that tests for these specific side channels. This script was integrated into the online TLS checker from Secure Sockets Layer (SSL) labs[1], which made it much more accessible to website operators with limited knowledge of cryptography. This example shows how automated side-channel detection mechanisms are helpful tools in testing existing implementations and during software testing as part of the development process. However, it should also be pointed out that such tools are rarely provided and many side channels are only reported to hardware and software manufacturers. This means users of devices may be unable to determine if they are affected and if

[1] `https://www.ssllabs.com/ssltest/`

updates provided by the manufacturer are actually effective in removing the side channel.

The automation gap Based on these observations, we can identify a discrepancy between the needs of software developers working on cryptographic implementations, the users of these cryptographic implementations, and the tools provided by cryptographers working on side-channel attacks. While it is certainly not possible to reliably detect all side channels in an automated way, such testing might be feasible for certain remote side channels. Automated side-channel detection tools would have to address the knowledge gap between software developers and cryptographers and be easy to use. If these tools require no human intervention, this would even allow integration into standard testing toolboxes and CI pipelines.

Goals of this thesis This thesis aims to achieve the following

- to formalize side-channel detection as an information leakage problem
- to explore the connection between machine learning algorithms and information leakage detection
- to propose automated solutions for side-channel detection
- to implement these solutions and apply them to remote side channels
- to identify and explore additional automation steps necessary for hardware side channels and assess their impact

1.1 State of the Art

Some automated detection methods for side channels exist but are limited in scope to specific use cases.

For example, to detect the Bleichenbacher side channels in the ROBOT attacks, Böck, Somorovsky, and Young [BSY18] used the statistical tests Fisher's exact test and chi-squared test to detect the side channels of specific aspects of the TLS server behavior. Current versions of the TLS attacker [Som16] are also able to check for specific Bleichenbacher side channels using a simpler fingerprinting method where any deviation in the fingerprint is interpreted as a side-channel vulnerability. One advantage of TLS attacker is the flexibility of the TLS client that supports various ways to manipulate the padded messages and to deviate from the usual handshake routine (as exploited by the ROBOT attacks). A drawback of these approaches is the inability to combine behavior features as each feature is tested independently.

When we look beyond the task of detecting side channels, fuzzing can be a helpful complementary method. In fuzzing, the software is confronted with

automatically generated invalid inputs and monitored for undesirable behavior like crashes. This was contributed to by de Ruiter and Poll [dP15] with their protocol state fuzzing of TLS. Their approach uses machine learning to recreate the internal state machine of TLS implementations based on the server's reaction to fuzzed input messages. While this would not detect all side channels, past Bleichenbacher side channels like ROBOT have often been created by improper error handling. Thus, removing unintended states and code paths discovered with this fuzzing technique removes possible sources of leakage and could already go some way to preventing side channels in the first place.

A recent development is DL-LA, a method created by Moos, Wegener, and Moradi [MWM21] to detect information leakage in hardware datasets using a deep learning classifier. Developed concurrently and independently of our approaches for remote side channels, their method also uses a machine learning classifier to detect leakage. It is able to detect arbitrary information leakage and is able to combine different behavioral features. However, their underlying assumption relies on a machine learning model being able to outperform a classifier that guesses randomly. In practice, this restricts their approach to balanced datasets.

In the hardware side-channel attack community, most attacks require careful crafting of the attack model architecture. This has recently inspired Wu, Perin, and Picek [WPP20] and Rijsdijk, Wu, Perin, and Picek [Rij+21] to create a white-box approach that automates this architecture creation. For our purposes, this is unsuitable, as the returned performance estimate that we could use to detect leakage is optimistically biased. Acharya, Ganji, and Forte [AGF22] recently proposed a different, black-box approach for automated hardware attacks. Their method deviates from the usual attack methodology as they don't use a single model but simultaneously optimize several models.

1.2 Our Contributions

We advance the state of the art in the following areas:

New Techniques for Side-Channel Detection with Machine Learning When it comes to the detection of local side channels, machine learning has been successfully used to improve attacks in the last few years. However, approaches to the automated *detection* of side-channel vulnerabilities have been lacking. Some statistical approaches have been used successfully, e.g. for the ROBOT attacks [BSY18], but they are limited in flexibility.

We provide a formalization of side-channel detection as an information leakage problem and propose a novel approach for the detection of information leakage using machine learning. This approach is the first to use the performance of the machine learning algorithm itself for *fully automated detection of remote side channels*. We also propose variations that are able to handle imbalanced datasets properly by comparing the algorithm performance to classifiers that reflect the

imbalance. In addition, we increase the resistance of our approach to noise and variations in side-channel behavior by employing an ensemble of machine learning algorithms and aggregating their results. We demonstrate that our approach outperforms state-of-the-art approaches experimentally, both on synthetic data and on real-world remote side channel datasets.

Application to Bleichenbacher's Attack We apply our automated approach to detect side channels in cryptographic protocol implementations, which automatically detects general *patterns* in network protocol traffic that might give rise to a padding oracle. We consider Bleichenbacher side-channel vulnerabilities in TLS as our specific use case, with the approach itself being transferable to other side channels. Over the previous decades, many research papers appearing at leading academic security conferences [Ble98; KPR03; JSS12; Deg+12; Bar+12; Mey+14; Zha+14; Avi+16; Fel+18; BSY18; Ron+19] showed that such vulnerabilities appear again and again in popular open-source software and widely-used commercial products. This makes the detection of such side channels with an automated tool especially useful, as issues with Bleichenbacher-like side channels can be spotted easily during the development process before the large-scale deployment of a vulnerable implementation.

Existing solutions using manual detection scripts are limited in capability to specific, known side channels. General tools like the TLS attacker are more capable in this regard, but still only consider certain behavioral elements of a TLS server, considering each element in isolation.

We implement and analyse our approach in *a new software solution* (the open-source AutoSCA-tool) which is able to analyse a given TLS server implementation automatically. Our solution is the first which relies entirely on the ability to learn patterns of interest for the specific implementation without being restricted to single features or the predictions of a pre-trained model. It is also able to provide an easy-to-understand side-channel assessment for software engineers.

We confirm that the AutoSCA-tool is reliably able to detect known vulnerabilities in TLS server implementations, like the bad version oracle in OpenSSL [KPR03] or the ROBOT vulnerabilities in commercial TLS appliances [BSY18]. We also used the tool to scan the most visited websites according to the Alexa Top 500 ranking and notified the operators of vulnerable servers. We additionally verify that recent versions of 13 different popular open-source TLS implementations contain no detectable side-channel vulnerabilities.

Automating Hardware Attacks In the area of attacks using local side channels, such as power consumption or electromagnetic emissions, machine learning has repeatedly demonstrated its usefulness. Hardware side-channel detection can be achieved by running the attack and checking if it is able to succeed. This is in contrast to padding oracle side-channel attacks, which can be ruined by a single false positive and for which the necessary padding oracle implementations

would have to be created individually for different leakages. However, the attacks themselves are far more difficult, as the measurements usually contain a lot of noise and their dependency on secret key material is highly complex. As such, fully automating the attacks themselves is the necessary next step to automated side-channel detection. While the most powerful attacks already use deep learning, the architecture used for the attack is hand-tailored to each dataset.

We use neural architecture search (NAS) to create a novel approach to hardware attacks that fully automates the crucial step of architecture design. In contrast to recent works by Wu, Perin, and Picek [WPP20] and Rijsdijk, Wu, Perin, and Picek [Rij+21], our approach is *fully black-box*, which means it returns a realistic estimate of the attack performance. Our method is also designed such that the attack is executed not only once but several times with different train-validate splits and on independent parts of the test database, resulting in a more substantial performance evaluation. We additionally performed a big parameter study, enabling the first comparison of different search strategies (including GREEDY and HYPERBAND) and dataset input shapes on 10 benchmark hardware datasets. The results demonstrate that our approach is on par with fixed architectures from Benadjila, Prouff, Strullu, Cagli, and Dumas [Ben+20] and Zaid, Bossuet, Habrard, and Venelli [Zai+19], even outperforming them on some datasets.

1.3 Outline

The remainder of the thesis is structured in the following way:

Chapter 2: Fundamentals This chapter introduces the general concept of side channels as well as some more specific details on padding side channels and hardware side channels. This is followed by a high-level introduction to the most important concepts of machine learning, as well as a definition of the notation used throughout the thesis.

Chapter 3: New Techniques for Side-Channel Detection with Machine Learning This chapter lays the theoretical foundation for automated side-channel detection using machine learning. It introduces the general method we created, as well as an empirical comparison of different ways to implement it using synthetic and real-world datasets.

Chapter 4: Application to Bleichenbacher's Attack This chapter uses the method presented in the previous chapter to create automated detection software for Bleichenbacher-like side channels. We use the software to test various TLS servers in lab environments and on the web, demonstrating its usefulness.

Chapter 5: Automating Hardware Attacks This chapter investigates automating hardware side-channel attack and detection. To this end we propose a novel

approach that uses neural architecture search (NAS) to automate the process of architecture selection necessary for applying deep learning, achieving full automation. We follow this up with a detailed parameter study on which combination of NAS search strategy and input shape performs the best on the 10 reference datasets we gathered and compare our approach to fixed architecture baselines.

Chapter 6: Conclusion and Outlook This chapter discusses the impact of this thesis on automated side-channel detection and its future development.

1.4 Publication Overview

The foundation for this thesis are the following papers. Each of them is discussed in more detail in the respective chapter. The following publications have been published in peer-reviewed conference proceedings:

Jan Peter Drees, Pritha Gupta, Eyke Hüllermeier, Tibor Jager, Alexander Konze, Claudia Priesterjahn, Arunselvan Ramaswamy, and Juraj Somorovsky. "Automated Detection of Side Channels in Cryptographic Protocols: DROWN the ROBOTs!" In: *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*. AISec '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 169–180. DOI: 10.1145/3474369.3486868

Pritha Gupta, Arunselvan Ramaswamy, Jan Peter Drees, Eyke Hüllermeier, Claudia Priesterjahn, and Tibor Jager. "Automated Information Leakage Detection: A New Method Combining Machine Learning and Hypothesis Testing with an Application to Side-channel Detection in Cryptographic Protocols." In: *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*. ICAART '22. Setúbal, Portugal: SciTePress, 2022, pp. 152–163. DOI: 10.5220/0010793000003116

Pritha Gupta, Jan Peter Drees, and Eyke Hüllermeier. "Automated Side-Channel Attacks Using Black-Box Neural Architecture Search". In: *Proceedings of the 18th International Conference on Availability, Reliability and Security*. ARES '23. Benevento, Italy: Association for Computing Machinery, 2023. DOI: 10.1145/3600160.3600161

2 Fundamentals

In this chapter, we introduce the fundamental concept of side-channel attacks in Section 2.1 before diving into relevant examples and related work for remote side channels in Section 2.1.1 as well as for local side channels in Section 2.1.2. Due to the considerable amount of machine learning (ML) methods applied throughout this thesis, Section 2.2 contains a very high-level introduction to the central concepts, using an application example to illustrate them. Section 2.3 then follows up on this by more formally defining these methods along with the notation used throughout this thesis. We conclude the necessary fundamentals by covering the statistical tests and analysis methods used in our approaches in Section 2.4.

2.1 Side-Channel Attacks

A side channel consists of (unintended) leakage of sensitive information from a computing device. A side-channel attack uses the leaked information to reconstruct the original, sensitive information. Such attacks were initially used by governments to spy on each other, for example when NATO (in the TEMPEST programs) and Soviet governments (in the PEMIN programs) used the electromagnetic emissions from cipher machines to decrypt important diplomatic communication during the early cold war [Eas21].

Nomenclature For our purposes, we need to differentiate between distinct components: A side-channel attack describes an algorithm that uses access to a formally defined *oracle* to output some desirable secret, like a reconstructed secret key, the decryption of a ciphertext, or a forged signature. In the context of Bleichenbacher's attack, the oracle gets an RSA-encrypted ciphertext as input and returns whether the contained message is properly formatted according to the PKCS#1v1.5 padding standard. Since this distinction is only possible with access to the plaintext, this is leaking *secret information*. As such, the oracle needs to be implemented by using the *side channel* in a given protocol *implementation*. A transport layer security (TLS) server returning different TLS alert messages depending on the correctness of the padding contains a possible Bleichenbacher side channel. The oracle would use the side channel by executing a new TLS handshake with the TLS server, using the input ciphertext in the client key exchange message. It would then observe the server's reaction and depending on the TLS alert message return "padding correct" or "padding incorrect" as the output. Bleichenbacher's attack then queries the oracle repeatedly with manipulated ciphertexts as the

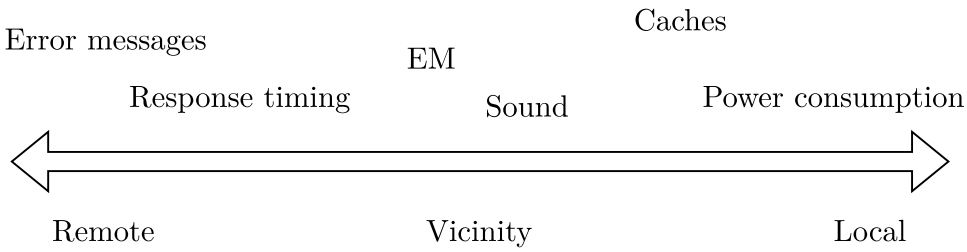


Figure 2.1: Examples of side channels and their classification according to [Spr+17]

input, gathering enough secret information to be able to return the plaintext or a forged signature after a few thousand queries.

Evolution of attacks This distinction between the side-channel *attack*, the *oracle*, the *side channel* itself, and the *implementation* can be helpful in understanding the properties of different attacks. For example, it is sufficient to remove the side channel in a particular implementation to eliminate the risk posed by the attack entirely. In turn, improvements to an oracle or the attack itself (like the ROBOT improvements [BSY18]) can mean that previously unexploitable or hidden side channels suddenly become an issue. And efficiency improvements to the attack (like the improvements by Bardou *et al.* [Bar+12]) can make a side-channel vulnerability exploitable that was previously considered impractical in the real world.

Classification Using the classification system proposed by Spreitzer, Moonsamy, Korak, and Mangard [Spr+17], we can categorize different side channels based on the way in which they can be exploited. Figure 2.1 shows some examples and roughly how we could classify them. On the extreme end of the scale, most power side channels require local physical access to the target device. The electromagnetic (EM) emitted by a device can sometimes be detected in a neighboring building, given the emissions are strong enough and the attacker has a sensitive antenna. However, some side-channel attacks using EM require the attacker to be millimeters away from the chip running the software, exemplifying how the categorization is only broadly possible. On the other end, a Bleichenbacher side channel using TLS alert messages is exploitable remotely by anyone connected to the internet. This means that the threat model of various side channels can differ significantly, from criminals trying to steal login credentials remotely to nation-state actors recovering confidential codes from captured military hardware. Another important distinction to make is that a side-channel attack does not use unintended implementation behavior to access information on the *main communication channel* [Spr+17]. As such attacks using buffer overflows like Heartbleed [Dur+14] are not considered side-channel attacks.

2.1.1 Remote Side Channels

For our purposes, any side channel that can be exploited over the internet can be considered a remote side channel. This category mostly consists of servers returning differing error messages or differences in their timing behavior.

Attacker model For remote side channels, our attacker model usually considers an adversary that has unlimited ability to interact with the implementation by sending messages over the network. This is a reasonable assumption for many protocols, as servers (for example web and mail servers) will respond to all requests essentially indefinitely. Most filtering limiting the traffic from a single source can be circumvented and will be ineffective against an adversary with distributed networks (botnets or virtual private networks (VPNs)) or sufficient patience. The side channel consists of the replies from the implementation, which encompasses both the intended protocol messages as well as the metadata and accompanying network traffic, like TCP packets or timing information. For some side channels, we also assume the attacker has acquired some ciphertexts they want to decrypt in advance, for example by sniffing traffic on the local network or a meddler-in-the-middle attack.

Padding oracles Padding oracle attacks are a special case of side-channel attacks that are usually exploitable remotely. During the encryption process of a short message, additional data called *padding* has to be added such that the resulting plaintext matches the length requirements of the encryption scheme. This is essential when using block ciphers, which can only operate on inputs where the length is a multiple of the block size. Adding randomized padding is also essential (but not necessarily sufficient) to be able to achieve IND-CPA secure encryption from deterministic public-key encryption schemes, like textbook RSA [KOS17]. As such, padding standards describe the method by which the sender adds this additional data and how the receiver removes it. In all of these attacks, the oracle returns the padding validity for a given ciphertext, or more formally, a binary correct/incorrect whether the decryption of the ciphertext conforms to the specification of the padding standard. Although this may seem like leaking only a tiny bit of information, if the attacker repeats the query to the oracle with slightly different variations of the same ciphertext over and over, they can narrow down the possible values of the plaintext until only a single option remains. This effectively allows the attacker to decrypt a given ciphertext without access to the secret key or to forge a signature in the case of Bleichenbacher's attack on RSA. To be able to repeat the queries, the ciphertexts produced by the encryption scheme need to be malleable, which allows the attacker to manipulate a ciphertext in a manner that results in predictable changes to the plaintext. In the case of RSA, this is due to its inherent multiplicative homomorphism, while in the case of the cipher block changing (CBC) mode of operation, a modification of the initialization vector (IV) results in predictable changes to the first block. The most famous attacks in

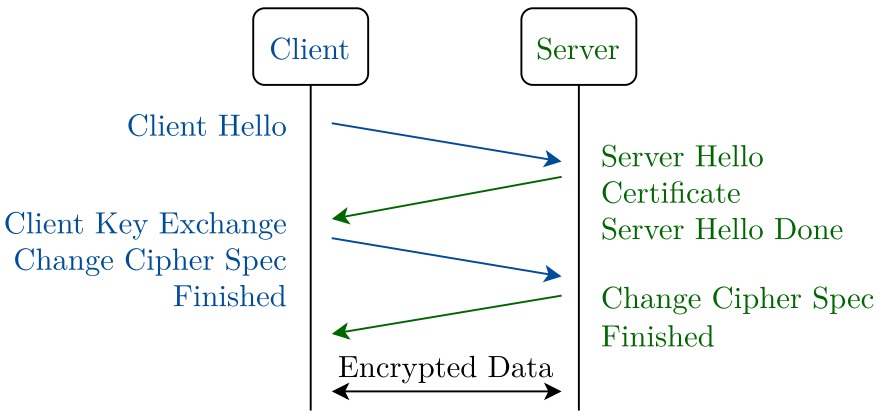


Figure 2.2: A typical TLS 1.2 handshake

this family are Bleichenbacher's attack on PKCS#1v1.5 Rivest–Shamir–Adleman (RSA) padding [Ble98], Vaudenay's attack on CBC padding [Vau02], and Manger's attack on OAEP RSA padding [Man01].

SSL and TLS TLS is perhaps the single most commonly used cryptographic protocol in existence. It is used when browsing the web and is currently supported by the vast majority of websites. Google reports that as of 1st of April 2023, 96 % of web pages visited by German Google Chrome users on Windows were loaded over TLS, although this percentage can vary by platform and country [Goo23]. TLS is a broad standard that can be employed in various situations, from transport encryption for web applications, emails, voice over IP (VoIP) calls, and VPN traffic. We should note here that the very first versions of TLS were called Secure Sockets Layer (SSL), but the protocol standard was renamed when it became an Internet standard in 1999 [DA99]. We use TLS to refer to all versions of the standard including the earlier SSL versions unless specifically noted otherwise.

The RSA key exchange TLS itself does not specify a single choice of cryptographic algorithm for key exchange, symmetric encryption, and integrity check but instead defines possible combinations that a device can implement. A TLS connection goes through the two phases shown in Figure 2.2: The first is the handshake, during which the parties authenticate each other and exchange cryptographic keys. These keys are then used by the transport protocol in the second phase to transmit the actual payload data in a bidirectional session securely. The very first key exchange included in TLS was the RSA key exchange, and several other algorithms have been added since. In the case of TLS key exchange, RSA is used as a public key encryption algorithm, with the TLS client generating a new secret key and encrypting it with the public key of the TLS server. Upon

reception of the encrypted data, the server uses its secret key to decrypt, and both parties are now in possession of the same secret that they use to derive the various authentication and encryption keys for the transport protocol. Another application of RSA is the digital signatures of the certificate of a TLS server, a file that the client uses to verify that a malicious meddler-in-the-middle is not impersonating the website it is connecting to.

The history of Bleichenbacher's attack In this thesis, we mostly consider Bleichenbacher's attack as an example of a remote side-channel attack. The specific workings of the actual attack on RSA, as well as the underlying mathematical properties, are covered in detail in Section 4.3.1. The attack was first published by Bleichenbacher [Ble98], and he estimated that a real-world attack on the RSA key exchange used in TLS would require executing a million TLS handshakes. As such, his attack gained the nickname of "the million message attack". His original attack targeted SSL version 3 [FKK11], but its fundamental logic applies to all protocols using the PKCS#1v1.5 padding method. As a consequence of exposing the issue with PKCS#1 padding, the PKCS#1 standard was updated from the vulnerable padding in version 1.5 [Kal98] to include a different padding type, OAEP, in version 2 [KS98]. However, when TLS version 1.0 [DA99] was standardized as a successor to SSL version 3, the standard did not switch to the new padding type and still requires PKCS#1v1.5 padding for RSA key exchange, all the way up to TLS version 1.2 [DR08]. Instead, the TLS standard was updated to explicitly state "The best way to avoid vulnerability to this attack is to treat incorrectly formatted messages in a manner *indistinguishable* from correctly formatted RSA blocks" in TLS version 1.0 [DA99, Section 7.4.7.1]. In 2003, Klíma, Pokorný, and Rosa published their attack on the RSA key exchange [KPR03], which used a different oracle than Bleichenbacher's attack. They used the behavior of TLS implementations rejecting key exchange messages that contained an invalid TLS version number as a side channel to construct a *bad version oracle*. Since the valid TLS version values were two known bytes, they were able to modify Bleichenbacher's attack to use these bytes in the middle of the message instead of the first two bytes. In the response, TLS version 1.1 expanded its section on handling Bleichenbacher to recommend that implementations should not respond with a TLS alert in case the received TLS version number was incorrect [DR06, Section 7.4.7.1]. When TLS version 1.2 was released in 2008, the section got expanded once again, with two different algorithm descriptions on how to handle incorrect paddings in a way that no timing side channel would result[DR08, Section 7.4.7.1]. However, a decade later there were still new Bleichenbacher side channels found when Böck, Somorovsky, and Young [BSY18] investigated the timeout behavior of closed source TLS servers when facing incomplete TLS handshakes. Their investigation revealed that 27 of the 100 most visited webpages (according to the Alexa ranking[2]) were vulnerable to Bleichenbacher attacks using their new

[2] `https://web.archive.org/web/20180321225122/https://www.alexa.com/topsites`

side channels. In the same year, TLS version 1.3 was standardized, which finally removed the RSA key exchange method altogether, but still allows RSA and the PKCS#1v1.5 padding for signing certificates [Res18]. Even now, many TLS servers still offer TLS version 1.2 or lower for backward compatibility reasons, and F5 reported that as of the end of 2021, 52 % of the surveyed web servers still supported RSA key exchange [WV21].

New Bleichenbacher side-channels and attacks In 2002, Klíma and Rosa [KR03] were able to generalize the attacks of Bleichenbacher and Manger, showing that the generation of a forged RSA signature could be done in all cases where a decryption side-channel attack was present. Later, Jager, Schinzel, and Somorovsky [JSS12] applied Bleichenbacher's attack on the XML encryption used in web services. Bardou, Focardi, Kawamoto, Simionato, Steel, and Tsay [Bar+12] were able to optimize the attack itself and apply it to various hardware implementations, among them the Estonian ID cards. Instead of requiring a "million messages", the improved attack on 1024-bit RSA could now complete with a median of 14500 oracle queries. Meyer, Somorovsky, Weiss, Schwenk, Schinzel, and Tews [Mey+14] found several new side channels in TLS implementations, including two timing side channels large enough to be detectable over a network. Zhang, Juels, Reiter, and Ristenpart [Zha+14] used a cache-based side channel to extract information across virtual machine boundaries in a cloud hosting context. This way, they were able to execute the attack on XML encryption from [JSS12] on a TLS server inside a different virtual machine running on the same physical hardware of a cloud hoster. In 2015, Jager, Schwenk, and Somorovsky [JSS15] were able to impersonate TLS 1.3 and QUIC servers, protocols that do not support the vulnerable RSA key exchange, by using a different server vulnerable to Bleichenbacher's attack, as long as the RSA key was shared between the servers. This meant that a single vulnerable server, for example, one supporting TLS 1.2 as a fallback or an old mail server, could be sufficient to attack users of state-of-the-art cryptography. This was followed by the DROWN attack [Avi+16], which demonstrated that a server supporting outdated cryptography, specifically export cipher suites for SSLv2, could be used to launch Bleichenbacher's attack. They adapted Bleichenbacher's attack such that a comparably easy brute-force attack on the weak export ciphers could be used as a stand-in for the usual side channel, enabling a similar attack on the RSA key exchange. Xiao, Li, Chen, and Zhang [Xia+17] were able to attack a TLS library running in the SGX enclave of Intel CPUs. Their attack used side channels discovered by analyzing the control flow of the TLS library to run Bleichenbacher's attack in a non-enclave process, successfully decrypting the TLS handshake executed by the library in the enclave. As explained above, Böck, Somorovsky, and Young [BSY18] discovered new side channels exposed by executing incomplete TLS handshakes. Felsch, Grothe, Schwenk, Czubak, and Szymanek [Fel+18] further expanded the reach of Bleichenbacher's attack to the IPSec VPN protocol. They attacked the RSA-based IKE handshake used in IPSec.

They were able to transfer a side-channel attack against the RSA key exchange to impersonate hosts using non-RSA key exchange methods that use RSA only for authentication. Ronen, Gillham, Genkin, Shamir, Wong, and Yarom [Ron+19] created another variant of the attack, this time by using microarchitectural side channels exposed by cache timing to execute a downgrade attack on the TLS handshake. Because of the limited time before an in-progress TLS handshake times out, they expanded Bleichenbacher's attack to support parallelized execution. Using this performance improvement, they were able to demonstrate the practical feasibility of their downgrade attacks. Finally, Kelesidis [Kel21] recently proposed further improvements to the attack, reducing the number of necessary oracle queries another 75 % compared to the improved attack in [Bar+12].

2.1.2 Local Side Channels

The term *local*, hardware, power, or physical side-channel attack refers to a class of attacks that rely on close physical proximity to a device to be able to detect the side-channel leakage. This is almost always done by receiving electromagnetic emissions from the device or by measuring its power consumption. These leakage types are sufficiently similar that common attack approaches can be used for both types.

Attacker model For local attacks, our attacker model is different from the one used in remote side channels. The first prerequisite is that the adversary has already obtained a sufficient level of hardware access. For our investigation, we assume the chip running the cryptographic algorithm is in physical possession of the adversary. This is one of the strongest assumptions possible, as this level of access usually means that all software protections, warning systems, etc. are rendered ineffective. However, it is still a reasonable assumption for many high-security applications, especially military hardware that can be captured by an enemy, unattended servers that rely on a trusted platform module (TPM) or secure cryptoprocessor for attestation, or smart cards that can be stolen or obtained by a malicious user. Another usual assumption is that the adversary either chooses or knows the external inputs and outputs of the attacked system [Pic+23]. For our purposes, we assume the attacker chooses a plaintext and sends it as input to the system. The system encrypts it using a secret key contained inside and outputs the associated ciphertext such that the attacker can observe it.

Measurement process In such a situation, an adversary can connect probes for power consumption on the pins of a chip or place probes for electromagnetic radiation within millimeters of the circuitry of the chip. Sometimes the chip packaging is even removed to allow for closer access to the emissions. These probes can then be connected to a dedicated measurement system, often a high-precision oscilloscope, which records the received signal. However, local side

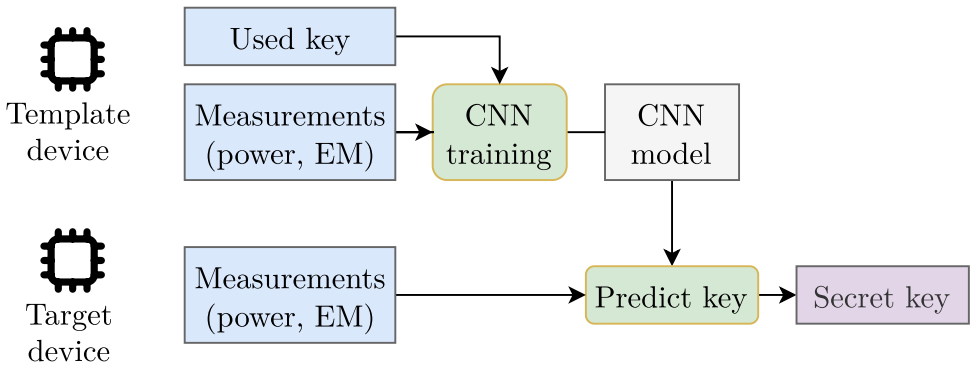


Figure 2.3: High-level concept of a deep learning template attack.

channels can also use more subtle and non-destructive measurement techniques like Hertzbleed [Wan+22], which allows determining the power consumption indirectly by measuring timing remotely and reconstructing the operating frequency of the processor. Other possibilities include the placement of amplifiers for electromagnetic radiation, which allows for attacks from a further distance [Eas21].

Attack scenarios The attack measurements can be divided into individual *traces*, with each trace representing the time-series data of the received signal for one cryptographic operation, e.g., one encryption or one generation of a signature. The first hardware attacks in the research community considered the premise of simple power analysis (SPA), which only uses one trace. In SPA, the attacker has gained access to the target device, measures its power consumption once, and then tries to deduce the secret from this measurement alone. SPA attacks can already be sufficient. Some algorithms like DES or modular exponentiation can result in an obvious pattern of operations that is observable in the trace [KJJ99; CFR10]. However, the attacks can get much more powerful if more than one trace is considered, as now their differences can be used to isolate the influence of the computations on the secret itself. These are called differential power analysis (DPA) and were also introduced by Kocher, Jaffe, and Jun [KJJ99].

Template attacks The biggest step up for hardware attacks was the introduction of *template attacks*. These attacks extend the attacker model and assume that the attacker can gain access to an identical, "cloned" copy of the target device, called the *template device*. Figure 2.3 illustrates this concept for a convolutional neural network (CNN) model. The attacker tries known secrets on the template device, observing the side-channel leakage of thousands of cryptographic computations. This can be used to create a model of how the choice of the secret key influences the side-channel leakage, either by statistical methods or by machine learning, usually deep learning. In the attack phase, this leakage model is then used to

reconstruct the secret used by the actual target device based on a few traces obtained from it. This extension to the attacker model is even stronger, but we can still argue that it applies in many cases. For almost all commercially available devices, including smart cards and TPMs, it can be as easy as buying a second device or ordering the same chip from a manufacturer. One caveat is however necessary: In academic datasets, the template attack is often only simulated and both datasets are actually gathered with the same measurement setup on the same physical device. They, therefore, represent the best case where the template device is virtually identical to the target device, overestimating the performance of attacks in the real world.

Application of machine learning When considering the structure of a template attack, it becomes apparent that the approach lends itself to applying machine learning (as is discussed in depth in Sections 2.2 and 2.3). The *profiling* dataset gathered from the template device becomes the training data and is labeled with the secret used (or some derivation of it). Consequently, the attack dataset gathered from the target device becomes the test dataset, but that requires that this dataset contains labels. In any analysis of an attack method, the actual secret key used in the attack measurements has to be known to enable evaluation of the attack's success. This means that generating labels for the test dataset is straightforward.

Performance metrics As Picek, Perin, Mariot, Wu, and Batina [Pic+23] explain, the performance metrics commonly used in hardware attacks differ significantly from the ones used for machine learning. Instead of averaging the prediction accuracy over several traces, a single prediction score is generated for the whole attack dataset. A single attack trace usually contains too much noise and too little information on the secret, preventing us from making an informed prediction at once. By aggregating these individual predictions the small bias towards the actual secret can be accumulated to give a reliable final prediction for the secret. As such, the reliability of an attack is determined with the *success rate*, which measures how often a given attack is expected to succeed, given a dataset of a certain size. To measure the efficiency, alternative metrics are used which focus on the necessary size of the attack dataset, which is usually the deciding factor in the runtime and real-world feasibility for an attack.

Countermeasures According to Mangard, Oswald, and Popp [MOP08], there are two categories of methods preventing local side-channel attacks, both aiming to decouple the measurable leakage from the secret. The first is hiding, which aims to make the measurable leakage independent of the data being processed. This can be done by ensuring all computations result in the same leakage or by fully randomizing the leakage. However, even with the most careful design, some leakage will remain. The second is masking, which makes the processed (intermediate) data

independent of the secret. One example would be *boolean masking*, which creates the intermediate data by XORing the secret with a random bitstring or *mask*. The computations are then adjusted to account for this, such that the output remains unchanged. As with a one-time pad, this effectively hides the secret from the adversary. Obviously, this relies on the adversary not being able to reconstruct the mask value which needs to contain sufficient entropy. Obtaining this entropy can be difficult on integrated hardware, as the only reliable option, dedicated hardware entropy generators, significantly increase the complexity and power consumption. Another weakness is the process of masking the secret, which in itself can become a target for the side-channel attack. This has been demonstrated before, with attacks first determining the mask value and using this knowledge to attack the actual secret value [Pic+23].

Defense effectiveness Chari, Jutla, Rao, and Rohatgi [Cha+99] developed a theoretic foundation that quantifies how much masking is necessary to achieve security against a certain adversary. Their model assumes the adversary is able to observe a restricted number of data paths in the implementation, giving rise to the notion of *higher-order masking* protecting against adversaries able to observe more than one path. While there have been several successful side-channel attacks against implementations using boolean masking, the most promising current method is *affine masking* as presented in [von01; Fum+11]. The ASCADv2 dataset [Ben+20] combines this technique with boolean masking, and currently, there are no known successful attacks on the dataset [Pic+23].

2.2 Machine Learning

Fundamentally, machine learning (ML) is a field that attempts to create algorithms that can automatically learn from past experiences to make future predictions [Mit97]. This is done by collecting training data, which is used by the learning algorithm to create and optimize a model that makes predictions based on data instead of pre-programmed rules [KBA96]. The following Section is a high-level introduction to this subject, followed by more detailed formal definitions in Section 2.3.

Three broad categories of ML approaches are usually used: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. In this thesis, we only use *supervised learning* methods where the training data consists of example pairs of input and output. The output label corresponds to the "ground truth" output for a given input, such that the algorithm can pick up on patterns and make predictions on unlabeled data.

Application example Let us consider a real-world example to illustrate the different ML concepts: An engineer at a post office is tasked with automating the sorting of mail. Incoming envelopes are scanned and the handwritten address of the

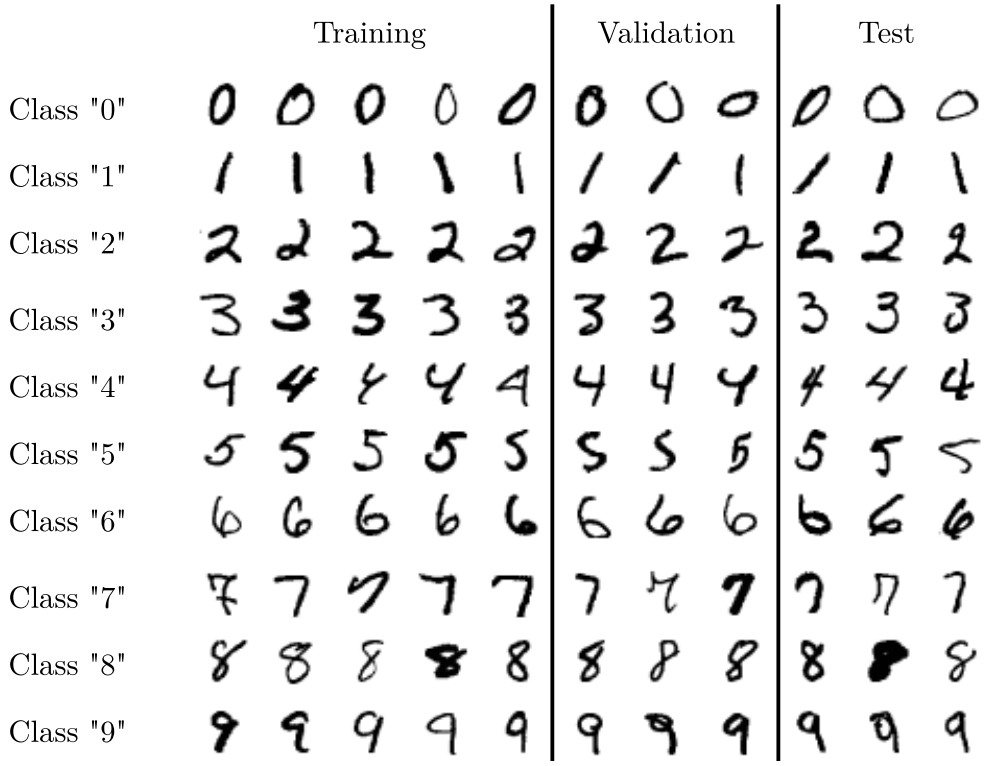


Figure 2.4: An example of the MNIST supervised learning dataset [LeC+98], adapted from [Ste17].

receiver needs to be determined by the automated system. In a supervised learning case, the engineer already has access to a number of scanned envelopes where the characters of the address have been identified manually. Our training dataset (as shown in Figure 2.4) consists of images of different handwritten characters, all labeled with the actual character depicted. The machine learning algorithm is tasked with learning from this data by identifying the similarities within the group of images depicting the same character. It would then return a model that is able to give a prediction for the character contained in an image without access to the associated label. This allows the system to recognize the characters on a new letter arriving at the post office and thus enables sorting the mail automatically.

Classification In the example, we have to define the desired output of the model. Since each character belongs to a distinct category or *class*, we want the model to give a single prediction: "This character is a 6". This type of task consisting of predicting a single class for a given input is called *classification*. In Figure 2.4, we see that the training data is labeled with the 10 classes that are possible when recognizing handwritten numbers. Instead of getting a single prediction, we can also use ML models that output a score for each of the 10 classes representing

how certain the model is that the input belongs to this class. Such outputs are common when dealing with deep learning, e.g. using CNNs. We can transform this output into a single prediction by returning the class with the highest score.

Features Having defined the output, how should the input to the model be represented? In the handwriting case, the original input might be a png file outputted from a scan software, but most machine learning algorithms expect a vector of real values as input, which could be implemented as an array of floats or integers. These individual values are called *features* and transforming the original data into this representation is called feature engineering. Clever representations can lead to fewer features with higher information content, which makes it easier for the algorithm to pick up on what is actually relevant, increasing its performance. In the example, the trivial approach would be to feed each png file as an array of bits into the algorithm. However, this means that the files contain unnecessary information, such as metadata, and that the machine learning algorithm is faced with raw data without any context. A better approach would be to transform the png file into an array of floats representing the grayscale lightness values of each pixel. This way the machine learning algorithm is only confronted with relevant data, and in a format that requires less interpretation. Because the performance of a ML model can depend heavily on feature engineering, it is usually done by experts with some knowledge in the respective application area.

Generalization In our case, we want the algorithm to be able to read the handwriting of many different people, not only the ones we used in our training dataset. This is the big challenge for every machine learning model: *Generalization*. That is, does the model work well on data it has not been exposed to before? For this to happen, the model needs to learn some kind of pattern in the data instead of blindly memorizing and reproducing the training examples. However, the way we train a model is restricted to optimization for the training dataset only. When the model only works well on the training data but performs much worse on other data, *overfitting* occurs. This would be bad for a mail sorting algorithm, as it cannot be trained on the handwriting of every single person sending mail, but is still expected to detect the addresses. Because overfitting is such a big issue, there are many methods to combat it, which we cover in the respective chapters.

Train-validate-test split So how do we evaluate the real-world performance of our model? Instead of directly applying it to route mail, which could be costly if the model makes frequent mistakes, we instead use it on a dataset with known labels. The purpose of this dataset is to assess the ability of the algorithm to generalize, detecting overfitting. For this, the original dataset of labeled images is split into different parts as shown in Figure 2.4. The first part is the training dataset, which is the data the ML model is trained on directly. The second part is the validation dataset on which we determine the performance of the

trained model. We ask the model to predict the labels of the characters in this dataset and compare the prediction to the label in the dataset, which act as the ground truth. Since the trained model has not been exposed to this data before, the performance on this part of the data is a pretty good indicator of the generalizability or usefulness in the real world. The validation data is used to adjust different parameters of the training process, such as learning rate and model complexity, in a process called *hyperparameter optimization*. After adjusting the hyperparameters, we use the third part of the data, or testing dataset, to again apply the model. This "holdout dataset" gives us the final performance estimate, as this data has not been used in the previous process at all.

Hyperparameter optimization As mentioned, the purpose of the validation dataset is to be used as the testing dataset for hyperparameter optimization (HPO). This process is necessary as most machine learning algorithms possess hyperparameters, that is, parameters that influence what kind of model can be created by the algorithm (e.g. the depth of a decision tree or the number of layers in a deep network) and how the model is trained (e.g. the learning rate of gradient-based models). The learning rate determines how much the model should be adjusted in each iteration of the training process, influencing the model training. The optimal choice of this parameter depends on the properties of the dataset as well as choices for other hyperparameters and is therefore impossible to determine in advance. Suboptimal hyperparameter choices can result in less accurate models, or even in the model not performing at all, and as such their tuning is a fundamental task in machine learning [HBF11]. While it would be possible to use a trial-and-error approach to tune these hyperparameters, the fact that a single hyperparameter can usually not be adjusted independently from others means the possible search space for hyperparameter combinations can be very large. Fortunately, this search for a good combination can be treated as an optimization problem for which approximate solutions can be created programmatically. The approach is the following: Pick a hyperparameter combination, train the model on the training dataset with these parameters, and evaluate the model on the validation dataset. This process is repeated for different combinations, exploring the search space of hyperparameter combinations using a search strategy that is guided by the performance of combinations that have already been tried. Finally, the combination with the best performance becomes the final hyperparameter choice and is used for the final model.

Cross-validation Using the process laid out above, we could already create a successful ML model that allows for the automatic recognition of handwritten characters. However, we are left with a single estimate of the model performance on the test dataset. The actual training process can have varying results based on randomized methods, which training data it is presented with, and even in which order the individual training samples are processed. Maybe we just got very

(un-)lucky and the test dataset happened to contain only legible handwriting? Our engineer might not dare introduce the system on this result alone, and rightly so. Luckily, we have another method we can use to increase our confidence in the performance results: The method of outer *cross-validation* repeats the entire training process using different splits of the dataset into training and validation data on the one side and test data on the other. This way, the training is done on a slightly different dataset each time, and the evaluation is on a different test dataset as well. The same process can be applied in the hyperparameter optimization process to create different train vs. validate splits, called *inner* cross-validation. When both of these methods are combined, we call the process *nested* cross-validation. Inner cross-validation is used to make the optimization more robust, while outer cross-validation can be used to detect overfitting more reliably. In outer cross-validation, the performance of models produced with different splits is aggregated to produce an overall performance estimate for the machine learning algorithm. This way, the engineer does not have to be worried as much about the system accidentally performing well on a single test dataset while failing on others.

2.3 Formal Definition of Machine Learning

In the following, we are formally defining the ML process as used in the remainder of this thesis. It is a unification of the notation used in [Gup+22] and [Dre+21].

Dataset Let $\mathcal{X} \in \mathbb{R}^d$ be the set of all possible inputs and let $\mathcal{Y}$ be the set of all possible *classes*. For the sake of simplicity, we let $\mathcal{Y} = \{0, 1, \ldots, K-1\}$, where $K \in \mathbb{N}$ with $2 \leq K < \infty$ represents the number of *classes*, also referred as class-label. The learning problem is called *multi-class classification* when the number of classes is $K > 2$ and *binary classification* if $K = 2$ [Mit97]. The training dataset is defined as $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, consisting of $|\mathcal{D}| = N$ training instances in form of tuples $(\boldsymbol{x}_i, y_i)$, such that $y_i \in \mathcal{Y}$ is the ground truth class-label associated with input instance $\boldsymbol{x}_i \in \mathcal{X}$. We assume it is generated by the unknown and possibly probabilistic *target function* $f : \mathcal{X} \to \mathcal{Y}$.

Hypothesis space The *hypothesis space* $\mathcal{H}$ defines the set of all candidate functions the learner can choose its predicted functions h from. This set is dependent on the specific algorithm, e.g. the hypothesis space for linear regression would only contain linear functions.

Classification The goal in binary classification is to find a *predicted function* $\hat{f} : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f. Formally speaking, the task of the learner is to induce a hypothesis $h \in \mathcal{H}$ with low generalization error (risk)

$$R(h) = \int_{\mathcal{X} \times \mathcal{Y}} L(y, h(\boldsymbol{x}))\, d\, P(\boldsymbol{x}, y) \tag{2.1}$$

where $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function, and P a joint probability measure modeling the underlying data-generating process of f. The loss function commonly used for classification is the 0-1 loss (using the indicator function $[\![\hat{y} \neq y]\!]$, which equals 1 *iff* $\hat{y} \neq y$, and 0 otherwise):

$$L_{01}(y, \hat{y}) := [\![\hat{y} \neq y]\!] \tag{2.2}$$

Empirical risk minimizer Since the underlying data-generating process and corresponding distribution $P(\boldsymbol{x}, y)$ is unknown to the learner, Equation (2.1) cannot be minimized directly. Instead, learning is accomplished by minimizing (a regularized version of) the *empirical risk*, which is determined by applying the hypothesis function h on the training data:

$$R_{emp}(h) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, h(\boldsymbol{x}_i)) \,. \tag{2.3}$$

The learning process returns the final hypothesis $\hat{f} \in \mathcal{H}$, a classification function that minimizes the risk calculated in Equation (2.3).

Evaluation Using the predicted function $\hat{f}$ we can determine its predictions $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_N)$ with $\hat{y}_i = f(x_i), \forall i \in \{1, \ldots, N\}$. We then compare the predictions to the ground truth class-label $\boldsymbol{y}$ and evaluate the difference using the measures used for classification as per [Koy+15].

Accuracy The *accuracy* is defined as the proportion of correct predictions:

$$\mathrm{m}_{ACC}(\hat{\boldsymbol{y}}, \boldsymbol{y}) := \frac{1}{N} \sum_{i=1}^{N} [\![\hat{y} = y]\!]. \tag{2.4}$$

Error rate In the case of classification, the *error rate* is defined based on the accuracy:

$$E(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 1 - \mathrm{m}_{ACC}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{2.5}$$

Confusion Matrix Many evaluation metrics for binary classifiers are defined using the number of *true positives* (TP), *true negatives* (TN), *false positives* (FP), and *false negatives* (FN). Formally, they are defined as:

$$\begin{aligned} \mathrm{TN}(\hat{\boldsymbol{y}}, \boldsymbol{y}) &= \textstyle\sum_{i=0}^{N} [\![y_i = 0, \hat{y}_i = 0]\!] \\ \mathrm{TP}(\hat{\boldsymbol{y}}, \boldsymbol{y}) &= \textstyle\sum_{i=0}^{N} [\![y_i = 1, \hat{y}_i = 1]\!] \\ \mathrm{FP}(\hat{\boldsymbol{y}}, \boldsymbol{y}) &= \textstyle\sum_{i=0}^{N} [\![y_i = 0, \hat{y}_i = 1]\!] \\ \mathrm{FN}(\hat{\boldsymbol{y}}, \boldsymbol{y}) &= \textstyle\sum_{i=0}^{N} [\![y_i = 1, \hat{y}_i = 0]\!] \end{aligned} \tag{2.6}$$

Using these, the *confusion matrix* is defined as:

$$\boldsymbol{M}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \begin{pmatrix} \mathrm{TN}(\hat{\boldsymbol{y}}, \boldsymbol{y}) & \mathrm{FP}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \\ \mathrm{FN}(\hat{\boldsymbol{y}}, \boldsymbol{y}) & \mathrm{TP}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \end{pmatrix} \tag{2.7}$$

F1-Score F1-Score is an accuracy measure that accounts for an imbalance between positive and negative instances in the dataset by incorporating both the FP and the FN and is defined as:

$$\mathrm{m}_{F_1}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{2\mathrm{TP}(\hat{\boldsymbol{y}},\boldsymbol{y})}{(2\mathrm{TP}(\hat{\boldsymbol{y}},\boldsymbol{y})+\mathrm{FN}(\hat{\boldsymbol{y}},\boldsymbol{y})+\mathrm{FP}(\hat{\boldsymbol{y}},\boldsymbol{y}))} \tag{2.8}$$

False Negative Rate FNR is defined as the ratio of FN to the total positive instances:

$$\mathrm{m}_{FNR}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\mathrm{FN}(\hat{\boldsymbol{y}},\boldsymbol{y})}{(\mathrm{FN}(\hat{\boldsymbol{y}},\boldsymbol{y})+\mathrm{TP}(\hat{\boldsymbol{y}},\boldsymbol{y}))} \tag{2.9}$$

False Positive Rate FPR is defined as the ratio of FP to the total negative instances:

$$\mathrm{m}_{FPR}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\mathrm{FP}(\hat{\boldsymbol{y}},\boldsymbol{y})}{(\mathrm{FP}(\hat{\boldsymbol{y}},\boldsymbol{y})+\mathrm{TN}(\hat{\boldsymbol{y}},\boldsymbol{y}))} \tag{2.10}$$

Cross-validation $\hat{f}$ only minimizes the *in-sample* error (E_{in}), but we wish to find a function that generalizes well, minimizing the *out-of-sample* error E_{out}. This is achieved with the train-validate-test split, returning a single estimate of E_{out} on the test dataset. To better evaluate the performance of $\hat{f}$, one employs outer cross-validation techniques, in which the dataset is split into many pairs k of train and validate vs. test datasets[3] and which allows several estimates of E_{out}. In inner cross-validation, the hyperparameter optimization goal for $\hat{f}$ is an estimate of E_{out} on the validation dataset achieved by creating k splits into train vs. validate datasets.

The methods used in this thesis are Monte Carlo cross-validation (MCCV) [Smy96; XL01] and k-fold cross-validation (KFCV) [RTL09]. In k-fold cross-validation (KFCV) the whole dataset is split into k equal parts, each to be used as the test dataset for one of the k evaluation rounds [RTL09]. The remainder of the dataset is then used for training (or training and validation datasets in the case of outer cross-validation) [VB12]. For Monte Carlo cross-validation (MCCV) each sample for the training dataset is selected at random without replacement from the full dataset. s defines the proportion of the dataset to be used for training, with the remaining $1 - s$ proportion of instances for a given training set constituting the test set [Smy96]. This process is repeated for a selected number of splits k, generating k pairs of train vs. test datasets at random.

In these techniques, different training datasets overlap, i.e. the same instance can appear in multiple training datasets [Smy96; RTL09]. The major difference is that in MCCV different testing datasets overlap, causing some bias in the

[3]The number of dataset splits k should not be confused with the number of classes K

estimation, while in KFCV each instance gets tested exactly once [Smy96; RTL09]. On the other hand, MCCV has an advantage over KFCV in terms of the number of possible ways the dataset can be split, as the maximum number of possible splits k can be very large [XL01]. So, KFCV provides a nearly unbiased performance estimate for algorithms with high variance, while MCCV gives a more biased performance estimate with less variance [VB12].

Hyperparameter optimization The learning algorithm uses hyperparameters to control the learning process. ML algorithms have different *hyperparameters* apart from the model parameters, that should be set for an algorithm, such as the number of estimators and the maximum depth of each decision tree (DT) for random forest (RF) or the learning rate of the perceptron learning algorithm (PLA). The class of optimization approaches used to solve the task of choosing a set of optimal hyperparameters for a learning algorithm is called HPO and the approach itself is called hyperparameter optimizer [HBF11]. Typically, a hyperparameter optimizer is provided with a search space, a defined range of possible values each hyperparameter can take for the given algorithm, and it runs for a given number of maximum trials. For each trial, it applies one combination of hyperparameter values from the search space and evaluates its performance using a loss function. Generally, we set aside a validation dataset sampled from the training dataset, for performance evaluation the loss function or metric used is called the *validation loss* or *validation accuracy*. For binary classification, the error rate is used as the validation loss, as defined in Equation (2.5).

For formalizing the HPO problem, we define a vector as $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$, for some $n \in \mathbb{N}$, which denotes n hyperparameters of the learner. Each hyperparameter can be an integer, real-valued or categorical, i.e., $\theta_i \in \mathbb{R}$, $\theta_i \in \mathbb{Z}$ or $\theta_i \in \{0, \ldots, c_i\}$, for some $c \in \mathbb{N}$, where c_i is the number of categories [FSH15]. For a given learner, each hyperparameter can only take a certain range of values, which is represented by $\Theta_i, \forall i \in [n]$, such that $\Theta_i = [c_1, c_2]$. The ranges depend on the type of parameter, with $c_1, c_2 \in \mathbb{R}$ for real-valued hyperparameters, $c_1, c_2 \in \mathbb{Z}$ for integers, and $c_1 = 0, c_2 \in \mathbb{N}$ for categorical hyperparameters. For finding the best hyperparameters, an objective function is defined as $g : \boldsymbol{\Theta} \mapsto \mathbb{R}$, which maps the hyperparameter vector $\boldsymbol{\theta}$ to a real-valued validation loss. This loss is generally obtained by evaluating the model for a given $\boldsymbol{\theta}$ on the validation dataset set aside earlier. For binary classification, the error rate is used as the validation loss, as defined in ℓ_{acc} eq. (2.5). The HPO task is to find an optimal combination of hyperparameters such that: $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} g(\boldsymbol{\theta})$

Hyperparameter optimizers A simple hyperparameter optimizer suggested in the literature is random search (RANDOM), in which the optimizer randomly samples the combination of values from given ranges for a certain number of trials and outputs the hyperparameters for which algorithm produces the best validation accuracy [FSH15].

A more sophisticated approach is to use the BAYESIAN optimization technique, which is more efficient for producing the optimal hyperparameters for the same number of trials [HBF11]. It builds a probabilistic model of the function g, by iteratively evaluating a set of parameters $\boldsymbol{\theta}$ to obtain the validation loss and then updating the underlying model. These methods use an acquisition function that samples the set of parameters $\boldsymbol{\theta}$ using the posterior distributions over objective functions. Under certain assumptions, Bayesian optimization techniques have been proven to converge to the optimal solution (hyperparameters), while RANDOM search does not [HBF11; FSH15].

Binary classifiers The following is a very high-level overview of the binary classifiers in this thesis.

Among the most commonly used binary classifiers proposed in the literature are perceptron learning algorithm (PLA) [FS99], logistic regression (LR) [YHL11], and Ridge classifier (RC) [HK70]. These algorithms are limited to solving binary classification problems involving linearly separable data. Since we cannot assume that the data is separable by a hyperplane, we also consider support vector machines (SVMs). SVM is a popular algorithm that uses a *kernel trick* (which can also be applied to linear models) to classify data that is not linearly separable. This results in its *candidate space* encompassing linear and non-linear functions [Pla99; CV95]. A decision tree (DT) is another promising approach. It learns a set of hierarchical rules, splitting the dataset into regions at each node of the tree [RM05].

Given the substantial difference in properties of the described algorithms, combining several classifiers to solve a classification problem can improve their reliability. These are called *ensemble*-based approaches, in which multiple classifiers are trained on the dataset and their predictions aggregated to obtain the final predicted function [Rok10]. One ensemble-based approach is *bagging*, where each classifier is trained only on a subsample created by sampling the given training dataset with replacement. The individual trained binary classifier are called the *base learners*, for which we use DTs. The two bagging approaches we consider are extra tree (ET), which uses mean aggregation of the predictions [GEW06], and random forest (RF), which uses majority voting for aggregation [Bre01]. Another way to build an ensemble is to use *boosting*, in which a set of weak learners are combined to create a single strong learner [Rok10]. The weak learners are successively trained and more weight is given to the wrongly classified instances each round. We again take DTs as the weak learners to build the complete binary classifier. Three popular approaches of this category are adaptive boosting (AB) [FS97], gradient boosting (GB), which improves the optimization of AB [Fri01], and histogram gradient boosting (HGB), which is overall faster than GB [Ke+17].

Feature Importance *Feature importance* refers to techniques that assign a score to each input feature based on their usefulness for predicting the class-label $\hat{y}$. Generally, decision-tree-based binary classifiers are used to calculate feature

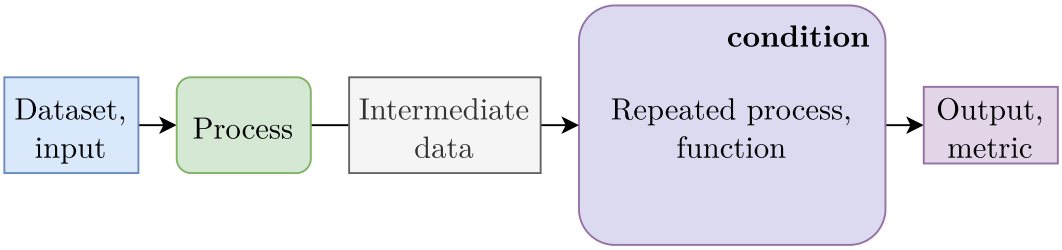


Figure 2.5: Example diagram illustrating the design used in this thesis.

importance. In a DT, the importance of a feature is based on the depth of the node on which the feature was used to split the data. The lower the depth in the DT, the higher the importance. In RF, each individual decision tree is used to calculate the feature importance. In order to reduce the variance and noise in the calculation, the overall feature importance is the mean of the importance in the individual trees.

Diagrams In this thesis, we use several diagrams to illustrate the detailed workings of the machine learning methods involved. These roughly use a shared design demonstrated in Figure 2.5.

2.4 Statistical Tests

In this section, we explain the statistical tests used for our proposed information leakage detection (ILD) approaches in Section 3.3 in more detail.

Paired t-Test The paired t-test (PTT) is used to compare two samples (generated from an underlying population) in which the observations in one sample can be paired with observations in the other sample [Dem06]. In our ILD application, we apply KFCV and use the same k train-test dataset pairs for evaluating a baseline classifier (e.g. majority class classifier) that we want to compare a binary classifier to. This produces k paired accuracy estimates of majority class classifier ($\boldsymbol{a}_{mc}$) with the binary classifier ($\boldsymbol{a}_j$). For binary classifier C_j, let $H_0(\boldsymbol{a}_j = \boldsymbol{a}_{mc})$ be the null hypothesis and $H_1(\boldsymbol{a}_j \neq \boldsymbol{a}_{mc})$ be the alternate hypothesis. $H_0(\boldsymbol{a}_j = \boldsymbol{a}_{mc})$ indicates that the underlying distribution of two populations is the same, which means that there is no difference between the performance of 2 binary classifiers [Dem06]. $H_1(\boldsymbol{a}_j \neq \boldsymbol{a}_{mc})$ instead implies that there is a significant difference between the performance of 2 binary classifiers.

The p-value quantifies the probability of accepting H_0 and is evaluated by determining the area under the Student's t-distribution curve at value t, which is $1 - \text{cdf}(t)$. The t-statistic is evaluated as $t = \frac{\mu}{\sigma/\sqrt{k}}$, with hypothesized value

$\mu = \frac{1}{k}\sum_{i=1}^{k} d_i$. We use the difference between the classifier accuracy and majority class classifier accuracy with $d_i = \boldsymbol{a}_{j_i} - \boldsymbol{a}_{mci}$ and $\sigma^2 = \sum_{i=1}^{k} \frac{(\mu - d_i)^2}{k-1}$. We have to adjust this p-value to correct for the dependency in estimates due to KFCV. Nadeau and Bengio [NB03] proposed to do this by modifying the variance σ in the following way: $\sigma^2_{Cor} = \sigma^2(\frac{1}{k} + \frac{1}{k-1})$. This variance σ^2_{Cor} is then used to calculate the value of the t-statistic as $t = \frac{\mu}{\sigma_{Cor}}$.

Due to its asymptotic nature, the paired t-test requires a large number of estimates k to produce a precise p-value. This in turn diminishes the effect of the correction term $\frac{1}{(k-1)}$ for σ^2_{Cor}. For a given dataset size, increasing k reduces the test set size, causing each accuracy estimate to become less precise.

Fisher's Exact Test Fisher's exact test (FET) is a non-parametric test that is used to calculate the probability of non-dependence between two classification methods. It analyses the contingency table $\boldsymbol{M}$ (see Equation (2.7)) containing the result of classifying objects by the two methods [Fis22]. For example, given a sample of people, we can divide them based on whether they have ever been to Australia and based on if they ever got attacked by a Kangaroo. Assuming that the sample is a good representation of people and that most people that get attacked by a Kangaroo are living in Australia, the FET would produce a very low p-value, implying that the two classification methods are correlated.

The p-value is computed using the Hypergeometric distribution as:

$$Pr(\boldsymbol{M}|N, R, r) = \frac{C^{(R)}_{(\mathrm{TN})} \times C^{(N-R)}_{(r-\mathrm{TN})}}{C^{(N)}_{(r)}} = \frac{C^{(\mathrm{FN+TN})}_{(\mathrm{TN})} \times C^{(\mathrm{FP+TP})}_{(\mathrm{FP})}}{C^{(\mathrm{TP+TN+FP+FN})}_{(\mathrm{TN+FP})}}$$

where $r = \mathrm{TN} + \mathrm{FP}$, $N = \mathrm{FP} + \mathrm{TP} + \mathrm{FN} + \mathrm{TN}$, $R = \mathrm{TN} + \mathrm{FN}$ and $C^{(n)}_{(r)}$ is the combinations of choosing r items from the given n items. The p-value is calculated by summing up the probabilities $Pr(\boldsymbol{M})$ for all tables having a probability equal to or smaller than that observed $\boldsymbol{M}$. This test considers all possible tables with the observed marginal counts for TN of the matrix $\boldsymbol{M}$ to calculate the chance of getting a table at least as "extreme".

Holm-Bonferroni correction When running several hypothesis tests where each rejected hypothesis results in an overall decision, care has to be taken when it comes to the rejection criteria α of the individual tests. For example, executing a number of tests that individually control the error rate (probability of false positive or Type 1 errors) at level α will result in the overall result exceeding level α. This is a result of the probability of at least one hypothesis test rejecting a hypothesis erroneously increasing with a growing number of tests. The Holm-Bonferroni method controls this family-wise error rate by adjusting the rejection criteria α for each individual hypothesis [Hol79]. We consider a family of null hypotheses $\mathcal{F} = H_1, \ldots, H_J$ and obtain p-values $p_1, \ldots, p_J$, from independently testing each classifier $C_j \in \mathcal{C}$, such that $J = |\mathcal{C}|$. For getting an aggregated decision, the significance level for the set $\mathcal{F}$ is not higher than the pre-specified threshold, e.g.,

$\alpha = 0.01$. The p-values are sorted in ascending order, i.e. $p_1 \leq p_2, \ldots p_{j-1} \leq p_J$, and for each hypothesis $H_j \in \mathcal{F}$, if $p_j < \frac{\alpha}{J+1-j}$, H_j is rejected. Let H_{m+1} be the first hypothesis for which the p-value does not validate rejection, i.e., $p_{m+1} > \frac{\alpha}{J-m}$. Then, the rejected hypotheses are $\{H_1, \ldots, H_m\}$ and the accepted hypotheses are $\{H_{m+1}, \ldots, H_J\}$.

3 New Techniques for Side-Channel Detection with Machine Learning

In this chapter, we create the theoretical foundation in preparation for the following chapters on applications. We start by introducing the overall scenario of information leakage, the underlying concept of side channels, in Section 3.1. This is followed by a formal definition of information leakage in Section 3.2. Based on this, we explore the connection between machine learning classifiers and information leakage, demonstrating how the performance of a binary classifier can be used as an indicator for information leakage. We use this concept in Section 3.3 to develop a concrete methodology for machine learning-based side-channel detection, proposing 4 new methods using statistical tests for the detection. We then proceed to compare these methods to two baseline methods on synthetic datasets and in a real-world side channel scenario in Section 3.4. We are able to demonstrate that our approaches based on Fisher's exact test (FET) outperform the state-of-the-art, reliably detecting information leakage in the majority of scenarios. After showing that our approaches are the first that are able to handle dataset imbalances properly, we conclude this chapter in Section 3.5.

Author's contribution The contents of this chapter are based on joint work together with Pritha Gupta, Arunselvan Ramaswamy, Eyke Hüllermeier, Claudia Priesterjahn, and Tibor Jager [Gup+22]. Pritha Gupta contributed the theory formalizing the relationship between the Bayes classifier accuracy and information leakage. Based on this she contributed the formal definition of information leakage detection, including the usage of t-test and Fisher's Exact Test to detect leakage. She contributed the synthetic datasets and design of the experimental setup, with the result analysis being performed jointly by Pritha Gupta and the author. The author's main contributions are the idea of using the Holm-Bonferroni correction and the creation of the OpenSSL real-world datasets.

3.1 Introduction

According to Hettwer, Gehrer, and Güneysu [HGG20], information leakage (IL) is defined as the unintended disclosure of sensitive information to an unauthorized individual or an eavesdropper via observable system information. Detecting these ILs is crucial since they can cause electrical blackouts and theft of valuable

or sensitive data like medical records and national security secrets [HGG20]. Information leakage detection (ILD) is the task of detecting IL in a given system.

Information leakage detection A system, intentionally or inadvertently, releases huge amounts of information publicly (called the *observable information*), which can be recorded by any outside observer. Specifically, IL takes place in this system if the observable information is connected, directly or indirectly, to secret information (secret keys, plaintexts) processed by the system, which may compromise security. The majority of current statistical methods estimate the mutual information between observable and secret information to quantify IL. However, these estimates suffer from the problem of the *curse of dimensionality* since it can be very difficult to determine which part of the observable information is actually connected to the secret. In addition, they strongly rely on time-consuming manual analysis by domain experts [CCG10]. As machine learning for ILD has been shown to be highly effective, current research focuses on the development and application of machine learning specifically for ILD [MWM21; Mus+18]. These approaches analyse the accuracy of a supervised learning model on the data extracted from the given system. The dataset is created by using the observable information as input and the secret information as the output [MWM21]. These methods, however, are domain-specific and unprepared to handle imbalanced datasets. This increases the likelihood that they would either overlook novel ILs (false negatives) or mistakenly identify leaks in the system (false positives) [Pic+18]. Our approach tackles this problem by using supervised learning algorithms and the evaluated confusion matrix to detect the IL which inherently takes the imbalance in the dataset into account [HK18].

Applications As a use-case for ILD, we consider the problem of side-channel detection in cryptographic systems. A cryptographic system unintentionally releases *observable information* via one of many possible channels, such as network messages, CPU caches, power consumption, or electromagnetic radiation. The *secret information* in this case could be a plaintext being encrypted, the secret key used in the cryptographic algorithm, or some other confidential information. These leakages are exploited by a side-channel attack (SCA) to reveal the secret information to an adversary, potentially rendering all implemented cryptographic protections irrelevant [MWM21; Mus+18]. The existence of a side channel in a cybersecurity system is equivalent to the occurrence of IL. Note however, that a system that contains a side channel is not necessarily vulnerable to a SCA, as sometimes the amount of information being leaked is simply not sufficient to reveal secret keys or plaintexts [Ber+17]. In other cases, leaking a single bit of secret information can be enough to allow powerful attacks [KR03]. As such, we have to regard all IL as dangerous until proven otherwise.

Side-channel detection In this field, the most relevant literature uses machine learning to perform SCAs, not preventing side channels through early detection of ILs [HGG20]. Current machine learning-based approaches are able to detect side channels, thus preventing SCA on the algorithmic and hardware levels and this has been presented by Zhang, Zheng, Nan, Hu, and Yu [Zha+20], Perianin, Carré, Dyseryn, Facon, and Guilley [Per+21], and Mushtaq, Akram, Bhatti, Chaudhry, Lapotre, and Gogniat [Mus+18]. These approaches apply the supervised-learning techniques using the observable system information as input to classify a system as vulnerable (with IL) or non-vulnerable (without IL). For generating the binary classification datasets, they extract observable information from the secured systems as input, label them as 0 (non-vulnerable) and then introduce known ILs in these systems and label them 1 (vulnerable). This process makes these approaches domain-specific and misses novel side channels [Per+21]. Current state-of-the-art automated learning-based approaches improve this by analyzing the accuracy of supervised-learning models on the binary classification data extracted from the given system, such that observable information is used as input and partial (sensitive) information is used as output [MWM21; Dre+21]. These approaches are restricted to detecting side channels accurately only if the extracted data is balanced, not noisy, and also produces a large number of false positives. The problem of imbalanced, noisy system datasets is very common in real-life scenarios [Zha+20].

Our Contributions We propose a novel approach that provides a general solution for detecting IL by testing the learnability of the binary classifiers on the extracted binary classification data from the system. Learnability is the ability of the classifier to learn from patterns in the training data to give reliable predictions, and this ability implies that there are useful patterns to be learned, which constitutes IL. To account for any imbalance in the dataset we use weighted versions of the binary classifiers and test the evaluated confusion matrices using FET [HK18]. The FET inherently takes the imbalance into account by indirectly using the Mathews correlation coefficient evaluation measure, which is zero if the predictions are obtained by guessing the label (random guessing) or predicting the majority label (majority class classifier) [CTJ21]. To account for the noise, we define an ensemble of binary classifiers which includes a deep multi-layer perceptron, and aggregate their FET results (p-values) to get the final verdict on the IL in a system. We show that our approach is more efficient (detection time) and accurate in detecting side channels in real-world cryptographic OpenSSL transport layer security (TLS) protocol implementations and ILs in synthetic scenarios as compared to the current state-of-the-art.

3.2 The Information Leakage Problem

In this section, we formalize the condition for information leakage (IL) in a given system using the binary classification problem described in Section 2.3. We also define the information leakage detection (ILD) task of classifying a given system as vulnerable or non-vulnerable.

Information leakage IL occurs in a system when *observable information* (measurable behavior) $\mathcal{X}$ is directly or indirectly correlated to secret information (secret keys, plaintexts) $\mathcal{Y}$ of the system. We can formulate this as an empirical task using the dataset $\mathcal{D}$ containing the measured behavior $\mathcal{X}$ and the known secret values y. In this setting, IL exists if there is some information present in the features $\mathcal{X}$ that can be used to derive the label y. By minimizing the empirical risk given in Equation (2.3) using a machine learning algorithm, we obtain a mapping $\hat{f}$ between observable input and secret output.

We suggest using the *Bayes predictor* f^b to check for dependencies (correlation) between $\mathcal{X}$ and $\mathcal{Y}$, equivalent to checking for IL in a system. The Bayes predictor can be seen as the "perfect" classifier, which uses the full knowledge about the underlying distribution that a dataset is generated from to make the best possible predictions for a given loss function. In our case, we use the (pointwise) *Bayes predictor* f^b that minimizes the expected 0-1 loss (L_{01}) of the prediction $\hat{y}$ for given input $\boldsymbol{x}$:

$$\begin{aligned} f^b(\boldsymbol{x}) &= \arg\min_{\hat{y}\in\mathcal{Y}} \sum\nolimits_{y\in\mathcal{Y}} L_{01}(\hat{y}, y)\, p(y|\boldsymbol{x}) \\ &= \arg\min_{\hat{y}\in\mathcal{Y}} E_y[L_{01}(\hat{y}, y)|\boldsymbol{x}], \end{aligned} \tag{3.1}$$

where $E_y[L_{01}]$ is the expected 0-1 loss with respect to $y \in \mathcal{Y}$ and $p(y|\boldsymbol{x})$ is the conditional probability of the class y given an instance $\boldsymbol{x}$. In the case where no side channel exists and $\mathcal{X}$ and $\mathcal{Y}$ are independent of each other, we know that the distribution $p(y)$ is simply the prior distribution of y. Given this distribution $p(y|\boldsymbol{x}) = p(y)$, the Bayes predictor f^b can be simplified to:

$$f^b(\boldsymbol{x}) = \arg\max_{y\in\mathcal{Y}} p(y) \tag{3.2}$$

Correspondingly, the prediction of f^b for every point $\boldsymbol{x} \in \mathcal{X}$ is label 0 if $p(1) < p(0)$ and label 1 if $p(1) > p(0)$. This implies the Bayes predictor behaves as a *majority class classifier*, a classifier that always predicts the single majority class that is present most often in the dataset, when $p(y|\boldsymbol{x}) = p(y)$. Hence, if f^b produces a 0-1 loss less than the 0-1 loss of a majority class classifier, we conclude $p(y|\boldsymbol{x}) \neq p(y)$, implying a dependency between $\mathcal{X}$ and $\mathcal{Y}$. However, note that an equal 0-1 loss does not necessarily imply $p(y|\boldsymbol{x}) = p(y)$. Based on this observation, we can consider a known distribution $p(y|\boldsymbol{x})$. If f^b produces a loss (significantly) lower than that of a majority class classifier, we imply that IL occurs in the system, else we assume no IL. If we further restrict $p(1) = p(0)$ corresponding to a perfectly

balanced dataset, the Bayes predictor becomes equivalent to a random guessing. In this special case, it would suffice to compare the f^b loss to that of a random guessing.

In reality, we cannot construct a Bayes classifier f^b simply because we do not know enough information about the underlying distribution. Instead, we can try to approximate f^b using an *empirical risk minimizer* $\hat{f}$ which we produce by training a sufficiently powerful machine learning classifier on $\mathcal{D}$ and minimizing Equation (2.3). Using this, we quantify the IL in a system as the difference between average 0-1 loss $(1 - \mathrm{m}_{ACC})$ for $\hat{f}$ and the majority class classifier. If this difference is significant enough, then we conclude that IL occurs in the system used to generate $\mathcal{D}$. This condition is the basis for our ILD approaches proposed in Section 3.3.

Information leakage detection Using this relationship, the problem of ILD is reduced to analyzing the learnability of binary classifiers on a given dataset. We can use the following hypothesis to test for IL in a real-world setting: Suppose the mapping function produced by the supervised learning algorithm accurately predicts the outputs using the inputs. In that case, the correlation between the input and output is sufficiently high and implies the system leaks information.

We require an ILD approach to return a simple *leakage/no leakage* evaluation for a given system. The task of the ILD approach is therefore to assign a label to the dataset $\mathcal{D}$ extracted from a system such that 0 indicates occurrence and 1 indicates the absence of IL in the system. The binary classification dataset $\mathcal{D}$ is constructed by representing the observable information of the system as inputs $\mathcal{X} \subset \mathbb{R}^d$ and secret information as outputs $\mathcal{Y} = \{0, 1\}$. Given a dataset $\mathcal{D}$ of size N, the task of detecting IL boils down to associating $\mathcal{D}$ with a label in $\{0, 1\}$, where 0 suggests "no information leakage" and 1 suggests "information leakage". Thus, we are interested in the function I defined as:

$$I : \bigcup_{N \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^N \to \{0, 1\}\,, \tag{3.3}$$

which takes a dataset $\mathcal{D}$ (extracted from the system) of any size as input and returns an assessment of the possible existence of IL in the given system as an output. We denote the mapping $\hat{I}$ as the predicted IL function produced by an ILD approach.

Datasets For many systems, we cannot decide if information leakage exists based on running an ILD approach on a single dataset $\mathcal{D}$. Instead, we extract several datasets using different methods and aggregate them into a "meta" dataset. One application would be detecting a Bleichenbacher side channel in a cryptographic system. Since there are several input types of manipulated PKCS#1v1.5 padding, the detection approach needs to check for IL separately for each input type. We define the overall IL-Dataset $\mathcal{L}$ as a collection of these individual datasets $\mathcal{D}$. Let $\mathcal{L} = \{(\mathcal{D}_i, z_i)\}_{i=1}^{N_I}$ be the IL-Dataset, such that $N_I \in \mathbb{N}, z_i \in \{0, 1\}, \forall i \in [N_I]$

Let $\boldsymbol{z} = (z_1, \ldots, z_{N_I})$ be the ground-truth vector, generated by the I, such that $z_i = I(\mathcal{D}_i), \forall i \in [N_I]$. Let $\hat{\boldsymbol{z}} = (\hat{z}_1, \ldots, \hat{z}_{N_I})$ be the corresponding prediction vector, such that $\hat{z}_i = \hat{I}(\mathcal{D}_i), \forall i \in [N_I]$. Since the ILD task produces binary decisions, their accuracy is measured using the binary classification evaluation metrics. Specifically, accuracy for the ground-truth $\boldsymbol{z}$ and predictions $\hat{\boldsymbol{z}}$ is calculated using $\mathrm{m}_{ACC}(\boldsymbol{z}, \hat{\boldsymbol{z}})$ defined in Equation (2.4). To avoid confusion, we refer to $\mathcal{L}$ as IL-Dataset and each $\mathcal{D}$ as the dataset.

3.3 Our Approaches for Information Leakage Detection

In this section, we describe our proposed ILD approaches using binary classification and statistical tests as shown in Figure 3.1. In addition, we describe an aggregation method based on using a set of binary classifiers and Holm-Bonferroni correction to make our approaches more robust.

3.3.1 Paired t-Test Approaches

From the machine learning perspective, IL occurs in a system if a binary classification algorithm trained on the dataset extracted from the system produces an accurate mapping between input (observable information) and output (secret information). This accuracy should be significantly better than that of the *Bayes predictor* evaluated on the dataset extracted from a secure system. For such systems, the inputs and outputs are independent of each other and the *Bayes predictor* becomes a majority class classifier as per Equation (3.2).

Paired comparisons The comparison between k accuracies obtained from k-fold cross-validation (KFCV) of the binary classifier and majority class classifier can be implemented with paired statistical tests between the performance estimates. These tests examine the probability (p-value) of observing the statistically significant difference between the paired samples (accuracies of the majority class classifier and binary classifier) [Dem06]. The p-value is the probability of obtaining test results (mean of the difference between the accuracies) at least as extreme as the observation, assuming that the null hypothesis (H_0) is true [Dem06]. The null hypothesis H_0 states that the accuracies are drawn from the same distribution, which would imply there is no difference in performance or, more formally, for the average difference between the paired samples drawn from the two populations to be zero (~ 0) [Dem06].

Paired test choices Out of many paired statistical tests, the most commonly used paired tests are the paired t-test and the Wilcoxon-signed rank test [Dem06]. These tests assume that each accuracy estimate is independent of all others. Using

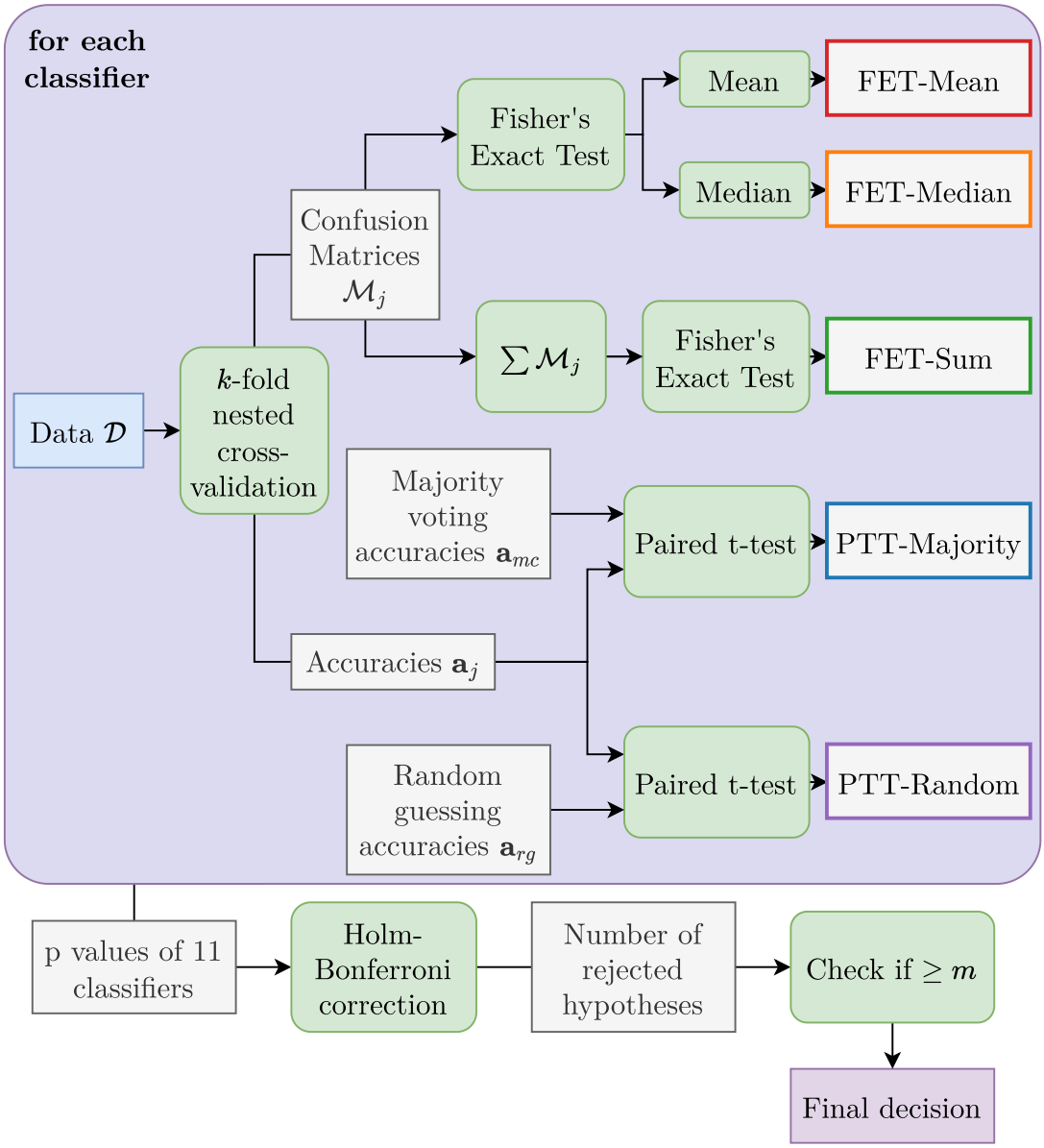


Figure 3.1: Our approaches for Information Leakage Detection

KFCV violates this assumption of independence because the training data across each fold overlaps. Nadeau and Bengio [NB03] showed that the violation of independence in the paired tests leads to overestimating the t statistic, resulting in tests being optimistically biased. We consequently choose to use their corrected version of the paired t-test because it accounts for the dependency in KFCV, as explained in Section 2.3. The shortcoming of the paired t-tests is their asymptotic nature and the assumption that the samples (difference between the accuracies) are normally distributed, which results in optimistically biased p-values.

3.3.2 Fisher's Exact Test Approaches

To address the problem of class imbalance and incorrect p-value estimation, we propose to use FET on the evaluated k confusion matrices (using KFCV) to detect IL. IL is likely to occur if there exists a (sufficiently strong) correlation between the inputs $\boldsymbol{x}$ and the outputs y in the given data $\mathcal{D}$. The predictions produced by the classifier C_j is defined as $\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{y}}$, where $\hat{f}$ is the *predicted function* as per Equation (2.3). $\hat{\boldsymbol{y}}$ can be interpreted as a single point encapsulating the complete information contained in the input vector $\boldsymbol{x}$. If there exists a correlation, then $\hat{\boldsymbol{y}}$ will contain input information that is relevant/used for predicting the correct outputs $(\mathrm{TP}, \mathrm{TN})$. We can therefore determine the existence of IL by examining the dependency between predictions $\hat{\boldsymbol{y}}_j$ of classifier C_j and the ground-truth $\boldsymbol{y}$.

We propose to apply FET on the confusion matrix for calculating the probability of independence between the model predictions $\hat{\boldsymbol{y}}$ and the actual labels $\boldsymbol{y}$ [Fis22]. FET is a non-parametric test that is used to calculate the probability of independence (non-dependence) between two classification methods, in this case, classification of instances according to ground-truth y and the binary classifier predictions $\hat{y}$ [Fis22]. The null hypothesis H_0 states that the model predictions $\hat{y}$ and ground-truth labels y are independent, implying the absence of IL. While the alternate hypothesis H_1 states that the model predictions $\hat{y}$ are (significantly) dependent on the ground-truth labels y, implying the occurrence of IL.

The advantage of using FET is that the p-value is calculated using the Hypergeometric distribution exactly rather than relying on an approximation that only becomes exact with sample size approaching infinity, as is the case for many other statistical tests. In addition to that, this approach directly tests the learnability of a binary classifier without having to consider a random guessing or majority class classifier.

3.3.3 On Robustness

For making our ILD approaches more robust, we define a set of 11 binary classifiers $\mathcal{C}$ that are evaluated on the extracted dataset $\mathcal{D}$ of the given system. The motivation of using an ensemble of binary classifiers, rather than just one binary classifier is that each binary classifier restricts their hypothesis space $\mathcal{H}$, based on the assumptions imposed on h. In addition, the statistical tests are asymmetric in

nature, and as such, they can only be used to reject H_0, which implies that we can prove the existence of IL but not its absence. To work around both restrictions, we use a set of multiple binary classifiers ($\mathcal{C}, |\mathcal{C}| = 11$) to detect IL more reliably inculcating greater trust in the absence of IL, if all classifiers fail to find an accurate matching.

The *set of binary classifiers* $\mathcal{C}$ ($|\mathcal{C}| = 11$) includes simple and commonly used linear classifiers, which assume linear dependencies between the input and output, such as Perceptron, Logistic Regression, and Ridge Classifier. $\mathcal{C}$ also includes support vector machine (SVM), which classifies the non-linearly separable data using a *kernel trick*, i.e., its hypothesis space also contains non-linear functions. $\mathcal{C}$ also includes decision tree (DT) and extra tree (ET), which learn a set of rules using a tree for classification [GEW06]. To learn more complex dependencies with very high accuracy, we also include *ensemble* based binary classifiers in $\mathcal{C}$. The *ensemble* based approaches train multiple binary classifiers (*base learners*) on the given dataset and the final prediction is obtained by aggregating the predictions of each *base learner*. The two most popular approaches proposed to build a diverse ensemble of learned base learners are *bagging* and *boosting* [KZP06]. To achieve diversification, *bagging* generates sub-sampled datasets from the given training dataset and *boosting* successively trains a set of weak learners (Decision Stumps) by giving more weight to the previously misclassified instances each round [KZP06]. The bagging-based approaches included in $\mathcal{C}$ are ET (mean aggregation) and random forest (RF) (majority voting aggregation). We choose adaptive boosting (AB) and gradient boosting (GB) from the available boosting-based approaches [KZP06]. We also include a deep multi-layer perceptron, as they are the universal approximators that, in theory, can approximate any continuous function between input and the output [Cyb89].

For evaluation of each binary classifier C_j, we use nested KFCV with hyper-parameter optimization to get k unbiased estimates of accuracies and confusion matrices [BG04]. As shown in Figure 3.1, for each binary classifier $C_j \in \mathcal{C}$, $\boldsymbol{a}_j = (a_{j1}, \ldots, a_{jk})$ denotes the k accuracies and $\mathcal{M}_j = \{\boldsymbol{M}_j^k\}_{k=0}^k$ denotes set of k confusion matrices.

Paired t-test approaches Our proposed approach based on the paired t-test (PTT) is **PTT-Majority**, which compares $\boldsymbol{a}_{mc}$, the accuracy of the majority class classifier, with $\boldsymbol{a}_j$, that of the binary classifier C_j. We denote the baseline approach used for the AutoSCA-tool described in Chapter 4 as **PTT-Random**, which uses a paired t-test to test if the binary classifier C_j ($\boldsymbol{a}_j$) performs significantly better than random guessing ($\boldsymbol{a}_{rg}$). Note that the relationship between the Bayes classifier and random guessing is only applicable for balanced datasets and does not hold for imbalanced datasets.

FET approaches We propose three new FET-based approaches **FET-Sum**, **FET-Mean** and **FET-Median**. Since we acquire k confusion matrices $\mathcal{M}_j$ for

each binary classifier C_j, we need further aggregation methods to reach a final FET p-value. As the test dataset in KFCV does not overlap for various folds, each confusion matrix $\boldsymbol{M}_j^k$ may be viewed as a separate estimate. Since FET is only applicable for 2×2 matrices containing natural numbers, we aggregate the confusion matrices using *sum* ($\sum_{k=1}^{k} \boldsymbol{M}_j^k$ for classifier C_j) and apply FET to obtain the final p-value. We refer to this approach as **FET-Sum**. The second method involves applying the FET to each confusion matrix $\boldsymbol{M}_j^k \in \mathcal{M}_j$ in order to determine k p-values, which are subsequently aggregated. Bhattacharya and Habtzghi [BH02] showed that the *median* aggregation operator provides the best estimation of the true p-value. We aggregate k p-values using the *median* operator and refer to this approach as **FET-Median**. We also investigate using the arithmetic *mean* operator for the aggregation instead and refer to this approach as **FET-Mean**.

We apply these approaches to each classifier $C_j \in \mathcal{C}$ and produce $|\mathcal{C}| = 11$ p-values as shown in Figure 3.1. To detect IL, we aggregate the p-values using the Holm-Bonferroni correction as described in Section 2.4. Using this correction yields the value m, which denotes the number of binary classifiers for which the null hypothesis H_0 was rejected.

Detection condition The sufficient condition for the existence of IL in the system is that even if one binary classifier is able to learn an accurate mapping between input and output, i.e. if $m = 1$ then IL occurs in the system. Using different types of binary classifiers and $m = 1$ makes our approach more general and can detect a diverse class of ILs in a system. Although $m = 1$ is the appropriate choice from an information-theoretic point of view, choosing $m > 1$ can be useful in real-world scenarios where false positives need to be minimized.

3.4 Empirical Evaluation

In this section, we provide an extensive evaluation of our proposed approaches, detecting the IL in synthetic and real-world scenarios. In particular, we show that our approach outperforms the state-of-the-art,[4] as presented in [Dre+21; MWM21], with respect to detection accuracy, efficiency, and generalization capability with respect to class imbalance in the datasets. We also describe the IL-Datasets used to illustrate our ideas.

3.4.1 Dataset Descriptions

As stated earlier, we need to generate a binary classification dataset from the system under consideration for IL detection. In this section, we describe the generation process for these datasets from synthetic and real-world systems. Recall

[4] Since the paper on the application of these techniques to Bleichenbacher's attack was published first, it is considered to be the state-of-the-art for this comparison

from Section 3.3.3, that k refers to the number of folds of nested KFCV used for evaluation of binary classifiers. *class imbalance parameter* is the proportion of positive instances in a given dataset $\mathcal{D}$, defined as: $r = \frac{|\{(\boldsymbol{x}_i, y_i) \in \mathcal{D} \mid y_i = 1\}|}{|\mathcal{D}|}$.

Synthetic Dataset Generation For simulating a realistic leakage detection scenario, we generate synthetic binary classification datasets $\mathcal{D}$ from *vulnerable* and *non-vulnerable* systems, using Algorithm 1. This approach enables us to generate datasets with varying degrees of imbalance r and parameter k for KFCV. For each system, the inputs $\boldsymbol{x}$ of the datasets are produced using the d-dimensional ($d = 10$) multi-variant normal distribution. To imitate a vulnerable system containing IL, the corresponding labels (output) are produced using a function $y = f(\boldsymbol{x})$, which causes the output to depend on the inputs. To imitate a non-vulnerable system which does not contain IL, the corresponding labels (output) are produced using Bernoulli distribution. Specifically, $y \sim \text{Bernoulli}(p = r, q = 1 - r)$ with class imbalance parameter r makes the output independent of the inputs. Algorithm 1 generates balanced IL-Datasets, containing 10 datasets extracted from the vulnerable systems and 10 from the non-vulnerable systems. Each $\mathcal{D}$ contains $200 \times k$ instances with dimensionality 10, out of which $r \times 200 \times k$ are labeled 1 and the rest as 0. This ensures that the number of instances in the test dataset, used for evaluation of a binary classifier ($|\mathcal{D}|/k = 200$) is the same across different scenarios, making accuracy and confusion matrix estimates fair ($\text{TP} + \text{FP} + \text{TN} + \text{FN} = 200$). We implement Algorithm 1 by modifying the `make_classification` function in scikit-learn [Ped+11].

The main goal of our empirical evaluation is to analyse how our proposed ILD approaches perform compared to baselines in regards to *efficiency (detection time)* and *generalization* capability with respect to the class imbalance parameter r. The total time taken by the ILD approaches is linear w.r.t. the dataset size. Because the test dataset size needs to remain constant for a fair comparison, we choose $|\mathcal{D}| = k \times 200$, and thus overall runtime is in $\mathcal{O}(k)$. For analyzing the *efficiency*, we generate 28 IL-Datasets with each dataset $\mathcal{D}$ of size $|\mathcal{D}| = k * 200$, for value of k ranging from 3 to 30 and fixed class imbalance $r = 0.1, 0.3, 0.5$, as detailed in Table 3.1. To gauge the *generalization* capability, we generate 25 IL-Datasets with each dataset $\mathcal{D}$ of size $|\mathcal{D}| = k * 200$ for fixed $k = 10, 20, 30$ with class imbalance r varying between 0.01 and 0.51, as detailed in Table 3.1.

OpenSSL Dataset Generation The real-world classification datasets are generated from the network traffic of 2 OpenSSL TLS servers, one of which is vulnerable to Bleichenbacher's attack (contains IL) and the other being non-vulnerable (secure, does not contain IL). We generate the data using the AutoSCA-tool,[5] the details of which are covered in Chapter 4. For now, it suffices to assume that the tool uses a modified TLS client to send requests with manipulated padding

[5] `https://github.com/ITSC-Group/autosca-tool`

Table 3.1: Overview of the IL-Datasets used for the experiments

Scenario	Fixed Parameter	IL-Dataset $\mathcal{L}$ configuration			Binary Classification Dataset $\mathcal{D}$ configuration			
		# Systems $\lvert\mathcal{L}\rvert$	# $z = 0$	# $z = 1$	$\lvert\mathcal{D}\rvert$	# $y = 0$	# $y = 1$	# Features
Efficiency k for KFCV, $3 \leq k \leq 30$	$r = 0.1$	20	10	10	$200 \times k$	$180 \times k$	$20 \times k$	10
	$r = 0.3$	20	10	10	$200 \times k$	$140 \times k$	$60 \times k$	10
	$r = 0.5$	20	10	10	$200 \times k$	$100 \times k$	$100 \times k$	10
Generalization class imbalance parameter r $0.01 \leq r \leq 0.51$	$k = 10$	20	10	10	2000	$2000 \times (1 - r)$	$2000 \times r$	10
	$k = 20$	20	10	10	4000	$4000 \times (1 - r)$	$4000 \times r$	10
	$k = 30$	20	10	10	6000	$6000 \times (1 - r)$	$600 \times r$	10
Configuration of the OpenSSL IL-Datasets								
OpenSSL0.9.7a (Vulnerable)	$r = 0.1$	20	-	10	11 124	10 012	1112	88
OpenSSL0.9.7b (Non-Vulnerable)	$r = 0.1$		10	-	11 078	9971	1107	88
OpenSSL0.9.7a (Vulnerable)	$r = 0.3$	20	-	10	14 302	10 012	4290	88
OpenSSL0.9.7b (Non-Vulnerable)	$r = 0.3$		10	-	14 244	9971	4273	88
OpenSSL0.9.7a (Vulnerable)	$r = 0.5$	20	-	10	19 991	10 012	9979	88
OpenSSL0.9.7b (Non-Vulnerable)	$r = 0.5$		10	-	19 995	9971	10 024	88

Algorithm 1 Generate IL-Dataset $\mathcal{L}$ for given k, r

Define $\mathcal{L} = \{\}$, $N = k \times 200$, N_L.
Sample weight-vector $\boldsymbol{\beta} \sim N(1, \sigma)$, $\sigma \sim [0, 2]$
Define $\boldsymbol{\mu}$ as vertices of d-dimensional hypercube.
for $j \in \{2 \times j - 1\}_{j=1}^{N_L/2}$ **do**
  Draw i.i.d. samples $\boldsymbol{x}_i \sim N(\boldsymbol{\mu}, \boldsymbol{I}_d), \forall i \in [N]$
  Define $\mathcal{D}_j = \{\}$, $\mathcal{D}_{j+1} = \{\}$. ▷$\mathcal{D}_j$: *With IL,* $\mathcal{D}_{j+1}$*: No IL*
  for $(i = 1; i <= N; i{+}{+})$ **do**
    Calculate score $s_i = \text{sigmoid}(\boldsymbol{x}_i \cdot \boldsymbol{\beta})$
    Label y_i: $y_i = [\![s_i < r]\!]$.
    $\mathcal{D}_j = \mathcal{D}_j \cup \{(\boldsymbol{x}_i, y_i)\}$ ▷*Add instance*
  end for
  for $(i = 1; i <= N; i{+}{+})$ **do**
    Label y_i: $y_i \sim \text{Bernoulli}(p = r, q = 1 - r)$.
    $\mathcal{D}_{j+1} = \mathcal{D}_{j+1} \cup \{(\boldsymbol{x}_i, y_i)\}$ ▷*Add instance*
  end for
  $\mathcal{L} \cup \{(\mathcal{D}_j, 1), (\mathcal{D}_{j+1}, 0)\}$ ▷*Add datasets*
end for
return $\mathcal{L}$

to a TLS server. In this setting, IL occurs when an attacker can deduce the manipulation in the message only by observing the server's reaction to the message. Therefore, the server's reaction is recorded as a network trace by the tool and exported to a labeled dataset suitable for classifier training. The most popular TLS server, OpenSSL, comes to mind when deciding which TLS server to employ for our experiment. According to the OpenSSL changelog,[6] a fix for the Klíma-Pokorny-Rosa [KPR03] bad version attack touched on in Section 2.1.1 was applied in version 0.9.7b. Consequently, version 0.9.7a contained IL in the form of a bad version side channel, while version 0.9.7b does not contain this IL. This provides the chance to collect suitable datasets from these servers and use them in our demonstration. To generate the dataset, we configure the modified TLS client to manipulate the TLS version bytes contained in the premaster secret (`PMS`) it sends to the OpenSSL server. For each handshake, the client flips a coin to either keep the correct TLS version in place or replace it with the non-existing "bad" version `0x42 0x42`.

In the resulting dataset, class label $y = 0$ is used for handshakes with the correct TLS version and class label $y = 1$ for handshakes with non-existing TLS version. This handshake process is then repeated 20 000 times, with each handshake being extracted into a single instance in the dataset. This approach produces datasets with approximately 10 000 instances per class. We refer to these (mostly) balanced datasets as $r = 0.5$ in our experiments. In addition, we also generate datasets with

[6] `https://www.openssl.org/news/changelog.html`

artificial class imbalance $r = 0.3$ and $r = 0.1$ by sampling from the full dataset, resulting in imbalanced datasets containing around 14 000 handshakes and 11 000 handshakes respectively. We generate IL-Datasets containing 10 datasets from 0.9.7a (with IL) and 10 datasets from 0.9.7b (without IL) as shown in Table 3.1. All real-valued features of the TLS and TCP layers in the messages sent by the server as a reaction to the manipulated TLS message are part of an instance in the dataset. This results in datasets with a large dimensionality, containing 88 features, only a handful of which are actually correlated to the output.

3.4.2 Implementation Details

The main goal of our empirical evaluation is to analyse how our proposed ILD approaches perform in comparison to the baselines in regards to *efficiency* (detection time), *generalization capability* with respect to the class imbalance parameter r, and overall ILD *accuracy*. Table 3.1 describes the IL-Datasets used for these experimental scenarios. We use PTT-*Random* as a baseline, which compares the accuracy of a random guessing to a set of binary classifiers (ensemble). For this ensemble, we use our defined set of binary classifiers described in Section 3.3.3. We also consider DL-LA as another baseline, which trains a deep multi-layer perceptron on a balanced binary classification dataset and propose that if its accuracy is significantly greater than 0.5, then IL exists in the given system [MWM21].

We apply nested KFCV with hyperparameter optimization on $|\mathcal{C}| = 11$ binary classifiers as described in Section 3.3.3. We employ the weighted versions of binary classifiers described by Hashemi and Karimi [HK18] to improve the accuracy of the binary classifiers for imbalanced datasets ($r < 0.5$). They penalize misclassification of positive ($y = 1$) and negative ($y = 0$) instances by $\frac{1}{r}$ and $\frac{1}{1-r}$, respectively. For our experiments, the rejection criteria for statistical tests is set to $\alpha = 0.01$, giving us 99 % confidence for our prediction. The binary classifiers, stratified KFCV, and evaluation measures were implemented using the scikit-learn library [Ped+11] and the statistical tests using SciPy [Vir+20]. The hyperparameters of each binary classifier were tuned using the Bayesian optimization technique with the Gaussian process surrogate model implemented by scikit-optimize [Hea+20]. The code for the experiments and the generation of plots with detailed documentation is publicly available on GitHub.[7]

3.4.3 Results

In this section, we discuss the results of the experiments outlined above. Recall that k is the number of folds used for conducting KFCV and r is the proportion of positive instances in the dataset $\mathcal{D}$. Each IL-Dataset contains binary classification datasets of size $200 \times k$. In Figures 3.2 to 3.4, we compare the detection accuracy

[7] `https://github.com/prithagupta/ML-ILD`

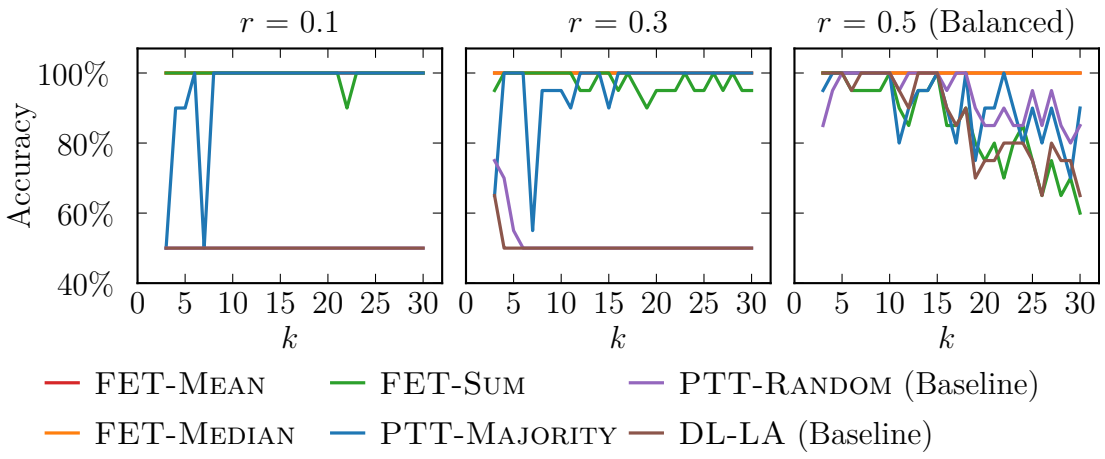


Figure 3.2: Accuracy of detection approaches on synthetic IL-Datasets evaluated using a varying number of folds k for KFCV

of FET-Mean, FET-Median, FET-Sum, PTT-Majority with the baselines PTT-Random and DL-LA.

Efficiency In Figure 3.2, the value k used for k-fold cross-validation KFCV is varied between 3 and 30 and shown on the X-axis, and the resulting accuracy of the ILD approaches along the Y-axis. Additionally, we compare the performance for different choices of class imbalance parameter r ($r = 0.1$, $r = 0.3$, and $r = 0.5$ (balanced)), producing three individual plots shown side-by-side.

Overall, we observe that the performance of FET-Mean and FET-Median ($\sim$ 100%) does not change with the value of k (number of cross-validation estimates) for all three IL-Datasets, while FET-Sum is very unstable for balanced dataset $r = 0.5$ and slightly unstable for imbalanced datasets. The PTT-Majority approach outperforms the baselines, but there is no single fixed value of k for which the approach is able to handle arbitrary imbalance. A good choice to prevent unstable and inaccurate results would require a small value $k < 7$ for balanced datasets ($r = 0.5$) and a large value $k > 7$ for imbalanced datasets $r = 0.1, 0.3$, which is an issue in applications where the imbalance is not known in advance. A possible reason for this behavior could be the paired t-test producing optimistically biased p-values due to the low deviation in estimated accuracy of the majority class classifier. We also observe that DL-LA performs similarly to PTT-Random, with both failing to detect IL in imbalanced datasets accurately. In general, we observed that FET-based approaches require a lower number of estimates k, and are thus more efficient in detecting ILs.

Handling imbalance In Figure 3.3, the value r (class imbalance) is varied between 0.05 and 0.5 and shown on the X-axis, and the resulting accuracy of the ILD approaches along the Y-axis. Additionally, we compare the performance for

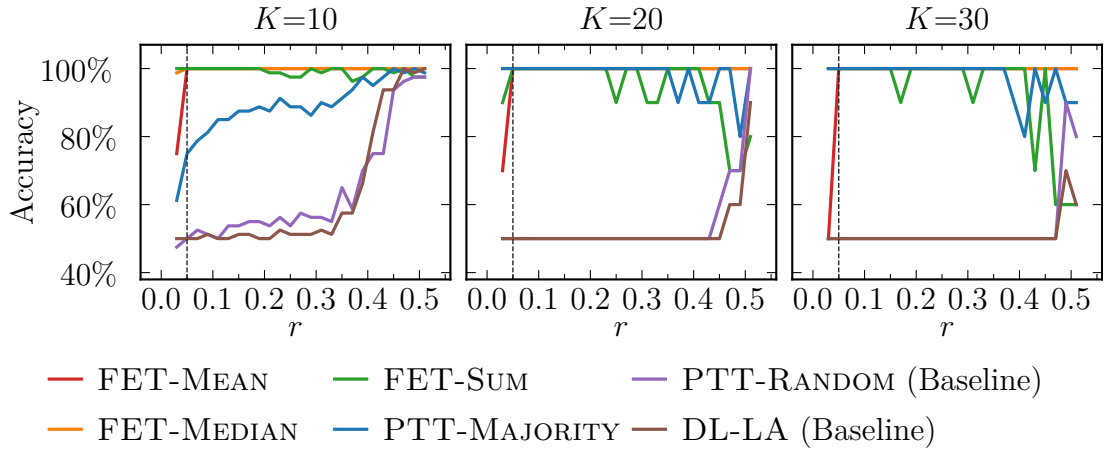


Figure 3.3: Accuracy of detection approaches on synthetic IL-Datasets containing datasets with varying imbalance r

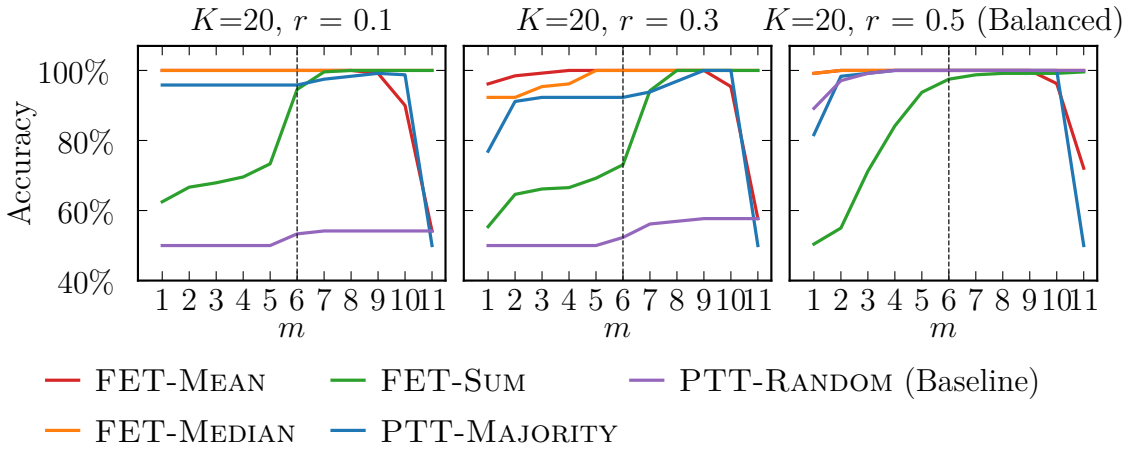


Figure 3.4: Accuracy of detection approaches on OpenSSL datasets with varying Holm-Bonferroni cut-off parameter m.

different choices of k ($k = 10$, $k = 20$, and $k = 30$), producing three individual plots shown side-by-side.

The FET-based approaches perform very well for all datasets $r \geq 0.05$, even if the number of estimates is as low as $k = 10$. As before, PTT-Majority achieves a high detection accuracy for larger numbers of estimates $k = 20, k = 30$ only with high imbalance $0.05 \leq r < 0.3$, with deteriorating accuracy for $r \geq 0.3$. The baselines are not able to detect ILs in imbalanced datasets. This is to be expected for PTT-Random as most of the binary classifiers easily achieve the same accuracy as a majority class classifier $(1 - r)$, which means that they always outperform random guessing (0.5) even when there is no IL.

OpenSSL dataset In Table 3.2, we summarize the overall performance in terms of FPR, FNR, Accuracy, and F1-Score (as defined in Section 2.3) of ILD approaches on the real datasets, fixing $k = 20$ based on the previous results. Overall FET-Mean and FET-Median outperform other approaches in detecting side channels in the OpenSSL case study. As anticipated, the baselines successfully detect side channels for balanced datasets but not for imbalanced datasets. The PTT-Majority approach works well for imbalanced datasets but produces false positives for balanced datasets. The FET-Sum approach overestimates ILs in the servers, resulting in false positives for both balanced and imbalanced datasets.

In Figure 3.4 we also explore the effectiveness of ILD approaches for different values of the Holm-Bonferroni parameter m on these datasets. The DL-LA approach is not included in these comparisons since it does not apply the Holm-Bonferroni correction. For very low values of m, some ILD approaches produce a large number of false positives, because the tests are underestimating the p-values. For very high values of m, the inability of some of the binary classifiers to learn an accurate enough mapping to support rejecting the null hypothesis results in a lot of false negatives. While $m = 1$ is the choice supported by information theory, it can result in false positives in practice. Based on the requirements at hand, it will therefore be necessary to tune m to balance avoiding false positives and false negatives.

FET-Median appears to be the optimal choice, achieving an accuracy of 100 % in almost all cases. The overall performance of FET-Median and FET-Mean is consistently very high (between 99 % and 100 %) for all but the most extreme choices of $m \geq 10$. FET-Median slightly outperforms FET-Mean because the median aggregation results in more accurate p-values [BH02]. PTT-Majority and FET-Sum are not reliable for estimating the correct p-values and produce large false positives, especially for $m < 4$.

3.5 Conclusion and Open Problems

We presented a novel machine learning-based framework to detect the possibility of IL in a given system. To do this, we first used its observable and secret system

Table 3.2: Results on the OpenSSL datasets for every approach using $k = 20$ and $m = 1$. The best entry is marked in bold.

	class imbalance $r = 0.1$				class imbalance $r = 0.3$				class imbalance $r = 0.5$ (Balanced)			
Approach	FPR	FNR	Accuracy	F1-Score	FPR	FNR	Accuracy	F1-Score	FPR	FNR	Accuracy	F1-Score
FET-Mean	**0.0021**	**0.0**	**0.999**	**0.999**	**0.0182**	**0.0**	**0.9909**	**0.9911**	**0.0164**	**0.0**	**0.9918**	**0.9919**
FET-Median	**0.0021**	**0.0**	**0.999**	**0.999**	0.0455	0.0	0.9773	0.9778	0.0327	0.0	0.9836	0.9839
FET-Sum	0.66	0.0	0.67	0.7519	0.84	0.0	0.58	0.7043	0.9564	0.0	0.5218	0.6766
PTT-Majority	0.0473	0.0	0.9764	0.9769	0.3309	0.0	0.8345	0.8585	0.3145	0.0	0.8427	0.8674
PTT-Random (Baseline)	1.0	0.0	0.5	0.6667	1.0	0.0	0.5	0.6667	0.1836	0.0	0.9082	0.9179
DL-LA (Baseline)	1.0	0.0	0.5	0.6667	1.0	0.0	0.5	0.6667	0.4727	0.0	0.7636	0.8111

data to create an adequate (binary) classification dataset. The dataset was then used to train an ensemble of classification models. We deduce the existence of IL when either the ensemble performance is significantly better than majority voting or using the more complex statistical test FET.

The major advantages of the presented approach over previous ones are:

- it accounts for imbalances in datasets
- it has a very low false positive rate
- it is robust to noise in the generated dataset
- it is time-efficient
- it outperforms the state-of-the-art machine learning-based approaches

These advantages are partly due to our IL inference using the non-parametric FET, applied on the confusion matrix of the learning models, as opposed to direct IL inference using learning accuracies. The Holm-Bonferroni correction technique contributes to the increased robustness in particular. We presented extensive empirical evidence for our claims and compared our approach to other baseline approaches, including a deep-learning-based one.

In the future, we aim to extend our work to detect new and unknown side channels. We are also interested in exploring IL detection when the generated dataset yields a multi-class classification problem with more than 2 classes. This calls for extending the proposed FET methods to take multi-class classification issues into account.

Research Question 1. *Is it possible to generalize the FET approach to* ≥ 3 *classes while maintaining the theoretical guarantees on optimality?*

Another interesting question that remains unanswered concerns the connections between information leakage in a purely informatic-theoretical view and the approaches outlined above.

Research Question 2. *Is there a different way to formulate the information leakage problem that leads to a more direct detection approach, ideally without involving any statistical tests?*

Alternatively, we might be able to eliminate the need for the Holm-Bonferroni correction altogether by replacing the set of classifiers with a single AutoML method, predicting the best pipeline, and then using statistical tests to detect IL.

4 Application to Bleichenbacher's Attack

In this chapter, we apply the methodology developed in the previous Chapter 3 to detecting Bleichenbacher side channels. Accordingly, we start by introducing the context for our work in Sections 4.1 and 4.2. We then cover the working of Bleichenbacher's side-channel attack including how it can be used to decrypt confidential data sent over transport layer security (TLS) in Section 4.3.1. We then revisit the methodology from Chapter 3 in Section 4.3.2 and derive a Bleichenbacher-specific version of it. We use this method to create the automated Bleichenbacher side-channel detection tool "AutoSCA-tool" which we describe in detail in Section 4.4. In Section 4.5, we evaluate this tool on various TLS servers containing Bleichenbacher side channels. We validate the functionality with a version of OpenSSL that is deliberately manipulated to contain an easy-to-detect side channel in Section 4.5.2. We can then apply the tool on the bad version oracle by Klíma, Pokorný, and Rosa [KPR03] in Section 4.5.3 and a recreation of the ROBOT side channels [BSY18] in Section 4.5.4. In Section 4.5.5 we then check for previously unknown side channels present in the most commonly visited webpages and current versions of open-source TLS servers. In Section 4.5.6, we demonstrate how the modularity and fully automatic capabilities of the AutoSCA-tool can be used to detect Bleichenbacher side channels as part of commercial TLS standard conformance testing. Section 4.6 concludes this chapter.

Author's contribution The following is joint work together with Pritha Gupta, Eyke Hüllermeier, Tibor Jager, Alexander Konze, Claudia Priesterjahn, Arunselvan Ramaswamy, and Juraj Somorovsky [Dre+21]. Pritha Gupta designed and implemented the machine learning module, including the feature importance feedback technique. A majority of the results in [Dre+21] were contributed by the author, encompassing the following technical contributions:

- Creating a modular framework that allows switching between different TLS clients and network recording tools
- Design and implementation of the feature extraction process
- Implementation of the framework, the "AutoSCA-tool"
- Designing and running the experiments and scans

- Result analysis and discussion were joint work by Pritha Gupta and the author

4.1 Introduction

Many recent attacks on cryptographic protocols deployed in practice do not break the cryptographic algorithms directly. Instead, they are based on *side-channel* information. For instance, this includes many recent attacks on TLS, probably the most widely-used and well-analysed cryptographic protocol on the Internet, such as DROWN [Avi+16], ROBOT [BSY18], Raccoon [Mer+21], POODLE [MDK14], and many more.

One particularly important family are *padding oracle attacks*, such as the attacks by Bleichenbacher [Ble98], Manger [Man01], and Vaudenay [Vau02]. Padding oracle attacks have found countless applications.

Since the original publication of Bleichenbacher's attack in 1998, follow-up works have developed many different approaches which construct the padding oracle required for this attack, which then often yields an efficient attack on the considered implementation or application [KPR03; JSS12; Deg+12; Bar+12; Mey+14; Zha+14; Avi+16; Fel+18; Ron+19].

Unexpected side channels Particularly surprising constructions of padding oracles were shown in the ROBOT attack by Böck, Somorovsky, and Young [BSY18]. This work demonstrated that padding oracles can appear in very subtle and quite unexpected forms. For example, one would probably expect that the TCP protocol, which is used to transport TLS messages, cannot provide an exploitable padding oracle, because it is merely a transport protocol that operates on a completely different network layer. The protocol is independent of the cryptographic keys used in higher-layer protocols (such as TLS running over TCP), and therefore should not be able to leak any information. However, Böck *et al.* showed, very surprisingly, that certain TLS implementations may terminate a TCP session in different ways, depending on whether a padding error occurred or not. This could be used to construct a new padding oracle. The authors of the ROBOT paper also found new vulnerabilities in commercial products by Cisco, Citrix, F5, Symantec, and Cavium, and even demonstrated a forgery of a valid digital signature using Facebook's Rivest–Shamir–Adleman (RSA) certificate that was based on a padding oracle provided by Facebook's custom TLS server implementation.

All padding oracle attacks appearing in the literature so far seem to have been found *manually*, that is, via careful analyses of TLS server responses by specialized expert security researchers, who thoroughly analysed one popular implementation after another. Of course, this approach does not scale well with a growing number of implementations. Furthermore, even for experts it is very difficult to find

"unexpected" side channels that go beyond what one is specifically looking for, such as the aforementioned TCP side channel [BSY18], for instance.

Research challenge The development of techniques that are capable of finding such cryptographic side channels automatically, without the need for time-consuming manual analysis, is a foundational open problem. The main difficulty is that in order to be able to identify even unexpected side channels, it is not sufficient to check against a list of known or typical vulnerabilities. Instead, one has to analyse the behavior of an implementation and efficiently recognize *general patterns* that depend on the validity of the padding, and thus might give rise to a padding oracle.

For example, in the context of TLS implementations, we would like to have a tool that can be executed against any concrete TLS implementation, possibly after every significant code modification or before every release of a new software version. Such a tool should not require any extensive and time-consuming manual analysis or supervision by an expert security researcher. Instead, it should be usable without expert knowledge, ideally in a fully-automated way that allows running automated tests, possibly in the regression testing phase of a CI/CD pipeline. This would significantly reduce the attack surface of practical applications.

Our contributions We propose an automated approach to detect side channels such as padding oracles in cryptographic protocol implementations, which uses a variety of classification algorithms from machine learning to automatically detect general *patterns* in network protocol traffic that might give rise to a padding oracle. Our solution does not merely use the predictions of a pre-trained model but relies on the ability to learn patterns of interest.

In order to analyse this general approach, we consider Bleichenbacher-like attacks on TLS as a concrete use case. This class of attacks provides a prime example of a cryptographic side-channel attack on the protocol level. A long sequence of research papers appearing at leading academic security conferences [Ble98; KPR03; JSS12; Deg+12; Bar+12; Mey+14; Zha+14; Avi+16; Fel+18; BSY18; Ron+19] showed that such vulnerabilities appear repeatedly in popular open-source software and widely-used commercial products. Hence, this is an ideal reference for the development and analysis of an automated methodology to detect side-channel attacks on the protocol level.

In order to minimize the probability that a padding oracle vulnerability remains undetected in the automated analysis, we propose to train an ensemble of machine learning algorithms and aggregate them. Concretely, we consider 10 algorithms from different families, including for instance Logistic Regression, Support Vector Machines, Decision Trees, Random Forest, and Boosting algorithms. Since detection of vulnerabilities is most useful when one can also provide information of the origin of recognized patterns (e. g., which particular protocol message exhibits the pattern), we focus on machine learning (ML) algorithms that are amenable to

feature importance techniques.

We implement and analyse this approach in a tool which automatically analyses a given TLS server implementation. The tool implements a TLS client to generate training and testing data and then applies the machine learning algorithms to detect potential side channels.

We confirm that the tool is indeed able to detect known vulnerabilities in TLS server implementations reliably. Concretely, the tool correctly identifies the vulnerability identified in [KPR03] in OpenSSL version 0.9.7a, while no vulnerability is identified in version 0.9.7b (which patches this vulnerability). We also confirm that the tool reliably detects all padding oracle vulnerabilities described in the ROBOT paper [BSY18]. Since some of these vulnerabilities were found in proprietary implementations (e. g., Facebook's and Cisco's) which are not publicly available for analysis, we patched an mbedTLS[8] server according to the description of the behavior of these implementations from the ROBOT paper [BSY18] to simulate these padding oracles. Finally, we analyse the most recent versions of 13 different popular open-source TLS implementations (cf. Table 4.2), but (as expected) without finding any new vulnerabilities. We conclude that the tool is able to reliably detect known vulnerabilities, with a general and generic approach. We consider this as an indicator that the approach will also work for future, new side channels that exhibit distinguishable patterns on the network layer.

To assist in removing identified potential vulnerabilities, the tool also provides detailed feedback about detected patterns to developers, for which we rely on *feature importance* techniques of the considered ML algorithms.

We hope that the ideas developed here may potentially detect new subtle and complex side channels in the future *before* large-scale deployment of an implementation.

Supplementary material We make the AutoSCA-tool[9] and all data[10] publicly available as open-source on Github.

4.2 Related Work

Attacks based on Bleichenbacher's The original padding oracle considered in [Ble98] was based on distinguishable error messages returned by a TLS server implementation.[11] Subsequent protocol versions then required indistinguishable error messages, in order to remove this particular way to construct a padding oracle. The difficulty of detecting and preventing *all* possible side channels that

[8] `https://tls.mbed.org/`

[9] `https://github.com/ITSC-Group/autosca-tool`

[10] `https://github.com/ITSC-Group/autosca-data`

[11] Early versions of TLS were actually called SSL, and re-named to TLS with the specification of TLS 1.0 by the IETF in 1999. We use the term TLS for all protocols versions.

may give rise to a padding oracle has been demonstrated by many research papers that appeared since the original publication of Bleichenbacher's attack in 1998. Klíma *et al.* [KPR03] introduced a new variant (a "bad version oracle", a special case of a padding oracle) and also used timing as a new way to construct the padding oracle required for Bleichenbacher's attack. Jager *et al.* [JSS12] showed that XML Encryption inherently provides a padding oracle, which is based on application layer properties of Web services and XML. Degabriele *et al.* [Deg+12] described attacks on the Europay-Mastercard-Visa (EMV) specification. Bardou *et al.* [Bar+12] developed a clever variant of Bleichenbacher's algorithm that may improve the performance of attacks significantly. They also found new padding oracles in several applications, including RSA SecurID tokens and several other hardware tokens, Siemens CardOS smartcards, hardware security modules, and even the cryptography implementation of the Estonian ID card. Meyer *et al.* [Mey+14] found several new padding oracles in the Java Secure Socket Extension (JSSE) TLS implementation and in hardware security appliances using the Cavium NITROX Secure Sockets Layer (SSL) accelerator chip. The DROWN attack [Avi+16] discovered another new vulnerability in OpenSSL that was present in OpenSSL releases from 1998 to early 2015, which gave rise to extremely efficient Bleichenbacher-style attacks, by leveraging an additional vulnerability in OpenSSL even in less than one minute on a single CPU. In 2018, Felsch *et al.* found new padding oracles in widely used IPSec implementations by Cisco, Huawei, Clavister, and ZyXEL [Fel+18]. Zhang *et al.* [Zha+14] and Ronen *et al.* [Ron+19] considered settings where the attacker is able to run code on the same physical machine as the victim, which circumvents many countermeasures to the aforementioned attacks. Even though this is a strong attacker model, it seems very reasonable in certain applications, such as cloud computing.

Automated scanning for vulnerabilities Recent analyses on new side-channel vulnerabilities come with large-scale evaluations of frequently used servers to estimate the attack impact. Such analyses were performed for ROBOT [BSY18], RACCOON [Mer+21], or CBC padding oracle attacks [Mer+19]. All these analyses have in common that the used scanners send test vectors to the servers and evaluated the potential side channels based on differences in server responses. Nevertheless, such an approach comes with potential false positives and negatives, resulting from unstable Internet connections and server behaviors; one broken TCP connection or connection timeout can change the server response resulting in a different behavior and thus in a potential side-channel report. Merget *et al.* attempted to solve these problems by rescanning vulnerable servers and by careful statistical tests [Mer+19]. The results of these approaches were integrated into common TLS scanning tools, such as SSLlabs[12] or testssl.sh.[13] Therefore, the tools are now able to cover a very wide range of *specific* and *known* vulnerabilities.

[12] `https://www.ssllabs.com/ssltest/`
[13] `https://testssl.sh/`

While the statistical tests developed in [Mer+19; Mer+21] are well-suited for precisely finding padding oracle vulnerabilities, the side channels they search for have to be manually defined by the researchers. For example, TLS-Attacker, which was used in [Mer+19; Mer+21], explicitly searches for side channels resulting from different messages, message counts, and TCP connection state differences. All these side channels have been defined after careful manual vulnerability assessments performed in the previous years [BSY18; Mer+19; Mer+21]. It is not guaranteed that the list of the side channels TLS-Attacker and other scanners search for is final. Unexpected behavior in the TLS implementation or the underlying TCP stack can reveal new side channels beyond message differences and TCP connection states, which are not explicitly analysed. This gap is addressed in our research; our tool observes the whole TLS communication and provides it to the machine learning algorithms, which are able to detect side channels *without previous assumptions* and explain the potential vulnerability to the developer.

Machine learning in side-channel analysis Previously, machine learning algorithms have been applied to detect side-channel attacks on the algorithmic and hardware level (cf. [Hos+11; Ler+15; MPP16b; Car+19], for instance, [HGG20] for a recent survey, as well as [Zha+20; Zai+21] for more recent works). To best of our knowledge, ours is the first approach to consider side-channel attacks on the cryptographic *protocol* level. A different research direction was proposed by Beck *et al.* , who analysed the automatic exploitation of adaptive chosen ciphertext attacks [BZG20]. In their work, they assumed a vulnerable implementation allowing an attacker to modify ciphertexts. They concentrated on the automatic exploitation development with SAT and SMT solvers based on the malleability characteristics of the encryption scheme. Our work extends this interdisciplinary research direction by analyzing ML algorithms for detecting new side channels.

4.3 Preliminaries

This section gives a brief introduction to Bleichenbacher's attack in the context of TLS in Section 4.3.1. We particularly consider TLS 1.2. We further summarize relevant concepts and terminology from machine learning in Section 4.3.2.

4.3.1 Bleichenbacher's Attack on TLS

The TLS 1.2 Handshake The TLS 1.2 [DR08] handshake, shown in Figure 2.2, is an essential first step in the establishment of a secure TLS connection. Performing the handshake, client and server agree on which cryptographic algorithms and parameters to use, they exchange the secret keys they later use to encrypt the actual data being transmitted, and the server proves its identity to the client:

The TLS Handshake consists of the following sequence of messages.

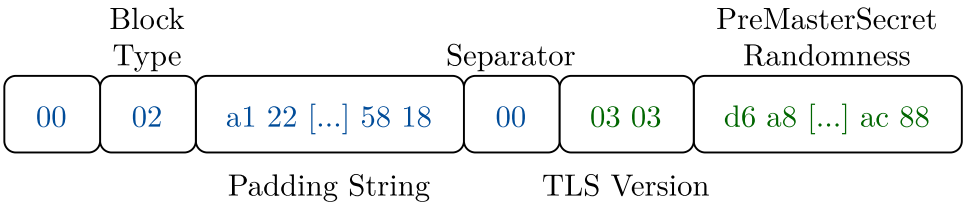


Figure 4.1: Padded Premaster Secret `PMS`.

(1) The client initiates the connection with the `ClientHello` message, and proposes different sets of cryptographic algorithms (so-called "cipher suites") it supports.

(2) The server selects a cipher suite and sends it to the client in the `ServerHello` message. It then proves its identity with a digital signature with respect to a certificate signed by a trusted third party. The `ServerHelloDone` message signals the end of this message flow. In the following, we consider a setting where a cipher suite based on RSA key transport is used.[14]

(3) The client uses the agreed-upon algorithms to perform the actual key exchange with the `ClientKeyExchange` (`CKE`). In the case of RSA-based key transport, the client generates a new random key called the premaster secret (`PMS`) and encrypts it with RSA, such that only the server can decrypt it. It then signals that the remainder of the conversation will be encrypted with this `PMS` by sending a `ChangeCipherSpec` (`CCS`) message. Finally, the `Finished` (`FIN`) message contains a cryptographic checksum overall key exchange messages, computed with a key derived from the `PMS`.

(4) The server decrypts the `PMS` and uses it to derive several cryptographic keys, which are then used to decrypt and verify the checksum of the `FIN` message. It concludes the handshake by sending `CCS` and `FIN` messages. Thereafter, all communication is protected using the keys derived from the `PMS`. Note that all cryptographic keys used for a TLS session are derived from `PMS` and other public values (such as nonces sent in plain text in the `ClientHello` and `ServerHello` messages).

RSA-PKCS#1v1.5 Encryption All TLS cipher suites based on RSA key transport use the RSA-PKCS#1v1.5 encryption scheme [Kal98] in order to transport the `PMS` from the client to the server. Essentially, this encryption scheme pads the `PMS` with constants and random bytes, as defined in Figure 4.1. The `PMS` is a random 48-byte string. The leading `00 02`, the separator byte `00`, and the TLS version number `03 03` (which refers to TLS 1.2) are constants. The padding string is chosen at random from all byte strings that do not contain a `00` byte (to

[14]The only cipher suite which is mandatory to implement in TLS 1.2 is of this type (`TLS_RSA_WITH_AES_128_CBC_SHA`).

guarantee that the separator byte is uniquely identified), and such that the total length of the padded string (including the `PMS` and all constants) is equal to the size of the RSA modulus N. The resulting padded string M is then encrypted with the "textbook" RSA encryption function as $c = M^e \bmod N$. We say that a ciphertext is *PKCS#1v1.5-conformant* or *has valid padding*, if $M = c^{1/e} \bmod N$ satisfies the padding scheme from Figure 4.1. There are several ways in which a ciphertext can be non-conformant, which we cover in Section 4.4.1.

Bleichenbacher's Attack Bleichenbacher's attack assumes that a "padding oracle" is available, which takes as input a ciphertext, and returns whether the ciphertext contains a plaintext with valid PKCS#1 v1.5 padding, with respect to a given target public key (N, e). Such an oracle may be constructed in many different ways, e.g., based on error messages returned by a TLS server implementation. A long sequence of research papers has developed many different ways to concretely construct such an oracle for certain concrete implementations or applications [Ble98; KPR03; JSS12; Deg+12; Bar+12; Zha+14; Mey+14; Avi+16; Fel+18; BSY18; Ron+19].

Bleichenbacher described a seminal algorithm [Ble98] which is able to use such an oracle in order to effciently compute the RSA decryption function $c \mapsto c^{1/e} \bmod N$ with respect to any number $c \bmod N$. Note that this can in particular be used to decrypt RSA PKCS#1 v1.5 ciphertexts, but also to compute valid RSA signatures with respect to the RSA key (N, e) contained in a server's certificate.

Essentially, the idea of Bleichenbacher's algorithm is as follows. Suppose that $c = M^e \bmod N$ be a PKCS#1-conformant ciphertext. This is without loss of generality, because if c is not, then one can use the oracle to "randomize" c by computing $\hat{c} = c\rho^e = (M\rho)^e \bmod N$ for random ρ, until $\hat{c}$ is PKCS#1-conformant, and then continue with $\hat{c}$. Thus, the number $c = M^{1/e} \bmod N$ lies in the interval $[2B, 3B)$, where B denotes the number modulo N whose binary representation is

$$B = \texttt{00 01 ... 00} = 2^{8(\ell-2)}$$

and where ℓ is the byte-length of the RSA modulus N.

Bleichenbacher's algorithm chooses a small integer s, computes

$$c' = (c \cdot s^e) \bmod N = (Ms)^e \bmod N,$$

If the padding oracle reveals that c' has a valid padding, then this implies that $2B \leq Ms - rN < 3B$, for some r, which is equivalent to

$$\frac{2B + rN}{s} \leq M < \frac{3B + rN}{s}.$$

Thus, M must lie in the interval

$$M \in [\lceil (2B + rN)/s \rceil, \lfloor (3B + rN)/s \rfloor).$$

By repeatedly choosing new s, this yields a set of intervals that narrows down the possible values of M, until only one possibility is left, which has to be the plaintext.

The perfomance of Bleichenbacher's algorithm mainly depends on the provided oracle, and how precisely it checks the validity of the padding. Bardou *et al.* described an improvement to Bleichenbacher's algorithm [Bar+12], and also analysed the concrete efficiency for various types of oracles.

4.3.2 Machine Learning

A binary classifier in our application considers two categories: 0 for secure implementation, and 1 for the considered side channel, cf. Figure 4.3. For our training dataset, we extract real-valued features from the network traffic ($\boldsymbol{x}$) and label each instance with the class-label, which represents the side channel corresponding to the type of error in the message (1) or if the message passed was correct. If there is an information leak or existence of a side channel in the system under test (SUT), then we can say that a classification algorithm can be used to learn an appropriate function/mapping between the network data and class-label. In particular, each data point is a two-tuple $(\boldsymbol{x}, y)$, where $\boldsymbol{x}$ is the real-valued d-dimensional feature vector extracted from network traffic, and y is its *ground-truth class-label*, see Section 2.3. The label y, takes one of two values which is determined by the property of the ciphertext. It is 0 when the padding is correct, and 1 when the padding is incorrect. We say that the SUT is vulnerable when the classification algorithm is able to learn on the binary classification data for the given manipulation, with high accuracy. This in turn implies an ability to distinguish between server reactions for the correct and incorrectly padded plaintext. Hence, when a server is secure, then all binary classifiers will not be able to label the instances $\boldsymbol{x}$ correctly, i.e. the accuracy should be very low. This corresponds to the observation in Section 3.2 that the Bayes classifier, the best possible binary classifier, is equivalent to random guessing if there is no information leakage. Using this concept, we determine the existence of the side channel with respect to a particular manipulation.

To ensure the robustness of our system, we employ multiple binary classification algorithms and aggregate the corresponding binary classifiers in order to decide on the vulnerability of the SUT. Recall that solving a binary classification task is akin to finding an unknown *target function*. As explained in Section 2.3 each binary classification algorithm is associated with a *candidate space* for the aforementioned target function. By employing a large set of binary classification algorithms, we are increasing the chance of finding mapping that is very close to the target function. Then we will be searching for the target function within the union of the *candidate space*. We obtain a set of binary classification models by training the set of binary classification algorithms [Mit97]. Each binary classification algorithm can therefore be thought of as searching its candidate space, for a *predicted function* that fits the given training dataset most accurately [Mit97]. In this thesis, we use Bayesian optimization techniques with the Gaussian processes to perform

the hyperparameter optimization (HPO) for different binary classifiers [FSH15]. The full list of the classifiers we chose, as well as an in-depth explanation of their workings, is covered in Section 2.3.

Hypothesis Tests Comparing the performance of two algorithms is a common problem in ML. In our approach, we want to compare the binary classifiers listed in the previous section with the random guessing (RG) baseline. This corresponds to the PTT-Random approach discussed in Chapter 3, which should work reasonably well for the balanced datasets we generate. For a given dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i), \ldots\}$, RG generates each class-label ($y_i \in \{0, 1\}$) uniformly at random without using the input $\boldsymbol{x}_i$. Assuming that the proportion of class-label 0 and 1 is equal in the given dataset, the accuracy of approximately 50% implies that the input features $\boldsymbol{x}$ are not used to predict corresponding class-labels $\hat{y}$. For $k = 2$ the RG produces an estimated average E_{out} or accuracy of 50 %. So, one can say that if a binary classifier is using input features $\boldsymbol{x}$ to predict the corresponding class-labels $\hat{y}$, its estimated out-of-sample accuracy would be greater than 50 % ($E_{out} > 0.5$). A binary classifier can only produce accuracy lower than 50 % if there exists noise in the dataset, i.e. if our testing dataset is not large enough or the number of accuracy estimates k is very low. Therefore, we use the Monte Carlo cross-validation (MCCV) approach to produce $k = 30$ estimates, with 30% of the data used for testing ($ts = 0.7$) [BG04].

One way to imply that the binary classifier can learn something about the class-labels using the input vectors $\boldsymbol{x}$ is to consider the differences of the mean accuracies of RG and the given binary classifier. However, this approach can be misleading as it is hard to know whether the difference between the mean accuracies ($1 - E_{out}$) is real or a result of a statistical fluke. For this reason, well-established techniques compare two resulting populations, rather than just their mean differences. For comparing the performance of two classifiers, there are two statistical tests which are proposed in the literature, the paired t-test [Dem06] and the Wilconson-Signed Rank test [Wil92].

Paired t-test The paired t-test is used to study if there is a statistical difference between two samples observed from two populations. Mathematically, it approaches the problem by assuming a null hypothesis $H_0(a_j == a_{rg})$. After applying the t-test, if the null hypothesis $H_0(a_j == a_{rg})$ is rejected, it indicates that the groups are different with high probability. Here, $a_j, \forall j \in \{1, \ldots, 10\}$ represents the population of the out-of-sample accuracy estimates of the binary classifier j and a_{rg} represents the population out-of-sample accuracy estimates of RG [BF04; Dem06].

The t-statistic value is used with the Student-t distribution with $N - 1$ degrees of freedom to quantify the p-value by calculating the area under the t-distribution

curve at value t, which is computed as

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(d_i = a_i - rg_i), \quad \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{(\mu - d_i)^2}{N-1}, \quad t = \frac{\mu}{\sigma}. \tag{4.1}$$

The p-value p_j is the probability of obtaining test results at least as extreme as the results observed during the test, assuming that the null hypothesis is correct [LR06; Dem06]. The critical value α of a statistical test defines the boundaries of the acceptance region of the test [LR06; Dem06], i.e. if we nhave p-value $p_j < \alpha$ for binary classifier c_j, then the null hypothesis $H_0(c_j == c_{rg})$ is rejected and we can say that c_j is significantly different/better than RG.

Corrected paired t-test A problem with using the Paired Student's t-test (and Wilconson-Signed Rank) is that the accuracy estimates of the binary classifiers are not independent. This is because the same data is used to train the model multiple times. This lack of independence in the evaluation means that the Paired Student's t-Test is optimistically biased. Nadeo and Bengio [NB03] showed that the violation of independence in the paired t-test might lead to an underestimation of the variance of differences. To solve this problem with the paired Student's t-test, they propose to correct the variance estimate by taking this dependency into account. The corrected variance is given by $\sigma^2_{Cor} = \sigma(\frac{1}{N} + \frac{ts}{1-ts})$, where ts is fraction of training datasets used by MCCV [BG04; NB03]. The t-statistic is calculated in the similar manner as in Equation (4.1), such that $t = \frac{\mu}{\sigma_{Cor}}$.

Holm-Bonferroni correction In statistics, this method is used to aggregate the the p-values of multiple hypothesis tests [Hol79]. To describe the process, we assume that we compared J classifiers with RG and produced p-values $p_1, \ldots, p_J$ and the corresponding hypothesis are $H_1, \ldots, H_J$. The significance level is defined for complete family, $\alpha = 0.01$. For each p-value, test whether $p_j < \frac{\alpha}{J+1-j}$, If so, reject H_j and the index j identifies the first p-value that is not low enough to validate rejection. Then the rejected hypotheses are $H_1, \ldots, H_{j-1}$ and the accepted hypotheses are $H_j, \ldots, H_J$. For our experiments, $j \geq 2$ (i.e., if at least one hypothesis is rejected) implies that performance significantly better than RG was achieved. In this case, we conclude that there exists a mapping between input space $\boldsymbol{x}$ to class-labels y, and reject the null hypothesis, for the family of classifiers. If $j = 1$ then no p-values were low enough for rejection, therefore no null hypotheses $H_1, \ldots, H_J$ are rejected (i.e., all null hypotheses are accepted). In this case, the analysis result will be given as "vulnerable", while the analysis concludes with "non-vulnerable" otherwise.

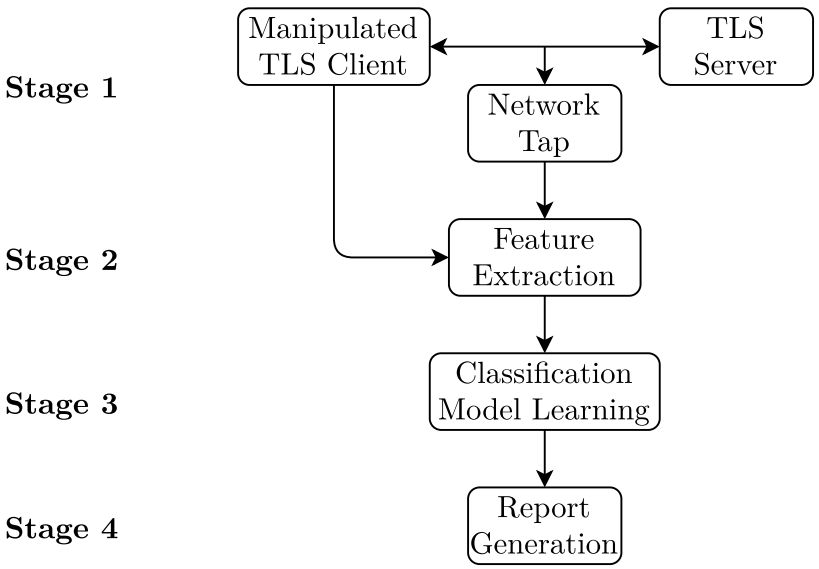


Figure 4.2: Components of the AutoSCA-tool

4.4 Implementation of Automated Side-Channel Detection

In this section, we describe how we implement our proposed approach in a software tool. The tool consists of the components shown in Figure 4.2, running after each other in four discrete stages:

Stage 1 The manipulated TLS client connects to the TLS server which is to be tested. The client then executes a pre-configured number of requests with manipulated padding while a network tap records all the exchanged messages. We describe this client in Section 4.4.1. We treat the server as a black box and run it in a docker container. For the network tapping, we record all traffic on the respective docker interface with tcpdump.

Stage 2 The raw data recorded in Stage 1 is transformed into a dataset that is suitable for machine learning. The feature extractor achieves this by extracting real-valued features from the messages contained in the network trace. It also matches these handshakes with the manipulation information from the manipulated TLS client, labeling each handshake with the associated padding manipulation. This is described in depth in Section 4.4.2.

Stage 3 The dataset is used to train and test classifiers. After training, the learned models are executed on the test data sets, testing the accuracy of their predictions. This process is repeated for different splits into test and training sets and the overall accuracy is determined. This process is presented in Section 4.4.3.

Stage 4 The results of the machine learning process are evaluated. The performance of each machine learning model is compared to a simple RG algorithm,

applying the Holm-Bonferroni test for significance. The final report then contains information on whether a model was able to significantly outperform RG, which would indicate the presence of a padding side channel. Feature importance is also included in the report as feedback to the software developer. This is shown in Section 4.4.4.

4.4.1 Manipulated TLS Client

For Bleichenbacher's attack, we need to recognize patterns in protocol network traffic that make it possible to distinguish messages with incorrect padding from messages with correct padding. Therefore we implement a TLS client that executes handshakes with valid and invalid paddings. The client builds upon TLS-Attacker [Som16],[15] a Java-based tool that allows for sending arbitrary TLS protocol messages with flexible modifications. It has already been used to detect new Bleichenbacher side channels [Som16] and implements several modified protocol flows. We use TLS-Attacker in our tool to execute n TLS handshakes (where n is a parameter, we choose $n \in \{500, 50000\}$), using a TLS Cipher Suite based on RSA key exchange.

The structure of the PKCS#1 v1.5 padded `CKE` message offers several distinct ways in which a handshake may not conform to the specification. We call each of these deviations a *manipulation*. TLS-Attacker already supports sending `CKE` messages with various padding manipulations. These manipulations include all attack vectors presented by Böck *et al.* in the ROBOT paper [BSY18], extended with attack vectors by Meyer *et al.* [Mey+14] and Klíma *et al.* [KPR03]. We use the following manipulations for our experiments:

- Correctly formatted PKCS#1 message: Standard-compliant message, the real `PMS` replaced with a random string of appropriate length
- Incorrect first byte: Replacing the first byte of the message, which should be 0x00, with a non-zero value (we chose a constant, 0x17)
- Incorrect Second Byte: Replacing the second byte of the message (block type), which should be 0x02, with a different constant (0x17)
- Invalid TLS version in PMS: Setting the TLS version bytes in the payload to an incorrect constant (0x42 0x42, a non-existing version)
- No 0x00 separator byte: Except for the first byte, all other bytes in the padded string that are 0x00 (particularly the separator byte) are replaced with 0x01
- 0x00 in PKCS#1 padding: The second byte of the padding string, which should be non-zero, is replaced with 0x00

[15] `https://github.com/tls-attacker/TLS-Attacker`

- 0x00 in PKCS#1 padding: Replacing the ninth byte of the padding string, which should be non-zero, with 0x00
- `PMS` is the empty string: Placing the 0x00 separator at the last byte, creating a payload of length 0.
- 0x00 On the last-but-one: Placing the 0x00 separator at the last-but-one byte, creating a payload of size 1.
- Correctly formatted, but short $|\texttt{PMS}| = 47$: Valid padding for a 47 bytes `PMS`, which should be 48 bytes long
- Correctly formatted, but 1 byte shorter: An otherwise correctly padded message, but with the the total length being one byte too short (not matching the RSA modulus size)

This aims to cover the majority of classes of padding errors one might expect. Of course, completeness cannot be guaranteed, but the set of considered manipulations can be easily extended, if considered useful in a particular context.

The client chooses at random whether and which manipulation to apply to a ciphertext. It then executes the handshake with the (manipulated) ciphertext. This process is repeated n times. The client logs which padding manipulation from the list above was applied to each handshake, which will be used in Stage 2 to match the observed network traffic to the manipulation. For example, when choosing "Incorrect first byte", the client manipulates the padding by replacing the first byte (which should be `0x00`) with a different constant. For the handshake message corresponding to a correct padding, the `PMS` is replaced with randomness, in order to ensure that the handshake will still fail as soon as the `FIN` message is processed. This corresponds to a step in the Bleichenbacher attack where the decrypted message has correct padding by chance and the padding oracle returns "correct padding", but where the actual `PMS` contained, which is unknown to the attacker, does not match the one used by the attacker in the `FIN` message. This way of randomly selecting a manipulation for each new handshake eliminates information leakage from the order of execution of the handshakes while producing balanced datasets for sufficiently large dataset sizes.

Our client also supports several different "workflows" of the TLS handshake. The first workflow we use is that of a "regular" handshake, consisting of `CKE`, `CCS`, and `FIN` messages. It is also necessary to test with a shortened workflow of a single `CKE`, without a `CCS` or `FIN` message, to trigger some vulnerabilities discovered in ROBOT [BSY18]. The workflow (full or shortened) and the padding manipulation are selected at random when executing a handshake.

Note that the missing `CCS` and `FIN` messages can result in server connection timeouts. Because of the timeout caused by the shortened workflow waiting for a potential response from the server, this becomes a limiting factor in client throughput. Consequently, the timeout for the client disconnecting from an idle

session needs to be set high enough to not miss any messages from the server. We used a timeout of one second for experiments in a local environment and three seconds for remote servers, which turned out to be appropriate for the respective network delays and gave the analysed TLS libraries enough time for their responses.

4.4.2 Feature Extraction

The "padding oracle" in Bleichenbacher's attack is essentially an abstraction of a way that enables the attacker to efficiently distinguish whether the tested server acts differently on validly or invalidly padded ciphertexts. We consider an attacker that is able to observe and record the entire network traffic. Therefore we need to ensure that the same information is available to the ML algorithms, including the labeled padding modifications performed by the TLS client.

The data obtained in Stage 1 is the traffic exchanged between the manipulated TLS client and the TLS server, recorded using tcpdump. This results in a .pcap file containing all messages in their original binary representation, as well as their metadata. The feature extractor transforms the raw data into a feature representation that is applicable for training a classifier. A standard approach is to use datasets consisting of labeled real-valued vectors, where the label contains the class the particular vector (called "instance") belongs to.

In our case, each handshake corresponds to a single instance in the dataset, with the padding manipulation used as the class label for the instance. Thus, handshakes with the same manipulation end up as instances of the same class in the dataset. An instance has to be represented by an n-dimensional real-valued vector, where each dimension corresponds to a feature. Hence, we have to reduce all network messages belonging to a handshake to a single vector. This is necessary because these messages are intrinsically linked with each other through the handshake process and have to be treated as a single entity in ML, in order to be able to capture patterns exhibited by combinations of messages.

To achieve this transformation, we build upon the popular network analysis tool Wireshark, which our tool automatically interacts with using the Python library pyshark. By taking all the protocol fields from the Wireshark output, we can transform the messages into real-valued feature vectors compatible with ML.

The result of this transformation is a vector of high dimensionality. We need to make sure its dimensionality is not too high, as this affects the performance of most ML algorithms negatively due to a phenomenon often called the "curse of dimensionality" or the "peaking phenomenon" [Hug68; TK09]. The peaking phenomenon states that the predictive power of a learning model (classifier) first increases with the number of features in the dataset, but after a certain number it starts deteriorating instead of improving steadily [TK09]. One way to mitigate this problem is by increasing the size of the dataset to at least 5 training instances for each dimension in the dataset, or by reducing the dimensionality of the feature vectors [TK09].

The dimensionality of our dataset is higher, if the feature vector contains more messages or more network protocol fields. We can discard all messages sent by the server before it receives the manipulated Client Key Exchange itself, as they cannot be influenced by the padding manipulation. This reduces the number of messages contained in the vector reducing the dimensionality at the same time.

We also chose to only export data on the TCP and TLS layer in our experiments. Our decision was based on the previous works on side-channel attacks [Mer+19; BSY18], which only detected behavioral differences on the TCP layer and above. Our reductions dramatically reduce the feature dimensionality, making the experiments feasible with limited resources.

Note that in our experiments in this chapter, we do not consider the timing of messages, even though several attacks construct a padding oracle using timing. The treatment of timing information requires further consideration. We decided to configure the current feature extractor to filter out all timing-related features in our experiments to prevent accidental leaks of padding status from the client side. A non-constant-time client implementation (like TLS-Attacker) could inadvertently leak information about the used padding manipulation into the timing features, causing false positives.

However, note that while our investigations in this chapter do not consider timing, it is possible to use the AutoSCA-tool for this purpose. This was demonstrated later by Berlinblau [Ber21], who identified which specific features contain client-side timing information and have to be excluded from the extraction process. Based on her work, Funke [Fun22] improved the AutoSCA-tool to be able to use a high-precision timing network card. When connecting to a vulnerable server on the local network, the tool was able to detect timing side channels of around 30 µs. However, similar to our investigations in this chapter, these improvements did not reveal any new side channels in current open-source TLS servers.

Finally, because we are using supervised machine learning, we label the vector with the padding manipulation applied by the client in this handshake.

4.4.3 Classification Model Learning

In this section, we discuss the steps taken by the ML component in order to determine the existence of a pattern in the dataset, which would imply the existence of a side channel. This process is illustrated in Figure 4.3.

Multi-class to binary conversion The dataset obtained in Stage 2 consists of handshakes, involving either a correctly padded message or one of 10 plaintext manipulations. When using a multi-label classifier on the dataset, we observed that the performance was poor. This was because of the fact that many different manipulations may lead to the same server behavior. In its stead we formulate K binary classification problems, since we are interested in distinguishing any of the K manipulations from the correct padding, as this indicates information

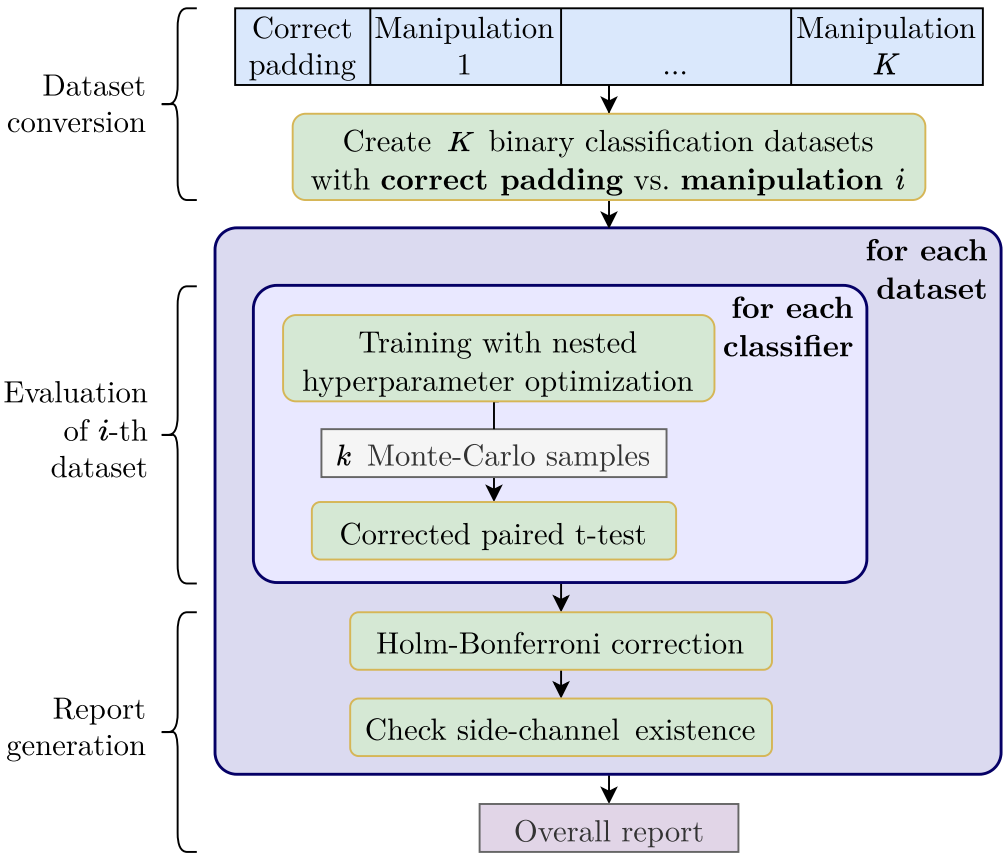


Figure 4.3: Concept of classification model learning (stage 3) and report generation (stage 4).

leakage. Recall that every instance $\boldsymbol{x}$ is associated with a discrete valued class-label $y \in \{0, 1, \ldots, K\} = [K]$. It takes value 0 when there is no manipulation, and values between 1 and K are directly related to the types of manipulation. Our binary classifier considers manipulation k, where $1 \leq k \leq K$ and tries to distinguish it from the correct plaintext for all K manipulations individually.

Evaluation At the core of the process described in Figure 4.3 is a familiy of binary classifiers that are trained to distinguish between correct padding and a specific manipulation. Evaluating the performance of the classifiers outside the given dataset is pertinent, since we do not want the classifiers to overfit the given dataset. Within our framework, overfitting refers to the problem of finding a pattern within the dataset that does not exist in general. We use cross-validation as discussed in Section 2.3 to be able to detect and avoid overfitting. Since we are interested in creating a robust detection method, we use the MCCV approach to get several accuracy estimates with less variance for our binary classifiers. Another

key step in training a binary classifier successfully is to tune its hyperparameters. For this we use HPO as presented in Section 2.3, with 2-fold cross-validation to get the best hyperparameters for the classifier. For the actual classifier training, we use 30 splits of the dataset into 70 % training data and 30 % validation data. This results in $k = 30$ Monte-Carlo samples of the predicted accuracy. Finally, note that we train multiple classifiers independently to increase the robustness of our verdict.

Since the binary classifier needs to be significantly better than the RG to imply information leakage, we apply the corrected paired t-test. This test provides a t-score and a p-value. The t-score quantifies the performance difference between the binary classifier and RG. The associated p-value gives us the confidence that the t-score represents the true performance difference. Note that a better performance than RG typically implies a high classification accuracy. To summarize, we return the p-values of the family of binary classifiers for each manipulation, which are subsequently interpreted in stage 4. In order to conduct the experiments in a fair and unbiased way, we use a family of classifiers listed in Section 4.3.2. We use the scikit-learn [Ped+11] library, which implements the required ML algorithms, cross-validation techniques and evaluation measures. For HPO we use the scikit-optimize library [Hea+20], which implements different hyperparameter optimizers compatible with the binary classifiers implemented in scikit-learn [Ped+11].

4.4.4 Error Correction and Report Generation

As explained in the previous section, in order to increase the test robustness, we use an ensemble of binary classifiers. Because it is not known in advance which classifier might perform particularly well on a new, unknown side channel, trying several in an ensemble increases the chance that a suitable one is among them.

Family-wise error rate Since we have multiple classifiers, the false positive rate when combining them is higher than if only a single one were used, as a single classifier outperforming the RG is already sufficient to conclude that implementation is vulnerable. The p-values obtained from the multiple independent tests of each classifier, therefore, need to be adjusted and then aggregated to give us a final verdict on the vulnerability of the server with respect to the manipulation. We use the Holm-Bonferroni test [Hol79] as described in Section 4.3.2 to correct this error.

Feedback We aim to provide detailed information about any detected side channels to support software developers in removing them. The first information we provide is which manipulations were distinguishable from correct padding. This indicates which padding check is at fault. Another important part of the report is whether this side channel was detectable with the full (`CKE`, `CCS`, `FIN`) handshake or the shortened one (`CKE` only). The random forest classifier additionally provides

information about the importance of features (c.f. Section 2.3) in the dataset. This can provide valuable feedback for the software developer about which parts of a network trace need to be investigated and what the cause of the potential padding oracle vulnerability could be. For example, as explained in the analysis below, a different TLS alert message in response to a padding failure can be easily pointed out, since this is the single most important feature in the dataset.

4.5 Analysis

We explore the capabilities of our proposed methodology by applying the tool described in Section 4.4 to various TLS server implementations. In Section 4.5.1, we explain our experimental setup. In Section 4.5.2, we describe the results of a first basic validation of the approach, which executes our tool against a completely insecure implementation, and one implementation which is considered secure. The test confirms that the vulnerabilities are found and that the secure implementation does not exhibit any noticeable patterns. In Sections 4.5.3 and 4.5.4 we continue the analysis by testing the ability to spot diverse real-world side channels. To this end, we investigate the side channels discovered in ROBOT [BSY18] and the side channel in OpenSSL Version 0.9.7a from [KPR03]. Again, the analysis confirms that the tool can successfully and reliably detect all of these side channels. Finally, in Section 4.5.5 we also present the results of an analysis of the most recent versions of a large number of popular open-source implementations, which does not yield any new unknown vulnerabilities.

4.5.1 Test Setup

The tool performs and captures $n \in \{500,\ 50000\}$ handshakes. Usually, we run 50,000 handshakes, since ML algorithms often tend to classify more reliably based on larger datasets. For the ROBOT attack, we experimented with 500 handshakes, as this size is better suited for a large-scale scan of public servers and more appropriate for a quick scan as part of automated regression testing. Depending on the network conditions and timeout settings, the 500 handshakes can be completed in as few as fifteen seconds, with the feature extraction taking about ten seconds. The training process of the classifiers is by far the most time-consuming stage of the process, but for a dataset of this size, it remains under three minutes. In comparison, the overall running time for the big 50,000 handshake datasets is just under two hours.

Our analysis showed that this is a reasonably large number of requests to provide sufficient confidence in the analysis result. Detailed analysis of the proper dataset size is covered in Appendix A. All handshakes are executed sequentially without any waiting time.

The servers are running on the same machine as the rest of the setup, isolated in docker containers. The network traffic is captured on the docker virtual network

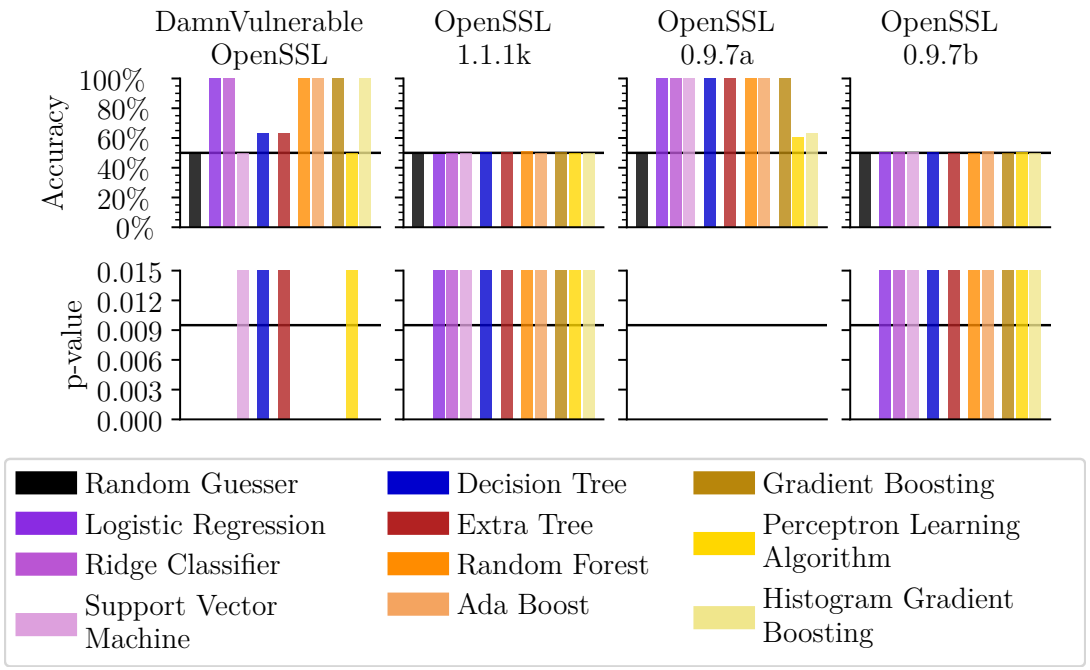


Figure 4.4: Performance of different classifiers. The two plots on the left are obtained in the basic approach validation, considering DamnVulnerableOpenSSL and OpenSSL 1.1.1k. The two plots on the right are obtained in the analysis of the Klíma-Pokorný-Rosa attack [KPR03].

interface, which ensures there is no interfering traffic. Since both server and client run on the same physical machine, we introduce an artificial 2 ms delay on the interface. Otherwise the very low delay could cause aggressive TCP retransmissions, which appear as noise in the collected dataset and make the automated detection of patterns more difficult and less reliable. The 2 ms delay produces TCP behavior that is closer to the real world, while still being low enough not to negatively influence the timeout of the client, which is set to 50 ms, or overall performance.

The tool is executed on an otherwise idle workstation with an AMD Ryzen 9 3950X 16-core CPU, 64GB RAM and an Nvidia RTX 2080 Super GPU, running Debian 10.7 and Python 3.7.3.

When executing the machine learning component, we get 30 accuracy estimates samples using $ns = 30$ MCCV train-test datasets. This sample size allows applying the Holm-Bonferroni test at a significance level of $\alpha = 0.01$, which is shown as a horizontal bar in the figures. This 99% confidence threshold appears to be a good tradeoff between false positives and false negatives.

4.5.2 Basic Approach Validation

We use the OpenSSL version 1.1.1k (released in August 2021) as our baseline for a non-vulnerable server implementation, as it is considered to be secure against

Table 4.1: Most important features leading to a side channel in DamnVulnerableOpenSSL, extracted automatically using the random forest algorithm.

Feature Name	Importance
Description of first TLS Alert	1.0
TCP ACK number of the 2nd TCP Disconnect	0.93
TCP RST of the 2nd TCP Disconnect	0.90

Bleichenbacher padding oracle attacks, based upon the scrutiny of both the open-source community as well as the attention of security researchers. Our baseline for a vulnerable server implementation is DamnVulnerableOpenSSL,[16] a patched TLS implementation which intentionally contains a padding oracle vulnerability for experimental and educational purposes.

Figure 4.4 shows how the different classifiers perform on our two baseline datasets. For the non-vulnerable OpenSSL 1.1.1k, all classifiers achieve an accuracy of approximately 50%. Consequently, none of the p-values exceeds the threshold for acceptance, i.e., to be considered potentially vulnerable. For DamnVulnerableOpenSSL, some classifiers perform quite well while some others, like perceptron learning algorithm (PLA) or support vector machine (SVM), do not outperform RG. This is also reflected in the associated p-values. The performance of the classifiers decision tree (DT) and extra tree (ET) is somewhat higher than 50%. However, their p-values indicate that we cannot rule out that this happened by chance. The performance of other classifiers, e.g random forest (RF), which significantly outperform RG when applying the Holm-Bonferroni correction, means that we can still conclude this server is vulnerable. We consider this as support of the approach of using many different machine learning algorithms in parallel.

Table 4.1 shows the most important features in the random forest algorithm for DamnVulnerableOpenSSL. This is the feedback that the tool provides to a software developer on the detected side channel. The highest importance is associated to the TLS alert message. Since DamnVulnerableOpenSSL was specifically crafted to return a different TLS alert for incorrect paddings (`handshake failure` instead of `bad record mac`), this confirms that the tool provides correct feedback.

Our approach was also able to spot another subtle change in the error handling; in case of a padding failure, the server disconnects the TCP connection slightly differently, by sending a TCP reset right after a TCP finished message, instead of waiting for the client to acknowledge the TCP finished. This causes the `RST` TCP flag on the second disconnect message to be set only when processing incorrect padding, which consequently appears as the third most important feature in telling the two classes apart. This behavior also causes a shift in the acknowledgement numbers, which was detected and used by the random forest algorithm, as indicated

[16] `https://github.com/tls-attacker/DamnVulnerableOpenSSL`

by the second-highest feature importance score. Hence, the approach is also capable of identifying such subtle side channels.

4.5.3 Detecting Klíma-Pokorný-Rosa Side Channels

As a next step, we analysed the detection of real-world side channels in old implementations. According to the OpenSSL changelog,[17] prior versions of this implementation exposed several side channels. The first changes were applied prior to version 0.9.5, where Bleichenbacher's attack was fixed after publication of the original paper in 1998. As mentioned in the release notes, this fix was not sufficient, as the error caused by a padding failure was not properly ignored. Even worse, the countermeasures were accidentally removed in version 0.9.5. This is explained in the release notes for version 0.9.6b (released in 2001), claiming that this version then contained the first working protection against Bleichenbacher's attack. Unfortunately, the source code of these versions is no longer available for download.

Versions 0.9.6j and 0.9.7b (released in 2003) then contained another change, this time to address the recently published Klíma-Pokorný-Rosa bad version oracle attack [KPR03]. Version 0.9.7a (vulnerable) and 0.9.7b (not vulnerable to the Klíma-Pokorný-Rosa attack) are available and can be compiled, so we use them to analyse the approach when faced with a bad version side channel.

TLS handshakes with manipulated `ClientKeyExchange` (`CKE`) messages containing an incorrect TLS version resulted in different server behavior of version 0.9.7a. This was correctly detected by our tool. The classifiers were able to significantly outperform RG. The tool consequently detected the Klíma-Pokorný-Rosa bad version side channel present in 0.9.7a. We then tested 0.9.7b and no classifier was able to outperform RG, with the tool returning a "not vulnerable" result. This indicates that the applied countermeasures are successful in preventing these side channels.

4.5.4 Detecting ROBOT Side Channels

Another range of real-world side channels to apply our tool on was presented in the ROBOT paper [BSY18], where Böck, Somorovsky and Young found numerous side channels in public web servers. Most of these were found in closed source TLS server implementations. The authors informed the affected vendors, most of which published updated software versions for their devices. Consequently, we expect these side channels to be no longer present in most web-facing TLS servers.

To verify our approach is also suitable for large-scale internet scans, we ran our tool on the domains in the Alexa Top 500 web page ranking.[18] Because this scan necessitates connecting to machines outside of our control, the workstation

[17] `https://www.openssl.org/news/changelog.html`

[18] `https://web.archive.org/web/20180321225122/https://www.alexa.com/topsites`

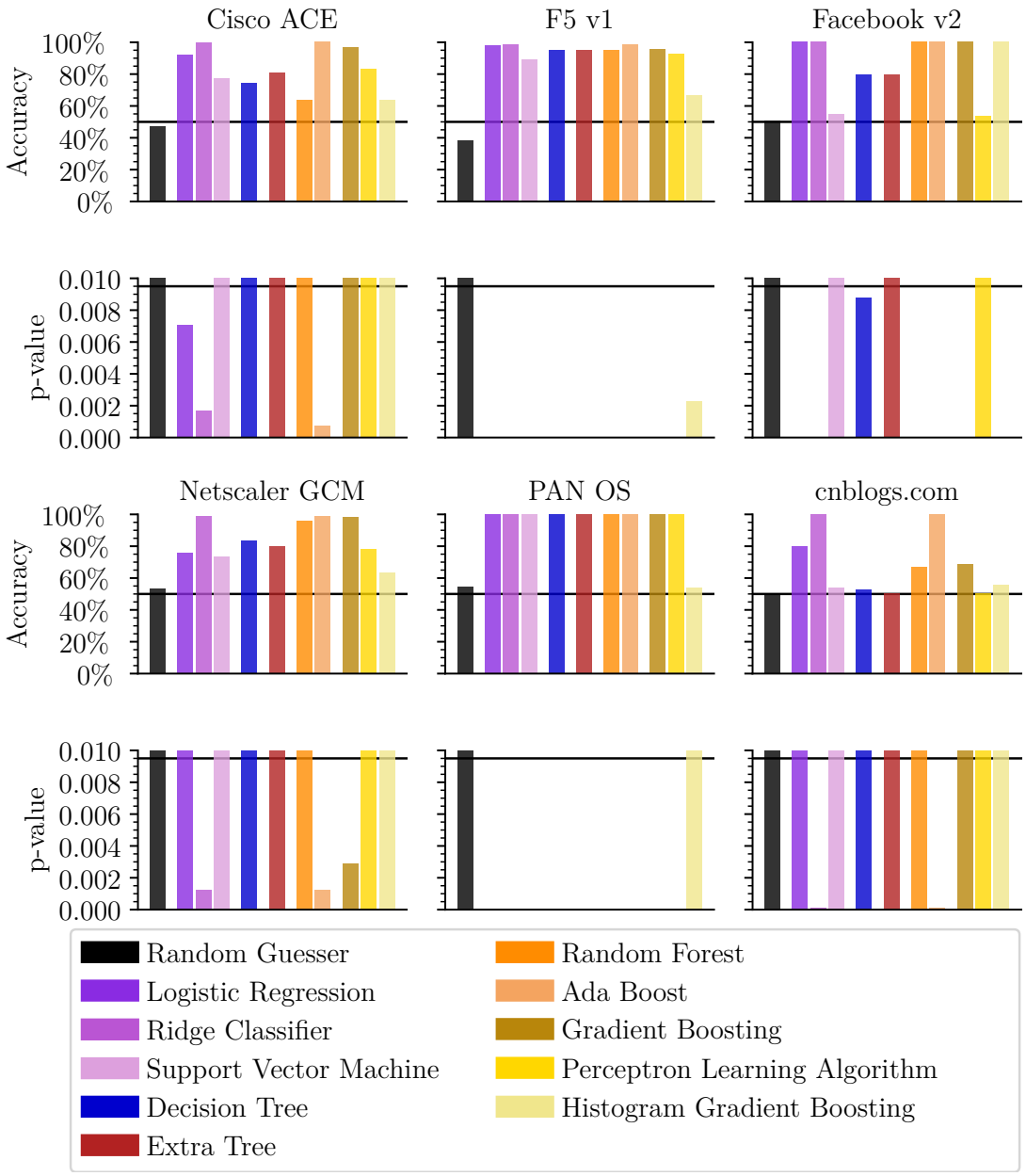


Figure 4.5: Performance of different classifiers for the ROBOT servers generated with manipulated `ClientKeyExchange` (`CKE`) messages containing 0x17 instead of 0x00 as the first byte of the PKCS#1 v1.5 padding. Each server was successfully labeled as vulnerable by at least one of the used classifiers.

used for the scans serves an informational web page with contact information for concerned server operations that want to opt out of the scans. The tool was set up to execute an independent test consisting of 500 handshakes with each of the domains in the Alexa Top 500 ranking. For 5 domains the hypothesis test rejected, indicating a possible vulnerability. To investigate further, a larger test with 50000 handshakes was executed with these servers. This revealed that 4 of the 5 were false positives. The fifth, cnblogs.com, is a genuinely vulnerable server, with the result of the initial 500-handshake scan shown in Figure 4.5. This was confirmed using the established ssllabs[19], tls-scanner[20] and testssl.sh[21] TLS scanners, which were able to detect the side channel as well. We then reached out to the server operators and notified them of the issue, but did not get any response.

This confirms that our approach scales to hundreds of web servers, albeit at the cost of an increased running time compared to other ROBOT scanners. On the other hand, our approach requires fewer assumptions about the nature of the side channel. Additionally, this test also proves that the tool can be applied outside of lab conditions in the real world, where network behavior has a bigger influence on the recorded traffic traces.

We have thus determined that almost all of the side channels covered in ROBOT have since been removed from high-profile web servers. To evaluate if our tool would be able to detect these side channels nonetheless, we deliberately recreated them by imitating their behavior in modified versions of mbedTLS. We decided to imitate five ROBOT vulnerabilities that cover a representative set of server behaviors:

- When F5 v1 (CVE-2017-6168) and Facebook v2 are tested with the reduced workflow of a single `ClientKeyExchange` (`CKE`) message, this should result in a TCP timeout caused by the server waiting for the `CCS` and `FIN` message. For incorrect paddings, these servers do not wait but abort prematurely with a TLS alert or a TCP disconnect (TCP finished message).

- Cisco ACE (CVE-2017-17428) responds with a TLS alert 47 instead of alert 20 if the padding check fails.

- Citrix Netscaler (CVE-2017-17382) gives a TLS alert 51 for correct padding but does not send any data for incorrect padding, causing the TCP session to time out.

- PAN OS (CVE-2017-17841) sends the same TLS alert 40 in both cases, but also sends a duplicate of the alert in case of a padding failure.

Again, we performed 500 handshakes to generate the datasets as we did in the Alexa 500 scan. As can be seen in Figure 4.5, the tool was able to correctly classify

[19] `https://www.ssllabs.com/ssltest`
[20] `https://github.com/tls-attacker/TLS-Scanner`
[21] `https://testssl.sh/`

Table 4.2: open-source TLS servers tested

Name	Version
BearSSL	0.6
BoringSSL	commit 3743aafd
Botan	2.17.3
Bouncy Castle	1.64
GnuTLS	3.7.0
LibreSSL	3.2.3
MatrixSSL	4.3.0
Mbed TLS	2.25.0
OCaml-TLS	0.12.8
OpenSSL	1.1.1k
s2n	0.10.25
tlslite-ng	0.8.0-alpha40
wolfSSL	4.4.0

all servers as vulnerable. This was confirmed by a Holm-Bonferroni test, with at least one classifier significantly outperforming RG in each experiment.

4.5.5 Testing open-source Implementations

After establishing that our method is indeed able to detect known side channels, we applied it to up-to-date open-source TLS server implementations. Table 4.2 shows all software versions we investigated. For this experiment, we executed 50.000 handshakes each.

After applying the Holm-Bonferroni correction, we concluded that no classifier significantly outperformed random guessing for any of the servers we tested. Our experiments excluded timing side channels and are limited to the range of manipulated handshakes executed by the TLS clients, and as such might miss novel attack methods. We are confident, however, that within these limitations not a single of these TLS servers is vulnerable to a conventional Bleichenbacher-like padding oracle attack.

4.5.6 Commercial Integration

Finally, with the previous experimental results inspiring confidence into the reliability of the detection method, the detection component was included in the TLS Server Inspector. This is a a commercial software offered by the company achelos GmbH that is used to test TLS servers, e.g. those used in sensitive applications like health care, for compliance to the TLS standard specification. The achelos engineers were able to integrate the detection mechanism due to the modular design of the AutoSCA-tool, which allowed them to reuse existing

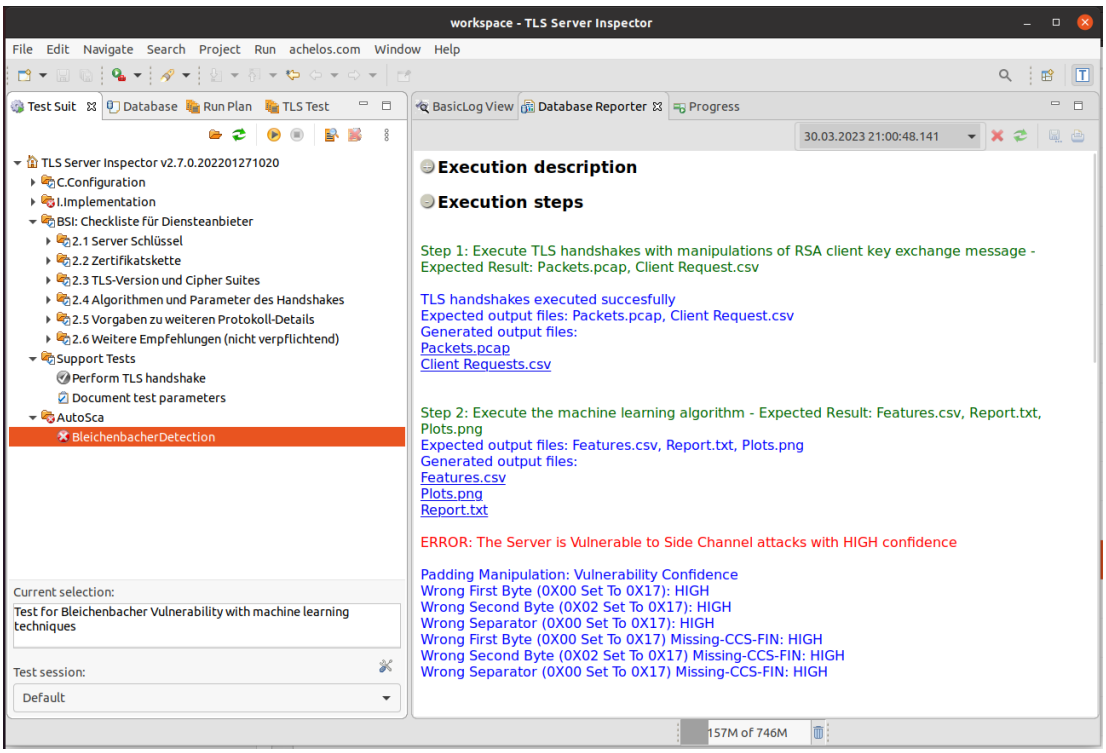


Figure 4.6: Screenshot of the automatic detection running in the TLS Server Inspector. The server tested contains a side channel, which is detected and causes the test case to fail. The main window shows the feedback view with details of the detected side channel.

software components like the TLS client and the network tap. Figure 4.6 shows the result of such a test in the case of a side channel being detected, with the full engineer feedback on the details of the side channel shown in the right panel. This demonstrates the fully automatic nature of our approach, allowing a software tester without special training in cryptography to execute the Bleichenbacher test case simply as part of a larger suite of security tests.

4.6 Conclusions and Open Problems

We propose, implement, and analyse an approach to automatically detect protocol-level side channels in implementations of cryptographic protocols such as TLS. We consider Bleichenbacher's attack as a concrete use case, due to its repeated appearance in many popular open-source implementations and widely-used commercial products.

A major advantage of our approach is that the side-channel vulnerability detection is fully automated, robust, and therefore scales much better than manual analysis. In particular, it could be applied as a standard test before every release of a new version of an implementation, possibly automatically in a CI/CD pipeline.

This would also prevent the accidental removal of countermeasures, as happened in version 0.9.5 of OpenSSL, for example.

Our analysis confirms that this approach, despite being fully automated, is able to reliably recognize the patterns in network traffic on which the padding oracles identified by Klíma, Pokorný, and Rosa [KPR03] and ROBOT [BSY18] are based. We did not yet find new weaknesses in the popular TLS implementations that we have analysed with our tool. However, we were able to reliably detect the aforementioned padding oracles with a general and generic approach, which provides confidence that it will also work for future, new side channels that exhibit distinguishable patterns on the network layer.

In our analyses, we showed that a single binary classification algorithm is not reliable enough to detect every side channel; the used classification algorithms performed differently based on the tested TLS server side channels. The robustness of our approach was achieved by the usage of an ensemble of machine learning algorithms for the task of detecting vulnerabilities. This is further reinforced by the use of a family-wise statistical test to greatly reduce the chance that a vulnerability found by our ensemble is an accident. This technical knowledge can be used by designing future approaches for side-channel analyses.

We have intentionally focused on ML algorithms amenable to *feature importance* techniques, because we consider an automated tool particularly useful, if it is able to provide concrete feedback on potential vulnerabilities. This is especially important if the detected side channel is subtle, as a software engineer using our tool might simply be unable to address the underlying issue otherwise. Adding deep learning methods implemented in scikit-learn to the ensemble of algorithms would be straightforward. This might be desirable as deep learning methods are usually considered to be more powerful in the case of very complex relationships between features and labels. However, deep learning is generally less easily interpretable and often treated as a black box.

Research Question 3. *Are there ways to add interpretability to deep learning methods such that they can be used in the context of side-channel detection as well?*

We believe that the techniques developed in this Chapter are applicable more generally, beyond Bleichenbacher's attack on the network layer. One obvious alternative would be Vaudenay's attack on CBC padding [Vau02]. However, non-padding side-channel attacks usually don't result in the clear padding correct/incorrect distinction considered here.

Research Question 4. *Is it possible to apply the method to side channels where the oracle has a larger number of possible answers?*

Possible future extensions could additionally take local states of implementations, as in [Zha+14; Ron+19], into account by using them as additional features.

The way in which our tool generates the data using its own TLS client ensures that the dataset remains balanced. This setup might not be suitable for other

applications, resulting in imbalanced data. Chapter 3 showed that some detection approaches are better suited to handle this imbalance, while others fail outright.

Research Question 5. *Which side channels might result in imbalanced data? Is our approach able to detect them with alternative detection methods, e.g. FET-Mean and FET-Median?*

5 Automating Hardware Attacks

In this chapter, we look beyond the remote side-channel detection investigated in the previous chapters. Instead, we focus on local side-channel attacks on devices like smart cards and trusted platform modules (TPMs), which we introduce in Section 5.1. As we explore in Section 5.2, machine learning and especially deep learning is already an established attack method for local side channels. The special attack methodology for these attacks is covered in Section 5.3.1, with the existing metrics already sufficiently capable to detect local side channels. To achieve side-channel detection, it is therefore sufficient to run the attack and check if it is able to succeed to some degree. However, state-of-the-art attacks are not fully automated, as the *architecture* of the deep learning models (covered in Section 5.3.2) has to be tailored to each device and dataset manually. We therefore propose to automate this process using neural architecture search (NAS) as presented in Section 5.3.4. This proposed approach also needs to result in a reliable performance estimate in a black-box setting, which we develop in Section 5.4. To analyse our attack performance for different input shapes and hyperparameter optimizer we performed a big parameter study on a supercomputer, which we describe in Section 5.5. In Section 5.6, we investigate the performance and determine the optimal combination of input shapes and hyperparameter optimizer and compare our NAS performance to that of hand-crafted architectures. This demonstrates that our attack method is able to achieve a similar attack efficiency to non-automated attack. This makes NAS a viable alternative for preliminary investigations of new datasets and enables automated side-channel detection for local side channels. Section 5.7 then concludes the chapter and presents future research directions.

Author's contribution The following chapter is based on joint work together with Pritha Gupta and Eyke Hüllermeier [GDH23]. Pritha Gupta contributed the implementation of the experimental setup, design and execution of the experiments, the idea to use input reshaping and specific search strategies, and collected the baseline models. The proposed methodology for black-box testing as well as the result analysis and interpretation were developed jointly by Pritha Gupta and the author of this thesis. The author's contributions are the collection and evaluation of the datasets used for the experiments and the performance analysis of the attack convergence.

5.1 Introduction

When it comes to hardware side channels, machine learning techniques have long been the tool of choice for attackers. One of the most powerful tools, deep learning, is particularly promising, as it is capable of learning very complex mappings between the secret key used in the encryption and the observed power consumption or electromagnetic emissions. One issue with using convolutional neural networks (CNNs) is designing an appropriate architecture for the network. A good architecture that matches the requirements of the dataset at hand can perform incredibly well, sometimes even predicting the correct key byte after observing a single attack trace. On the other side, a bad architecture may fail to give any useful predictions, no matter how much training data it is given. The hardware side channel community has been struggling with this, failing to come up with good and general rules for architecture designs that are suitable for arbitrary attack datasets. There is one way around this issue, though: Instead of trying to come up with architecture design guidelines by manual architecture tweaking, why not treat the choice of architecture as just another machine learning (ML) hyperparameter that can be optimized automatically? This can be achieved with NAS, which explores a pre-defined hyperparameter space of CNN architectures by repeatedly training models and determining their performance. If done correctly, this approach should be able to easily identify which architecture works well for any given dataset, performing at least as well as carefully constructed manual models.

The NAS approach was first applied to hardware side channels recently by Wu, Perin, and Picek [WPP20] and Rijsdijk, Wu, Perin, and Picek [Rij+21], paving the road for automatic architecture design. These approaches use the secret key of the traces in the attack dataset to perform the search for an optimal architecture [WPP20; Rij+21]. This is only acceptable in a white-box setting, as the attack dataset can no longer be used as a test dataset to get an unbiased performance estimation. The strategy for performing the actual NAS has a large impact on the quality of the final architecture, with [WPP20] exploring Random and Bayesian search strategies and [Rij+21] proposing a search strategy based on reinforcement learning. The experimental analysis of these works was also limited in scope, as only a small number of datasets were considered. Even though the state-of-the-art CNN architectures were inspired by 2-D image classification models, ML-based side-channel attacks (SCAs) have only been applied to them using 1-D inputs.

Our contributions

- We propose a NAS approach that relies only on using the profiling dataset for optimization, which makes it suitable for a black-box setting. In addition, it is set up to perform several independent attacks, which produces more reliable performance estimates.

- We expand the previous NAS experiments into a large-scale parameter study, investigating the impact of search strategy by considering four different search strategies, including Greedy and Hyperband, as well as 2-D CNNs. Our evaluation is performed on 10 publicly available reference datasets in both the identity (ID) and Hamming weight (HW) leakage model.

- We also conduct a performance comparison between the CNN architectures obtained from our NAS approach and the state-of-the-art fixed architectures proposed by Benadjila, Prouff, Strullu, Cagli, and Dumas [Ben+20] and Zaid, Bossuet, Habrard, and Venelli [Zai+19].

5.2 Related Work

Breaking cryptographic implementations using their side-channel emissions has a long history, particularly in the intelligence community, who has been aware of such issues since the 1950s. In the scientific world, a breakthrough was achieved in 1999 with the introduction of differential power analysis (DPA) by Kocher, Jaffe, and Jun [KJJ99]. This attack needs to observe a large number of cryptographic operations, e.g. en- or decryptions, while measuring the power consumption or electromagnetics (EMs) of the target device. These traces, consisting of thousands of measurements over time, are then matched to possible computations of the executed function using statistical methods, revealing the encryption keys.

This approach of attacking the target device directly requires thousands of observations during the attack, making it difficult to execute in real-world scenarios where only a handful of traces can be obtained. If a device sufficiently similar to the target device can be obtained, for example by buying a second copy of the device, its profile (a model of its leakage) can be created by observing a large number of cryptographic operations with known keys and plaintexts. In the attack phase, fewer measurements need to be obtained from the actual target device, which are subsequently matched up with this leakage model. Chari, Rao, and Rohatgi [CRR03] developed template attacks for this scenario, where several parts of an attack trace are matched to the distributions of the template traces.

Creating such a function linking secret key inputs to output traces was soon recognized to be a possible application for ML algorithms. These models are trained on the profiling traces and predict the secret key used on the attack traces. Early ML-based SCAs were already capable of dealing with measurement noise and misalignment in the traces [Hos+11]. Soon, the ML models grew more sophisticated and the attacks became more successful, capable of breaking devices that have been explicitly hardened against side-channel attacks after observing only a handful of attack traces [Hos+11; Ler+15; PHG17; LBM15; Heu+20; GHO15; CDP17; Pic+23]

Tuning a ML model properly by choosing appropriate hyperparameters is paramount for its success, with well-tuned models outperforming template at-

tacks [Pic+17; LBM15]. Deep learning is especially promising, as it is capable of approximating any continuous function in idealized settings where the universal approximation assumption holds [Cyb89]. The deep learning networks multilayer perceptrons (MLPs) and CNNs have proven to be very powerful for performing SCAs [Zai+19; SAS21]. However, these models have to be provided with an architecture, e.g. different types of neural layers arranged after each other, each with their own parameters [Zai+19; Wou+20]. Designing an appropriate architecture can be more of an art than a science, prompting a wave of experimentation with different architectures, both created from scratch as well as existing ones taken from image classification tasks [Ben+20]. In order to alleviate this issue, Perin, Chmielewski, and Picek [PCP20] proposed using ensembles of multiple networks and aggregating their predictions. While this approach improves the generalization properties of existing CNNs architectures for hardware side channels, it also increases the computational cost and the number of trainable parameters of the model without addressing the underlying issue of architecture design.

This challenge to design optimal neural architectures is not unique to the hardware side-channel community. It has led to active exploration in the new area of NAS, which treats the design of neural architectures as another optimization problem for which approximate solutions can be determined automatically [Ren+21]. This idea has been picked up recently by Wu, Perin, and Picek [WPP20] and Rijsdijk, Wu, Perin, and Picek [Rij+21] and applied to hardware side channels for the first time. These works apply a white-box scenarios and use the attack traces for optimizing the architecture with Random and Bayesian search strategies [WPP20] or using reinforcement learning [Rij+21]. These initial investigations show very promising results, being able to produce very capable CNN and MLP models on the ASCAD benchmark dataset created by Benadjila, Prouff, Strullu, Cagli, and Dumas [Ben+20] and the CHES_CTF dataset.

Another technique for hyperparameter optimization and neural architecture design that was developed by Acharya, Ganji, and Forte concurrently to ours is InfoNEAT [AGF22]. They created an algorithm that evolves several neural network architectures and their hyperparameters simultaneously. Instead of the usual approach where a single neural network needs to predict the whole key byte, it trains a separate neural network for each possible key byte value, using a one-versus-all multi-class classification approach. In combination with an architecture selection based on information-theoretic metrics this makes it uniquely suited for hardware attacks. Unfortunately, this also means that their results cannot be directly compared with more traditional approaches striving for a single architecture.

5.3 Background

In a hardware SCA, an attacker wants to determine the secret key used in a cryptographic operation, e.g. an encryption operation, running on a target device

they can observe. For non-profiled attacks, the attacker is limited to observing the device without access to the private key being used, relying entirely on their observations of EM radiation or power consumption, for example [Pic+23]. In many cases, it is reasonable to assume that an attacker can also gain access to a second device matching the target device, called a "profiling" device, e.g. by obtaining an identical model [Pic+23]. This enables a profiled SCA, where the attacker can build a behavioral profile of the target device by running a large number of cryptographic operations with known secret keys on the profiling device. They thus obtain a set of N observation traces $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ in the first profiling phase. Each profiling trace is a time series of d instances of measured power consumption or electromagnetic radiation, represented by a d-dimensional real-valued vector $\boldsymbol{x}_i \in \mathbb{R}^d, \forall i \in \{1, \ldots, N\}$. In the attack phase, this profile is used to recover the secret key from the observed behavior of the target device.

5.3.1 Supervised Learning for Profiled Side-Channel Attacks

The application of ML to hardware attacks differs slightly from the method described in Section 2.3.

Application to AES-128 In the Advanced Encryption Standard (AES), the non-linear `SubBytes` method is being applied byte-wise to the inputs containing round keys derived from the full secret key. Because `SubBytes` uses an input-dependent S-box array lookup, this method is usually the target for SCAs. Another advantage of targeting this method is the independent operation on each input byte, allowing independently attacking specific round key bytes. Without loss of generality, we only consider attacking a single, specific key byte in a specific round of AES-128, as the same attack can be applied to multiple key bytes across multiple AES rounds to retrieve the full key [WPP20].

Profiling Dataset Structure The attacker records the traces $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ from the profiling device. Each of these N profiling traces corresponds to a single known secret key byte $k_i \in \mathcal{K}$ (with $\mathcal{K} = \{0, ..., 255\}$) and a known plaintext byte p_i. In case the attacker used different keys for each profiling trace, the key bytes $k_1, \ldots, k_N$ are also different in each trace, while $k_1 = k_2 = \ldots = k_N = k$ if the attacker used the same key for each profiling trace. The profiling trace is then labeled with $y_i = \phi(p_i, k_i)$ using a function ϕ. The function ϕ maps the plaintext p_i and the key k_i to a value that is assumed to relate to the deterministic part of the measured leakage $\boldsymbol{x}_i$ [Pic+18]. This mapping depends on the assumed leakage model and is usually defined using the AES S-box function $sbox()$ itself: $\phi(p_i, k_i) = sbox(p_i \oplus k_i)$. This identity leakage (ID) leakage model results in 256 possible labels corresponding to the possible values of the input byte. This labeling results in the profiling dataset $\mathcal{D}_{\text{profiling}} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, which is then used by the profiling supervised learning algorithm to build a profiling model.

Supervised Learning The task of the profile is to predict the secret key value k_i that was used in the cryptographic operation observed in attack trace $\boldsymbol{x}_i$, for which the true secret key value is unknown. This can be formalized as a supervised learning task, where the learner is provided with a set of training data $\mathcal{D}_{\text{profiling}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N} \subset \mathcal{X} \times \mathcal{Y}$ of size $N \in \mathbb{N}$, with $\mathcal{X} = \mathbb{R}^d$ the input space (in our case the measured traces) and $\mathcal{Y} = \{0, \ldots, C-1\}$ the output space (the 256 possible labels or "classes" produced by $\phi(p_i, k_i)$ as defined above). The task of the learning algorithm is to find a target function $f : \mathcal{X} \to \mathcal{Y}$ which, given any query $\boldsymbol{x} \in \mathcal{X}$ as input, predicts the corresponding output y in an accurate manner. Instead of simply predicting a single label, the most commonly used approach is to give a probability score for each candidate label. This allows the attacker to use the model on more than a single attack trace, aggregating the probabilities over multiple observations.

The function f can often be parameterized by parameters $\boldsymbol{w} \in \mathbb{R}^n$, where n is the number of trainable parameters. Typically, the target function is represented using a probabilistic scoring function $S : \mathcal{X} \to [0,1]^C$, which is also parameterized by $\boldsymbol{w}$. For a given instance $(\boldsymbol{x}_i, y_i)$, this function assigns a probability score for each label, such that $\boldsymbol{s}_i \coloneqq S_{\boldsymbol{w}}(\boldsymbol{x}_i) = (s_{i,0}, \ldots, s_{i,C-1})$, where $s_{i,j} \coloneqq S_{\boldsymbol{w}}(\boldsymbol{x}_i)[j]$ corresponds to the probability score for label $j \in \mathcal{Y}$ for the given instance $\boldsymbol{x}_i$. Typically, neural networks are used to estimate the parameters $\boldsymbol{w}$ of the target function f. These networks implement a scoring function $U : \mathcal{X} \to \mathbb{R}^C$, which assigns a real-valued score for each label, such that $U_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) = \boldsymbol{u} = (u_0, \ldots, u_{C-1})$, where $u_j \coloneqq \boldsymbol{u}[j]$. These scores are then transformed into (pseudo-)probabilities by means of the *softmax function*:

$$S_{\hat{\boldsymbol{w}}}(\boldsymbol{x})[j] = \frac{\exp(\boldsymbol{u}[j])}{\sum_{k=0}^{C-1} \exp(\boldsymbol{u}[k])} \,. \tag{5.1}$$

The aim of supervised learning is to learn a $\boldsymbol{w}^*$ with minimal expected loss:

$$\boldsymbol{w}^* \in \arg\min_{\boldsymbol{w}} \int L(S_{\boldsymbol{w}}(\boldsymbol{x}), y)\, dP(\boldsymbol{x}, y) \;, \tag{5.2}$$

where L is a loss function $[0,1]^C \times \mathcal{Y} \to \mathbb{R}$ and P the (unknown) data-generating process. One way to approximate $\boldsymbol{w}^*$ is to minimize the *empirical risk* on the profiling dataset $\mathcal{D}_{\text{profiling}}$:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^n} R_{emp}(\boldsymbol{w}) \tag{5.3}$$

with

$$R_{emp}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} L(S_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) = \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{s}_i, y_i) \,. \tag{5.4}$$

categorical cross-entropy (CCE) is often used as the loss function in SCA [Pic+23]:

$$L(\boldsymbol{s}_i, y_i) = L_{CCE}(\boldsymbol{s}_i, y_i) = -\sum_{j=0}^{C-1} [\![y_i = j]\!] \log(\boldsymbol{s}_{i,j}) \,, \tag{5.5}$$

where $[\![z]\!]$ is the indicator function returning 1 if condition z is true and 0 otherwise. Finally, the target function f is defined as $f_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) = \arg\max_{j \in \mathcal{Y}} S_{\hat{\boldsymbol{w}}}(\boldsymbol{x})[j]$.

Attack Methodology The attacker records the attack traces $\boldsymbol{x}_1, \dots \boldsymbol{x}_{N_a}$ from the device under attack, by sending N_a plaintexts (or ciphertexts) $p_1, \dots p_{N_a}$. Each of these N_a attack traces $\boldsymbol{x}_i$ corresponds to the unknown key byte $k^* \in \mathcal{K}$ (with $\mathcal{K} = \{0, \dots, 255\}$) of the device and a known plaintext p_i. In order to perform the attack, the attacker needs to consider every possible key byte candidate $k \in \mathcal{K}$. For each instance $(\boldsymbol{x}_i, p_i)$, a label is generated for every key byte candidate $k \in \mathcal{K}$ using the same $\phi(p_i, k)$ function used during the profiling phase. The resulting labels are denoted by the vector $\boldsymbol{y}_i = (y_{i,0}, \dots y_{i,K-1})$, such that $y_{i,k} = \phi(p_i, k), \forall k \in \mathcal{K}$. The labeling results in the attack dataset $\mathcal{D}_{attack} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), \dots, (\boldsymbol{x}_N, \boldsymbol{y}_{N_a})\}$, which is then used by the learned profiling model to acquire the secret key byte k^* of the device. To perform the attack, the learned probabilistic scoring function $S_{\hat{\boldsymbol{w}}}$ is used to acquire the scores for every possible key byte candidate $k \in \mathcal{K}$. For a given attack instance $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, the scores of every key byte candidate are denoted by the vector $\hat{\boldsymbol{s}}_i \coloneqq (S_{\hat{\boldsymbol{w}}}(\boldsymbol{x}_i)[y_{i,0}], \dots S_{\hat{\boldsymbol{w}}}(\boldsymbol{x}_i)[y_{i,K-1}]) = (\hat{s}_{i,0}, \dots, \hat{s}_{i,K-1})$, such that $\hat{s}_{i,k} \coloneqq S_{\hat{\boldsymbol{w}}}(\boldsymbol{x}_i)[y_{i,k}]$ represents the score of the key byte candidate $k \in \mathcal{K}$ [Ben+20]. Using these predicted scores, the cumulative score for each key byte candidate $k \in \mathcal{K}$ is calculated over several attack traces using the maximum log-likelihood [Ben+20; Pic+23]:

$$\boldsymbol{d}_{N_a}[k] = \log\left(\prod_{i=1}^{N_a} \hat{s}_{i,k}\right) = \sum_{i=1}^{N_a} \log(\hat{s}_{i,k}) \tag{5.6}$$

Using the likelihood to acquire the cumulative scores is an outlier-sensitive operation, as a single low score value can completely disqualify the true key [Lom+14]. To increase robustness and reduce sensitivity toward low scores, the attack is run multiple times on shuffled traces of the attack dataset to obtain the corresponding cumulative scores $\boldsymbol{d}_{N_a}[k]$.

Guessing Entropy The guessing entropy (GE) is the number of guesses that are required by a model to predict the correct key byte k^* [Mas94]. It is acquired using the ranking vector, which contains the position of each key: $\boldsymbol{r}_{N_a}[\tilde{k}] = 1 + \left(\sum_{k \in \mathcal{K} \setminus \tilde{k}} [\![\boldsymbol{d}_{N_a}[k] > \boldsymbol{d}_{N_a}[\tilde{k}]]\!]\right), \forall \tilde{k} \in \mathcal{K}$, and the *guessing entropy* of k^* is $\boldsymbol{r}_{N_a}[k^*]$, or r_{k^*}. Because of the repeated attacks, we acquire multiple GE values, which we average to determine the final estimated GE. The $Q_{t_{GE}}$ value is the minimum number of attack traces that are required for the very first guess of the model to be correct, i.e. $\boldsymbol{r}_{Q_{t_{GE}}}[k^*] = 1$, and it can be used to describe the efficiency of the attack model [Rij+21]. In case the available attack traces N_a are not sufficient, this value is not well-defined, but for the sake of being able to perform aggregation in the experiments, we choose to set it to N_a.

5.3.2 Convolutional Neural Networks

A Neural Network consists of a series of interconnected layers containing Neurons that connect an input layer that is activated according to observation with an output layer corresponding to the prediction of the model for this observation. The structure of these interconnections as well as the method of layer operation can vary significantly and defines the overall Neural Architecture. The MLPs is a very simple Neural Network, only employing fully connected, or "dense", layers. These were shown to perform SCA efficiently in case there are no countermeasures applied by the system, but often fail for more challenging tasks [Ben+20; MPP16a].

In recent work, the CNNs have proven to be very effective in learning a multiclass classification model and breaking a system via hardware side channels, even if such a system implements countermeasures [Zai+19]. CNNs have shown to be very robust towards the most common countermeasures, namely masking [MPP16a; MDP19] and desynchronization [CDP17]. A CNN contains convolutional and pooling layers in addition to dense layers as shown in Figure 5.1. A CNN can be viewed as an MLP where only each neuron of the layer l is connected to a set of neurons of the layer $l-1$, therefore can perform all the operations that can be performed by an MLP [Kle17; Zai+19]. In addition to that, the CNN architecture imposes inductive biases that are useful for many important applications and that the MLP networks would have to learn [Kle17; Zai+19]. A recent study has shown that overall, CNNs are more efficient and better suited to perform SCA on hardware datasets than MLPs [Cha+22], which is why we chose to focus on different CNN architectures.

The convolutional block consists of the convolutional layer and a pooling layer and the dense block consists of the dense (fully-connected layer). The batch normalization operation is typically applied after the convolutional layer and dense layer. Each layer has some trainable parameters $\hat{\boldsymbol{w}}$ which are used to get the final target function f (c.f. Section 5.3.1) and some hyperparameters. The hyperparameters are configuration variables of the layer external to the learning model (f) and hugely influence finding an optimal target function f. Now, we briefly describe the operations performed by these layers.

Convolutional Operation This operation basically re-estimates the value of the input value by taking a weighted average of the neighboring values. The weights are defined using a kernel of some size w_k (w_k for 1-D data or $w_k \times w_k$ for 2-D data) and these weights are learned using back propagation algorithm [Zai+19]. This kernel is shifted over the input data (1-D vector or 2-D maps) with a stride until the entire data is covered. The convolutional operation is performed for every shift and produces a weighted average value. Typically this operation is applied multiple times using different kernels and this number is called the filter size f_i of the convolutional layer. If this operation is applied without padding, then the dimensionality of output decreases, and this operation is called the valid padding operation. In order to preserve the dimension, the data is padded with 0,

and this operation is called same padding [Zai+19].

The number of trainable parameters for convolutional layers are $[in \times f_i \times w_k \times out] + out$ for 1-D data and $[in \times f_i \times w_k \times w_k \times out] + out$ for 2-D data, where in denotes the number of inputs and out denotes the number of outputs [Rij+21]. The two hyperparameters which need to be searched for an optimal CNN architecture are the kernel size and number of filters for each convolutional layer as listed in Table 2.

Pooling Operation This operation applies down-sampling on the input acquired from the previous layer and produces a condensed representation. This operation reduces the number of trainable parameters of the CNN and avoids over-fitting [Zai+19]. The pooling operation of some size w_p (w_p for 1-D data or $w_p \times w_p$ for 2-D data) and stride, is shifted across the input and reduces it by applying a max operation or an average operation. Similar to convolutional operation, pooling operation could also be applied with (preserves dimensionality) or without padding (dimensionality decreases), and the operations are called "same" or "valid padding" respectively. This layer does not have any trainable parameters and the hyperparameters which need to be searched for an optimal CNN architecture are the poolsize w_p, number of strides, and pooling operation type as listed in Table 2.

Dense Layers This layer consists of weights $\boldsymbol{W} \in \mathbb{R}^{d \times n_h}$ and biases $\boldsymbol{b} \in \mathbb{R}^{n_h}$, where d is the dimensionality of the input $\boldsymbol{x} \in \mathbb{R}^d$ and n_h is the number of hidden units of the layer [Ben+20]. The output of this layer is evaluated using the formula $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$. Typically, an activation function (e.g. ReLU, Elu) is also applied to each element of the output and the weights and biases are learned using the back-propagation algorithm [Ben+20]. The last dense layer is applied using softmax function (c.f. Section 5.3.1), which converts real-valued scores to *softmax-scores*. The number of trainable parameters for dense layers is the sum of n_h for each input plus the bias b: $n = (in \times n_h + n_h \times out) + (n_h + out)$ where in denotes the number of inputs and out denotes the number of outputs [Rij+21]. The hyperparameter which needs to be searched is the number of hidden units for each dense layer as listed in Table 2.

Batch-Normalization Layer This layer was introduced to lower internal covariance shift in neural network and thus making the convergence faster [IS15]. It was possible to use larger learning rates for the training process. This layer normalizes every data point x_i in a training batch by estimating the expected mean and the variance of the training batch. The number of trainable parameters for batch normalization is $4 \times d$, where d is the dimensionality of the input. For NAS, we can choose to either apply it or not in each convolutional block and in each dense block.

5.3.3 Leakage Model

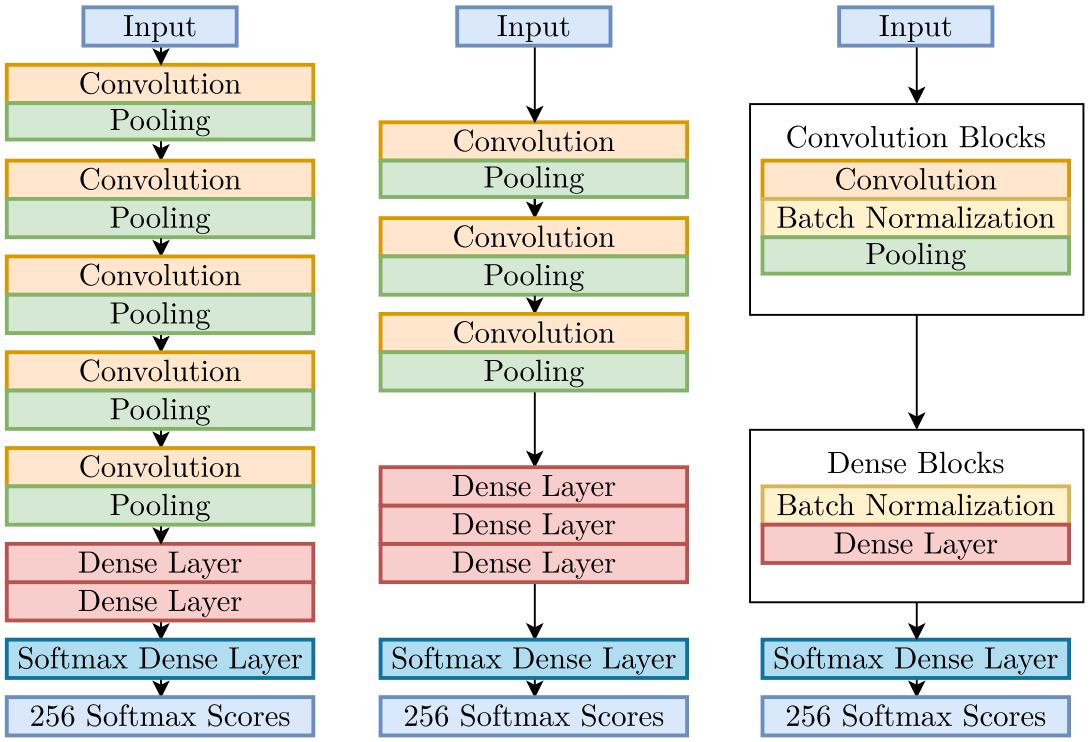
The leakage model defines which information is expected to be leaking from the device in the measurements. Since we focus on AES-128 in our experiments, we assume the output to the S-box function is leaked. Additionally, we only target a single S-box corresponding to a single key byte in the very first execution step of `SubBytes` in the first AES round. We believe our results apply to other key bytes and later rounds, as determined by [GJS19]. We investigate two types of leakage models for this output byte, the Hamming weight (HW) leakage model and the identity (ID) leakage model.

Identity Leakage Model In this model, the attacker assumes that the leakage l or power consumption is directly linked to the entire S-box output. For the 8-bit S-box used in AES, this leakage model results in 256 classes representing every possible value of the input byte. The dataset is then labeled with $\phi(p_i, k_i) = sbox(p_i \oplus k_i)$.

Hamming Weight Leakage Model In this model, the attacker assumes that the leakage l or power consumption is directly linked to the number of bits set to 1 in the S-box output, which is equivalent to its Hamming weight (HW). For the 8-bit S-box used in AES, this leakage categorizes 256 possible inputs into 9 classes, from 0 bits set to 8 bits set. This is done with the labeling function $\phi(p_i, k_i) = HW\left(sbox\left(p_i \oplus k_i\right)\right)$. This causes several outputs to map to the same class, since e.g., the output values 1 and 4 both belong to HW class 1, and the full output value cannot be recovered. Using the redundant information over several S-boxes and `SubBytes` rounds, as well as the relationship between them, this still allows full key recovery, as for example demonstrated in the CHES 2018 CTF challenge by the AGSJWS team [GJS19]. Choosing this leakage model produces a large class imbalance because while only a single output maps to class 0 and 8 each, 70 outputs map to class 4. This can have a large effect on a machine learning process and may require custom metrics to account for the imbalance [Pic+18].

5.3.4 Neural Architecture Search

The first handcrafted CNN architecture is shown in Figure 5.1a, which Benadjila, Prouff, Strullu, Cagli, and Dumas [Ben+20] proposed to attack the ASCAD dataset. This architecture was later optimized manually to produce dataset-specific smaller architectures (for example Figure 5.1a shows the architecture produced by Zaid, Bossuet, Habrard, and Venelli [Zai+19] for attacking ASCAD_f desync50 and ASCAD_f desync100). This shows that an optimal CNN architecture is dependent on the dataset and designing it manually requires expert knowledge. This challenge is not unique to side-channel attacks, and consequently, there have been recent developments in automating this process by employing "Neural Architecture Search". NAS treats the task of finding a suitable architecture for a given dataset as a simple optimization problem (using objective) that can be solved



(a) ASCAD baseline from [Ben+20]

(b) Zaid baseline from [Zai+19]

(c) Our NAS base architecture

Figure 5.1: CNN architecture comparison

automatically. NAS takes a search space $\mathcal{A}$ containing possible architectures and the dataset as input and, using a specified search strategy, automatically searches for the optimal architecture as shown in Figure 5.2. Typically, NAS uses an evaluation metric, e.g. accuracy or a loss function as its *objective*, which is used as a criterion to evaluate or measure the performance of an architecture. The dataset is split into training data $\mathcal{D}_{train}$, which is is used for training a new architecture $A \in \mathcal{A}$ and validation data $\mathcal{D}_{val}$ which is used to evaluate the performance A. In the end, the NAS produces the best-performing architecture according to the defined *objective* [EMH19]. This motivated the usage of NAS approach, which takes the profiling dataset as input and automatically produces an optimal CNN architecture to perform the SCA for a given dataset [EMH19].

Previous Proposals Recent works propose using NAS for SCA [WPP20; Rij+21]. They first proposed different white-box metrics for defining the objective for performing the NAS using RANDOM and BAYESIAN search strategy [WPP20]. These white-box metrics determine the cumulative score of the secret key byte on the labeled attack dataset in order to evaluate the performance of an architecture. This work was extended by proposing a novel reinforcement learning based NAS

approach which uses the white-box objective as the reward function for learning the Q-function [Rij+21]. The Q-function is used to guide the search and choose the hyperparameters of the next architecture to be evaluated. This approach uses guessing entropy and $Q_{t_{GE}}$ value of the secret key byte k^* of the system, to evaluate its white-box objective or the reward function. The drawback of these approaches is that it uses the attack dataset to evaluate the objective function to find the architecture, which poses two major issues. First, it no longer allows the detection of overfitting, e.g. where a model is specifically matching only the data it has been exposed to before, performing exceptionally well on training data, but the model does not generalize, meaning that its performance with any other data is poor. Since the attack dataset is also used for hyperparameter-optimization, the architecture is specifically fitted to the attack dataset. To detect overfitting, a holdout dataset, which is deliberately withheld during the entire model selection, parameter tuning and training process, has to be used, since only the performance on this holdout can predict the generalization capabilities of the model. If this holdout dataset is used in any part of the process, even if it was just manually inspected to select which specific classifier to train on it, data-snooping occurs and it is no longer useful for assessing generalization [Jen00]. This is the case for [Rij+21] and [WPP20], and therefore we cannot be sure if overfitting occurred in these experiments. Second, testing a model on the same attack dataset for which its architecture has been optimized will necessarily result in an over-estimation of its real-world attack performance where the architecture cannot be optimized for the unknown key in the attack dataset. This is acceptable in a white-box or gray-box setting where some parts of the attacked device are assumed to be known, but is not compatible with the black-box setting we assume, where the attack dataset would be considered unlabeled for the purposes of training. The performance results reported in [Rij+21] and [WPP20] are thus not necessarily representative of real-world performance on an unlabeled attack dataset and cannot be meaningfully compared to our work.

5.4 Our Approach

We aim to produce an unbiased, optimal CNN architecture in a black-box setting with the help of NAS. Since NAS was designed primarily to use only the training dataset for finding an optimal architecture, we devised an approach, illustrated in Figure 5.2, which can satisfy these black-box requirements. This follows the standard training-test-validation split used in many ML methods. For this the profiling dataset is split into validation and training dataset, such that the training dataset $\mathcal{D}_{train}$ is used to train the architecture under consideration and validation data $\mathcal{D}_{val}$ is used to estimate the performance of said architecture. In our approach, the search strategy uses a predefined search space $\mathcal{A}$ (c.f. Table 2) containing CNN architectures (1-D or 2-D depending on the input shape) to perform NAS. Typically, the search strategy initially suggests some architectures

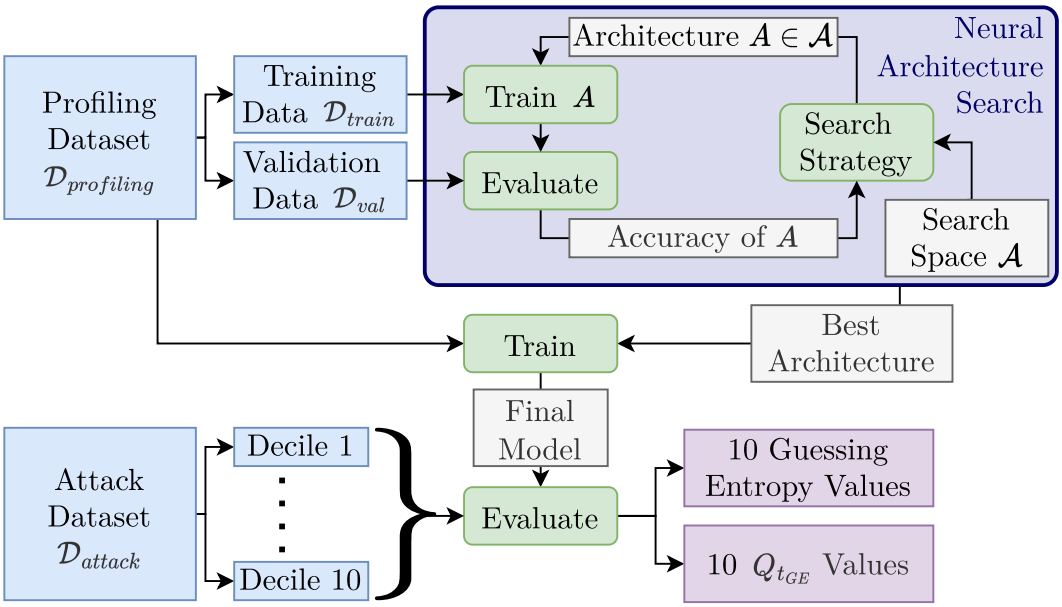
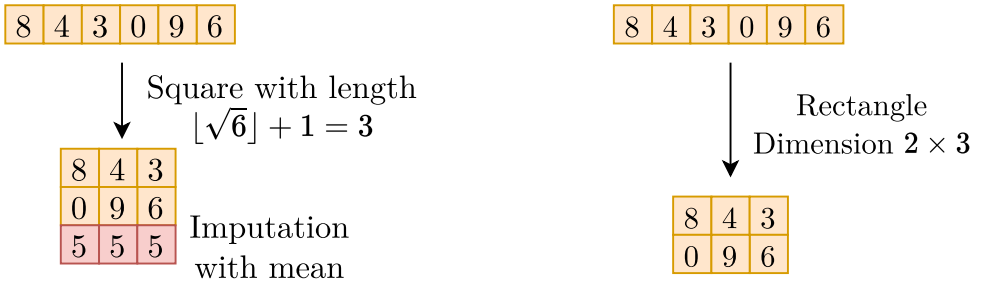


Figure 5.2: Schematic of our NAS approach for black-box attacks

$A \in \mathcal{A}$ randomly (exploration) and uses their evaluated accuracy to make future suggestions (exploitation). The training data ($\mathcal{D}_{train}$) is used for training the architectures A and the validation data ($\mathcal{D}_{val}$) is used for evaluating the accuracy of A. The attack dataset ($\mathcal{D}_{attack}$) acts as the holdout data for independent evaluation of the final model. Since we are only using the profiling and not the attack dataset for searching and evaluating the architecture $A \in \mathcal{A}$ under consideration, this makes our NAS optimization a *black-box* approach [Rij+21]. This is in contrast to Wu, Perin, and Picek [WPP20] and Rijsdijk, Wu, Perin, and Picek [Rij+21], which use the attack dataset instead of a validation dataset for guiding the search, making their approach *white-box* since the attack dataset can then not be used to give an unbiased performance evaluation. In the end, NAS suggests the architecture which has the highest accuracy. The best-performing architecture is then trained on the complete profiling dataset ($\mathcal{D}_{\text{profiling}}$), which improves the performance while only incurring a marginal computational overhead compared to the actual search. Similar to the folds used in cross-validation, the attack dataset ($\mathcal{D}_{attack}$) is split into 10 equal parts (deciles) and each part is used to independently evaluate the attack efficiency of this model using the *guessing entropy* and $Q_{t_{GE}}$ measures defined in Section 5.3.1.

5.4.1 Two-Dimensional Input Reshaping

The measurement traces of the datasets have to be transformed into the proper shape for the neural network to process. The most straightforward transformation is to produce a One-Dimensional input from the time series input, define a



(a) SQUARE Input Conversion (b) RECTANGLE Input Conversion

Figure 5.3: Conversion technique of 1-D input to 2-D inputs

search space containing 1-D CNN architectures (c.f. Table 2), and apply NAS to find an optimal architecture. While most of the 1-D CNN architectures that are explored for performing SCA to break AES-128 encryption were originally inspired by popular 2-D CNN architectures proposed for image classification, like VGGNet, Inceptionv3 [Ben+20], they are only applied on one-dimensional inputs. Recent work has shown that using 2-D CNNs could increase the accuracy and efficiency for breaking post-quantum key-exchange (PQKE) protocols [Kas+21; Het+20]. This motivates us to also explore using two-dimensional inputs for AES-128 attacks, applying NAS on a search space containing 2-D CNN architectures (c.f. Table 2).

In order to convert the one-dimensional input of length d to a two-dimensional input, we propose to use two techniques SQUARE and RECTANGLE. In the SQUARE approach shown in Figure 5.3a, we create a perfect square of dimension $\lfloor\sqrt{d}\rfloor + 1$, but since most feature spaces will not result in a full square we need to fill the remaining places with the mean value of the instance. For forming a RECTANGLE input, we create the rectangle shown in Figure 5.3b, which is as close as possible to a square without the need to fill it with mean values. These dimensions can be determined by using the two factors of d which are closest to $\sqrt{d}$.

5.4.2 Search Strategies

The search strategy used in NAS has a huge influence both on search performance and runtime. The previous work on applying NAS for performing SCA only considered BAYESIAN and RANDOM search strategies [WPP20], which are not necessarily time-efficient [EMH19; Li+17]. We explore the four search strategies RANDOM, GREEDY, HYPERBAND, and BAYESIAN with 1000 fixed trials and use their respective implementations provided by AutoKeras [JSH19].

Typically, search strategies first explore the search space by trying many substantially different architectures. This is followed by exploitation, in which well-performing architectures are further improved via small changes. Because each search algorithm is only allowed to consider a limited number of architectures (in our case 1000), it should have a balanced trade-off between exploration and

exploitation in order to perform efficiently and accurately.

Random The RANDOM search technique chooses a unique architecture configuration uniformly at random (with replacement) from the complete search space and evaluates its performance. This process is repeated for a specified number of trials but is preempted if it has exhausted the search space or the same configuration is chosen multiple times. This means it focuses only on exploration without any exploitation.

Greedy The GREEDY search algorithm proposed by Jin [Jin21] works in two distinct stages, exploration and exploitation. In the exploration stage, it evaluates uniformly randomly chosen models for a limited number of trials [JSH19]. In the exploitation stage, it generates models which are neighboring the best-performing model from the first stage and exclusively try to improve this model. This is done by traversing a hierarchical hyperparameter tree representing the hyperparameters of the best-performing model, as well as possible changes such that the new hyperparameter values remain close to that of the best-performing model with high probability [Jin21]. This tree is rebuilt if a better architecture is found, at which point this architecture becomes the starting point for subsequent exploitation. The algorithm continues the exploitation process until either the trial limit is reached or until it has exhausted the entire search space. The GREEDY search algorithm is time-efficient, but it might get stuck in local optima since it performs limited exploration, e.g. for only 1 % of the maximum number of trials in `AutoKeras` [Jin21; JSH19].

Hyperband The HYPERBAND search algorithm is based on the successive halving algorithm [Li+17]. The Successive Halving algorithm divides the resources (time, epochs) equally into a specific number of hyperparameter configurations. In each time step (2-3 epochs) it checks their performance and at the end keeps the top-half best-performing configurations. This process is repeated until only the best-performing configurations are left. The HYPERBAND algorithm is time-efficient and balances the trade-off between exploration (check many models with a low budget) and exploitation (provide a high budget to the best-performing architectures) very well [Li+17].

Bayesian The BAYESIAN search algorithm is based on Bayesian optimization, which assumes a (black-box) function $f' : \boldsymbol{\Theta} \to \mathbb{R}$ over the hyperparameter space $\boldsymbol{\theta} \in \boldsymbol{\Theta} = (\Theta_1, \ldots, \Theta_n)$, such that $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ represents one hyperparameter configuration [FSH15]. Each hyperparameter space can be an integer, real or categorical, i.e., $\Theta_i \in \mathbb{R}$ or $\Theta_i \in \mathbb{Z}$ or $\Theta_i \in \{0, \ldots, c\}$, for some $c \in \mathbb{N}$, where c is the number of categories. For finding the best configuration function f', which maps a configuration $\boldsymbol{\theta}$ to the estimated real-valued validation accuracy, a probabilistic model using a Gaussian process is used. This probabilistic model

provides the properly balanced trade-off between exploitation and exploration of the search space [Li+17]. At the end of the 1000 fixed trials the best configuration is obtained as $\theta^* = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} f'(\theta)$.

5.5 Setup of Our Parameter Study

The performance of the final model returned by performing NAS as described in Section 5.4 heavily depends on two factors or parameters: First, the search strategy employed for NAS, which has a huge influence both on search performance and runtime, and second, the shape of the input features. In order to determine the best-performing options for both of these factors, we need to perform a parameter study. The methodology for this study is outlined in Section 5.5.1, while Appendix B presents the datasets we used for the experiments and Section 5.5.3 covers the hardware necessary for it. Because we investigate how our automated NAS setup compares to manually crafted baseline architectures, we additionally trained the fixed CNN architectures we describe in Section 5.5.2.

5.5.1 Methodology

We proposed a black-box NAS approach to automatically perform a SCA as described in Section 5.4. The two main phases of our empirical study are first analyzing the expected success rate of our approach for a given parameter combination of the search strategy and input shape. Using this, we determine the best parameter combination of the search strategy and input shape, which should be used by NAS to perform SCA. In addition, we would also like to compare the attack performance of the CNN architectures produced by the NAS configured with the best parameter combination of the search strategy and input shape with the state-of-the-art handcrafted CNN architectures.

Parameter Study The attack performance of the final architecture produced by the NAS is heavily dependent on the search strategy used and the input shape of the data. For the parameter study, we consider 4 search strategies, namely Random, Greedy, Hyperband and Bayesian (c.f. Section 5.4.2). Our input data could be One-Dimensional (1-D CNN), rectangular (2-D CNN) or square (2-D CNN) shaped (c.f. Section 5.4.1). For each of the 12 possible parameter combinations of input shape and search strategy we apply NAS to acquire the best-performing model on 10 datasets (c.f. Appendix B). This model is then trained on a complete profiling dataset $\mathcal{D}_{\text{profiling}}$ and evaluated on 10 equal parts (deciles) of the attack dataset $\mathcal{D}_{attack}$. In the end, we acquire 10 guessing entropy values and 10 $Q_{t_{GE}}$ values, which are aggregated to achieve the final performance of the experiment run.

Loss Function Investigation We initially intended to also investigate the influence of the loss function in our parameter study, considering the most commonly used loss function CCE and comparing it to 6 other specialized loss functions, among them ranking loss (RKL) and focal-loss categorical cross-entropy (FLCCE) [Ker+22]. The parameter study was set up to combine all possible combinations of 3 input shapes, 4 search strategies, and 7 loss functions, and then run on the supercomputer (c.f. Section 5.5.3). When analyzing the results, we discovered that a small bug in the ML library we used, `AutoKeras`, prevented the loss function parameter to be passed onto the actual CNN training module, forcing all of the experiments to use the default value CCE instead. We therefore effectively evaluated our NAS approach for the same choice of input shape and search strategy 7 times, using CCE as the loss function each time. Each of the 7 runs produced a different split of profiling dataset $\mathcal{D}_{\text{profiling}}$ into training $\mathcal{D}_{train}$ and validation $\mathcal{D}_{val}$ dataset, resulting in a different best architecture returned by NAS and a different final model. These experiments had already been executed and consumed significant resources, so we had to consider if the results should be outright discarded or not. Because of the randomization involved, it is necessary to run such an experiment several times and average the results to get a statistically significant and "realistic" estimation of performance [LT20].[22] Consequently, we decided to keep all of the original experiments and aggregate the results of the 7 models as independent repetitions of the same NAS experiment. Unfortunately, we are not be able to determine the impact of the loss function.

Evaluation of Single Experiment Run A single experiment run for our study consists of applying NAS configured with a unique combination of input shape, search strategy, dataset, and leakage model. For each experiment, there are a total of 70 attacks being executed: 7 different models are trained based on unique random seeds, then the attack is executed for each of the 10 attack deciles for each model. The results of the 70 attacks are then aggregated to determine the performance with the following metrics.

Success Rate A single attack is considered to be successful if the final guessing entropy is 1, e.g. the correct key byte is at the top of the predictions with $r_{k^*} = 1$ [TPR13]. The success rate is defined as the empirically determined probability of an attack being successful. For our 70 attacks, we can therefore count the number of successful attacks and divide by the total number of attacks [TPR13], returning a percentage value.

Attack Efficiency $Q_{t_{GE}}$ Even though several approaches might achieve a high success rate, it is still preferable for them to use fewer attack traces, making them more efficient. This can be measured by the $Q_{t_{GE}}$ value, which counts the number of attack traces that are required to achieve a successful attack. To calculate the

[22] As recommended in `https://github.com/keras-team/autokeras/issues/359`

$Q_{t_{GE}}$ for an experiment run, we can average the $Q_{t_{GE}}$ values of the 70 individual attacks. This value can be sensitive to outliers, e.g. unsuccessful attacks where $Q_{t_{GE}}$ is set to N_a.

Evolution of Guessing Entropy Another analysis method of the model performance involves plotting the development of the GE value "over time", after observing a certain number of attack traces. As detailed in Section 5.3.1, this GE is already the result of running the attack 100 times on shuffled attack traces. For our analysis, we aggregate the 10 attack runs by averaging and reporting the 7 NAS models individually.

Implementation Details To implement NAS for different search strategies and search spaces as well as input shapes, we extend the `AutoModel`, `DenseBlock`, `ConvBlock` and `ClassificationHead` classes of the popular NAS python library `AutoKeras` [JSH19]. The code for the experiments and the generation of plots with detailed documentation is publicly available on GitHub[23].

Architecture Search Space To perform NAS, we need to define the search space containing a reasonable range of CNN architectures and hyperparameters. Fundamentally, our architecture consists of layer blocks as shown in Figure 5.1c, with up to 5 convolution blocks followed by up to three dense blocks and a softmax dense layer. In addition to the architecture, our search space also contains the respective hyperparameters for each layers, such as kernel sizes for convolutional layers, the activation function, or the number of hidden units in a dense block. We have to define slightly different search spaces for 2-D and 1-D CNN architectures to avoid generating invalid configurations (see Appendix C for the full details). We selected the range for each hyperparameter by analyzing the related work [Rij+21; WPP20], ensuring that our search space includes all possible 1-D baseline CNN architectures [Zai+19; Ben+20]. As such the search strategies could in theory find these fixed architectures and match their performance, e.g. RANDOM is certain to find these architectures eventually. We also included the range of each hyperparameter from the search space designed for 1-D CNN architectures proposed by the current work done on NAS for performing SCA [Rij+21; WPP20]. If the characteristics of the dataset are known in advance, it is possible to reduce this search space to make the search more efficient. We chose not to tailor the search space to the dataset like this, as this way the architecture design remains fully automated, requiring no manual analysis of the dataset. Using our search space, the total number of possible hyperparameter configurations is 40 758 681 600 for 1-D CNNs and 2 264 371 200 for 2-D CNNs. Each search strategy is provided with the same budget limit of 1000 trials, which means that only around 5×10^{-5} % of 2-D search space and 2.5×10^{-6} % of 1-D search space can be explored in order to produce an optimal CNN architecture.

[23] `https://github.com/prithagupta/deep-learning-sca`

Datasets We want to investigate the behavior of NAS in a wide range of settings using well-known datasets to enable direct comparisons to other works in hardware side-channel attacks. We, therefore, chose to focus on five datasets that have already been investigated before: ASCAD v1, DPA contest v4.1, AES_RD, AES_HD, and CHES CTF 2018 AES-128. All of these datasets record implementations of AES-128, which means the same overall approach should work automatically for each of them, without any need for dataset-specific tuning. However, the systems incorporate different types of leakage countermeasures, e.g. masking or random delays. For ASCAD v1 we used both fixed keys (ASCAD_f) and variable key dasets (ASCAD_r) as well as their desynchronized counterparts (desync50 and desync100). The details of all these datasets are listed in Appendix B.

5.5.2 Baseline Architectures

We also need to train fixed baseline architectures for comparison with our NAS approach. We chose to use the ASCAD architecture as proposed in [Ben+20] and ZAID architectures as proposed in [Zai+19]. The ASCAD baseline is the CNN model inspired from the image recognition baseline VGG-16 CNN model [SZ15] and shown in Figure 5.1a. Zaid *et al.* proposed several specific architectures for ASCAD_f, ASCAD_f 50ms, ASCAD_f 100ms, AES_HD, AES_RD and DPAv4 datasets [Zai+19]. For ASCAD_r, ASCAD_r 50ms, ASCAD_r 100ms there is no specific proposal, so we use the corresponding baselines proposed for the fixed-key versions (ASCAD_f, ASCAD_f 50ms, ASCAD_f 100ms). Since CHES CTF is known to be a very hard dataset, we use the deepest CNN proposed by ZAID [Zai+19], which is the architecture for the ASCAD_f 100ms dataset.

We also need to point out that the ZAID baseline is not uncontroversial in the hardware side-channel community, as follow-up work by Wouters, Arribas, Gierlichs, and Preneel [Wou+20] was able to reduce the size of the proposed architecture by 52 % without a reduction in performance. Additionally, they were able to disprove some of the claims about the design methodology used to arrive at the specific architecture choice in [Zai+19]. This does not disqualify these architectures for our investigations, as the increased computational demand by the larger network is not part of our comparison, but clearly demonstrates that further optimization to these architectures is still possible.

5.5.3 Computing Hardware and Runtime

Our experiments necessitate training millions of CNN models in total, which requires thousands of hours of GPU time. We ran them in parallel on a supercomputer equipped with GPU nodes, which allowed us to finish the entire parameter study in a few weeks. These GPU nodes consist of two AMD Milan 7763 CPUs running at 2.45 GHz, 512 GB of main memory, and four Nvidia A100 GPUs equipped with NVLink and 40 GB HBM2 GPU memory. A single NAS

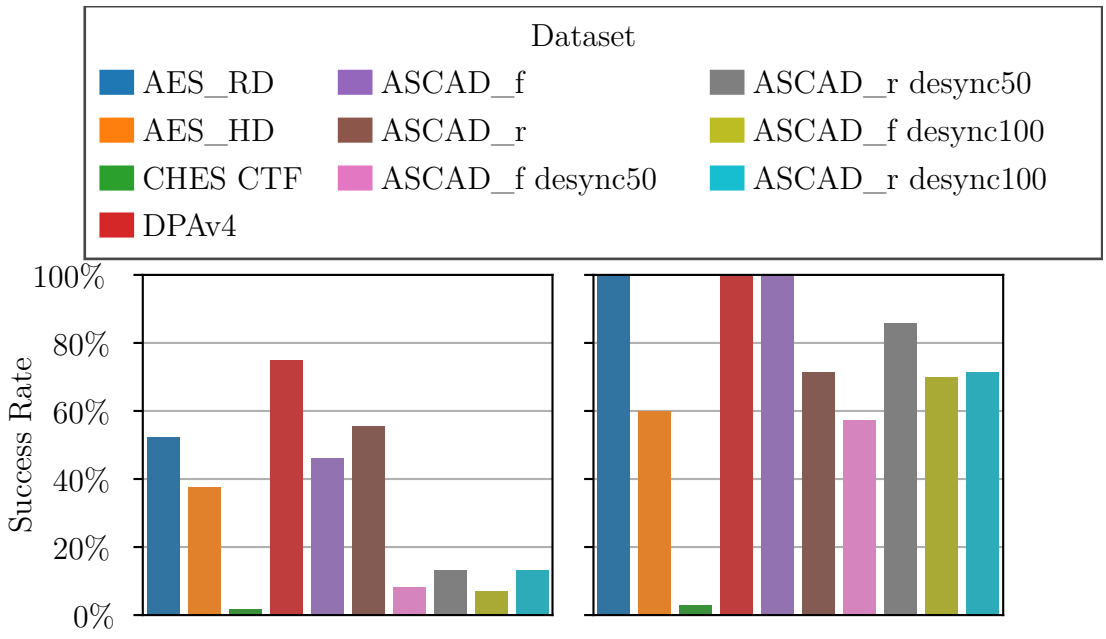
experiment, which consists of determining the best architecture through repeated intermediate model training followed by a final model training, takes less than 2 days on these shared GPU nodes. All search strategies were provided with a budget for 1000 trials, but only Random and Bayesian search strategies used up the entire budget every time. The Greedy strategy used only a portion of the provided budget, which resulted in a maximum running times of 5 hours, while the preemption technique of Hyperband results in a reduced maximum running time of 12 hours. We also ran some experiments on consumer hardware in the form of a $2500 gaming PC, where the longest experiment took 22 hours. We consider this runtime to be a reasonable assumption for the lower limit of computational resources that an attacker might be willing to commit to a single attack, with well-equipped attackers likely far exceeding these constraints.

5.6 Parameter Study Results

We ran the full parameter study outlined in Section 5.5, combining the possible options for search strategy and input shape. These were applied to the 10 datasets detailed in Appendix B, both for an ID leakage model as well as an HW leakage model. Additionally, we trained the baseline architectures described in Section 5.5.2 for each of the datasets.

5.6.1 Overall Reliability

First, we want to determine the overall reliability of our NAS approach, so we determined the success rate in all the experiments executed for each dataset in the ID leakage model. In our context, an attack is considered successful if the final guessing entropy is 1 after processing all the traces in the respective attack dataset decile. We show this per-dataset attack success rate in Figure 5.4a), giving a rough indication of how difficult attacking each dataset is. The first observation is the relative ease with which the DPAv4 dataset can be attacked: Even when taking all the suboptimal combinations of the search strategy and input shape into account, over 75 % of the attacks on it are successful. This is hardly surprising, as the dataset variant we consider effectively contains no countermeasures (see Appendix B for details). When comparing the non-desynchronized version of ASCAD_f and ASCAD_r to their desynchronized counterparts, the degradation in reliability caused by the increased difficulty incurred by desynchronization is clear, although some of the attacks are still able to succeed. We also need to point out the disastrous performance of NAS when applied to the CHES CTF dataset, where only a handful of attacks were able to recover the full identity value. This can be traced back to a reduced number of 500 attack traces available for the attack because of our decile split, with convergence plots (see Section 5.6.3 for the full details) indicating that all models would have succeeded given a larger attack dataset.



(a) Over all parameter choices (b) Using 1-D and RANDOM

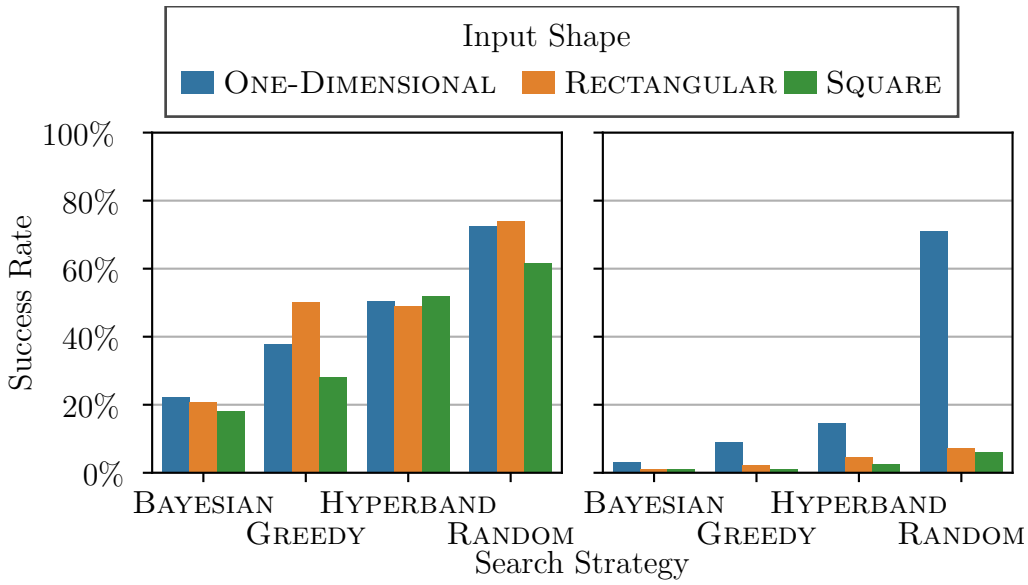
Figure 5.4: Success rate of all experiment runs for each dataset. An experiment is successful if the final model is able to predict the correct key byte after processing all provided attack traces)

Hamming Weight Model NAS struggled a little more in experiments using the HW leakage model, with generally slower convergence and similar success rate, which is why we don't show them here. Because the model only predicts 9 classes, it already takes more attack traces for the guessing entropy to reduce since a single prediction is more limited in information content than the 256 classes in the ID model. Upon manual investigation of the models produced by NAS, a large portion of them would simply predict a HW of 4 regardless of input. Due to the inherent class imbalance of HW datasets [Pic+18], this "blind" strategy produces a decent accuracy of over 27 % on random inputs. A NAS search algorithm using accuracy as the optimizing goal can get stuck on this local optimum, degrading the overall success rate of an attack.

5.6.2 Optimal Neural Architecture Search Parameters

In order to answer the question of which NAS parameter to choose, we plotted the success rate for every possible NAS parameter combination of input shape and search strategy. We chose to plot them separately for synchronized and desynchronized datasets because of the large difference in performance.

Synchronized Datasets Figure 5.5a shows the influence of the choice of each possible NAS parameter combination (input shape and the search strategy) on the



(a) Synchronized datasets (b) Desynchronized datasets

Figure 5.5: Influence of NAS parameters on the success rate.

overall performance of synchronized datasets for ID leakage. This clearly shows that the choice of search strategy has a significant impact on the success rate, which rises from around 20 % to around 70 % when going from Bayesian search via Greedy and Hyperband to Random search. Random search strategy clearly outperforms the other strategies, but it comes at the price of being slower, taking about 5 times longer than Greedy and Hyperband (c.f. Section 5.5.3), mostly because it keeps exploring the search space until the limit of 1000 trials is reached. For synchronized datasets, using the Hyperband strategy might be a viable alternative, as it still produces a CNN with a success rate of around 50 %, while being substantially faster than Random search. When comparing input shapes, there does not appear to be a consistent difference between 1-D and 2-D for synchronized datasets, demonstrating the ability of NAS to adjust to vastly different situations. This also shows that while using 2-D CNNs can be a viable option, they fail to provide any improvements compared to the 1-D input shape.

Desynchronized Datasets Figure 5.5b shows the influence of the parameter choices on the success rate for desynchronized datasets. The difference between Random and its competitors grows even larger on these datasets, to the point that choosing Hyperband for its speed is no longer a viable option. When desynchronization gets introduced, the input shape starts to play a major role. The 1-D CNN architecture is able to compensate for desynchronization to a large degree, with its performance essentially unchanged when paired with Random search. Wouters, Arribas, Gierlichs, and Preneel [Wou+20] observed a similar behavior where the first convolutional block successfully removes the desynchronization in

the dataset by using the convolutional and pooling operation on neighboring values in the trace. Converting one-dimensional to two-dimensional inputs changes the local relationship between neighboring values, which makes re-synchronizing the traces more difficult. Clearly, 2-D architectures struggle with re-synchronization in our experiments and are not a viable option for desynchronized datasets.

Overall Reliability of Optimal NAS Parameters As discussed above, it is clear that using RANDOM search strategy with ONE-DIMENSIONAL inputs gives the highest chances of producing a CNN model which can break the system successfully, especially when desynchronization is involved. We wanted to determine what success rate can be achieved for this specific combination, plotting it per-dataset in Figure 5.4b. The results show that for all other datasets apart from CHES CTF, over 57 % of the attacks are successful, sometimes even with a phenomenal success rate of 100 %. We can conclude that the combination of ONE-DIMENSIONAL input shape and RANDOM search strategy appears to be by far the best choice when it comes to reliably creating successful attack models.

5.6.3 Comparison with Fixed Architectures

As determined in Section 5.6.2, using the RANDOM search strategy and ONE-DIMENSIONAL input shape yields CNN architectures which can perform SCA with a high success rate. We are also interested in the efficiency of the architectures produced by ONE-DIMENSIONAL and RANDOM, as well as how they compare to traditional fixed architectures.

Comparison of $Q_{t_{GE}}$ We determined the efficiency of the models with their $Q_{t_{GE}}$ value, which counts the number of attack traces necessary for the model prediction to reach a GE of 1, with lower $Q_{t_{GE}}$ indicating a more efficient attack. We report the median $Q_{t_{GE}}$ for the 7 models NAS produced with RANDOM search strategy on ONE-DIMENSIONAL inputs as well as the model produced ASCAD and ZAID baseline architectures in Table 5.1. On the "easy" datasets AES_RD and DPAv4 our NAS is able to perform instantaneous attacks, requiring a single attack trace. We observe that our NAS models outperform the baselines for 4 out of 10 datasets, but especially the ZAID baselines specializing for desynchronization are much better suited for datasets with desynchronization, where NAS is only able to achieve better efficiency than the ASCAD baseline. However, these specialized models were not able to achieve successful attacks at all for the CHES CTF, synchronized ASCAD, and ASCAD_f desync50 datasets, indicated by the $Q_{t_{GE}}$ matching the total number of attack traces. This indicates some issues when relying solely on the $Q_{t_{GE}}$ for efficiency: The averaged $Q_{t_{GE}}$ is influenced by the success rate, as unsuccessful attacks will be considered with the total number of attack traces.

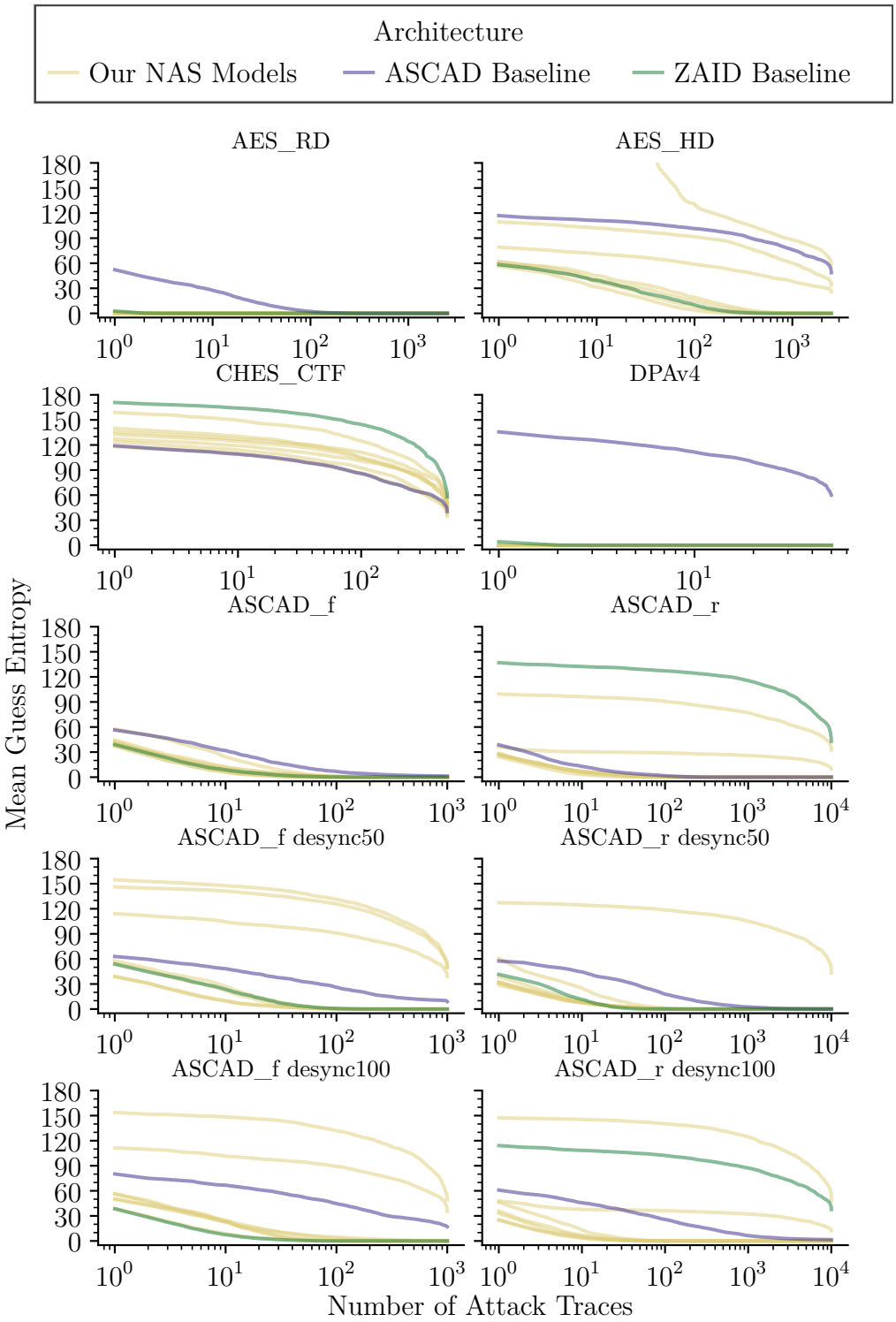


Figure 5.6: Guess entropy convergence of the 7 NAS models compared to the fixed architecture baseline models for each dataset.

Table 5.1: Comparison of the median $Q_{t_{GE}}$ metric for the different datasets on ID leakage. Entries where the attack failed (guessing entropy did not reach a rank of 1) are highlighted in italic and the best model for each dataset is highlighted in bold.

Dataset	ASCAD baseline	ZAID baseline	Our NAS
AES_RD	270.3	3.0	**1.0**
AES_HD	*2500.0*	**504.4**	1172.8
CHES CTF	**498.5**	*500.0*	*500.0*
DPAv4	49.3	2.9	**1.0**
ASCAD_f	703.5	**116.5**	118.3
ASCAD_r	322.0	*10000.0*	**136.3**
ASCAD_f desync50	920.9	**136.9**	202.0
ASCAD_r desync50	4449.0	**56.1**	139.7
ASCAD_f desync100	974.6	**82.5**	391.7
ASCAD_r desync100	6664.2	*10000.0*	**87.9**

Comparison of GE Convergence For the datasets with ID leakage, we therefore also plotted the GE to get a more detailed comparison. Again we compare the evolution of the models produced by NAS using Random search on One-Dimensional inputs against the two baselines and plot their GE convergence in Figure 5.6. Instead of averaging the 7 NAS models, we plot each model individually and only take the average GE over the 10 attack deciles of each model. This plot reveals that for most of the datasets, the NAS models match the baseline architectures, with the very best NAS model outperforming the baseline in the majority of the datasets (6 of 10). But just considering the best model is not representative of the general performance: On AES_HD, for example, it becomes apparent that while most NAS architectures perform similarly to the ZAID baseline, 3 out of the 7 models become outliers with significantly slower convergence. One of the models even started with a diverging prediction, only achieving a decent guess entropy towards the very end of the dataset. Similar behavior can be observed with ASCAD_r, where 5 out of 7 models outperform the ASCAD baseline and two outlier models suffer much slower convergence. For the desynchronized datasets, a similar issue occurs, and in total NAS outliers appear to be present in 6 of the 10 datasets we considered.

When taking a closer look at the ZAID baseline for ASCAD_r and ASCAD_r desync100, the baseline itself appears to be affected by a random outlier model, even though it is a fixed architecture. This failure to reproduce the performance of the ZAID baselines on the random key variant of the fixed key dataset it was specifically optimized for points to further issues with the fixed architecture approach. The issue with reproducibility pointed out by Wouters, Arribas, Gierlichs, and Preneel [Wou+20] appears in our experiments as well. While the $Q_{t_{GE}}$ reported in [Zai+19]

appear to agree reasonably well with our results on the ASCAD_f dataset, for ASCAD_f desync100 the reported mean $Q_{t_{GE}}$ of 270 in [Zai+19] and median $Q_{t_{GE}}$ of 82.5 in our experiments disagree. While our approach differs from their experiments e.g. by splitting the attack dataset into 10 parts, this does not suffice to explain the observed difference.

Generalization We observed the best NAS architecture going from outperforming the state-of-the-art on ASCAD_r and AES_HD to being vastly inferior simply because of a different train-validation dataset split. This indicates that the performance of NAS, like most ML processes, can sometimes vary heavily with small changes and randomization. Considering that the default approach for evaluating ML-based SCAs usually does not repeat the experiment with different train-validation-test splits or other cross-validation techniques, their generalizability is unclear. When looking at CHES_CTF, it also becomes obvious why splitting attack datasets into independent deciles as we did is not done more frequently: The attack dataset appears to be simply too small to be split into 10 parts, as all of the models, although appearing to be converging, simply did not reach a GE of 1 at the end.

When taking all of this into account, we can nonetheless conclude that NAS comes very close to state-of-the-art fixed architectures in terms of attack efficiency, but it is not able to do so consistently. This is promising, as it means that manual per-dataset architecture design can potentially be avoided altogether with NAS.

5.7 Conclusion and Open Problems

In this Chapter, we proposed to perform a black-box objective (using only profiling dataset) NAS to perform the SCA on hardware systems containing ID and HW leakages. In order to understand the impact of different NAS parameters like the search strategy and input shape of the data (1-D or 2-D) on the performance of the best model, we performed a detailed parameter study. We considered the 4 search strategies implemented by `AutoKeras`, Random, Greedy, Hyperband, and Bayesian. We also considered converting the original one-dimensional inputs to two-dimensional rectangular and square inputs, enabling us to use 2-D CNNs architectures. These choices were combined to create a large-scale parameter study, investigating the influence of search strategy and input shape on 10 different datasets and in the ID and HW leakage models. In order to get a better estimate of the success rate of each combination, we repeat the attack 10 times over different attack dataset parts, for 7 independent NAS models. Upon detailed analysis of the results of the study, we concluded that using the Random search strategy on one-dimensional inputs yields the best-performing CNN architectures for the medium-sized computational budgets we used. The attack performance is overall worse in the HW leakage model, presumably due to its inherent class imbalance. We also compared the efficiency of these NAS models with previously proposed

state-of-the-art CNN baselines. We showed that for most of the synchronized datasets, the 7 models produced by applying NAS were more efficient and required less attack traces than the baseline.

Considering that our approach was able to match the performance of hand-crafted architectures, NAS allows for fully automated attacks on devices or datasets with unknown characteristics. Our experiments highlight the importance of exploration, which needs to be considered when choosing search strategies for hardware attacks in the future. Given that a real-world attacker is likely to commit even bigger budgets to an attack, which enables RANDOM search to find even better architectures, our results should be considered only a lower limit to the capabilities of actual attackers. The apparent ability of NAS to generate useful architectures in various circumstances also allows for non-biased comparison of side-channel attack methods where current comparisons are skewed by the fixed architecture that is considered, e.g. loss functions. A big issue we observed was the susceptibility of the ML models to small variations in the training datasets, which current works applying ML to SCA are not accounting for. We were able to spot this issue because of the repeated training with different training-validation splits, but a broader discussion on how to achieve more consistent performance evaluations when applying ML to SCA needs to take place.

Research Question 6. *Considering the big spread in ML performance observed for different train-validate splits, how much of the reported advantage of recent hardware attack methods (e.g. specialized loss functions) is actually attributable to the method itself? Are these results reproducible at all or are they simply dependent on specific architecture choices that favor the proposed method and random variations in the training process? What would their performance look like when using our method?*

Possible future improvements could mitigate the imbalance in the HW leakage model, which negatively affected our results. This could be addressed by moving away from optimizing for accuracy alone towards more elaborate metrics such as balanced accuracy, Matthew's correlation coefficient, or AUC-score [GBV20]. Additionally, there are different class weighting techniques proposed in the literature which penalize the misclassification of a minority class more than that of a majority class, which could improve the convergence and learning of a CNN [HK18]. One limitation of our study is that we only considered previously attacked datasets, which allows for a direct comparison with other approaches.

Research Question 7. *Does NAS allow for successful attacks on datasets for which currently no attacks with fixed architectures exist, e.g. ASCADv2?*

We constrained our study to four search strategies, but there are more sophisticated alternatives that have been observed to be more time-efficient and more effective than e.g. RANDOM search in finding an optimal architecture [Ren+21].

Another possible efficiency improvement would be early stopping, where some hyperparameter optimization runs are aborted early if the performance is particularly underwhelming or if no further improvement occurs.

Research Question 8. *What is the lower limit to the maximum number of trials where NAS stops to be a viable solution? How much better do the attacks get when considering even larger computational budgets for* RANDOM *search?*

6 Conclusion and Outlook

Impact of this Thesis We have created a theoretical framework utilizing machine learning to achieve the task of detecting information leakage, which we can apply to cryptographic side channels. The methods we developed have laid the ground for fully automatic side-channel detection tools like the AutoSCA-tool. We have demonstrated the capabilities of the tool in the real world and its first industrial application, the transport layer security (TLS) inspector software, already uses it to detect Bleichenbacher-like side channels in TLS. When it comes to local attacks, our improvements use NAS to eliminate the necessity of domain expert input for the architecture design phase of the machine learning attacks. This takes the community a further step towards full automation, both on the attack and on the detection side. Overall, our contributions to both remote and local side-channel detection demonstrate that automated side-channel detection is indeed feasible and practical.

The future of side-channel attacks The twenty-five-year-long history of Bleichenbacher side channels has shown that publishing an attack and updating the relevant RFC will not suffice to fix it, and even in 2023 new Bleichenbacher vulnerabilities are being discovered[24]. Even worse, the number of contexts in which the attack can be applied keeps growing, and accordingly also the variety of potentially vulnerable soft- and hardware. While the Rivest–Shamir–Adleman (RSA) key exchange that the original attack targeted has been removed from the newest TLS standard 1.3, the need for backward compatibility and the continued use of RSA for signatures in TLS means that the danger of Bleichenbacher vulnerabilities is still not eliminated. It is already apparent that local side-channel attacks on newly introduced post-quantum cryptography are possible, and the hardware community is working on attacking symmetric algorithms set to replace AES [GLS22].

While our approach of automating the detection of remote side channels with machine learning scales much better than manual investigation, we are still dependent on the knowledge about client misbehavior that can trigger behavior differences in servers. Introducing a fuzzing-like approach for this task could be the big enabler of previously unknown and unexpected side-channel detection. On the hardware side, the ever-improving attacks result in shrinking margins of error when it comes to manufacturers eliminating leakage. Here, new protections are already available that should counter all but the most advanced attacks (for example those that combine multiple probe locations with immense computing

[24]CVE-2023-0361, GnuTLS issue #1050

power). However, their adoption with new hardware generations is naturally slower than that of cryptographic software, and as such side-channel detection will still remain an important challenge in the upcoming decades.

We have observed that the publication of new attacks will gather substantial attention from the research community and the media alike. At the same time, the detection and prevention of side channels are seemingly under-represented. Considering the outlook that side channels are here to stay for the foreseeable future, more effort should be spent on prevention. This encompasses minimizing possible side-channel issues during the design phase of new cryptographic algorithms and improving how we tackle the side channels we already know about, for which machine learning-based automatic detection is an important step forward.

Bibliography

[AGF22] Rabin Y. Acharya, Fatemeh Ganji, and Domenic Forte. "Information Theory-based Evolution of Neural Networks for Side-channel Analysis". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.1 (2022), pp. 401–437. DOI: 10.46586/tches.v2023.i1.401-437. URL: https://tches.iacr.org/index.php/TCHES/article/view/9957.

[Avi+16] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. "DROWN: Breaking TLS Using SSLv2". In: *USENIX Security 2016: 25th USENIX Security Symposium*. Ed. by Thorsten Holz and Stefan Savage. Austin, TX, USA: USENIX Association, 2016, pp. 689–706.

[Bar+12] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, and Joe-Kai Tsay. "Efficient Padding Oracle Attacks on Cryptographic Hardware". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 608–625. DOI: 10.1007/978-3-642-32009-5_36.

[Ben+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. "Deep learning for side-channel analysis and introduction to ASCAD database". In: *Journal of Cryptographic Engineering* 10.2 (June 2020), pp. 163–188. DOI: 10.1007/s13389-019-00220-8.

[Ber+17] Daniel J. Bernstein, Joachim Breitner, Daniel Genkin, Leon Groot Bruinderink, Nadia Heninger, Tanja Lange, Christine van Vredendaal, and Yuval Yarom. "Sliding Right into Disaster: Left-to-Right Sliding Windows Leak". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, 2017, pp. 555–576. DOI: 10.1007/978-3-319-66787-4_27.

[Ber21] Anastasija Berlinblau. "Detection of Timing Side Channels: Extending the AutoSCA Tool". Bachelor's Thesis. Bergische Universität Wuppertal, 2021.

[BF04] Remco R. Bouckaert and Eibe Frank. “Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 3–12.

[BG04] Yoshua Bengio and Yves Grandvalet. “No Unbiased Estimator of the Variance of K-Fold Cross-Validation”. In: *Journal of Machine Learning Research* 5 (2004), pp. 1089–1105.

[BH02] Bhaskar Bhattacharya and Desale Habtzghi. “Median of the p Value under the Alternative Hypothesis”. In: *The American Statistician* 56.3 (Nov. 2002), pp. 202–206. URL: http://www.jstor.org/stable/3087299.

[Bha+14] Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. “Analysis and Improvements of the DPA Contest v4 Implementation”. In: *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings*. Ed. by Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont. Vol. 8804. Lecture Notes in Computer Science. Springer, 2014, pp. 201–218. DOI: 10.1007/978-3-319-12060-7_14.

[Ble98] Daniel Bleichenbacher. “Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1”. In: *Advances in Cryptology – CRYPTO’98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1998, pp. 1–12. DOI: 10.1007/BFb0055716.

[Bre01] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[BSY18] Hanno Böck, Juraj Somorovsky, and Craig Young. “Return Of Bleichenbacher’s Oracle Threat (ROBOT)”. In: *USENIX Security 2018: 27th USENIX Security Symposium*. Ed. by William Enck and Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, 2018, pp. 817–849.

[BZG20] Gabrielle Beck, Maximilian Zinkus, and Matthew Green. “Automating the Development of Chosen Ciphertext Attacks”. In: *USENIX Security 2020: 29th USENIX Security Symposium*. Ed. by Srdjan Capkun and Franziska Roesner. USENIX Association, 2020, pp. 1821–1837.

[Car+19] Mathieu Carbone, Vincent Conin, Marie-Angela Cornélie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. “Deep Learning to Evaluate Secure RSA Implementations”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.2 (2019). https://tches.iacr.org/

index.php/TCHES/article/view/7388, pp. 132–161. DOI: 10.13154/tches.v2019.i2.132-161.

[CCG10] Konstantinos Chatzikokolakis, Tom Chothia, and Apratim Guha. "Statistical measurement of information leakage". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Javier Esparza and Rupak Majumdar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 390–404.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. "Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, 2017, pp. 45–68. DOI: 10.1007/978-3-319-66787-4_3.

[CFR10] Jean-Christophe Courrège, Benoit Feix, and Mylène Roussellet. "Simple Power Analysis on Exponentiation Revisited". In: *Smart Card Research and Advanced Application*. Ed. by Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 65–79. DOI: 10.1007/978-3-642-12510-2_6.

[Cha+22] Lipeng Chang, Yuechuan Wei, Shuiyu He, and Xiaozhong Pan. "Research on Side-Channel Analysis Based on Deep Learning with Different Sample Data". In: *Applied Sciences* 12.16 (2022), p. 8246. DOI: 10.3390/app12168246.

[Cha+99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. "Towards Sound Approaches to Counteract Power-Analysis Attacks". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1999, pp. 398–412. DOI: 10.1007/3-540-48405-1_26.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. "An Efficient Method for Random Delay Generation in Embedded Software". In: *Cryptographic Hardware and Embedded Systems – CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Lecture Notes in Computer Science. Lausanne, Switzerland: Springer, Heidelberg, Germany, 2009, pp. 156–170. DOI: 10.1007/978-3-642-04138-9_12.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. "Template Attacks". In: *Cryptographic Hardware and Embedded Systems – CHES 2002*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Redwood Shores, CA, USA: Springer, Heidelberg, Germany, 2003, pp. 13–28. DOI: 10.1007/3-540-36400-5_3.

[CTJ21] Davide Chicco, Niklas Tötsch, and Giuseppe Jurman. "The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation". In: *BioData Mining* 14.1 (2021), p. 13. DOI: 10.1186/s13040-021-00244-z.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018.

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/BF02551274.

[DA99] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. RFC 2246 (Historic). RFC. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507, 7919. Fremont, CA, USA: RFC Editor, Jan. 1999. DOI: 10.17487/RFC2246. URL: https://www.rfc-editor.org/rfc/rfc2246.txt.

[Deg+12] Jean Paul Degabriele, Anja Lehmann, Kenneth G. Paterson, Nigel P. Smart, and Mario Strefler. "On the Joint Security of Encryption and Signature in EMV". In: *Topics in Cryptology – CT-RSA 2012*. Ed. by Orr Dunkelman. Vol. 7178. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 116–135. DOI: 10.1007/978-3-642-27954-6_8.

[Dem06] Janez Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets". In: *Journal of Machine Learning Research* 7 (Dec. 2006), pp. 1–30. DOI: 10.5555/1248547.1248548. URL: https://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf.

[dP15] Joeri de Ruiter and Erik Poll. "Protocol State Fuzzing of TLS Implementations". In: *USENIX Security 2015: 24th USENIX Security Symposium*. Ed. by Jaeyeon Jung and Thorsten Holz. Washington, DC, USA: USENIX Association, 2015, pp. 193–206.

[DPW11] Jean Paul Degabriele, Kenny Paterson, and Gaven Watson. "Provable Security in the Real World". In: *IEEE Security & Privacy* 9.3 (2011), pp. 33–41. DOI: 10.1109/MSP.2010.200.

[DR06] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*. RFC 4346 (Historic). RFC. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507, 7919. Fremont, CA, USA: RFC Editor, Apr. 2006. DOI: 10.17487/RFC4346. URL: https://www.rfc-editor.org/rfc/rfc4346.txt.

[DR08] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). RFC. Obsoleted by RFC 8446, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919, 8447, 9155. Fremont, CA, USA: RFC Editor, Aug. 2008. DOI: 10.17487/RFC5246. URL: https://www.rfc-editor.org/rfc/rfc5246.txt.

[Dre+21] Jan Peter Drees, Pritha Gupta, Eyke Hüllermeier, Tibor Jager, Alexander Konze, Claudia Priesterjahn, Arunselvan Ramaswamy, and Juraj Somorovsky. "Automated Detection of Side Channels in Cryptographic Protocols: DROWN the ROBOTs!" In: *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*. AISec '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 169–180. DOI: 10.1145/3474369.3486868.

[Dur+14] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. "The Matter of Heartbleed". In: *Proceedings of the 2014 ACM Internet Measurement Conference*. IMC '14. Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 475–488. DOI: 10.1145/2663716.2663755.

[Eas21] David Easter. "The impact of 'Tempest'on Anglo-American communications security and intelligence, 1943–1970". In: *Intelligence and National Security* 36.1 (2021), pp. 1–16. DOI: 10.1080/02684527.2020.1798604.

[EMH19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey". In: *Journal of Machine Learning Research* 20 (2019), 55:1–55:21. URL: http://jmlr.org/papers/v20/18-598.html.

[Fel+18] Dennis Felsch, Martin Grothe, Jörg Schwenk, Adam Czubak, and Marcin Szymanek. "The Dangers of Key Reuse: Practical Attacks on IPsec IKE". In: *USENIX Security 2018: 27th USENIX Security Symposium*. Ed. by William Enck and Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, 2018, pp. 567–583.

[Fis22] R. A. Fisher. "On the Interpretation of χ^2 from Contingency Tables, and the Calculation of P". In: *Journal of the Royal Statistical Society* 85.1 (Nov. 1922), pp. 87–94. DOI: 10.2307/2340521.

[FKK11] A. Freier, P. Karlton, and P. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. RFC 6101 (Historic). RFC. Fremont, CA, USA: RFC Editor, Aug. 2011. DOI: 10.17487/RFC6101. URL: https://www.rfc-editor.org/rfc/rfc6101.txt.

[Fri01] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* 29 (Nov. 2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451. URL: http://www.jstor.org/stable/2699986.

[FS97] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of computer and system sciences* 55.1 (Aug. 1997), pp. 119–139. DOI: 10.1006/jcss.1997.1504.

[FS99] Yoav Freund and Robert E. Schapire. "Large Margin Classification Using the Perceptron Algorithm". In: *Machine Learning* 37.3 (Dec. 1999), pp. 277–296. DOI: 10.1023/A:1007662407062. URL: https://doi.org/10.1023/A:1007662407062.

[FSH15] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. "Initializing Bayesian Hyperparameter Optimization via Meta-Learning". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 1128–1135.

[Fum+11] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. "Affine Masking against Higher-Order Side Channel Analysis". In: *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. Lecture Notes in Computer Science. Waterloo, Ontario, Canada: Springer, Heidelberg, Germany, 2011, pp. 262–280. DOI: 10.1007/978-3-642-19574-7_18.

[Fun22] Dennis Michael Funke. "Pushing the AutoSCA Tool to Picosecond Precision: Improving Timing Side Channel Detection". Bachelor's Thesis. Bergische Universität Wuppertal, 2022. DOI: 10.13140/RG.2.2.33070.08005.

[GBV20] Margherita Grandini, Enrico Bagli, and Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. 2020. DOI: 10.48550/ARXIV.2008.05756. arXiv: 2008.05756.

[GDH23] Pritha Gupta, Jan Peter Drees, and Eyke Hüllermeier. "Automated Side-Channel Attacks Using Black-Box Neural Architecture Search". In: *Proceedings of the 18th International Conference on Availability, Reliability and Security*. ARES '23. Benevento, Italy: Association for Computing Machinery, 2023. DOI: 10.1145/3600160.3600161.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely Randomized Trees". In: *Machine Learning* 63.1 (Apr. 2006), pp. 3–42. DOI: 10.1007/s10994-006-6226-1. URL: https://doi.org/10.1007/s10994-006-6226-1.

[GHO15] Richard Gilmore, Neil Hanley, and Maire O'Neill. "Neural network-based attack on a masked implementation of AES". In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2015, pp. 106–111. DOI: 10.1109/HST.2015.7140247.

[GJS19] Aron Gohr, Sven Jacob, and Werner Schindler. *CHES 2018 Side Channel Contest CTF - Solution of the AES Challenges*. Cryptology ePrint Archive, Report 2019/094. https://eprint.iacr.org/2019/094. 2019.

[GLS22] Aron Gohr, Friederike Laus, and Werner Schindler. "Breaking Masked Implementations of the Clyde-Cipher by Means of Side-Channel Analysis A Report on the CHES Challenge Side-Channel Contest 2020". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.4 (2022), pp. 397–437. DOI: 10.46586/tches.v2022.i4.397-437.

[Goo23] Google. *HTTPS encryption on the web*. 2023. URL: https://transparencyreport.google.com/https/overview (visited on 03/22/2023).

[Gup+22] Pritha Gupta, Arunselvan Ramaswamy, Jan Peter Drees, Eyke Hüllermeier, Claudia Priesterjahn, and Tibor Jager. "Automated Information Leakage Detection: A New Method Combining Machine Learning and Hypothesis Testing with an Application to Side-channel Detection in Cryptographic Protocols." In: *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*. ICAART '22. Setúbal, Portugal: SciTePress, 2022, pp. 152–163. DOI: 10.5220/0010793000003116.

[HBF11] Matthew Hoffman, Eric Brochu, and Nando de Freitas. "Portfolio Allocation for Bayesian Optimization". In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. UAI'11. Barcelona, Spain: AUAI Press, 2011, pp. 327–336.

[Hea+20] Tim Head, Manoj Kumar, Holger Nahrstaedt, Gilles Louppe, and Iaroslav Shcherbatyi. *scikit-optimize/scikit-optimize*. Version v0.8.1. Sept. 2020. DOI: 10.5281/zenodo.4014775. URL: https://doi.org/10.5281/zenodo.4014775.

[Het+20] Benjamin Hettwer, Tobias Horn, Stefan Gehrer, and Tim Güneysu. "Encoding Power Traces as Images for Efficient Side-Channel Analysis". In: *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2020, pp. 46–56. DOI: 10.1109/HOST45689.2020.9300289.

[Heu+20] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. "Lightweight Ciphers and Their Side-Channel Resilience". In: *IEEE Transactions on Computers* 69.10 (2020), pp. 1434–1448. DOI: 10.1109/TC.2017.2757921.

[HGG20] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. "Applications of machine learning techniques in side-channel attacks: a survey". In: *Journal of Cryptographic Engineering* 10.2 (June 2020), pp. 135–162. DOI: 10.1007/s13389-019-00212-8.

[HK18] Mahdi Hashemi and Hassan A. Karimi. "Weighted Machine Learning". In: *Statistics, Optimization & Information Computing* 6.4 (Nov. 2018), pp. 497–525. DOI: 10.19139/soic.v6i4.479.

[HK70] Arthur E. Hoerl and Robert W. Kennard. "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: *Technometrics* 12.1 (1970), pp. 55–67. DOI: 10.1080/00401706.1970.10488634.

[Hol79] Sture Holm. "A Simple Sequentially Rejective Multiple Test Procedure". In: *Scandinavian Journal of Statistics* 6.2 (1979). Full publication date: 1979, pp. 65–70. DOI: 10.2307/4615733. URL: http://www.jstor.org/stable/4615733.

[Hos+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. "Machine learning in side-channel analysis: a first study". In: *Journal of Cryptographic Engineering* 1.4 (Dec. 2011), pp. 293–302. DOI: 10.1007/s13389-011-0023-x.

[Hug68] G. Hughes. "On the mean accuracy of statistical pattern recognizers". In: *IEEE Transactions on Information Theory* 14.1 (1968), pp. 55–63. DOI: 10.1109/TIT.1968.1054102.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[Jen00] David Jensen. "Data Snooping, Dredging and Fishing: The Dark Side of Data Mining a SIGKDD99 Panel Report". In: *SIGKDD Explor. Newsl.* 1.2 (Jan. 2000), pp. 52–54. DOI: 10.1145/846183.846195.

[Jin21] Haifeng Jin. "Efficient neural architecture search for automated deep learning". PhD thesis. Texas A&M University, 2021. URL: https://oaktrust.library.tamu.edu/bitstream/handle/1969.1/193093/JIN-DISSERTATION-2021.pdf.

[JKM18] Tibor Jager, Saqib A. Kakvi, and Alexander May. "On the Security of the PKCS#1 v1.5 Signature Scheme". In: *ACM CCS 2018: 25th Conference on Computer and Communications Security*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. Toronto, ON, Canada: ACM Press, 2018, pp. 1195–1208. DOI: 10.1145/3243734.3243798.

[JSH19] Haifeng Jin, Qingquan Song, and Xia Hu. "Auto-Keras: An Efficient Neural Architecture Search System". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Ed. by Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis. ACM, 2019, pp. 1946–1956. DOI: 10.1145/3292500.3330648.

[JSS12] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky. "Bleichenbacher's Attack Strikes again: Breaking PKCS#1 v1.5 in XML Encryption". In: *ESORICS 2012: 17th European Symposium on Research in Computer Security*. Ed. by Sara Foresti, Moti Yung, and Fabio Martinelli. Vol. 7459. Lecture Notes in Computer Science. Pisa, Italy: Springer, Heidelberg, Germany, 2012, pp. 752–769. DOI: 10.1007/978-3-642-33167-1_43.

[JSS15] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption". In: *ACM CCS 2015: 22nd Conference on Computer and Communications Security*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. Denver, CO, USA: ACM Press, 2015, pp. 1185–1196. DOI: 10.1145/2810103.2813657.

[Kal98] B. Kaliski. *PKCS #1: RSA Encryption Version 1.5*. RFC 2313 (Informational). RFC. Obsoleted by RFC 2437. Fremont, CA, USA: RFC Editor, Mar. 1998. DOI: 10.17487/RFC2313. URL: https://www.rfc-editor.org/rfc/rfc2313.txt.

[Kas+21] Priyank Kashyap, Furkan Aydin, Seetal Potluri, Paul D. Franzon, and Aydin Aysu. "2Deep: Enhancing Side-Channel Attacks on Lattice-Based Key-Exchange via 2-D Deep Learning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.6 (2021), pp. 1217–1229. DOI: 10.1109/TCAD.2020.3038701.

[Kat+09] Toshihiro Katashita, Akashi Satoh, Takeshi Sugawara, Naofumi Homma, and Takafumi Aoki. "Development of side-channel attack standard evaluation environment". In: *19th European Conference on Circuit Theory and Design, ECCTD 2009, Antalya, Turkey, August 23-27, 2009*. IEEE, 2009, pp. 403–408. DOI: 10.1109/ECCTD.2009.5275001.

[KBA96] John R. Koza, Forrest H. Bennett, and Martin A. Andre Davidand Keane. “Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming”. In: *Artificial Intelligence in Design ’96*. Ed. by John S. Gero and Fay Sudweeks. Springer Netherlands, 1996, pp. 151–170. DOI: 10.1007/978-94-009-0279-4_9. URL: https://doi.org/10.1007/978-94-009-0279-4_9.

[Ke+17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Vol. 30. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157. DOI: 10.5555/3294996.3295074.

[Kel21] Evgnosia-Alexandra Kelesidis. “An Optimization of Bleichenbacher’s Oracle Padding Attack”. In: *Innovative Security Solutions for Information Technology and Communications - 14th International Conference, SecITC 2021, Virtual Event, November 25-26, 2021, Revised Selected Papers*. Ed. by Peter Y. A. Ryan and Cristian Toma. Vol. 13195. Lecture Notes in Computer Science. Springer, 2021, pp. 145–155. DOI: 10.1007/978-3-031-17510-7_10. URL: https://doi.org/10.1007/978-3-031-17510-7_10.

[Ker+22] Maikel Kerkhof, Lichao Wu, Guilherme Perin, and Stjepan Picek. “Focus is Key to Success: A Focal Loss Function for Deep Learning-Based Side-Channel Analysis”. In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by Josep Balasch and Colin O’Flynn. Cham: Springer International Publishing, 2022, pp. 29–48. DOI: 10.1007/978-3-030-99766-3_2.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis”. In: *Advances in Cryptology – CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[Kle17] Matthew Kleinsmith. *CNNs from different viewpoints*. Feb. 2017. URL: https://medium.com/impactai/cnns-from-different-viewpoints-fab7f52d159c.

[KOS17] Eike Kiltz, Adam O’Neill, and Adam D. Smith. “Instantiability of RSA-OAEP Under Chosen-Plaintext Attack”. In: *Journal of Cryptology* 30.3 (July 2017), pp. 889–919. DOI: 10.1007/s00145-016-9238-4.

[Koy+15] Oluwasanmi Koyejo, Pradeep Ravikumar, Nagarajan Natarajan, and Inderjit S. Dhillon. “Consistent Multilabel Classification”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’15. Montreal, Canada: MIT Press, 2015, pp. 3321–3329. DOI: 10.5555/2969442.2969610.

[KPR03] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. "Attacking RSA-Based Sessions in SSL/TLS". In: *Cryptographic Hardware and Embedded Systems – CHES 2003*. Ed. by Colin D. Walter, Çetin Kaya Koç, and Christof Paar. Vol. 2779. Lecture Notes in Computer Science. Cologne, Germany: Springer, Heidelberg, Germany, 2003, pp. 426–440. DOI: 10.1007/978-3-540-45238-6_33.

[KR03] Vlastimil Klíma and Tomás Rosa. "Further Results and Considerations on Side Channel Attacks on RSA". In: *Cryptographic Hardware and Embedded Systems – CHES 2002*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Redwood Shores, CA, USA: Springer, Heidelberg, Germany, 2003, pp. 244–259. DOI: 10.1007/3-540-36400-5_19.

[KS98] B. Kaliski and J. Staddon. *PKCS #1: RSA Cryptography Specifications Version 2.0*. RFC 2437 (Informational). RFC. Obsoleted by RFC 3447. Fremont, CA, USA: RFC Editor, Oct. 1998. DOI: 10.17487/RFC2437. URL: https://www.rfc-editor.org/rfc/rfc2437.txt.

[KZP06] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas. "Machine learning: a review of classification and combining techniques". In: *Artificial Intelligence Review* 26.3 (2006), pp. 159–190. DOI: 10.1007/s10462-007-9052-3.

[LBM15] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. "A machine learning approach against a masked AES - Reaching the limit of side-channel attacks with a learning model". In: *Journal of Cryptographic Engineering* 5.2 (June 2015), pp. 123–139. DOI: 10.1007/s13389-014-0089-3.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[Ler+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. "Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis)". In: *COSADE 2015: 6th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Stefan Mangard and Axel Y. Poschmann: vol. 9064. Lecture Notes in Computer Science. Berlin, Germany: Springer, Heidelberg, Germany, 2015, pp. 20–33. DOI: 10.1007/978-3-319-21476-4_2.

[Li+17] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research* 18.1 (Jan. 2017), pp. 6765–6816. DOI: 10.5555/3122009.3242042.

[Lom+14] Victor Lomné, Emmanuel Prouff, Matthieu Rivain, Thomas Roche, and Adrian Thillard. "How to Estimate the Success Rate of Higher-Order Side-Channel Attacks". In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. Lecture Notes in Computer Science. Busan, South Korea: Springer, Heidelberg, Germany, 2014, pp. 35–54. DOI: 10.1007/978-3-662-44709-3_3.

[LR06] Erich L Lehmann and Joseph P Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006. URL: https://link.springer.com/book/10.1007/0-387-27605-X.

[LT20] Liam Li and Ameet Talwalkar. "Random Search and Reproducibility for Neural Architecture Search". In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. Ed. by Ryan P. Adams and Vibhav Gogate. Vol. 115. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 367–377. URL: https://proceedings.mlr.press/v115/li20c.html.

[Man01] James Manger. "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0". In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2001, pp. 230–238. DOI: 10.1007/3-540-44647-8_14.

[Mas94] J.L. Massey. "Guessing and entropy". In: *Proceedings of 1994 IEEE International Symposium on Information Theory*. 1994, p. 204. DOI: 10.1109/ISIT.1994.394764.

[MDK14] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. *This POODLE Bites: Exploiting The SSL 3.0 Fallback*. 2014. URL: https://www.openssl.org/~bodo/ssl-poodle.pdf.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. "A Comprehensive Study of Deep Learning for Side-Channel Analysis". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.1 (2019). https://tches.iacr.org/index.php/TCHES/article/view/8402, pp. 348–375. DOI: 10.13154/tches.v2020.i1.348-375.

[Mer+19] Robert Merget, Juraj Somorovsky, Nimrod Aviram, Craig Young, Janis Fliegenschmidt, Jörg Schwenk, and Yuval Shavitt. "Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities". In: *USENIX Security 2019: 28th USENIX Security Symposium*. Ed. by Nadia Heninger and Patrick Traynor. Santa Clara, CA, USA: USENIX Association, 2019, pp. 1029–1046.

[Mer+21] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. "Raccoon Attack: Finding and Exploiting Most-Significant-Bit-Oracles in TLS-DH(E)". In: *USENIX Security 2021: 30th USENIX Security Symposium*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 213–230.

[Mey+14] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. "Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks". In: *USENIX Security 2014: 23rd USENIX Security Symposium*. Ed. by Kevin Fu and Jaeyeon Jung. San Diego, CA, USA: USENIX Association, 2014, pp. 733–748.

[Mit97] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. URL: https://www.worldcat.org/oclc/61321007.

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008. DOI: 10.1007/978-0-387-38162-6.

[MPP16a] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. "Breaking Cryptographic Implementations Using Deep Learning Techniques". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Cham: Springer International Publishing, 2016, pp. 3–26. DOI: 10.1007/978-3-319-49445-6_1.

[MPP16b] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel 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 Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Vol. 10076. Lecture Notes in Computer Science. Springer, 2016, pp. 3–26. DOI: 10.1007/978-3-319-49445-6_1. URL: https://doi.org/10.1007/978-3-319-49445-6_1.

[Mus+18] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Vianney Lapotre, and Guy Gogniat. "NIGHTs-WATCH: A Cache-Based Side-Channel Intrusion Detector Using Hardware Performance Counters". In: *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '18. Los Angeles, California: Association for Computing Machinery, 2018. DOI: 10.1145/3214292.3214293. URL: https://doi.org/10.1145/3214292.3214293.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. "DL-LA: Deep Learning Leakage Assessment". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.3 (2021). https://tches.iacr.org/index.php/TCHES/article/view/8986, pp. 552–598. DOI: 10.46586/tches.v2021.i3.552-598.

[NB03] Claude Nadeau and Yoshua Bengio. "Inference for the Generalization Error". In: *Machine Learning* 52.3 (Sept. 2003), pp. 239–281. DOI: 10.1023/A:1024068626366. URL: https://doi.org/10.1023/A:1024068626366.

[PCP20] Guilherme Perin, Łukasz Chmielewski, and Stjepan Picek. "Strength in Numbers: Improving Generalization with Ensembles in Machine Learning-based Profiled SCA". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.4 (2020). https://tches.iacr.org/index.php/TCHES/article/view/8686, pp. 337–364. DOI: 10.13154/tches.v2020.i4.337-364.

[Ped+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. "Scikit-Learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (Nov. 2011), pp. 2825–2830.

[Per+21] Thomas Perianin, Sebastien Carré, Victor Dyseryn, Adrien Facon, and Sylvain Guilley. "End-to-end automated cache-timing attack driven by machine learning". In: *Journal of Cryptographic Engineering* 11.2 (2021), pp. 135–146. DOI: 10.1007/s13389-020-00228-5. URL: https://doi.org/10.1007/s13389-020-00228-5.

[PHG17] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. "Template attack versus Bayes classifier". In: *Journal of Cryptographic Engineering* 7.4 (Nov. 2017), pp. 343–351. DOI: 10.1007/s13389-017-0172-7.

[Pic+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. "Side-channel analysis and machine learning: A practical perspective". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 4095–4102. DOI: 10.1109/IJCNN.2017.7966373.

[Pic+18] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. "The Curse of Class Imbalance in Side-channel Evaluation". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.1 (2018). https://tches.iacr.org/index.php/TCHES/article/view/7339, pp. 209–237. DOI: 10.13154/tches.v2019.i1.209-237.

[Pic+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. "SoK: Deep Learning-Based Physical Side-Channel Analysis". In: *ACM Computing Surveys* 55.11 (Feb. 2023). DOI: 10.1145/3569577.

[Pla99] John C. Platt. "Fast Training of Support Vector Machines Using Sequential Minimal Optimization". In: *Advances in Kernel Methods: Support Vector Learning*. Cambridge, MA, USA: MIT Press, 1999, pp. 185–208. DOI: 10.5555/299094.299105. URL: https://dl.acm.org/doi/10.5555/299094.299105.

[PWP21] Guilherme Perin, Lichao Wu, and Stjepan Picek. *AISY - Deep Learning-based Framework for Side-channel Analysis*. Cryptology ePrint Archive, Report 2021/357. https://eprint.iacr.org/2021/357. 2021.

[Ren+21] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions". In: *ACM Computing Surveys (CSUR)* 54.4 (May 2021). DOI: 10.1145/3447582.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/rfc/rfc8446.txt.

[Rij+21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. "Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.3 (2021). https://tches.iacr.org/index.php/TCHES/article/view/8989, pp. 677–707. DOI: 10.46586/tches.v2021.i3.677-707.

[RM05] Lior Rokach and Oded Maimon. "Decision Trees". In: *Data Mining and Knowledge Discovery Handbook*. Ed. by Oded Maimon and Lior Rokach. Boston, MA: Springer US, 2005, pp. 165–192. DOI: 10.1007/0-387-25465-X_9.

[Rok10] Lior Rokach. "Ensemble-based classifiers". In: *Artificial Intelligence Review* 33.1 (Feb. 2010), pp. 1–39. DOI: 10.1007/s10462-009-9124-7. URL: https://doi.org/10.1007/s10462-009-9124-7.

[Ron+19] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. "The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations". In: *2019 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2019, pp. 435–452. DOI: 10.1109/SP.2019.00062.

[RTL09] Payam Refaeilzadeh, Lei Tang, and Huan Liu. “Cross-Validation”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 532–538. DOI: 10.1007/978-0-387-39940-9_565. URL: https://doi.org/10.1007/978-0-387-39940-9_565.

[SAS21] Mehwish Shaikh, Qasim Ali Arain, and Salahuddin Saddar. “Paradigm Shift of Machine Learning to Deep Learning in Side Channel Attacks - A Survey”. In: *2021 6th International Multi-Topic ICT Conference (IMTIC)*. 2021, pp. 1–6. DOI: 10.1109/IMTIC53841.2021.9719689.

[Smy96] Padhraic Smyth. “Clustering Using Monte Carlo Cross-Validation”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD’96. Portland, Oregon: AAAI Press, 1996, pp. 126–133.

[Som16] Juraj Somorovsky. “Systematic Fuzzing and Testing of TLS Libraries”. In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. Vienna, Austria: ACM Press, 2016, pp. 1492–1504. DOI: 10.1145/2976749.2978411.

[Spr+17] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. “Systematic classification of side-channel attacks: A case study for mobile devices”. In: *IEEE Communications Surveys & Tutorials* 20.1 (2017), pp. 465–488. DOI: 10.1109/COMST.2017.2779824.

[Ste17] Josef Steppan. *A few samples from the MNIST test dataset*. File: MnistExamples.png. 2017. URL: https://commons.wikimedia.org/wiki/File:MnistExamples.png.

[SZ15] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. DOI: 10.48550/arXiv.1409.1556.

[TK09] Sergios Theodoridis and Konstantinos Koutroumbas. “Chapter 5 - Feature Selection”. In: *Pattern Recognition (Fourth Edition)*. Ed. by Sergios Theodoridis and Konstantinos Koutroumbas. Fourth Edition. Boston: Academic Press, 2009, pp. 261 –322. DOI: https://doi.org/10.1016/B978-1-59749-272-0.50007-4. URL: http://www.sciencedirect.com/science/article/pii/B9781597492720500074.

[TPR13] Adrian Thillard, Emmanuel Prouff, and Thomas Roche. “Success through Confidence: Evaluating the Effectiveness of a Side-Channel Attack”. In: *Cryptographic Hardware and Embedded Systems – CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. Lecture

Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2013, pp. 21–36. DOI: 10.1007/978-3-642-40349-1_2.

[Vau02] Serge Vaudenay. "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS..." In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, 2002, pp. 534–546. DOI: 10.1007/3-540-46035-7_35.

[VB12] Gitte Vanwinckelen and Hendrik Blockeel. "On estimating model accuracy with repeated cross-validation". In: *BeneLearn 2012: Proceedings of the 21st Belgian-Dutch conference on machine learning*. 2012, pp. 39–44.

[Vir+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[von01] Manfred von Willich. "A Technique with an Information-Theoretic Basis for Protecting Secret Data from Differential Power Attacks". In: *8th IMA International Conference on Cryptography and Coding*. Ed. by Bahram Honary. Vol. 2260. Lecture Notes in Computer Science. Cirencester, UK: Springer, Heidelberg, Germany, 2001, pp. 44–62.

[Wan+22] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. "Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86". In: *USENIX Security 2022: 31st USENIX Security Symposium*. Ed. by Kevin R. B. Butler and Kurt Thomas. Boston, MA, USA: USENIX Association, 2022, pp. 679–697.

[Wil92] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Breakthroughs in Statistics: Methodology and Distribution*. Ed. by Samuel Kotz and Norman L. Johnson. New York, NY: Springer New York, 1992, pp. 196–202. DOI: 10.1007/978-1-4612-4380-9_16. URL: https://doi.org/10.1007/978-1-4612-4380-9_16.

[Wou+20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. "Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.3 (2020). https://tches.iacr.org/index.php/TCHES/article/view/8586, pp. 147–168. DOI: 10.13154/tches.v2020.i3.147-168.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. *I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis*. Cryptology ePrint Archive, Report 2020/1293. https://eprint.iacr.org/2020/1293. 2020.

[WV21] David Warburton and Sander Vinberg. *The 2021 TLS Telemetry Report*. 2021. URL: https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report (visited on 04/01/2023).

[Xia+17] Yuan Xiao, Mengyuan Li, Sanchuan Chen, and Yinqian Zhang. "STACCO: Differentially Analyzing Side-Channel Traces for Detecting SSL/TLS Vulnerabilities in Secure Enclaves". In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. Dallas, TX, USA: ACM Press, 2017, pp. 859–874. DOI: 10.1145/3133956.3134016.

[XL01] Qing-Song Xu and Yi-Zeng Liang. "Monte Carlo cross validation". In: *Chemometrics and Intelligent Laboratory Systems* 56.1 (2001), pp. 1–11. DOI: https://doi.org/10.1016/S0169-7439(00)00122-2. URL: http://www.sciencedirect.com/science/article/pii/S0169743900001222.

[YHL11] Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. "Dual Coordinate Descent Methods for Logistic Regression and Maximum Entropy Models". In: *Machine Learning* 85.1-2 (Oct. 2011), pp. 41–75. DOI: 10.1007/s10994-010-5221-8. URL: https://doi.org/10.1007/s10994-010-5221-8.

[Zai+19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. "Methodology for Efficient CNN Architectures in Profiling Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.1 (2019). https://tches.iacr.org/index.php/TCHES/article/view/8391, pp. 1–36. DOI: 10.13154/tches.v2020.i1.1-36.

[Zai+21] Gabriel Zaid, Lilian Bossuet, François Dassance, Amaury Habrard, and Alexandre Venelli. "Ranking Loss: Maximizing the Success Rate in Deep Learning Side-Channel Analysis". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (2021).

https://tches.iacr.org/index.php/TCHES/article/view/8726, pp. 25–55. DOI: 10.46586/tches.v2021.i1.25-55.

[Zha+14] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. "Cross-Tenant Side-Channel Attacks in PaaS Clouds". In: *ACM CCS 2014: 21st Conference on Computer and Communications Security*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. Scottsdale, AZ, USA: ACM Press, 2014, pp. 990–1003. DOI: 10.1145/2660267.2660356.

[Zha+20] Jiajia Zhang, Mengce Zheng, Jiehui Nan, Honggang Hu, and Nenghai Yu. "A Novel Evaluation Metric for Deep Learning-Based Side Channel Analysis". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.3 (2020). https://tches.iacr.org/index.php/TCHES/article/view/8583, pp. 73–96. DOI: 10.13154/tches.v2020.i3.73-96.

[ZZT09] Li Zhuang, Feng Zhou, and J Doug Tygar. "Keyboard acoustic emanations revisited". In: *ACM Transactions on Information and System Security (TISSEC)* 13.1 (2009), pp. 1–26.

Appendix

A Minimum Bleichenbacher Dataset Sizes

For a given learning model, in-sample error (or accuracy) E_{in} is its performance on the training dataset, and the out-of-sample error (or accuracy) E_{out} is its performance on the test data. The plot consisting of the evolution of the two error/accuracy scores as the size of the training set increases are called learning curves. For small n, the in-sample accuracy E_{in} might be much lower since it's quite easy to perfectly fit fewer data points, however, the out-of-sample error E_{out} will be very large. The reason behind this might be that the learning model is built around a small data, and it almost certainly won't be able to generalize accurately on data the learner hasn't seen before. Generally, for the large training set size, the learning model cannot fit perfectly anymore the training set and the training error becomes larger, while the out-of-sample error E_{out} decreases, because the model is trained on more data, so it manages to fit the test dataset better. After a certain number of training instances N, the E_{out} stays roughly the same, which implies that adding more training instances won't produce significantly better models. According to the VC-dimension theory, this N represents the sample complexity of an algorithm, i.e. the number of training examples that are required to successfully learn a target function. More precisely, the sample complexity is the number of training-samples that we need to supply to the algorithm, so that the function returned by the algorithm is within an arbitrarily small error of the best possible function, with probability arbitrarily close to 1.

We plot different the learning curves of the subset of binary classifiers, perceptron learning algorithm (PLA) from linear models, decision tree (DT), support vector machine (SVM), random forest (RF) from bagging and histogram gradient boosting (HGB) from boosting models, as shown in Figure 1. As seen in Figure 1, most of the binary classifiers show no significant decrease in the out-of-sample error after a certain number of instances. We can use this concept to determine the number of samples that should be generated by the system under test (SUT) for each class-label. For OpenSSL 1.1.1k, we can see that the PLAs and SVM require less than 200 instances to converge, i.e. to reach minimum possible error rate i.e. 0.5 same as random guessing (RG). While more complex binary classifiers like DT, RF and HGB are not able to converge even with 3600 instances. So, we cannot say with confidence if the system is not vulnerable, since it can be the case that the vulnerability exists but the existing models are not able to capture it yet. While for the DamnVulnerable OpenSSL server, almost all complex

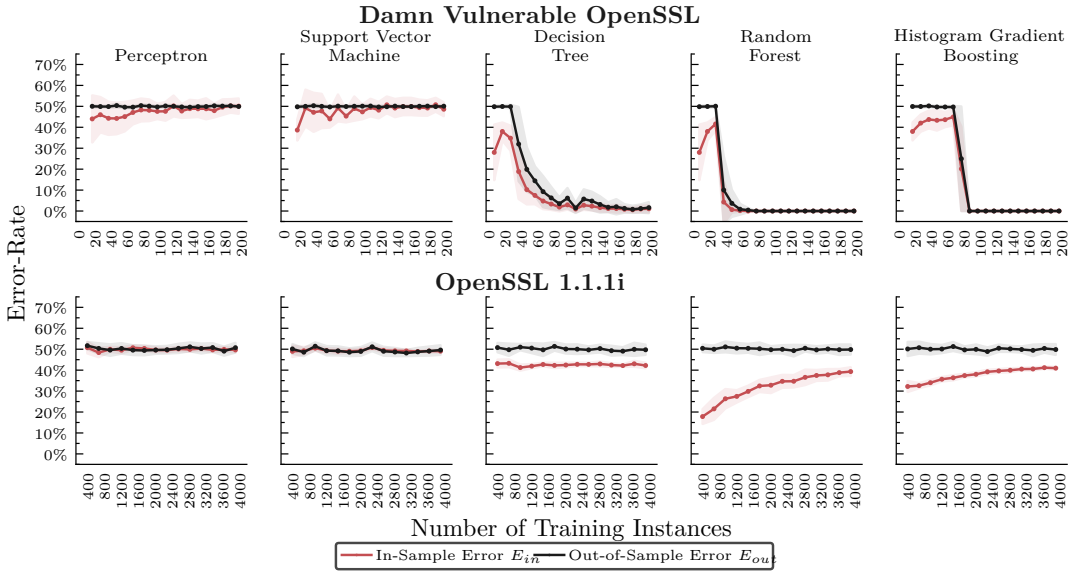


Figure 1: Learning curves of different classifiers for class-label "Wrong first byte (`0x00` set to `0x17`)" for DamnVulnerableOpenSSL and OpenSSL 1.1.1i

models like DT, bagging and boosting techniques require less than 50 instances to converge, i.e. to reach the minimum possible error rate of 0.0. The learning curves for PLA is interesting, as it requires more instances (400). As we have seen in Section 4.4.3, even if one of the classifiers can significantly perform better than RG, after applying the Holm-Bonferroni adjustment, the SUT will be marked as vulnerable to the given class-label. Summarizing these curves, we can imply that if there exists a vulnerability or side channel in the SUT, our approach should be able to detect it for the countable number of instances with high probability. But, we cannot imply non-vulnerability or absence of side channels in the SUT since it might be the case that the side channel exists but the current models are not able to identify it yet.

B Hardware Datasets

The following lists the 10 hardware datasets used in the experiments shown in Chapter 5. It appears in the appendix of [GDH23], with this dataset collection contributed by the author to the paper, among the other contributions discussed in Chapter 5. Details about the number of features, traces, as well as URLs where the datasets can be obtained are shown in Table 1. Further datasets and their details have been collected by the author and published on Github[25].

ASCAD The ASCAD dataset[26] was proposed as a benchmark dataset, containing electromagnetic (EM) radiation measurements obtained from an ATMega8515 device (8-bit microcontroller with AVR architecture) running AES-128 [Ben+20]. The implementation is coded in assembly language and uses 1st order boolean masking. The measurements were executed by placing a copper coil close to the device and recording the EM emissions. We utilize the original v1 dataset in both fixed key (ASCAD_f) and variable key (ASCAD_r) variants. The traces in these datasets have been temporally aligned and only a subset of samples are selected, based on the signal-to-noise ratio [Ben+20]. Additionally, we consider variants created by Benadjila *et al.* where the traces have been randomly desynchronized by a maximum of 50 as well as 100 samples, simulating imprecise temporal alignment of the traces.

CHES CTF The CHES CTF dataset[27] is a reduced (50 000 traces) and preprocessed dataset included in the AISY framework [PWP21], derived from the original measurements (500 000 traces) done for the CHES 2018 AES CTF challenge[28]. The software implementation of AES-128 is protected using 1st order boolean masking and runs on a STM32 microcontroller. The dataset we use contains a single fixed key in the training dataset and a different fixed key in the attack dataset.

AES_RD AES_RD[29] was collected from an AES-128 software implementation incorporating random delays as a hiding countermeasure [CK09]. The measurements target the power consumption of an 8-bit ATMEL AVR microcontroller. The random delay (RD) countermeasure adds a random number of up to 16 additional microcontroller instructions at several places throughout the processing of the actual AES instructions, rendering the attack more difficult because of the resulting trace misalignment. The target of the attacks, the sub-byte operation in the first AES round, is therefore separated from the start of the execution (start

[25] `https://github.com/ITSC-Group/sca-datasets`
[26] `https://github.com/ANSSI-FR/ASCAD`
[27] `http://aisylabdatasets.ewi.tudelft.nl/ches_ctf.h5`
[28] `https://chesctf.riscure.com/2018/content?show=training`
[29] `https://github.com/gabzai/Methodology-for-efficient-CNN-architectures-in-SCA/tree/master/AES_HD`

of the trace) by 32 of these random delays [CK09]. We use the converted dataset as analysed in [Zai+19].

AES_HD AES_HD[30] contains EM measurements obtained from a Xilinx Virtex-5 FPGA (on the SASEBO-GII board [Kat+09]) running an unprotected AES-128 VHDL implementation [Pic+18]. It records the AES decryption operation instead of the encryption operation and targets the register writing in the last AES round. It assumes the difference in the last round leaks, and thus uses a distance leakage model based on the ciphertext bytes c_i^j used in the decryption, specifically the 12th (c_i^{11}) and 8th (c_i^7) ciphertext bytes. We calculate the label as $\phi(c_i, k_i) = sbox^{-1}(c_i^{11} \oplus k_i) \oplus c_i^7$, but as opposed to the Hamming Distance (HD) model used in [Pic+18] we just use this identity instead of its Hamming Weight. The dataset contains measurements obtained "using a high sensitivity near-field EM probe, placed over a decoupling capacitor on the power line"[Pic+18]. We again use the converted dataset as analysed in [Zai+19].

DPAv4 The DPAv4 dataset[31] was used in the fourth edition of the DPA contest[32] and contains traces obtained from a software implementation running on a microcontroller [Bha+14]. The measurements are obtained by recording the power consumption of the ATMEL AVR-163 microcontroller running an AES-128 implementation protected with rotating S-Box masking. We use the "improved" masked AES-128 target contained in dataset version 4.2 from [Zai+19]. In order to be consistent while comparing our approach with the performance of the baselines proposed by [Zai+19], we assume that the mask value is known, essentially nullifying the masking. Accordingly, we generate the labels using $\phi(p_i, k_i) = sbox(p_i \oplus k_i) \oplus M_i$, with M_i denoting the mask value. As with [Zai+19], we target the first byte being processed in the first AES round.

[30] https://github.com/AESHD/AES_HD_Dataset/

[31] https://github.com/gabzai/Methodology-for-efficient-CNN-architectures-in-SCA/tree/master/DPA-contest%20v4

[32] http://www.dpacontest.org/v4/42_traces.php/

Table 1: Details of the datasets considered

Dataset	# Features	# Profiling traces	# Attack traces	Target byte	URL
ASCAD_f	700	50000	10000	2	ASCAD.h5 from https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key
ASCAD_f desync50	700	50000	10000	2	ASCAD_desync50.h5 from "
ASCAD_f desync100	700	50000	10000	2	ASCAD_desync100.h5 from "
ASCAD_r	1400	50000	100000	2	ASCAD.h5 from https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key/
ASCAD_r desync50	1400	50000	100000	2	ASCAD_desync50.h5 from "
ASCAD_r desync100	1400	50000	100000	2	ASCAD_desync100.h5 from "
CHES CTF	2200	45000	5000	2	http://aisylabdatasets.ewi.tudelft.nl/ches_ctf.h5
AES_HD	1250	50000	25000	0	https://github.com/gabzai/Methodology-for-efficient-CNN-architectures-in-SCA/blob/master/AES_HD/AES_HD_dataset.zip
AES_RD	3500	25000	25000	0	https://github.com/gabzai/Methodology-for-efficient-CNN-architectures-in-SCA/tree/master/AES_RD/AES_RD_dataset
DPAv4	4000	4500	5000	0	https://github.com/gabzai/Methodology-for-efficient-CNN-architectures-in-SCA/blob/master/DPA-contest%20v4/DPAv4_dataset.zip

C Neural Architecture Search Space

The hyperparameter search space used for the experiments in Chapter 5 is discussed here. Table 2 contains the full details on all hyperparameters we are tuning.

The hyperparameters for the convolutional layer are the kernel size and the number of filters, for the pooling layer the poolsize w_p, the number of strides and pooling operation type and for a dense layer the number of hidden units. The network, layer, and dense block hyperparameters are the same for both 1-D convolutional neural network (CNN) search space and 2-D CNN search space. Additionally, the range of convolutional kernel size, convolutional filters, and pooling types for each convolutional block is also the same for both search spaces. To avoid the formation of invalid 2-D CNN architectures, the range of pooling strides is smaller and the poolsize value is set using the kernel size, which is the suggested default of `AutoKeras` [JSH19]. This makes the search space relatively smaller for 2-D CNNs. The padding type is set to the "same" padding type for convolutional layer and "valid" padding type for pooling layer for 1-D CNNs search space. To avoid the formation of invalid 2-D CNN, the padding type for the convolutional and pooling layer is set using the kernel size of the convolutional layer, using the formulae proposed by `AutoKeras` [JSH19].

Table 2: Overview of the Search Space for our neural architecture search (NAS) approach

Hyperparameter Type	Hyperparameter	Possible Options
Whole Network	Optimizer	{'adam' , 'adam_with_weight_decay'}
	Learning rate	{1e-1, 5e-2, 1e-2, 5e-3, 1e-3, 5e-4, 1e-4, 5e-5, 1e-5}
Every Layer	Dropout	$\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$
	Use Batch Normalization	{True, False}
	Activation Function	{'relu', 'selu', 'elu', 'tanh'}
Convolutional Block	# Blocks	$\{1, 2, 3, 4, 5\}$
	Convolutional Kernel Size	{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
	Convolutional Filters	{2, 8, 16, 32, 64, 128, 256}
	Pooling Type	{'max' , 'average'}
	Pooling Strides 1-D CNN	{2, 3, 4, 5, 6, 7, 8, 9, 10}
	Pooling Poolsize 1-D CNN	{2, 3, 4, 5}
	Pooling Strides 2-D CNN	{2, 4}
	Pooling Poolsize 2-D CNN	Convolutional Kernel Size-1
Dense Block	# Blocks	{1, 2, 3}
	Hidden Units	{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}