Master's Thesis Computing Science

The more, the merrier? A step-by-step inter-device analysis for transfer learning side-channel attacks

On how previously trained networks can be used for tackling the portability problem

Lizzy Grootjen
s1001148

May 30, 2024

First supervisor/assessor:
Prof. dr. Ileana R. Buhan

Second co-supervisor:
Zhuoran Liu

Second assessor:
Prof. dr. Stjepan Picek

Radboud University

Abstract

In this study, we examine the impact of the physical layer on the performance of deep learning models applied to side-channel analysis. We explore the relationship between inter-device variations and model performance using identical devices. We consider the effects of transfer learning, varying target bytes, and simulated noise on attack performance. Our findings indicate minimal inter-device variations among identical devices, allowing the direct application of the base model across devices with comparable results to those obtained using transfer learning. We conclude that transfer learning is not necessary to enhance model performance. We also show that incorporating Gaussian noise into the training data improves model robustness while adding Gaussian noise to the attack traces reduces performance.

Keywords — transfer learning, side-channel analysis, deep learning, cross-device attacks, noise

Contents

Chapter 1

Introduction

With the emerging IoT devices, small chips run cryptographic software to protect against attackers. The security community has explored the possibility of attacking devices without mathematically breaking the cryptographic software [LCC08] [RD20]. This emerged into a new field of hardware security called side-channel analysis. Each device emits data based on its physical properties - the so-called side-channel. There are several techniques to analyze this data, from statistical tests to a relatively new field - machine learning.

In the last few years, the field of using deep-learning for side-channel analysis has been investigated [PPM+23]. Training such networks is computationally expensive, due to the large amounts of data. Most research used the ASCAD dataset or their own collected data, of which most are 8-bit devices. However, current IoT devices are shifting towards operating on 32-bit devices. On top of that, for every attack, the deep neural network needs to be retrained due to the differences between devices, as it is not possible to have one general neural network usable for all side-channel scenarios [KPH+19].

This difference in physical properties between devices is called the *portability problem*. When performing profiling attacks - attacks where an extra device is used to model the distribution of the target device - there is always a slight difference between the profiling device and the target device. This is due to architectural differences, configuration differences, and different types of noise within the device.

When performing a side-channel attack, a certain attacker model is assumed. For this research, we assume the attacker has full access to a similar device compared to the device under attack. The attacker also knows what encryption implementation is running on the device under attack. This way, the attacker can control the input plaintext and key and generate power traces from this profiling device.

This research aims to close the gap in deep learning side-channel analysis research by using 32-bit devices. We also investigate the relevance of the portability problem for 15 identical devices. Finally, we investigate the possibilities of applying transfer learning when performing a deep-learning attack to account for the inter-device differences and thus tackle the portability problem.

To answer the research question, we collected data from 15 identical 32-bit ARM Cortex M4 devices running a firmware implementation of tinyAES-128. ChipWhisperer has been used to collect the power traces. For each device combination, the inter-device variations are investigated. Then, the possibilities of using transfer learning on an existing good-performing model are investigated, and how inter-device variations affect the performance of a successful attack.

In short, the main contributions are:

1. Collect data from 15 identical 32-bit devices running tinyAES-128;

2. Explore how 32-bit devices perform compared to the 8-bit devices in literature;
3. Explore how transfer learning can improve the accuracy of cross-device attacks;
4. Explore how 15 devices vary among them and how this influences the key rank output by the deep neural network.
5. Provide a tool to compare ChipWhisperer devices to the 15 devices used in this research.

Chapter 2

Preliminaries

This chapter discusses any preliminary knowledge required for this work. This includes the architecture of the ARM Cortex M4 chip, techniques for side-channel analysis, and different deep-learning techniques.

2.1 Target device: ARM Cortex M4

Embedded devices are everywhere: in your car, smart home system, and Fitbit. More devices are connected to the internet - called the "Internet of Things" - and communicate with each other. When information is transmitted, it can be categorized into two types: sensitive information and regular information. Sensitive information needs to be protected to prevent learning its content. Protecting sensitive data can be done on several levels: the software security layer, the network security layer, and the hardware security layer [Ins]. In this thesis, the focus is on the security of the hardware layer.

2.1.1 Architecture

The architecture of a device is known as the description of the structure of a computer system and its components [PH90]. A classical Von Neumann architecture consists of two main parts: the CPU and the memory unit. In the memory unit, a set of instructions is stored. Every clock cycle, a new instruction is fetched from memory and executed by the CPU. Different architectures are also available, such as the Harvard architecture [Sar23]. The ARM Cortex M4 is an example of a RISC architecture, which implements the Harvard architecture [Ibr]. Figure 2.1 displays a schematic overview of the processor.

This processor features dedicated Digital Signal Processing (DSP) IP blocks, including an optional Floating-Point Unit (FPU). Therefore, this processor is easy for embedded systems, such as IoT, motor control, power management, embedded audio, industrial and home automation, and healthcare and wellness applications [ST].

2.1.2 ISA and micro-architecture

The architecture of a device can be organized into several subcomponents: one of them is the instruction set architecture. The instruction set architecture is part of an abstract model, which describes how the CPU is controlled by the software [ARM]. An implementation of this abstract model is called the micro-architecture. For the ARM Cortex M4, every chip has the same instruction set architecture. However, depending on the vendors, the implementation of this instruction set might vary.

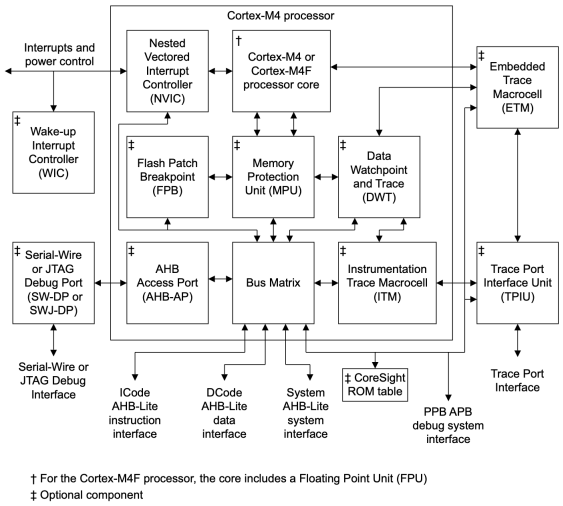


Figure 2.1: The structure of the ARM Cortex M4 processor. Image from [ARM10]

Not every instruction takes the same amount of power and time to execute. Therefore, in the field of side-channel analysis, it is important to know which kind of micro-architecture is implemented on the target device. To streamline the terminology, Zhang et al. made a distinction for devices and their micro-architectures [ZSX+20] as follows:

1. Same device: there is access to exactly one device used for the task.
2. Identical devices: the relationship between two devices are physical copies of the same chip model. The design and all configurations are identical.
3. Homogenous devices: In homogenous devices, the architecture of both devices is the same, but the micro-architecture and configuration are different. This happens when different manufacturers make a chip.
4. Heterogenous devices: this situation is similar to a real-life scenario: two devices that differ on a chip level in all aspects, including micro-architecture and power dissipations.

This terminology is used during this thesis to refer to the relationship of the devices.

2.2 Advanced Encryption Standard - AES

To map the physical differences between devices, it is important to investigate what is exactly running on these devices. Commonly, an attacker is trying to obtain sensitive or secret information. This information is usually secured by using encryption. Currently, AES is the most widely used symmetric cryptography algorithm. Most literature in the side-channel community uses AES as the standard algorithm for attacking through side-channel [BPS+18]. The Advanced Encryption Standard - AES - is a symmetric key block cipher. Each block consists of 128 bits, and the key size varies between 128, 192, and 256. A schematic overview is displayed in figure 2.2.

Firstly, from the provided key by the user, subkeys are determined using the AES keyschedule. After that, each round round consists of four parts:

1. `SubBytes`: This is a non-linear transformation of the state to prevent correlation between input and output at a byte level. This function makes use of a substitution box (also called

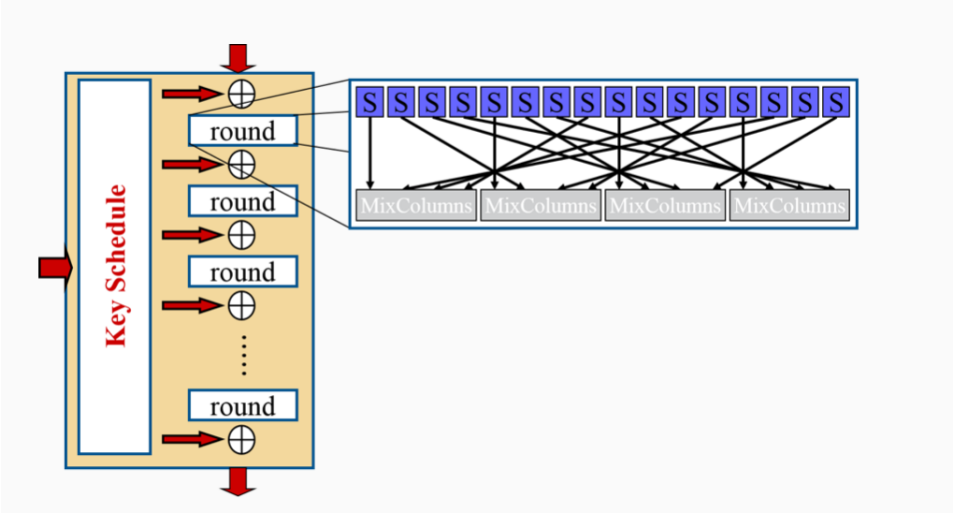


Figure 2.2: Visualisation of AES. This figure shows the different rounds of AES, with the four functions happening within a round. [Dae20]

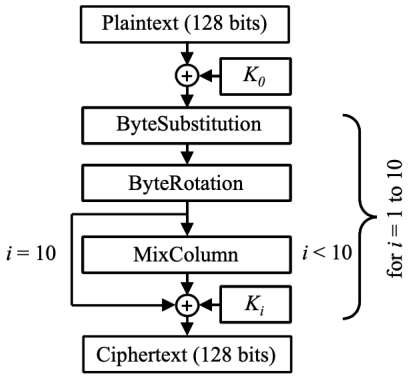


Figure 2.3: Schematic breakdown of AES. This image shows the four functions performed within one AES round. Image from [RDJ+01]

S-box) which is derived from the multiplicative inverse over $GF(2^8)$, as it is known for its good non-linear properties. It is usually implemented as a lookup table.

2. `ShiftRows`: This creates diffusion within the cipher.

3. `MixColumns`: This creates diffusion within the cipher. When in the final round, MixColumns is skipped.

4. `AddRoundKey`: in figure 2.3, this is displayed as the key addition after `MixColumns`. `AddRoundKey` adds the subkey (derived from the secret key) to the current state.

These four parts are also highlighted in 2.3. From a side-channel perspective, the step `SubBytes` is the most interesting part. The input and output size of the s-box is 8 bits, which results in a small sensitive variable based on the input and a subkey.

2.2.1 Power consumption

When executing encryption algorithms, some calculations may take more time and/or power compared to other calculations. There are some naturally harder calculations, such as division. The ARM Cortex M4 takes 2-12 rounds for executing a division, as opposed to 1 cycle for addition, subtraction, and multiplication [ARM10]. The power consumed depends on the current instruction or instruction pairs [KU17].

To correctly analyse the power consumption of a side channel, an investigation is needed on what components define the power consumption. Mangard et al. wrote a comprehensive guide on side-channel analysis [LCC08]. They provide a good definition in section 4.1 on the different components of a power trace, quoted below.

> "Each point of a power trace can be modeled as the sum of an operation-dependent component P_{op}, a data-dependent component P_{data}, electronic noise $P_{elnoise}$, and a constant component P_{const}"

$$P_{total} = P_{op} + P_{data} + P_{elnoise} + P_{const} \tag{2.1}$$

2.2.2 Reference implementation

For this research, the unmasked firmware implementation of tinyAES-128 supplied by the vendors of ChipWhisperer is used. The code belonging to this implementation can be found at [Chi]. In the figure 2.4, the power consumption of the ARM Cortex M4 processor running tinyAES-128 is displayed. As can be seen, the chip consumes different amounts of power at different points in time. Through visual inspection, it is already possible to distinguish the eight rounds performed by AES.

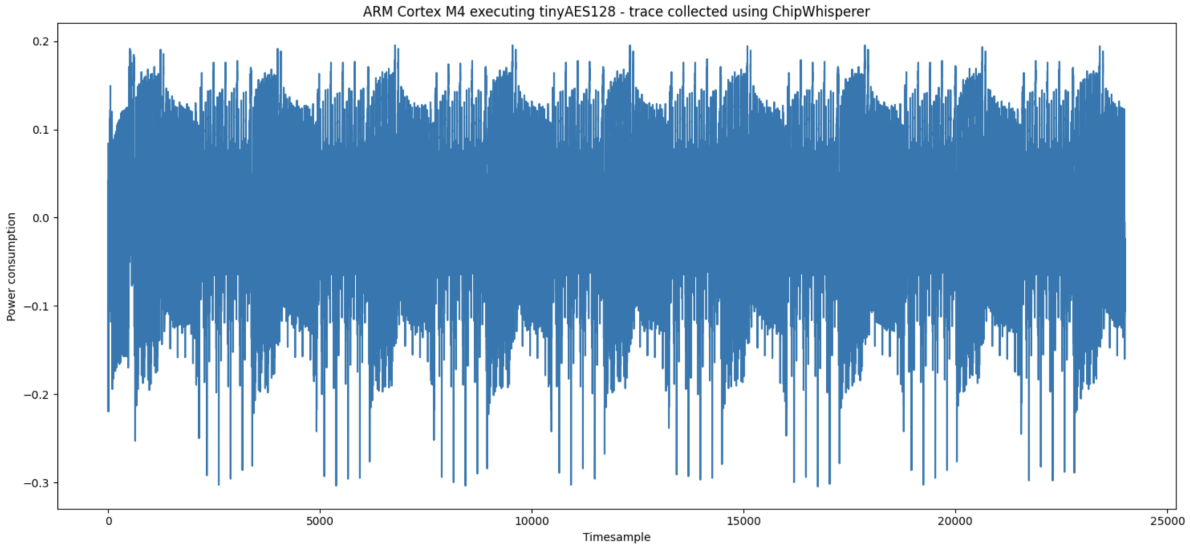


Figure 2.4: One trace collected using ChipWhisperer from an ARM Cortex M4 running tinyAES

This power consumption is not only influenced by the chip's instructions and operations. Its data also influences it. This has been shown by Das et al [DGD+19]: they found a difference in power consumption on the same device with a different key: inter-key variations. The inter-key variations would be modeled as P_{data} according to the definition in 2.1.

2.3 Side-channel analysis

As described earlier, every device emits data based on its physical properties. This is called the *side channel*. Examples of side channels are power consumption, electromagnetic emissions, and time of execution, which can be exploited. The leakage through the side channels is highly dependent on the micro-architecture of a device [ABPP22]. To perform a good analysis, knowing what kind of device(s) are targeted is important. In this thesis, the focus is on the power side channel.

When analyzing the data from a side channel, this can be categorized into two types of attacks: non-profiled and profiled. In non-profiled attacks, the attacker performs a statistical analysis without any assumption on the distribution of the target device. In other words, this attack is being deployed without creating a profile of the target device. In these attacks, usually, a statistical distinguisher is used to identify the power traces. Profiled attacks are based on the assumption that the target device follows an unknown distribution [PPM+23]. This distribution can be approximated to create a profile from the target device. For each of the categories, a few common attacks are highlighted.

2.3.1 Non-profiled attacks

The most common techniques for non-profiled power analysis involve statistical inference, for which many traces are collected from the device under attack. This is a non-profiling attack, as the attacker does not use a copy of the target device. Examples are simple power analysis (SPA) and differential power analysis (DPA). In SPA, an average trace is computed from all the traces, which is then visually inspected [BB22] [LCC08]. In section 2.2.2, it has already been highlighted that the separate AES rounds are visible from the trace. DPA uses a statistical test to verify or dismiss a hypothesis [KJJ99] [LCC08]. The attacker chooses a bit from the sensitive variable v - which should be relatively small. Usually, the output of the s-box is taken as the sensitive variable, as explained in section 2. A schematic overview of the sensitive variable v is displayed in figure 2.5.

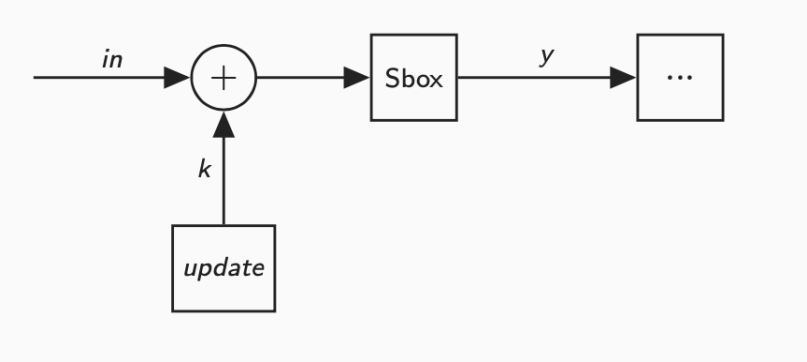


Figure 2.5: Variable y chosen as the sensitive variable v - output of s-box. Source: [BB22]

When making a DPA attack, the attacker computes the value of the output S-box for a chosen constant plaintext and each possible key value. For each possible key guess, the target bit can either have two values: 0 or 1. The traces are sorted on this target bit for every possible key guess. When the difference between those two classes is the biggest, that is the key with the most probability of being correct. In figure 2.6, this is schematically visualized.

It is also possible to use different distinguishers for this attack. Another common distinguisher is the correlation - this is called a correlational power attack (CPA). Next to a CPA the distinguished, a leakage model is also required - most commonly is the Hamming Weight. The

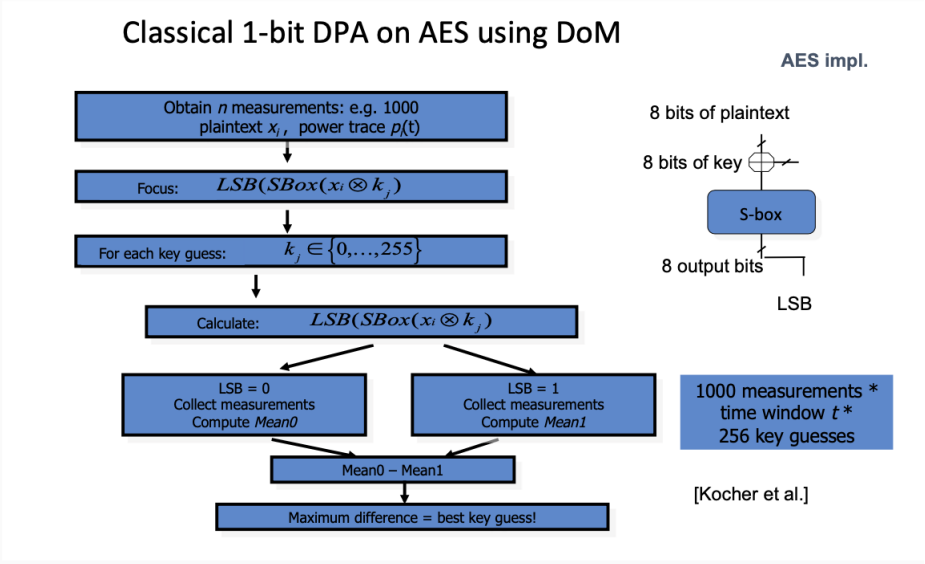


Figure 2.6: DPA attack with distinguisher of means on the least significant bit. This figure shows the grouping of power traces based on a key guess on the value of the least significant bit. Source: [BB22]

idea is that the Hamming weight models the power consumption of a device: when a certain bit-value is 0, less power is consumed compared to a bit-value 1. This hamming weight is again calculated over the sensitive variable v. With the Hamming Weight, an estimate is made on how much power this output will consume on the device. Then, where the hamming weight correlates the most with the traces is probably the best key guess [LCC08]. Figure 2.7 shows a higher-level overview of statistical attacks.

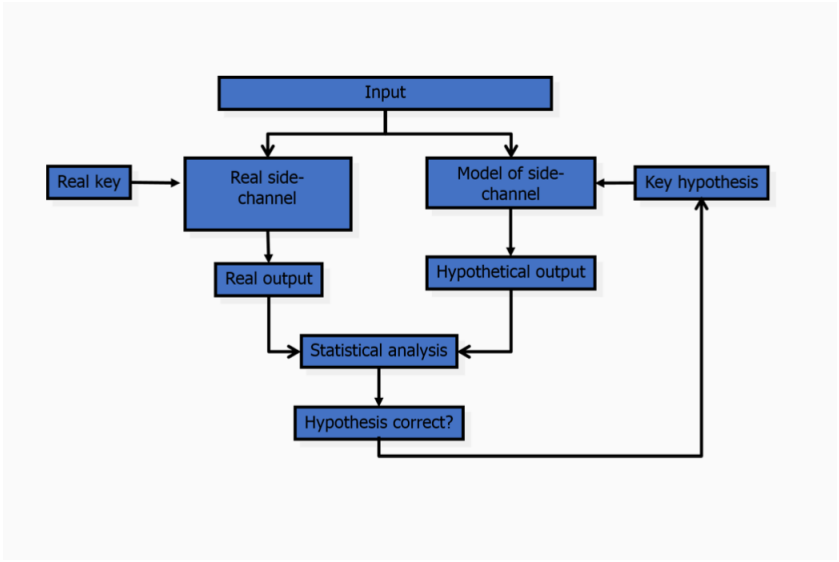


Figure 2.7: High-level overview on statistical attacks. A hypothetical model is compared to the real side channel for statistical attacks. This hypothetical model is built for every possible key guess. When the hypothesis is correct, it is most likely the correct key guess. Source: [BB22]

2.3.2 Profiled attacks

There is also another class of side-channel attacks: profiling attacks. In this type of attack, it is assumed that measurements follow an unknown distribution [PPM+23]. One of the strongest attacks is the template attack by [CRR02]. This attack assumes the leakage of a target device follows a multivariate Gaussian distribution. In this attack, the profiling stage consists of computing the statistical parameters for this distribution [PPM+23]. This is done by computing the hamming weight as leakage for every possible key byte. In the attacking phase, the attacker computes the probability that a newly obtained power trace belongs to either one of the parameters.

In the last years, analysis techniques from the field of artificial intelligence have been deployed as profiling attacks. Methods such as random forests, naive Bayes and support vector machines have been applied to side-channel analysis [BLR13] [HZ12] [HGDM+11] [LBM14] [LPMS18] [BCH+20]. When Hospodar et al. applied least-squares support vector machines as a profiling attack for side-channel analysis, they found that the choice of parameters is of a higher influence on performance compared to the amount of attack traces available. Lerman highlighted the curse of dimensionality in using statistical attacks, and how dimensionality reduction techniques sometimes can outperform modern machine learning approaches [LPMS18].

2.3.3 The portability problem

As an attacker, it is hard to make a correct assumption on the distribution of the measurements. For SPA and DPA, it is assumed that the attacker has access to the direct target device. However, it can take millions of measurements to obtain the key with these statistical attacks [PPM+23]. In contrast to profiling attacks, attackers tend to acquire a copy of the device under attack: an identical device. The attacker uses this identical device to create a profile from the device under attack. Bhasin showed that inter-device difference (and to a certain extent inter-key difference) can influence the performance of the model [BCH+20]. The difference in leakage distribution between the profiling device and the device under attack is called the *portability problem*.

Referring back to our components in power traces, the inter-device difference would be modelled as a strong varying $P_{el.noise}$ devices, where the inter-key difference would be modelled as P_{data}. The components P_{op} and P_{const} will be constant when using an identical device.

2.3.4 Assumptions and success metrics

For this work, two metrics are considered: the signal-to-noise ratio and the key rank. The signal-to-noise ratio is not part of an attack but is used to evaluate the leakage of a power trace. The key rank is used to evaluate the performance of a side-channel attack. Papagiannopoulos summarised the most commonly used side-channel metrics including their benefits and limitations [PGA+23].

The signal-to-noise ratio is a metric to calculate the ratio between signal and noise using their variance. A signal-to-noise ratio with a value above 1 indicates for a power trace that there exists more signal than noise. This signal-to-noise ratio can be calculated in two ways: with simulated traces and with real traces [PGA+23]. In our case, real traces are used. The signal-to-noise ratio is calculated using a leakage model, in our case the hamming weight. Traces are sorted according to their hamming weight value into groups. For each group, the average trace is calculated. The noise for a certain group is then calculated by subtracting the mean trace. Finally, the variance for the signal trace is divided by the noise trace [Buh]. The signal-to-noise ratio in this work is used as a way to determine the point of interest in a trace.

To evaluate the performance of a side-channel attack, a success metric has to be chosen. Regular metrics such as accuracy for profiling attacks do not cover the complete view in the

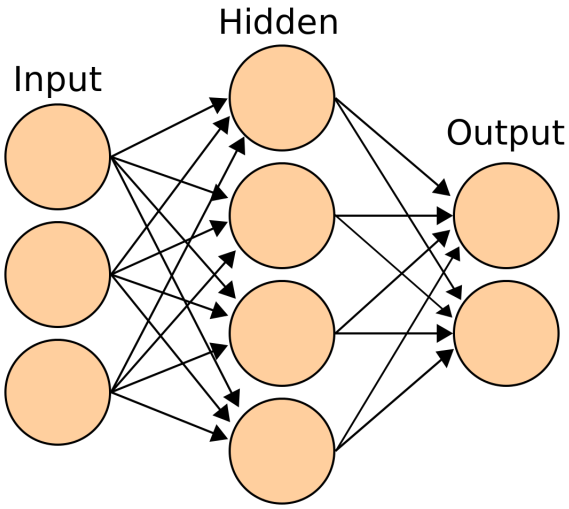


Figure 2.8: A multilayer perceptron with fully connected layers. Source: Wikipedia

security community. For example, if the key guesses are sorted according to probability in a profiling attack, and the correct key is the second highest probability, this can still be a security threat. The accuracy metric would only consider the first guess. For this, a metric called *key rank* is most commonly used in the side-channel community. For a target byte in the key, all 256 possible values are ranked. When the attacker has access to more attack traces, the key guesses can become more accurate. The key rank is the position of a key guess in the sorted vector of scores. This score is calculated using the maximum log-likelihood. The *full key rank* is the extended version of the key rank: which is the number of full key candidates to enumerate before reaching the correct full key [PGA+23]. Usually, the key rank is plotted against the number of attack traces. There are other success metrics as well, but this work is focused on the key rank.

2.4 Machine learning

Neural networks are well known for learning distributions within a dataset - which is exactly the problem at hand when an attacker is trying to do a profiling attack [Bis94]. However, neural networks are notorious for their training phase - also called the profiling phase in the side-channel community. Not does training only take many computational resources, but it also requires large amounts of data [AZH+21] [Ham93]. Choosing the right architecture for the neural network highly influences the performance and generalisability [Ala19] [AZH+21].

Neural network architectures

Artificial neural networks are computational models which are inspired by the structure of the brain [Bis94]. Each network consists of neurons and connections, which are usually sub-organised in layers. Classically, a neural network consists of an input layer, several hidden layers and an output layer. Based on the connections and the weight of the connections, input is processed through each layer. Based on a differentiable optimisation function, the feedback is backpropagated through the network. This is the ”learning”-step, where weights are updated accordingly.

Neural networks can have different topologies - also called architectures. There are roughly two main architectures, mainly based on how they process data: feedforward networks and re-

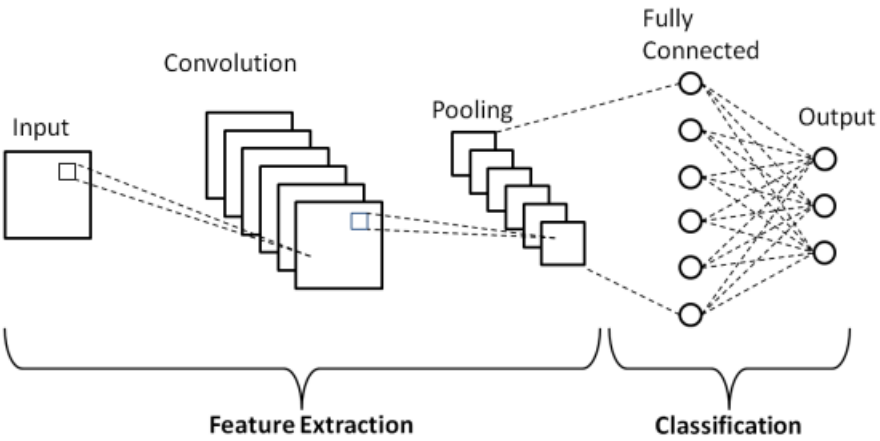


Figure 2.9: A convolutional neural network. This figure shows the distinction between the feature extraction and the classification part. Source: [PR19]

current neural networks [KR92]. In feedforward networks, the information flows uni-directional: there are no recurrent loops. In recurrent neural networks, the data can flow both ways. For this work, only feedforward networks are considered. The multilayer perceptron is a type of feedforward network, which is known for solving non-linearly separable tasks [Cyb89]. A schematic of a multilayer perceptron is shown in 2.8. This network is characterised by having at least three layers - an input layer, one or more hidden layers and an output layer - which are all fully connected.

Another neural network structure which is widely applied is the convolutional neural network [AZH+21]. This network is a type of deep feedforward network, where convolutional layers are used to extract relevant features to learn from them [LBD+89]. It was first designed for image classifications, but 1D data with patterns has been proven to be suitable for convolutional neural networks [KLN18]. Convolutional layers are defined by a kernel size - the size of the window extracting features - and the stride - how much the window moves each step. Convolutional layers are in almost all cases followed by a pooling layer, to reduce dimensionality. A schematic of a convolutional neural network has been displayed in figure 2.9.

Transfer learning

As said before, training such neural networks requires a high amount of data and computation resources. However, it is possible to use pre-trained networks on other similar tasks as well. This is called transfer learning [Boz20]. These pre-trained networks can sometimes be used directly or first finetuned to the new task. Finetuning is done by taking your pre-trained network, and deciding on freezing certain layers - i.e. they do not update while finetuning - or initialising certain layers with the weights of the pre-trained network. It is important to know how the new dataset relates to the original trained network. This determines how much finetuning is needed on the original pre-trained network. Depending on the similarity of your data, there are roughly four cases [JIM20]:

1. A small but similar dataset: in this case, your data is similar to the data of your original network. However, there is not much data to fine-tune the network again. In this case, one could choose to freeze all layers of the pre-trained network (both convolutional layers as fully connected layers) but the last one. The last layer - which is the classification layer - is initialised randomly and then finetuned on the new dataset. In the context of side-channel

analysis, a similar dataset might be data collected from an identical device or maybe a homogenous device.

2. A small but different dataset: when your dataset is different, your feature extraction needs to be finetuned as well. In this case, all layers are frozen, except the last (few) convolutional layers and the last fully connected layer. In the context of side-channel analysis, this might be the case with heterogeneous devices, but possibly also with homogenous devices.

3. A large but similar dataset: When there is access to a larger dataset, there are more possibilities for improving your network. In this case, it is not needed to freeze layers, as there is the capacity to retrain them. Therefore, all layers are initialised with the weights from the pre-trained network. Only the last layer is randomly initialised to be fine-tuned to the new dataset. When the attacker has access to such a larger similar dataset, it might be possible that homogenous devices can also be attacked correctly in this approach.

4. A large but different dataset: in this case, it is also possible to train the network from scratch. Therefore, the choice is more or less up to the user to initialise the weights of the convolutional layers (partially) from the pre-trained network. The model might converge earlier and have overall better performance. However, the last layer should be initialised at random. For side-channel analysis, depending on how many inter-device variations there are, it might be possible to attack heterogeneous devices through this method. However, this has not been investigated yet.

Chapter 3

Related Work

In the past years, the field of side-channel analysis in combination with artificial intelligence has been moving rapidly. Papers have been published from different points of view. In this chapter, the current state-of-the-art on profiled attacks is discussed. Then, papers covering the portability are summarised. Finally, some work is performed on using transfer learning for side-channel analysis.

3.1 Current state-of-the-art on profiled attacks

Picek et al. [PPM+23] has recently written a systematisation of knowledge on deep learning in side-channel analysis. They highlight current state-of-the-art techniques in various aspects, such as publicly available SCA datasets, research on different data augmentation models, cross-device models to overcome portability, different attack metrics and the importance of loss functions. It is a broad research paper meant to highlight all important findings in the SCA community. For each subarea, the main findings and challenges are listed. Recommendations are also given to make research in this area more structured. One of these recommendations was to further investigate the possibilities of transfer learning, as Thapar et al. showed promising results [TAM20].

One of the publicly available SCA datasets which was mentioned by Picek et al. is the ASCAD dataset. Benadjila et al. have developed this dataset to facilitate the study of deep learning of side-channel analysis [BPS+18]. Through electromagnetic emissions they collected power traces from two devices running AES: the 8-bit ATMega8515 MCU and the 32-bit Cortex-M ARM. The focus of this study was twofold: firstly, develop a dataset similar to the MNIST dataset in the machine learning community. This way, there is consistency in benchmarking and improving networks based on this shared dataset. Secondly, they used this dataset to throughout investigate different machine-learning models for side-channel analysis. These models varied from support vector machines and random forests to different neural network architectures. They also wanted to emphasise the importance of transparency for hyper-parameter tuning in the deep learning models, to enhance reproducibility. They concluded that MLPs and CNNs are the most promising machine learning models. The advantage of MLPs is their simplicity, but they require traces to be preprocessed and aligned. On the other hand, CNNs can deal with desynchronised traces and still perform well. The main drawback of CNNs is that they are harder to train. CNNs based on the VGG16 structure showed the highest performance. Even though convolutional neural networks are classically used for image classification, they still reached good results for 1-dimensional data such as power traces [KLN18] [KPH+19].

Kim et al. [KPH+19] have developed a similar convolutional neural network inspired by the VGG16 design. This new convolutional neural network has similar design principles as the

convolutional neural network designed by [BPS+18], but has a factor of 10 fewer parameters and has three more convolutional blocks. The performance of this model tested on ASCAD is also higher with this model. Kim et al. emphasise that there is no such thing as a "Free Lunch": one general neural network can't be generalisable on different SCA problems. To reuse previously trained networks, some adjustments might be needed to a certain neural network architecture if a new problem is tackled. Another contribution made by Kim et al. showed that adding a Gaussian noise tensor after the first batch normalisation layer improves the robustness of the model, as compared to task-specific noise. This prevents overfitting of the model.

3.2 Approaching Portability

The portability problem is not left untouched. Different researchers have made attempts to overcome this problem by using various approaches.

Das et al. highlight that most previous work done on side-channel analysis with deep learning focuses on attacking the same device - mostly from the ASCAD or DPAv4 dataset [DGD+19]. Therefore, they decided to collect traces from different devices using ChipWhisperer, to facilitate a cross-device side-channel attack. As ChipWhisperer delivers aligned traces, Das et al. opted for a fully connected deep neural network network. In total, they used eight identical 8-bit ATMEGA devices, four for profiling and four for attacking. Their main contribution is to design a fully connected 256-class deep neural network which is trained, validated and tested on multiple identical devices. They showed that a single-trace attack is possible with an accuracy of 99% with 10k training traces and 99.9% with 200k traces. Some specific key bytes are more likely to be misclassified. They also showed that inter-device variations can influence accuracy, but these variations become smaller when multiple devices are included in the train set.

Bhasin et al. highlight the same problem: previous work only focuses on profiling and attacking the same device [BCH+20]. However, using different devices for profiling and attacking is considered hard due to inter-device variations, which is also known as the portability problem. The training data is closely related to the device properties. Therefore, they highlight that the portability issue arises due to a suboptimal validation phase. To tackle this problem, a new model is introduced: the Multiple Device Model. The idea behind this model is to augment the profiling phase - which includes training and validating - with more devices, such that the model performs better on unseen devices. They used in total four identical 8-bit ATMEGA devices to investigate four scenarios using a regular side-channel setup: same device and same key, same device and different key, identical device and same key, and identical device with different key. They found that expanding their data to multiple identical devices of their Atmega328p 8-bit microcontroller target device improved the guessing entropy. The main difference between Bhasin et al. and Das et al. is that Bhasin et al. collected their data through probes and obtaining unaligned traces, whereas Das et al. collected their data through ChipWhisperer.

Zhang et al. [ZSX+20] propose a whole new analysing method to overcome the portability problem: Frequency and Learning based Power Analysis (FL-PA). Instead of augmenting the training dataset with traces from multiple devices as Bhasin and Das et al. did, this method first transforms the power traces with Fast Fourier Transform to the frequency domain, before training a deep neural network. They think the failure of a regular template attack on homogenous or heterogenous devices is attributed to the selection of POI region and cycles of instruction. Those are related to the clock in the time domain. By first transforming the traces, the timing differences between devices are eliminated. In total, they have ten devices with different relations: there are identical devices, homogeneous devices and heterogeneous devices. They evaluated their method across all possible combinations of profiling- and attacking devices. They found that all attacks succeeded within 600 attack traces compared to over 1000 traces with a deep

learning attack in the time domain. With their new method, they trained the neural network on their local PIC devices and attacked the DPAContest v4 dataset. This showed to be successful within 800 traces.

All highlighted papers above use 8-bit micro-controllers as target devices. Most IoT devices are currently transitioning to 32-bit microcontrollers, leaving new research opportunities.

3.3 Transfer learning and SCA

Next to augmenting the dataset and preprocessing your data, there are other options to improve the portability issue for side-channel attacks as well. One of these options is to investigate the possibilities of transfer learning. There are a few recent papers which touch upon the usage of transfer learning for DNNs in SCA.

As highlighted earlier, Thapar et al. already investigated the possibilities of transfer learning for side-channel analysis [TAM20]. They propose a deep-learning technique using transfer learning: TransSCA. This method improves the possibility of a real-world attack scenario and reduces training costs. The main difference between this model and regular DL-SCA is the fine-tuning phase. This is done by freezing some layers and fine-tuning the unfrozen layers. They found that it matters which layers are frozen when entering the fine-tuning phase (i.e. transfer learning phase). In this work, the last layer of the model was frozen to obtain the best results. They use simulated power traces to model different FPGA families running AES128.

A similar approach was taken by Genevey-Metat et al. [GMGH20]: they wanted to use transfer learning for three scenarios in a SCA setting, of which one was a cross-device side-channel attack in the power domain. Their main investigation was based on three attacker models with different powers:

1. access to a pre-trained network
2. access to a clone dataset (i.e. identical device)
3. access to both

For this experiment, they used a ChipWhisperer light combined with a CW308 UFO board with STM32Fx target devices. In total, traces were collected from four devices for finetuning the pre-trained network. They investigated re-training the pre-trained model and freezing all the convolutional layers, which gave similar results. In the end, they chose to retrain the model to prevent making wrong assumptions. The authors indicated that investigating the possibilities of finetuning the last layer would be a good extension of this work. The third attacker model (access to a pre-trained network which is retrained and a clone dataset) improved the accuracy slightly.

The paper by Yu et al. investigates the possibilities of using meta-transfer learning [YSPJ21]. This is a mixture of transfer learning and meta-learning, which reduces training time and costs. This method makes it possible to train on device A, and then use the meta-transfer learning approach to attack device B. The paper tested their method against different 32-bit microcontrollers, even across domains. The results look promising in terms of training time and the amount of training data needed, however, they did not address the portability problem.

It is also possible to use transfer learning across domains: Cao et al. developed a deep learning model for cross-device side-channel attack (CDPA) with domain adaptation [CZLG21]. For their research, they used 8 identical ATMEGA devices and three identical SAKURA-G devices. As a base model, they used a convolutional neural network. Transfer learning was used to attack across domains: from the EM domain to the power domain across an identical 8-bit device. A fine-tuning phase was needed to transfer this knowledge and it improved results for some of the

device combinations. A cross-domain experiment using transfer learning was also conducted by Genevy-Metat et al [GMGH20]. Using the pre-trained CNN network from the ASCAD dataset, they experimented with the same three cases as in their cross-device experiment. Only in the case where the attacker has access to the pre-trained network and the clone dataset, it was possible to perform a transfer learning attack across domains.

Some research has already been done on using transfer learning to attack across devices. Most of the research used 8-bit devices as targets, whereas most current IoT devices are 32-bit. The model by Genevy-Metat et al. ended up retraining the network, as it gave similar results to the fine-tuning phase. From the currently existing literature, it is unclear from this set of papers if transfer-learning is more beneficial than using DL-SCA, due to different setups and different finetuning phases. The most promising paper by Yu et al. did not cover portability on 32-bit devices.

Chapter 4

Experimental setup

Within the previous two chapters, it has been shown that there is a gap between relevant literature and the real world. First of all, most research on portability has been done on 8-bit identical devices, while currently there is a shift to 32-bit microcontrollers in IoT. Second, the research done on transfer learning for side-channel analysis is inconclusive concerning the portability problem and transfer learning specifications.

4.1 Definitions and Research Question

To close the gap between current literature, this research focuses on the inter-device difference of 32-bit identical devices and how this affects transfer learning. On top of that, the effect of noise in traces is investigated on the performance of the model. Therefore, the research question of this thesis is as follows:

"How does transfer learning compare to a regular deep neural network for a successful side-channel attack with both clean and noisy traces on identical 32-bit devices?"

To answer this research question, a definition of "successful attack" needs to be provided. For this work, a successful attack is defined as follows:

"A successful attack is when the target byte is guessed correctly in exactly one guess"

In the context of deep learning, this means that the network should output a probability distribution of all possible values for the target byte and the correct target byte should have the highest probability.

For this research, we assume the following attacker model: Firstly, the attacker should know which encryption algorithm is running on the device under attack. To train the base model, an identical device is required as the device under attack, which means the target device and training device have a similar micro-architecture. The attacker should have full control of this identical device - the training device. To obtain the traces from the training device, the attacker lets this training device compute the same algorithm as the device under attack. This algorithm should be a firmware or software implementation, and this should be operated on one byte at a time. On top of that, the attacker can determine the plaintexts and keys, to gather the corresponding power traces. This would lead to the dataset that facilitates the supervised training of the base model.

4.2 Devices & specifications

For this experiment, the devices for trace collection used are ChipWhisperers Lite with a 32-bit STM32F303 target board, which has an ARM Cortex M4 chip. ChipWhisperer is running firmware version 0.65. The ARM Cortex M4 chip is executing a simple serial AES implementation, which comes standard with ChipWhisperer. This is the tinyAES128 implementation [Chi]. In total, there is access to 15 identical devices.

As the traces collected with ChipWhisperer are relatively clean, and we are dealing with clone devices, we decided to add some noise to the data. It has been shown by Kim et al. that adding non-Gaussian noise improves the robustness of the network and supports generalization. For this research, artificial electrical noise is added to the traces to investigate the robustness of the model. This noise is taken from a Gaussian distribution.

4.3 Data collection

The ChipWhisperers have been used to acquire the power consumption of the ARM Cortex M4 chip running tiny AES-128. For this work, two groups of data are collected:

1. The dataset for intra- and inter-device analysis
2. The dataset for the deep learning model

The dataset for intra- and inter-device analysis consists of two devices: A and B. Within this experiment, a total of 6 sessions are collected. Sessions 1 and 2 on device A were collected on the same day as session 1 of device B. Sessions 3 and 4 on device A were collected on the same day as session 2 of device B. Each session consists of 500 traces with 12000 time samples, for which the plaintext and keys were the same for all sessions. In this dataset, one trace is collected with the maximum amount of possible time samples allowed by ChipWhisperer. A schematic of the dataset used for device analysis is displayed in figure 4.1.

The second dataset for the deep learning model consists of 15 devices in total. Device A is our ground device, on which the model is trained. We collected 50k traces from our ground device A. These traces have random plaintexts and a fixed key. For all devices, there are 10k traces collected with the same properties as the ground device which is used for testing. This includes an additional dataset for device A. On top of that, for each device, there are 500 traces collected with fixed plaintexts and fixed keys for the device analysis. For each device, the serial number is written to a `txt`-file for unique identification. The traces are saved separately in a `.npy`-file. Each file consists of three arrays with corresponding key values: `trace`, `textin` and `keys`. The schematic of the transfer learning dataset is displayed in figure 4.2.

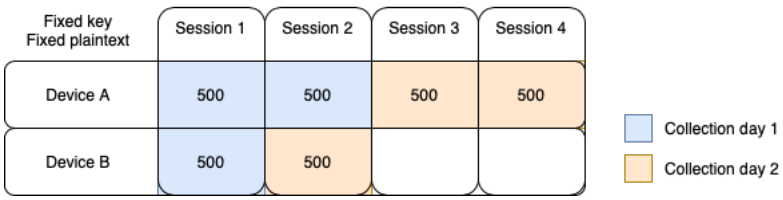


Figure 4.1: Dataset structure for the device analysis. The number indicates the amount of traces collected. Every trace is saved together with its plaintext and key.

	Profiling	Attacking														
	A (ground)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Data_fixed Fixed key Fixed plaintext	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500	500
Data Fixed key Variable plaintext	50k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k

Figure 4.2: Dataset for training the deep neural network. The numbers indicate the amount of traces collected. Every trace is saved together with its plaintext and key.

4.4 Design choices & neural network model

When using deep neural networks for data analysis, it is important to keep a grasp on what we are exactly modelling. In this case, the data being fed to the network are power traces with a variable plaintext and a fixed key for each device. As discussed in the preliminaries, the power traces consist of several components, as described in formula 2.1 defined by Mangard et al. In this research, identical devices are used which are running the same algorithm with the same input data. As the plaintext is variable, the variations captured by the neural network trained on our ground device would be the intra-device differences on $P_{data} + P_{el.noise}$. In this work, it is investigated if it is needed to close the portability gap between the profiling and attack device. In other words, would the base model be able to deal with inter-device differences caused by environmental influences and manufacturing processes?

As shown in the literature review, there have been investigations on which deep neural network architectures are beneficial for the analysis of power traces. In this research, the possibilities of transfer learning are investigated, thus for comparable research, the currently best-working deep neural network structure is used as the base model. That would be the convolutional neural network (CNN) model developed by Kim et al, and is displayed in figure 4.3.

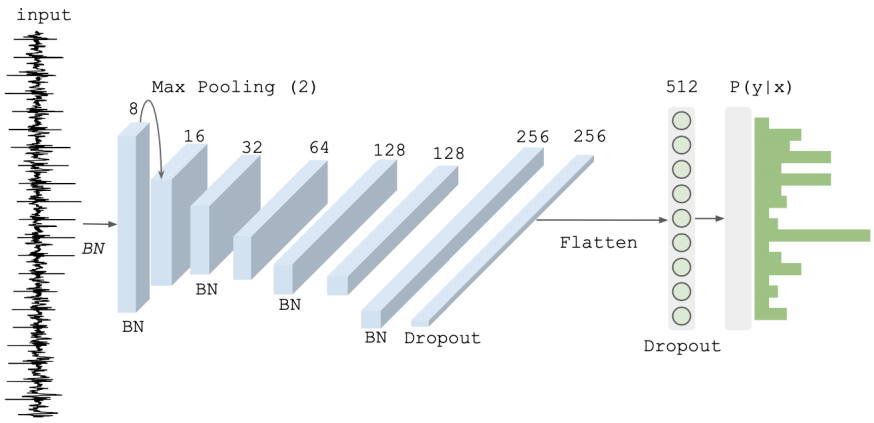


Figure 4.3: CNN model from Kim et al. used for this research [KPH+19]

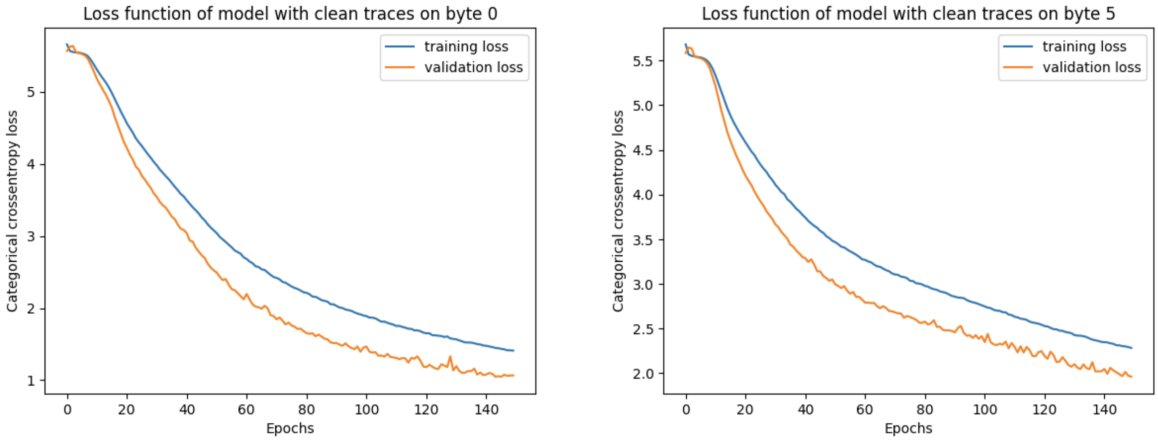
To apply transfer learning to this chosen base model, we need to determine which layers are frozen during finetuning. In the preliminaries, it is already briefly discussed how transfer learning can be applied to different types of datasets. In this case, the training dataset from device A is more or less similar to the test datasets from device B until O, as they are all identical devices. Therefore, in this research, it is chosen to freeze all layers after training the

base model. Then, the last fully connected classification layer will be removed and replaced with a new classification layer with random initialised weights.

In total, we trained four models based on the CNN by Kim et al. for different target bytes and noise levels:

1. Model trained on clean traces to attack byte 0
2. Model trained on clean traces to attack byte 5
3. Model trained on traces with simulated noise to attack byte 0
4. Model trained on traces with simulated noise to attack byte 5

For training the base CNN model on our dataset, a similar training setup is done compared to the model by Kim et al. This includes a categorical cross-entropy loss with the Adam optimiser and a learning rate of 0.0001. A validation split of 0.1 has been applied. The models are trained with the 50k traces collected from device A on a point of interest of index 1300 to 2000. In the latter two models, simulated Gaussian noise is added with a standard deviation of 0.01. The labels were calculated as $label = sbox[plaintext \oplus key]$. For the first two cases, the model was trained for 150 epochs. Figures 4.4a and 4.4b display the training and validation loss for target bytes 0 and 5. As can be seen from the plots, for a full convergence the network should have trained for more epochs, which was not possible due to limits in resources.



(a) Loss for byte 0 on the clean model

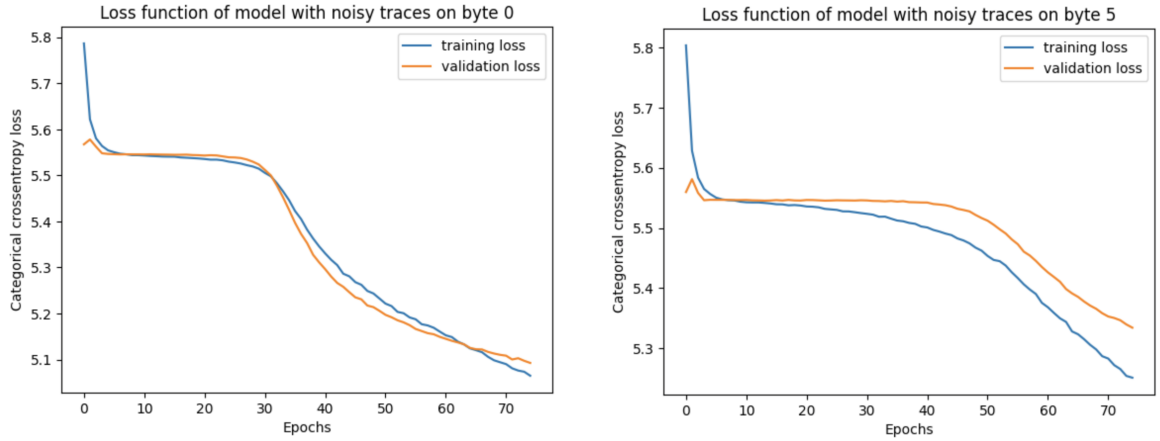
(b) Loss for byte 5 on the clean model

Figure 4.4: Training and validation loss for the models trained on the clean traces from ChipWhisperer. There is a minimal difference in training on byte 0 and byte 5. The models were almost fully converged.

The third and the fourth models were trained with 75 epochs. Their loss graphs are displayed in figure 4.5a and 4.5b. From these figures, it follows that the model was able to converge faster compared to the clean traces, and thus fewer epochs were needed. The number of epochs was chosen the same for both models for smooth comparison.

When comparing the transfer learning model to the base model, the following setup is used:

1. applying the trained base model directly across different identical devices using 1000 attack traces
2. finetuning the trained base model on 9000 attack traces from the target device, then attack using 1000 attack traces



(a) Loss for byte 0 on the noisy model trained

(b) Loss for byte 5 on the noisy model

Figure 4.5: Training and validation loss for the models trained on traces with simulated noise. The model converged better and with fewer epochs compared to the models in 4.4a and 4.4b. It was easier for the model to train on byte 0 compared to byte 5

In the case of fine-tuning the clean base model, a total of 30 epochs were used. When fine-tuning the base model trained on noisy traces, only 10 epochs were needed. When conducting this experiment, 10-fold cross-validation has been applied on the test set. All other parameters were the same as training. All target devices were attacked using clean traces and noisy traces. The full setup is displayed in figure 4.6.

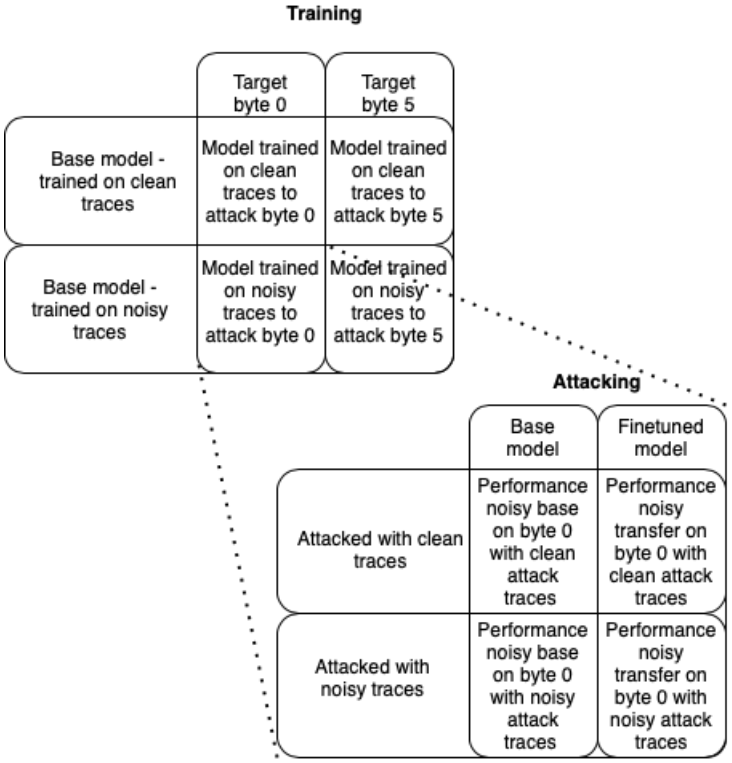


Figure 4.6: Experimental setup for training and attacking. Each attack is done on all target devices. This setup leads to 16 cases with each case having 15 attack devices.

Chapter 5

Investigation on portability of 32-bit devices

To make an educated guess on what information the neural network is modeling, a step-by-step device analysis is performed on the 15 identical ARM Cortex M4 devices. Figure 5.1 displays one trace of the ARM Cortex M4 running tiny AES-128. The maximum possible time samples (+-24000) allowed by ChipWhisperer are collected. To save storage space for the datasets, a maximum of 12000 time samples are collected for each dataset - indicated in red.

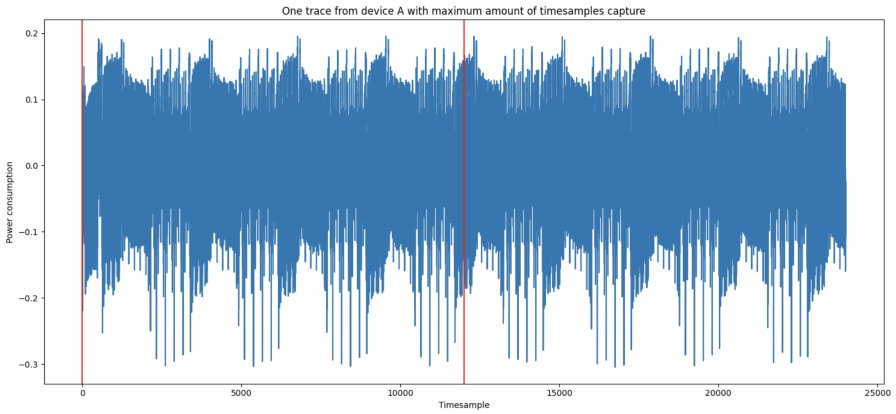


Figure 5.1: Trace collected from device A with maximum possible time samples. The parts indicated with red lines are collected for all the datasets to save storage space.

This chapter discusses several experiments to answer the posed research question. Section 5.1 gives an overview of how intra- and inter-device variations can behave. Section 5.2 is meant to determine the point of interest. This is relevant for the deep neural network, as it is undesirable to input the whole trace. Section 5.3 conducts a cross-device investigation to find similarities and differences across devices. The signal-to-noise ratio is used to determine the visibility of the signals for a certain byte. In section 5.4, the effect of transfer learning is compared for all different devices. Finally, in 5.5, Gaussian noise is added to the training data to test the robustness of the model. The effect of transfer learning is again compared for all different devices.

5.1 Intra- and inter-device variations

The first analysis is focussed on intra- and inter-device variations. Its goal is to investigate how intra- and inter-device variations behave. We did this by collecting traces from different devices

on the same day and from the same device but at different moments in time. The variations are modeled using the average power trace and the standard deviation per time sample.

5.1.1 Results and discussion

Conclusions. In this first experiment, we conclude from figure 5.3, 5.2, 5.4, 5.7, 5.5 and 5.6 that the inter-device variations are greater compared to the intra-variations of a device. From these figures, we also conclude that the inter-device variations reveal the pattern of the AES algorithm.
Analysis. As indicated in section 4.3, for each device, two types of datasets are collected. For this experiment, the first dataset on device analysis is used. In figure 5.2, the average traces of devices A and B on the first session are plotted together. The power consumption on the y-axis is the difference in power consumption compared to a device's baseline. As shown from the plot, device A consumes more power than device B.

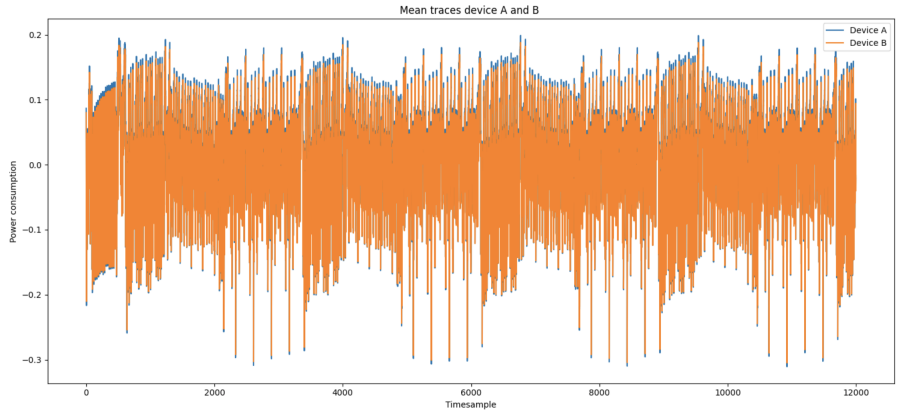


Figure 5.2: Average trace of devices A and B layered on top of each other. This figure shows that device A consumes more power compared to device B.

The differences are hardly visible when comparing the two sessions collected on the same day of device A and the two sessions collected on different days. This is plotted in figure 5.3.

The differences in these traces are calculated to make the variations more visible. In the first plot of figure 5.4, the average traces of device A sessions 1 and 2 are subtracted. In the second plot of figure 5.4, the average trace of device A session 1 and device B session 1 is subtracted. The first plot shows that the intra-device differences are small, concluded from the y-axis scale. The second plot shows that the inter-device differences are larger than the intra-device differences but are still relatively small. A remarkable thing about the second plot is that the pattern of the AES-trace shows. This tells us that the inter-device difference is larger at time samples with a higher amplitude than at time samples with a smaller amplitude.

The experiments above are repeated but with the standard deviation across time samples within a session as a metric. In figure 5.7, the standard deviation of device A on session 1 is plotted next to the standard deviation of device B on session 1. As can be seen, device A's standard deviation is greater than device B's standard deviation. There is also a standard deviation artifact on device A around time sample 700, which does not occur in device B. This plot shows the pattern of AES appearing, telling us that the standard deviation on higher amplitudes is greater than the standard deviation on the lower amplitudes. The pattern is stronger on device A compared to device B.

Figure 5.6 shows the standard deviation of device A across sessions 1 and 2, as well as sessions 1 and 3. The first thing to note is that the standard deviations are small. However, there is

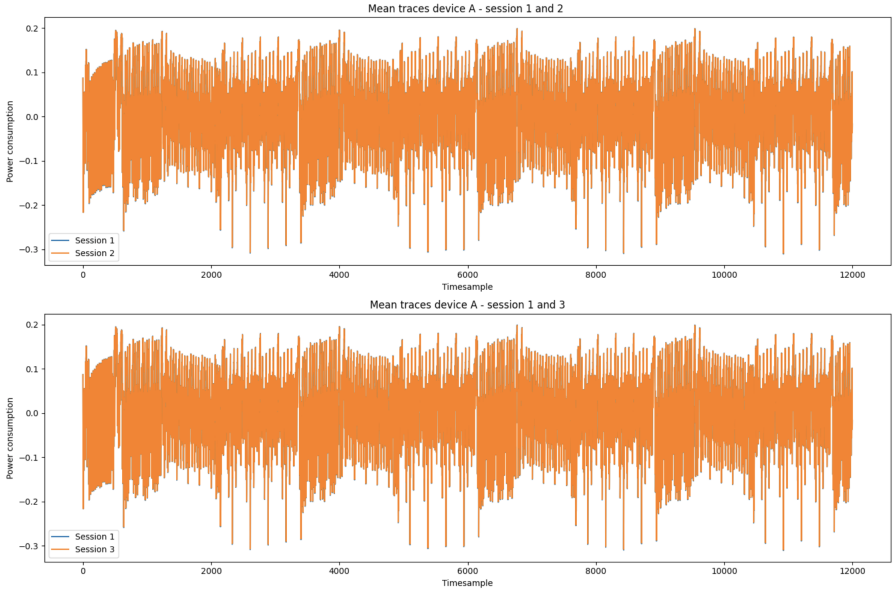


Figure 5.3: Average trace of different sessions from device A layered on top of each other. This plot shows that the intra-device variations are hardly visible, even when they are collected on a different day

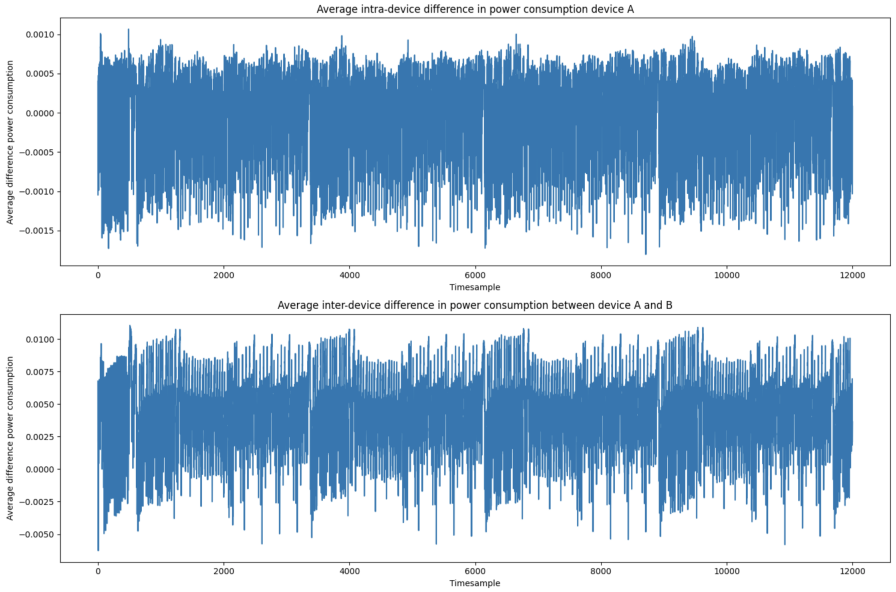


Figure 5.4: Average difference between device A and B and between sessions A1 and A2. This plot shows that the intra-device difference is very small. The inter-device difference reveals the AES pattern.

a visible difference in the standard deviation at different times. The difference in standard deviation on sessions 1 and 2 is smaller compared to sessions 1 and 3. Although the variations are small, an intra-device difference exists across different moments in time. The difference between sessions 1 and 2 are also displayed in the first plot of 5.7. The difference in standard deviation on device A's artifact is higher than the other time samples.

In the second plot of 5.7, the difference in standard deviation between the first sessions of

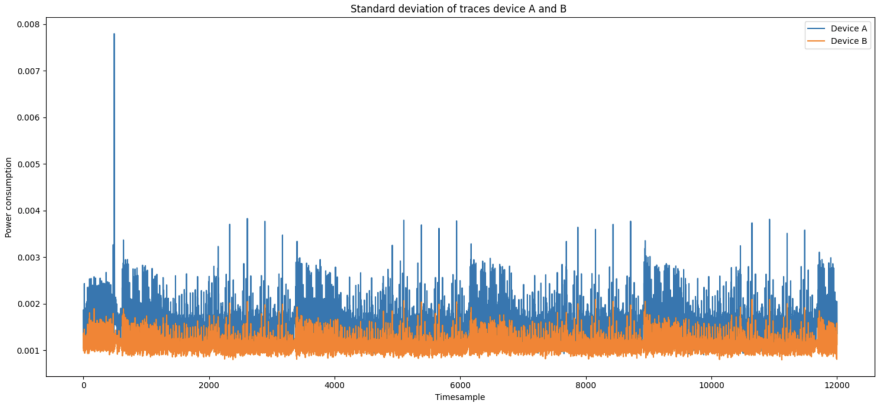


Figure 5.5: Standard deviation per time sample of devices A and B layered on each other. This plot shows that the standard deviation of device A is overall higher compared to B. Device A also has an artifact around time sample 700.

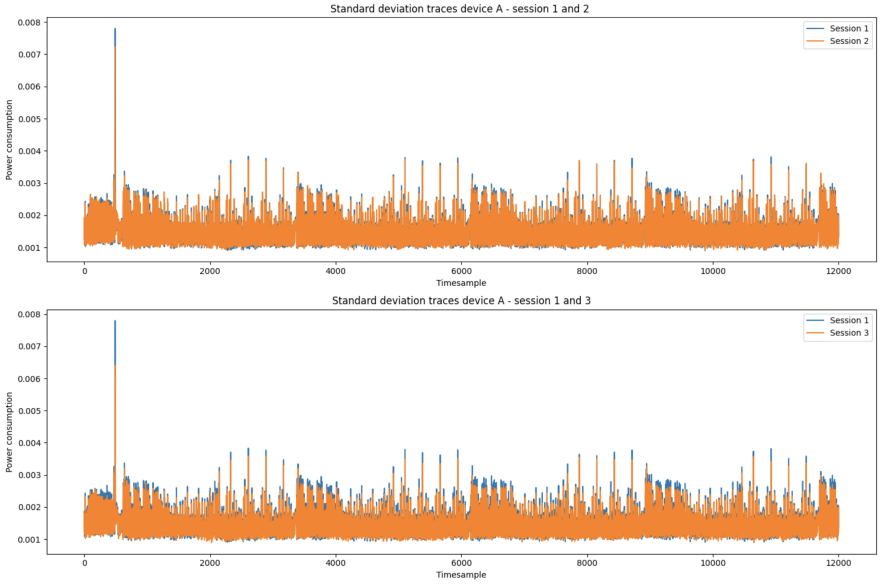


Figure 5.6: Standard deviation per time sample of different sessions from device A layered on each other. This plot shows that the difference in standard deviation is bigger when the sessions are collected on different days.

devices A and B are plotted, still showing the artifact of device A. The difference in standard deviation on the trace pattern is visible but somewhat less than the experiment using average traces.

5.2 Determining point of interest

Now that we know how devices differ during different moments in time and between devices, it is time to investigate which part of the trace is used for attacking. Therefore, this experiment indicates which time samples are used for attacking. The signal-to-noise ratio and correlation are used as a metric to determine the point of interest.

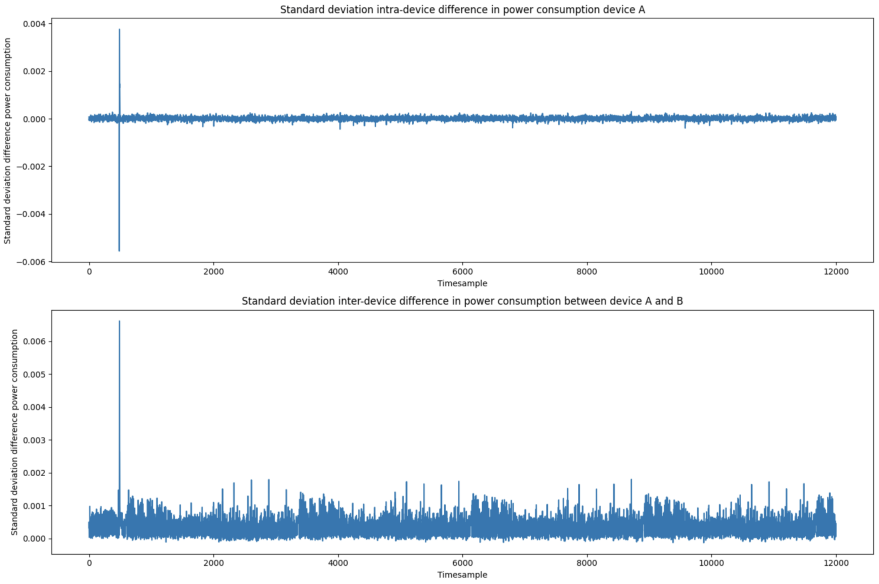


Figure 5.7: Difference in standard deviation between device A and B and between sessions A1 and A2. This figure shows that the intra-device difference in standard deviation is very small. Still, the artifact is present. When calculating the inter-device standard deviation, the AES pattern is revealed slightly.

5.2.1 Results and discussion

Conclusions. In this experiment, we conclude from figure 5.11, 5.9, 5.10 and 5.8 that the point of interest for this work - namely the S-box output from the first AES round - is between time-sample 1300 and 2000, and this window is the same for all target devices.

Analysis. The profiling dataset determines the point of interest as we train the neural network on our ground device, A. This dataset has 50k traces with variable plaintexts and fixed keys. For this work, the targeted bytes of the S-box are byte 0 and byte 5. To determine where the calculation of the S-box is taking place in the trace, the correlation is calculated between each time sample of the trace and the leakage model for byte 0 and byte 5. To investigate where the S-box calculation ends, byte 15 is also included in this investigation. For the rest of this work, byte 15 is not used as a target. The ranking of the first three most probable time samples representing the output of S-box on byte 0 and byte 5 are calculated.

Figure 5.8 shows that the time sample with the highest correlation for byte 0 is 1326, and for byte 5 is 1541 for ground device A. The time sample where the S-box calculates the final byte 15, is at sample 1973. In figure 5.11, the leakage plots are displayed for all three bytes. The base model planning to use takes as input 700 time samples. Using this information, we selected a time sample of 1300 until 2000 as input for the deep neural network. Figure 5.12 shows the point of interest for all upcoming investigations.

```
The timesample with the highest correlation for output sbox byte 0 is: 1326
The timesample with the highest correlation for output sbox byte 5 is: 1541
The timesample with the highest correlation for output sbox byte 15 is: 1973
```

Figure 5.8: Timesamples with the highest correlation for a certain byte. This figure shows that the window of the first round of AES concerning the output of the S-box is between time sample 1300 and 2000

Highest correlation for the point of interest on byte 0 for each test device

Correlating timesample	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Highest correlation	1326	1326	1325	1326	1326	1326	1326	1326	1326	1326	1326	1326	1326	1326	1326
Second highest correlation	1325	1327	1326	1327	1325	1325	1325	1327	1325	1327	1327	1327	1327	1325	1325
Third highest correlation	1327	1325	1327	1325	1327	1327	1327	1325	1327	1325	1325	1325	1325	1327	1327

Figure 5.9: Timesample ranks for byte 0. In this figure, only device C has a slightly different time sample for the highest correlation.

Highest correlation for the point of interest on byte 5 for each test device

Correlating timesample	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Highest correlation	1541	1541	1541	1541	1541	1541	1541	1541	1541	1541	1541	1541	1541	1541	1541
Second highest correlation	1542	1542	1542	1542	1542	1542	1542	1542	1542	1542	1542	1542	1542	1542	1542
Third highest correlation	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543	1543

Figure 5.10: Timesample ranks for byte 5. This figure shows that for all target devices, the same time sample has the highest correlation on byte 5.

As it is not sure if the point of interest for all the test devices is the same, a ranking has been made for the test devices on which time sample correlates the most with byte 0 and byte 5. This is displayed in figure 5.9 for byte 0 and figure 5.10 for byte 5. As shown in figure 5.9, only device C has a different time sample with the highest correlation for byte 0 compared to the ground device. However, the second highest correlation does equal the same time sample as the ground device. In figure 5.10, it is shown that each test device has the same highest correlating time sample as the ground device for byte 5. From this information, we can conclude that the neural network model will most likely not need to model the variation of the point of interest across devices, as they are mostly the same.

5.3 Investigate test devices

A comparison investigation is made across the test devices to determine how the test devices differ fundamentally from the ground device A. The goal of this experiment is to visualize the difference of $P_{el.noise} + P_{const}$ in identical devices on a byte level of 10-time samples and the 1300-2000 time samples window, as shown in figure 5.12. Another goal is to investigate the strength of the signal concerning the noise by using the signal-to-noise ratio for each test device. The dataset containing the profiling and attacking traces is used for this investigation.

5.3.1 Results and discussion

Conclusions. From this experiment, we conclude from 5.13a that device E correlates the least with our ground device A. Figure 5.13b visualizes that devices M and H show the biggest difference. From figure 5.14, we see that the standard deviation of device K is the highest. From these plots, we conclude that the overall inter-device variations are small. Figures 5.15 and A show that the signal-to-noise ratio for byte 0 and byte 5 are very high. Therefore, we conclude that the traces are relatively clean. When we added the Gaussian noise to the traces, the signal-to-noise ratio became a factor 10 smaller. The figures in A show that the signal for byte 0 is still stronger than the noise, but for byte 5, the signal is as strong as the noise. This behavior is the same for all target devices.

Analysis. The dataset with 500 traces for each target device with fixed plaintext and keys is used for this comparison. The differences between devices are measured using correlation and

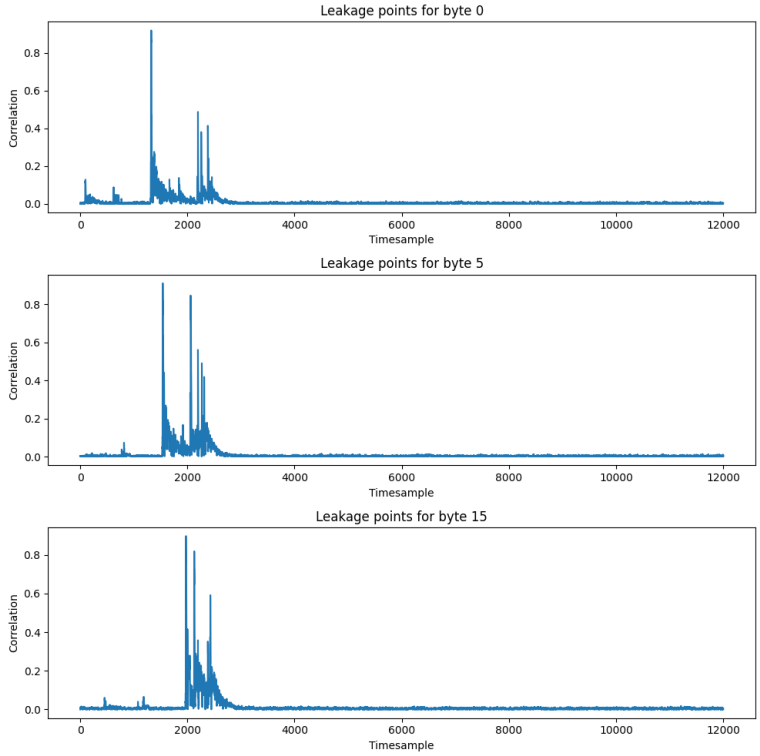


Figure 5.11: Leakage plots for each byte. This plot shows a visible leakage and an obvious correlation for byte 0. On byte 5, there is another spot as well, but this is not in the top-3 highest correlation as concluded from 5.10. Byte 15 shows where the first round of `SubBytes` ends.

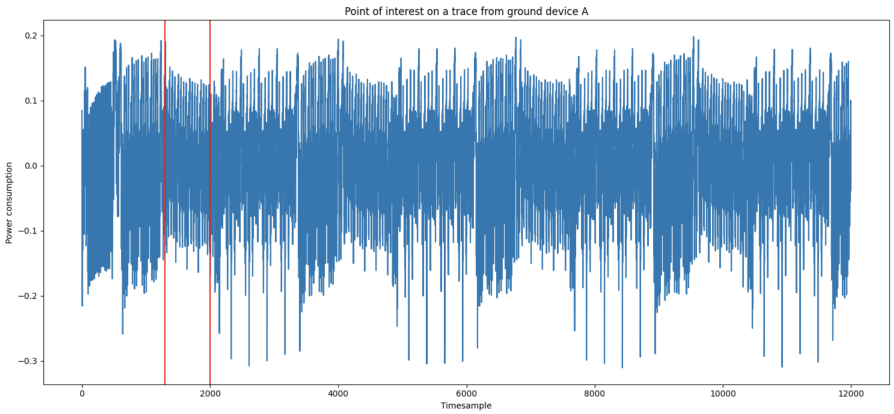
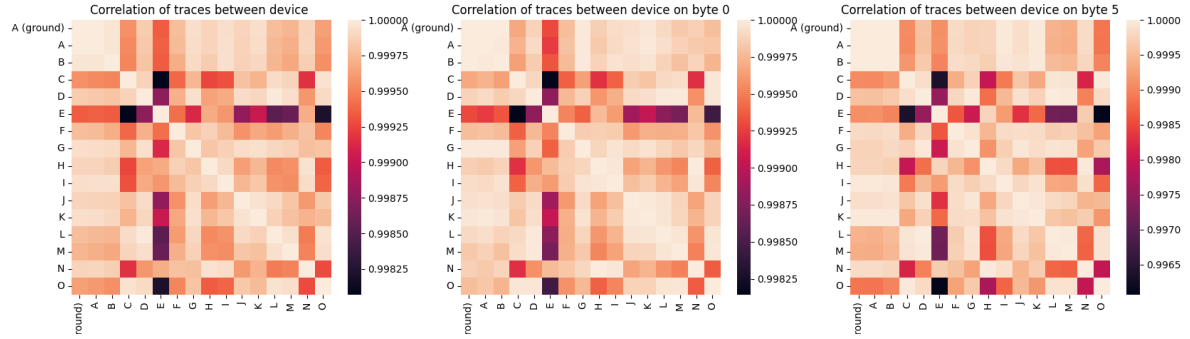
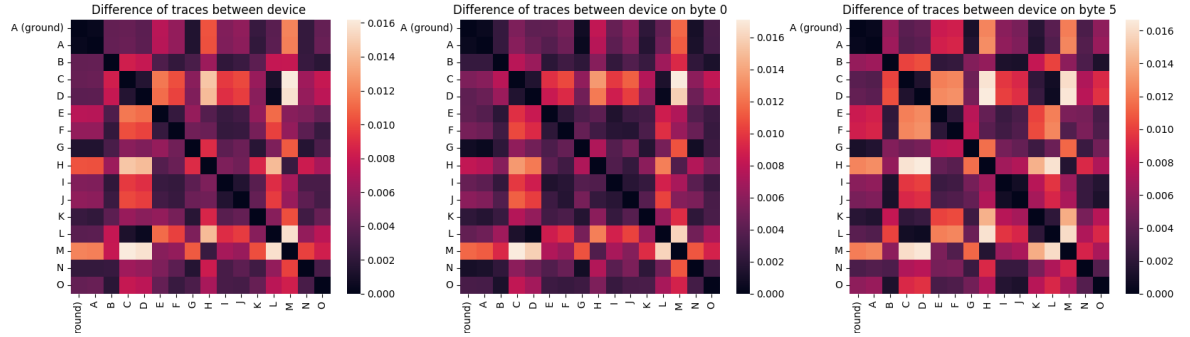


Figure 5.12: Point of interest for all upcoming investigations. This is the part of the trace where the first round of AES performs the `SubBytes` operation.

difference. Additionally, for each device, the standard deviation is investigated. The heatmaps with inter-device correlations and differences are shown in 5.13a and 5.13b. As expected, the correlation between the ground device A and the test device A is 1, where the difference is 0. This is because we use the same device but with the data collected in different sessions. The inter-session difference is not visible on this level. From the axis of the plots, it can be seen that the differences are minimal. Device E correlates the least with ground device A from all our



(a) Heatmap of correlation between test devices. This plot shows that device E correlates the least with our ground device A.



(b) Heatmap of difference between test devices. This plot shows that devices M and H have the biggest difference from our ground device A. The differences become stronger at byte 5.

Figure 5.13: Heatmaps for test device investigation. There does not seem to be a relationship between trace correlation and trace difference.

test devices compared to the other devices. Another highlight is that device E generally stands out compared to the other devices. When looking at the difference between devices, devices M and H show the biggest gap compared to device A. In this case, device M stands out the most compared to other test devices. The variations between devices become stronger when looking at the byte level, especially on byte 5.

For each device, the standard deviation is calculated from the average trace on the point of interest. This is displayed in figure 5.14. As can be seen, the standard deviation on device K on the overall window is the highest, as well as on byte 0. On byte 5, the standard deviation on device E is the highest. The standard deviation does get lower when zooming in on a byte level compared to the whole point-of-interest window.

As indicated earlier, the inter-device variations on identical devices are relatively small. To investigate the effect of noise on the model's performance, simulated noise from a Gaussian distribution with a standard deviation of 0.01 is added. This value has been chosen by inspecting the plots on 5.14, where the highest standard deviation is 0.08, and the lowest standard deviation is 0.0375. On top of that, this value still results in a signal-to-noise ratio around 1 or higher, such that the deep neural network can recognize the signal properly.

This is displayed in 5.15 for devices A and B. On the right side, a zoomed-in plot is shown on its behaviour with added noise. These plots are displayed for all test devices in the appendix A.

From these plots, it can be concluded that the traces collected from ChipWhisperer are very clean, as the signal-to-noise ratio is high. The signal for byte 0 is more prominent than that for

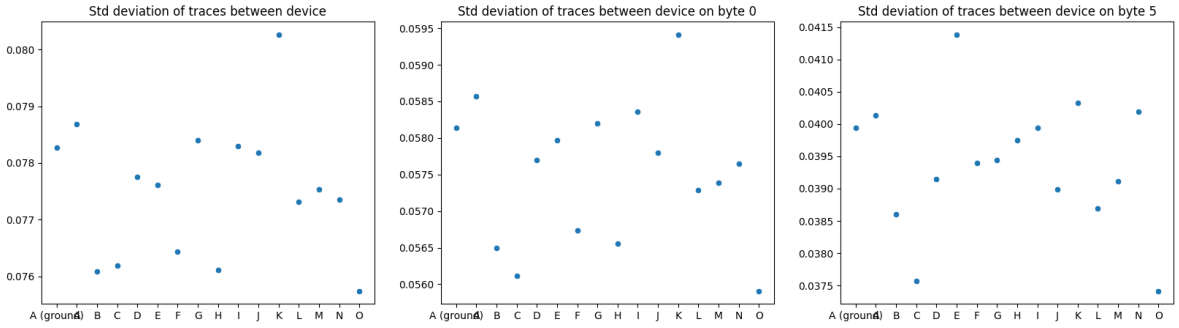


Figure 5.14: Standard deviation for each device. Overall, device M has the highest standard deviation. At byte 5, device E has the highest standard deviation.

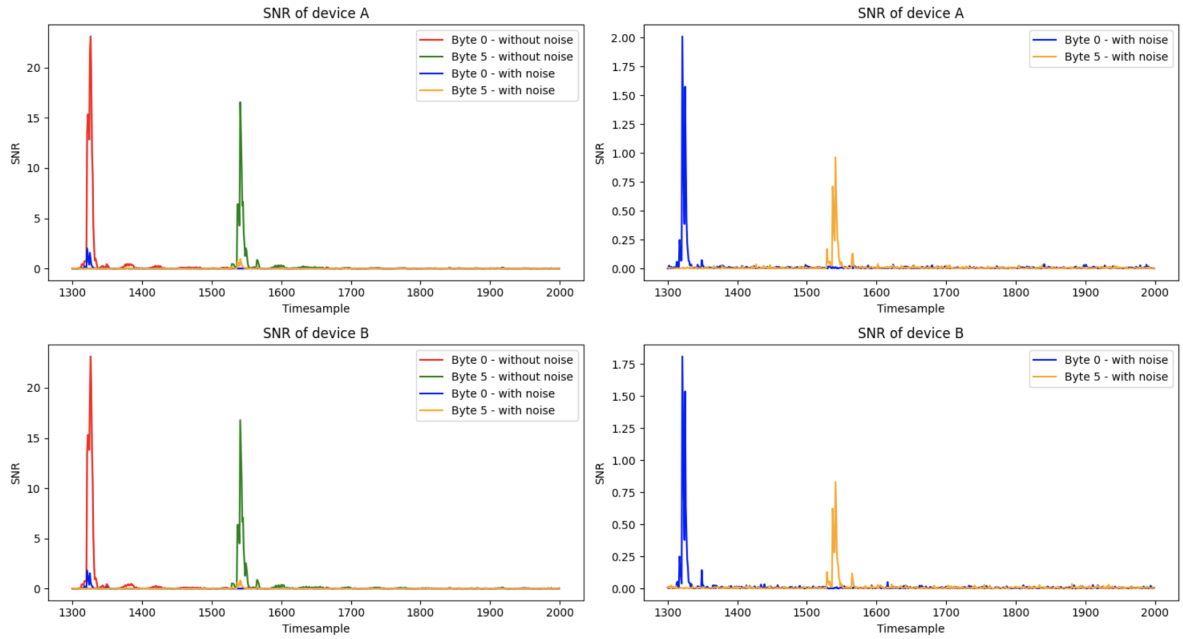


Figure 5.15: Signal-to-Noise ratio for devices A and B. The left plots show that the signal-to-noise ratio without added noise is high, where byte 0 is stronger than byte 5. With the added Gaussian noise, the signal-to-noise ratio is between 1.75 and 2 for byte 0 and around 1.00 for byte 5. Appendix A shows similar behavior for all test devices.

byte 5. When adding the noise, the signal-to-noise ratio decreased by 10. For byte 0, the signal is still two times stronger than the noise. For byte 5, the signal is almost as equal to the noise. For all test devices, similar behaviour is observed.

5.4 DL-SCA attack with transfer learning on a clean base model

This experiment investigates the influence of inter-device variations when applying a cross-device deep learning attack. This experiment's goal is threefold: first, investigate the difference between using the base model and the transfer model to investigate the extent of the portability problem on identical devices. Secondly, investigate if accuracy is different by attacking byte 0 or byte 5. Thirdly, investigate how noisy attack traces influence the success of the attack.

5.4.1 Results and discussion

Conclusions. From figure 5.16, we conclude there is no portability issue when using identical devices for a cross-device attack with clean traces. Only devices E and H were slightly harder to attack with target byte 5 using the base model. Figure 5.17 shows that using noisy attack traces influences the base model's performance. We also saw from figure 5.17 that applying the transfer learning model using noisy attack traces does not improve the attack performance. Using the transfer learning model, there is a visible difference between attacking byte 0 and byte 5. For an attacker, it usually took more traces for a successful attack on byte 5 compared to byte 0. From figure 5.17, we conclude that it was more difficult to successfully attack byte 0 on device M. Finally, using the transfer learning model, it was harder to perform a successful attack with noisy attack traces on byte 5 for the devices H and M.

Analysis. As seen from 5.16, it was possible to do a cross-device attack using both the trained base model and the transfer learning model when using the clean traces received from Chip-Whisperer. A small difference is observed when attacking byte 5 on devices E and H. This is as expected, as these devices were the least similar to our ground device. Other than that, it can be concluded that there is no portability issue when the inter-device variations are small. The base model can be applied directly across different identical devices.

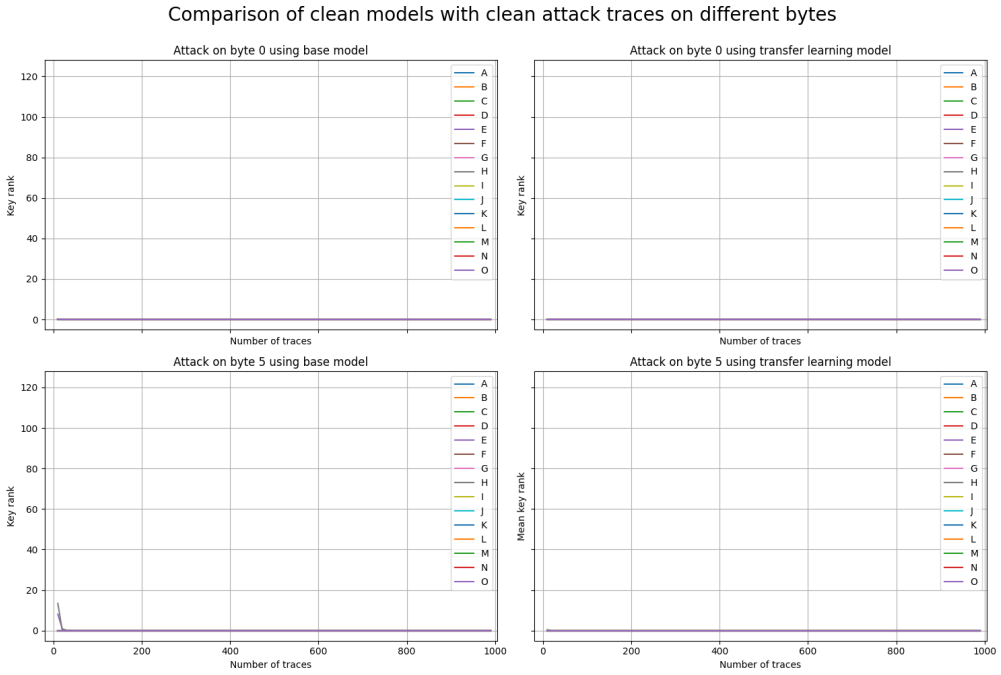


Figure 5.16: Performance of clean base model compared to transfer learning model on byte 0 and byte 5 with clean attack traces. This plot shows no portability issue when performing an identical cross-device attack. The base model and transfer learning model performed equally well on both byte 0 and byte 5.

The same experiment was repeated to challenge the model by feeding the model noisy attack traces. This is displayed in figure 5.17. It can be concluded that it is possible to do a cross-device attack with noisy attack traces using the base model. Therefore, no portability issue arises here. However, the transfer learning model did not perform as well. Only with a sufficient amount of noisy attack traces is it possible to perform a successful attack. It can be seen that devices H and M where harder to attack using the transfer learning model.

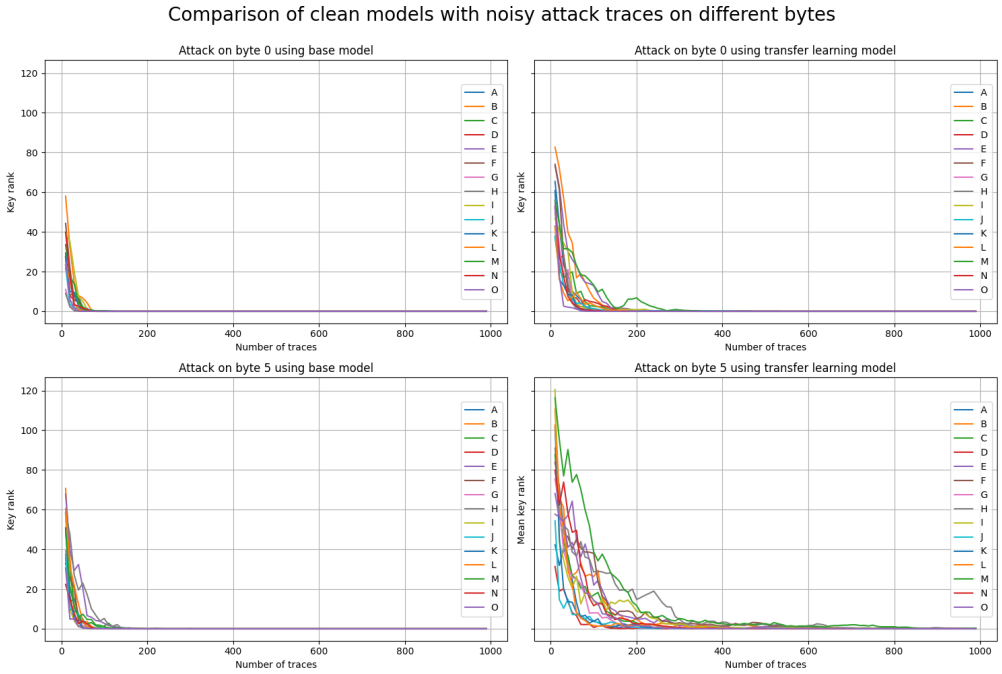


Figure 5.17: Performance of clean base model compared to transfer learning model on byte 0 and byte 5 with noisy attack traces. This plot shows it was better to perform the cross-device attack using the base model on both target bytes.

From this, it can be concluded that when the inter-device variations are small, applying transfer learning is unnecessary to enhance the model's performance. It is even better not to apply transfer learning when the attack traces are noisier than the base model. In the appendix B, the plots for each device are displayed separately for clarity.

5.5 DL-SCA attack with transfer learning on a noisy base model

In the final experiment, we investigate using a model trained on noisy traces. The goal of this experiment is threefold: first, investigate the difference between using the base model and the transfer model to investigate the extent of the portability problem on identical devices when noise is added to the training data. Second, investigate if there is a difference in accuracy by attacking byte 0 or byte 5. Third, investigate how noisy traces influence attack accuracy.

5.5.1 Results and discussion

Conclusions. From figure 5.18, we conclude there is no portability issue when performing a cross-device attack with clean attack traces. This figure also shows that attacking with the finetuned model is equally good on byte 0. This figure shows that devices M and H are harder to attack.

Figure 5.19 shows that the use of noisy attack traces does not influence the base model's performance. This figure also shows that transfer learning did not enhance the model's performance when using noisy attack traces. There is a visible difference in performance on attacking byte 0 compared to attacking byte 5.

Analysis. In figure 5.18, the key rank is displayed when the noisy base model is used. For byte 0, there was not a big difference between the base model's performance compared to the transfer

learning model. The device considered slightly harder to attack was device M.

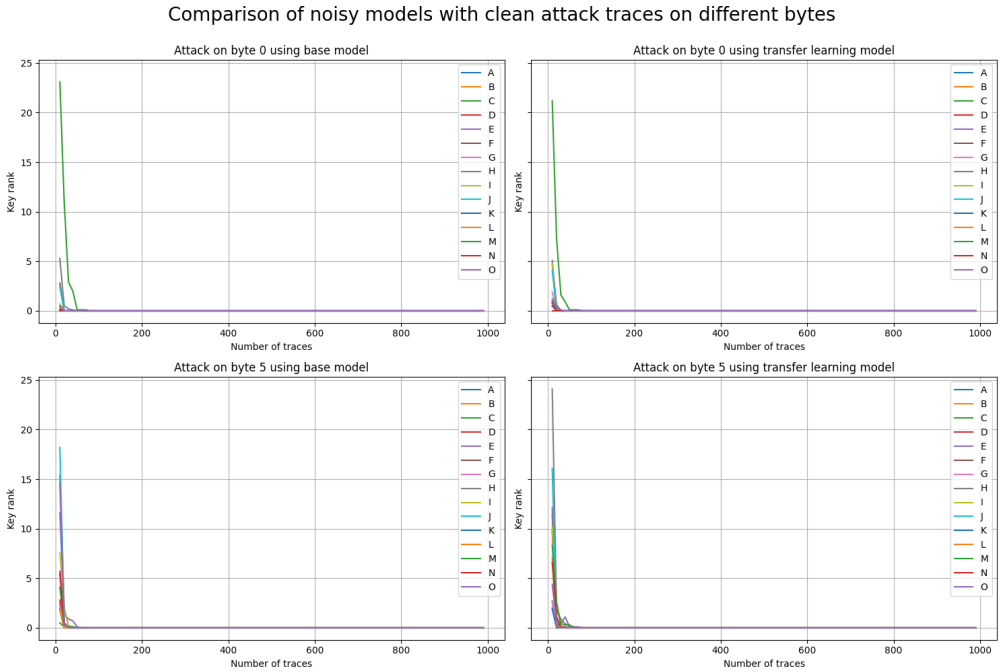


Figure 5.18: Performance of noisy base model compared to transfer learning model on byte 0 and byte 5 with clean attack traces. This plot shows no portability problem when performing a cross-device attack using the base and transfer learning models trained on added noise.

Adding noise to the attack traces hardly influences the base model's performance, as shown in 5.19. It also did not improve the performance of the transfer learning model. Both target bytes were harder to attack when noise was added. The separate plots of figure 5.18 and 5.19 are displayed in the appendix C.

From this experiment, it can be concluded that adding noise to the data does influence the network's performance. The model's attack byte 0 and byte 5 were different, and attacking with noisy traces did not improve performance compared to attacking with clean traces.

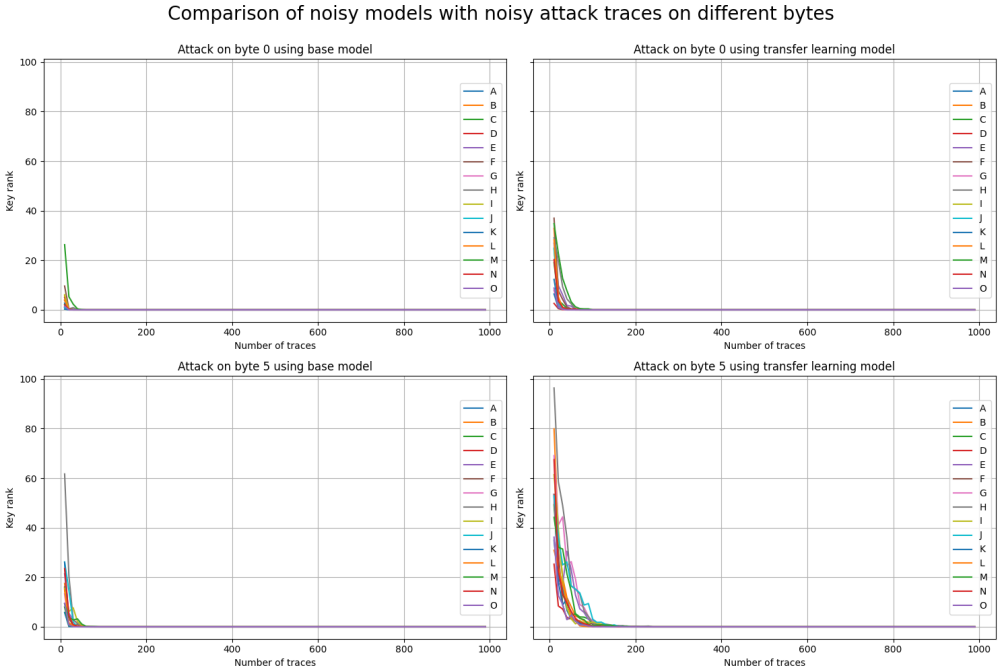


Figure 5.19: Performance of noisy base model compared to transfer learning model on byte 0 and byte 5 with noisy attack traces. This plot shows no portability problem on the base model performing a cross-device attack. When transfer learning was applied, the performance decreased. Transfer learning did perform better on the Gaussian-trained model compared to the model in 5.17.

Chapter 6

Discussion and future work

The device analysis shows that the inter-device variations are relatively small when using identical devices. The data collected using ChipWhisperer yielded a high signal-to-noise ratio. The analysis also showed that point-of-interest is obvious from the data and equal for all target devices. Thus, the deep neural network did not have to account for this. As there is little variation within all target devices concerning correlations, differences, standard deviations, signal-to-noise ratio, and time samples of interest, the network would probably perform well in classifying the attack traces across devices. This was the case: a well-trained base model outperformed the fine-tuned transfer-learning model. Even for a well-trained base model tested on attack data with simulated noise, it was still possible to perform a successful attack. Finally, we also saw that the least similar devices compared to our ground device were harder to attack compared to more similar devices. Adding transfer learning did not account for these variations.

Transfer learning also showed a difference in performance at the attack bytes when noise was added to the attack traces. This was already seen in the signal-to-noise ratio, where byte 5 has a weaker ratio than byte 0. From the experiments, it is obvious that having clean attack traces is more influenced than the target byte or the model used. When clean attack traces were applied, it was possible to perform a successful attack.

It is also shown that adding Gaussian noise adds to the network's generalisation, both training as the finetuning required fewer epochs in the case of a model trained on traces with simulated noise. The main difference compared to the added noise by Kim et al. [KPH+19] is that the noise is added as a tensor on a specific layer while learning.

This work shows that applying transfer learning does not improve the model's performance compared to identical devices. A well-trained base model for an identical cross-device attack outperformed a fine-tuned transfer learning model. There can be several reasons for this result: firstly, the portability problem for identical devices is not as problematic as for other devices. Secondly, the assumption that the feature extraction stays the same across devices and the classification needs to be fine-tuned is wrong. This is also what was found in the research from Thapar et al. [TAM20], where they chose to freeze the classification layer and fine-tune the convolutional layers. However, they had a pretrained network from a homogeneous device. Another reason for this result is that the fine-tuning for the transfer learning model has not converged yet. The fine-tuning was done in a batch due to the size of the study. Therefore, there were no individual tweaks for each device to make sure the model was fully converged. In work by Genevy-Metat et al., [GMGH20], they found that retraining and fine-tuning the last layer yielded similar results, again suggesting that transfer learning is not applicable in this domain when using a convolutional neural network. The result from Genevy-Metat et al. is in line with what we found in this work. Adding to the findings of Genevy-Metat et al, it is possible to predict model performance by doing a device analysis on the clone devices.

For future work, it is possible to investigate the applications of transfer learning for homogenous or heterogenous devices. The literature shows that the portability problem can still play a role in different kinds of cross-device attacks. Another option is to perform a similar study compared to this but using variable keys for each trace. Usually, variable keys are considered harder compared to fixed keys. Another option for future work is to use a protected implementation of tinyAES-128. In this situation, transfer learning can still be interesting as the deep learning model must also model information on the used countermeasures.

Chapter 7

Conclusions

This work investigated how transfer learning compares to a regular deep neural network for a successful side-channel attack on 32-bit devices. We investigated this for clean traces and traces with simulated noise.

From the device analysis and deep learning attack, we can conclude that the convolutional neural network can be applied for a cross-device attack regardless of the inter-device differences of these identical devices. In almost all cases, the base and transfer learning models lead both to a successful attack. The transfer learning model did worse when the attack traces were noisy.

The results of this work show that the portability problem is limited at the level of identical devices. A cross-device analysis beforehand does indicate the performance of the side-channel attack. This work also showed that a well-trained model can outperform a transfer-learning model. Adding noise to the training data contributes to the generalisation of the model. Finally, clean attack traces are important for a successful attack in both model scenarios.

We suggest investigating the relationship between the profiling and attacking device first. In the future, this could lead to making an educated guess on the type of model and parameters to use for a certain problem. As indicated by other literature, there is no silver bullet for solving all side-channel problems. However, picking the right bullet for the correct goal might reduce obstacles.

Bibliography

[ABPP22] Vipul Arora, Ileana Buhan, Guilherme Perin, and Stjepan Picek. A Tale of Two Boards: On the Influence of Microarchitecture on Side-Channel Leakage. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 13173 LNCS:80–96, 2022.

[Ala19] Cagdas Hakan Aladag. Architecture Selection in Neural Networks by Statistical and Machine Learning. *Oriental journal of computer science and technology*, 12(Issue 3):76–89, 2019.

[ARM] ARM. ARM ISA definition.

[ARM10] ARM. Cortex M4: Reference Manual. pages 11–13, 2010.

[AZH+21] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. *Review of deep learning: concepts, CNN architectures, challenges, applications, future directions*, volume 8. Springer International Publishing, 2021.

[BB22] Lejla Batina and Ileana Buhan. Side-channel analysis : Differential Power Analysis and Countermeasures Physical Attacks on Secure Systems , Spring 2022, 2022.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the Portability: A Warriors Guide through Realistic Profiled Side-channel Analysis. 2020.

[Bis94] Chris M Bishop. Neural networks and their applications. *Review of Scientific Instruments*, 65(6):1803–1832, 6 1994.

[BLR13] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications*, pages 263–276, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Boz20] Stevo Bozinovski. Reminder of the First Paper on Transfer Learning in Neural Networks, 1976. *Informatica*, 44(3):291–302, 9 2020.

[BPS+18] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database-Long Paper. *Journal of Cryptographic Engineering*, pages 1–46, 2018.

[Buh] Ileana Buhan. Computing the Signal-to-Noise Ratio (SNR) for SCA.

[Chi] ChipWhisperer (NewAE Technology). TinyAES-128 Firmware.

[CRR02] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template Attacks. *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, 2002.

[Cyb89] G Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[CZLG21] Pei Cao, Chi Zhang, Xiangjun Lu, and Dawu Gu. Cross-device profiled side-channel attack with unsupervised domain adaptation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):27–56, 2021.

[Dae20] Joan (Radboud University) Daemen. Block Ciphers, 2020.

[DGD+19] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-DeepSCA: Cross-device deep learning side channel attack. *Proceedings - Design Automation Conference*, 1, 2019.

[GMGH20] Christophe Genevey-Metat, Benoît Gérard, and Annelie Heuser. On What to Learn: Train or Adapt a Deeply Learned Profile? *Cryptology ePrint Archive*, (Report 2020/952):1–19, 2020.

[Ham93] D Hammerstrom. Working with neural networks. *IEEE Spectrum*, 30(7):46–53, 7 1993.

[HGDM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: A first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide. In Werner Schindler and Sorin A Huss, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 249–264, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Ibr] Mohammed H Ibrahim. ARM processors. Technical report.

[Ins] Identity Management Institute. LAYERED SECURITY MODEL.

[JIM20] HAFEEZ JIMOH. How Transfer Learning works, 2020.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[KLN18] Taejun Kim, Jongpil Lee, and Juhan Nam. Sample-Level CNN Architectures for Music Auto-Tagging Using Raw Waveforms. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018-April:366–370, 2018.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):148–179, 2019.

[KR92] M N Karim and S L Rivera. Comparison of feed-forward and recurrent neural networks for bioprocess state estimation. *Computers & Chemical Engineering*, 16:S369–S377, 1992.

[KU17] V. A Kulkarni and G. R Udupi. Instruction Level Power Consumption Estimation – Issues and Review. *Journal of Multidisciplinary Engineering Science and Technology*, 4(2):6776 – 6781, 2017.

[LBD+89] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In D Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: An approach based on machine learning. *International Journal of Applied Cryptography*, 3(2):97–115, 2014.

[LCC08] Thanh Ha Le, Cécile Canovas, and Jessy Clédière. An overview of side channel analysis attacks. *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS '08*, pages 33–43, 2008.

[LPMS18] Liran Lerman, Romain Poussier, Olivier Markowitch, and François Xavier Standaert. Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: extended version. *Journal of Cryptographic Engineering*, 8(4):301–313, 2018.

[PGA+23] Kostas Papagiannopoulos, Ognjen Glamočanin, Melissa Azouaoui, Dorian Ros, Francesco Regazzoni, and Mirjana Stojilović. The Side-channel Metrics Cheat Sheet. *ACM Computing Surveys*, 55(10), 2023.

[PH90] David A Patterson and John L Hennessy. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[PPM+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep Learning-based Physical Side-channel Analysis. *ACM Computing Surveys*, 55(11), 2023.

[PR19] Van Hiep Phung and Eun Joo Rhee. A High-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences (Switzerland)*, 9(21), 2019.

[RD20] Mark Randolph and William Diehl. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptography*, 4(2):1–33, 2020.

[RDJ+01] Atri Rudra, Pradeep Dubey, Charanjit Jutla, Vijay Kumar, Josyula Rao, Pankaj Rohatgi, Çetin Koç, David Naccache, and Christof Paar. Cryptographic Hardware and Embedded Systems — CHES 2001. 2162(September 2001):171–184, 2001.

[Sar23] Smruti R Sarangi. Basic Computer Architecture Version 2.2. 2023.

[ST] ST. Arm® Cortex®-M4 in a nutshell.

[TAM20] Dhruv Thapar, Manaar Alam, and Debdeep Mukhopadhyay. TranSCA: Cross-Family Profiled Side-Channel Attacks using Transfer Learning on Deep Neural Networks. 2020.

[YSPJ21] Honggang Yu, Haoqi Shan, Maximillian Panoff, and Yier Jin. Cross-Device Profiled Side-Channel Attacks using Meta-Transfer Learning. *Proceedings - Design Automation Conference*, 2021-Decem:703–708, 2021.

[ZSX+20] Fan Zhang, Bin Shao, Guorui Xu, Bolin Yang, Ziqi Yang, Zhan Qin, Kui Ren, and Ziqi Yang. From homogeneous to heterogeneous: Leveraging deep learning based power analysis across devices. *Proceedings - Design Automation Conference*, 2020-July, 2020.

Appendix A

Signal to noise ratio

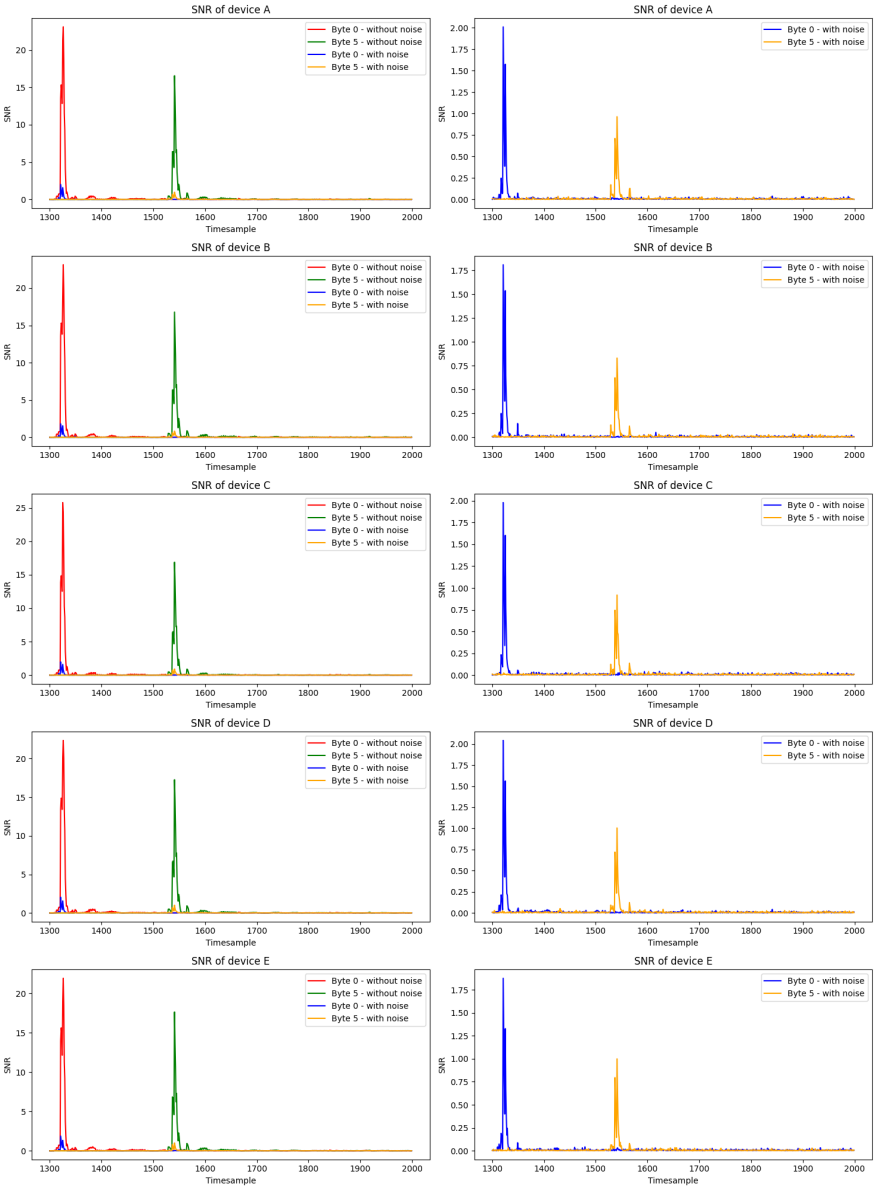


Figure A.1: Signal-to-noise ratio for all test devices - device A-E

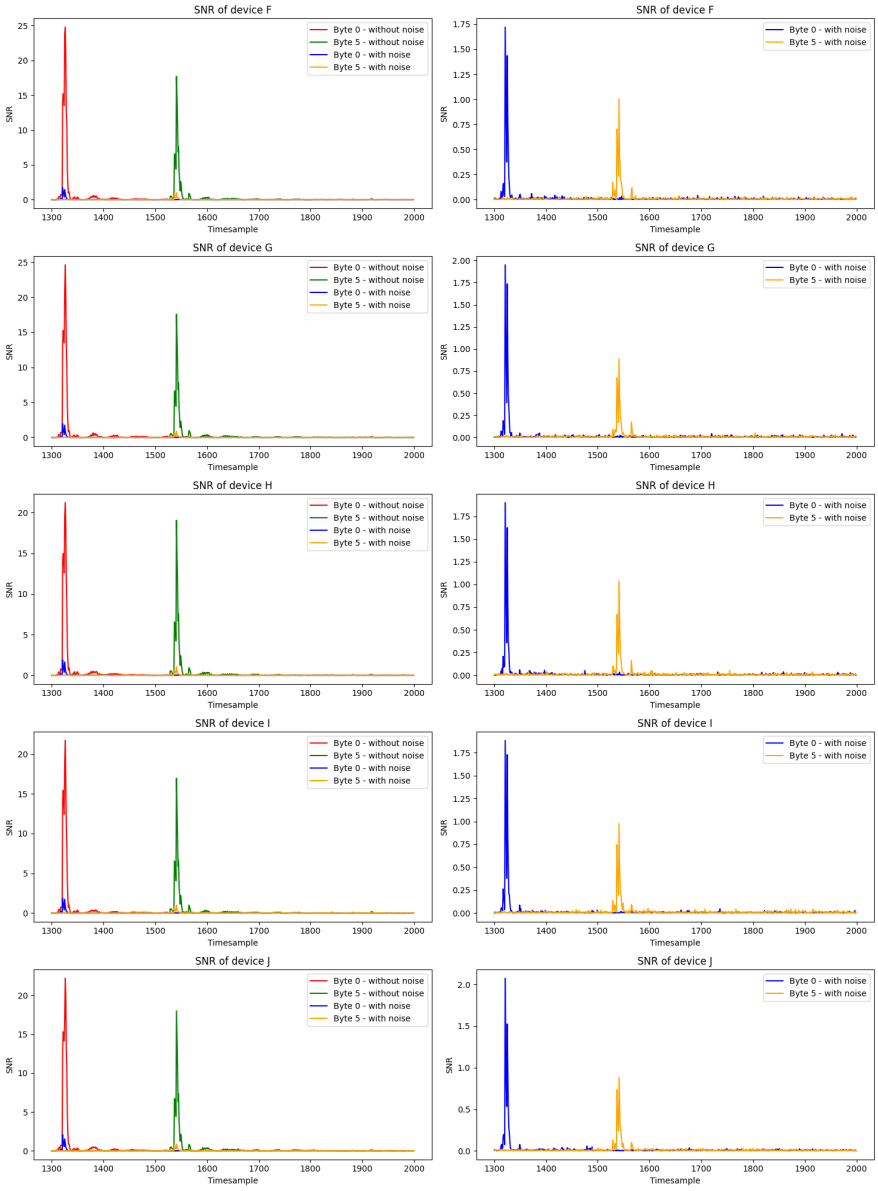


Figure A.2: Signal-to-noise ratio for all test devices - device F-J

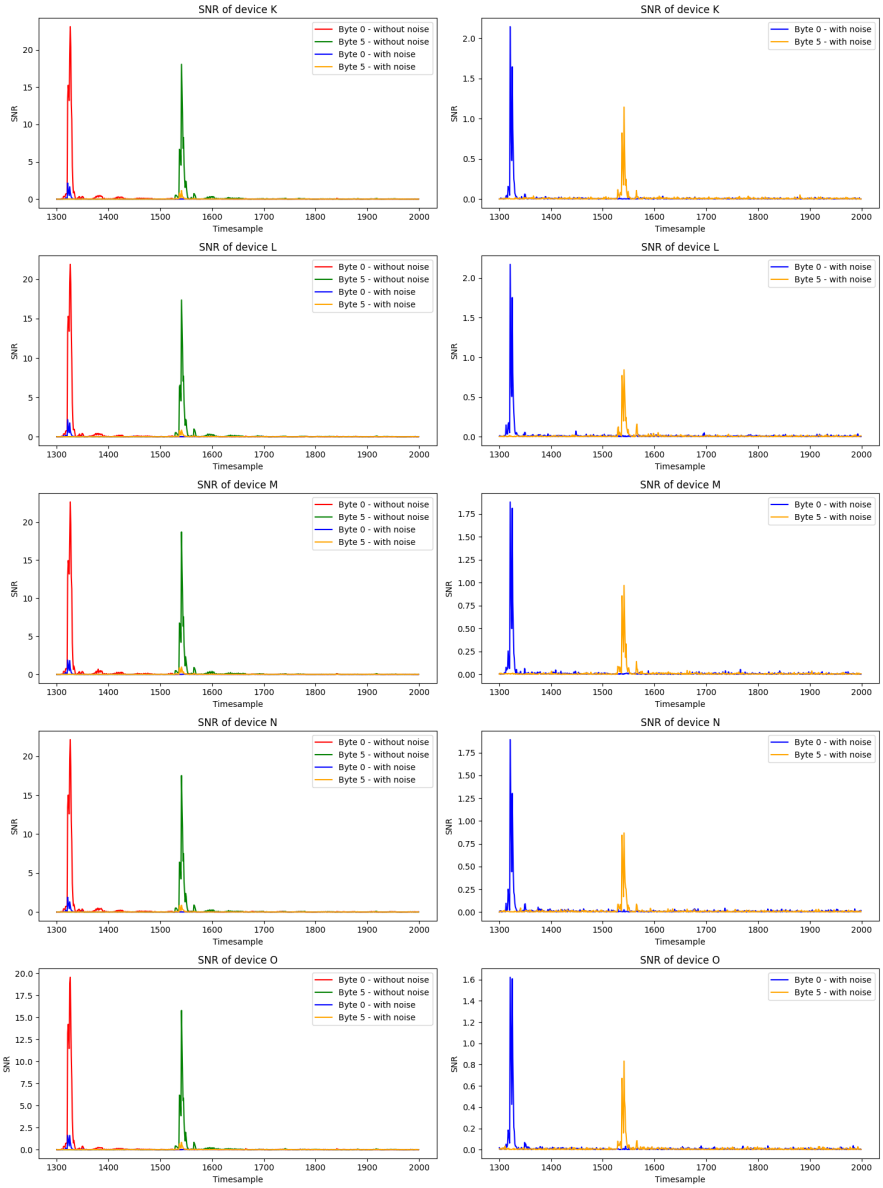
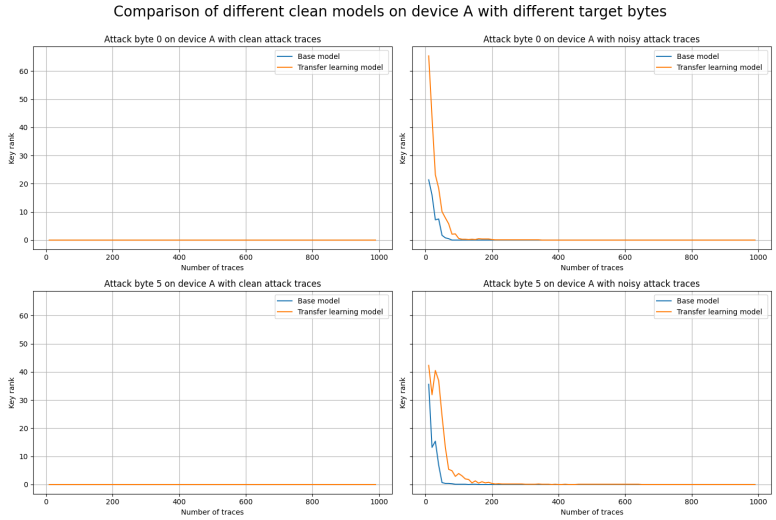


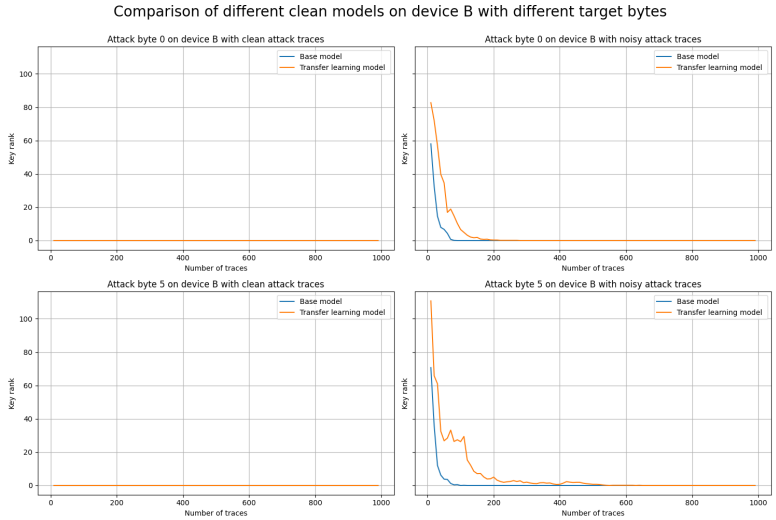
Figure A.3: Signal-to-noise ratio for all test devices - device K-O

Appendix B

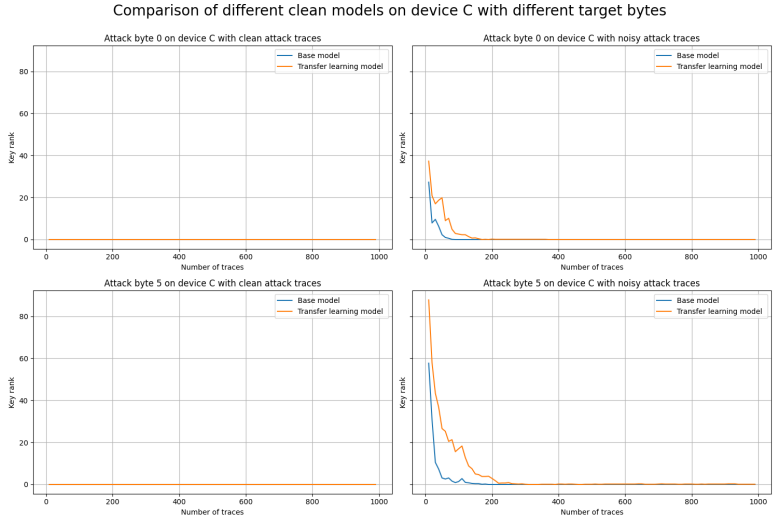
Key ranks for each device for base models trained on clean traces



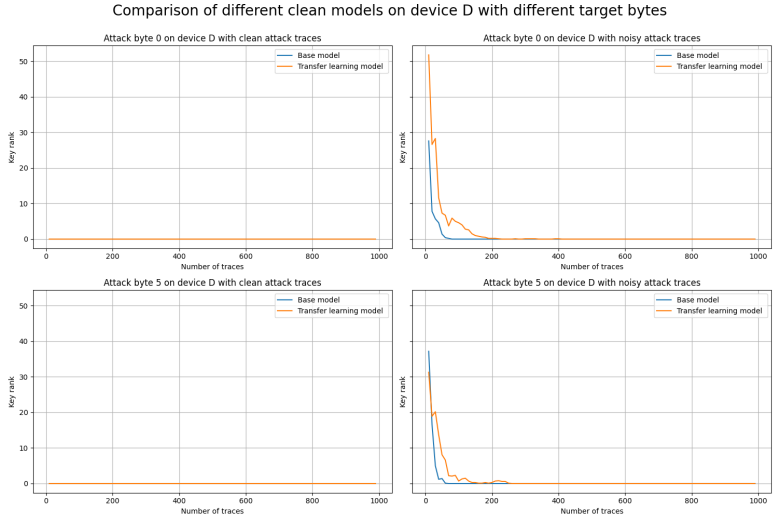
(a) Performance of base model and transfer learning model on byte 0 and 5 for device A



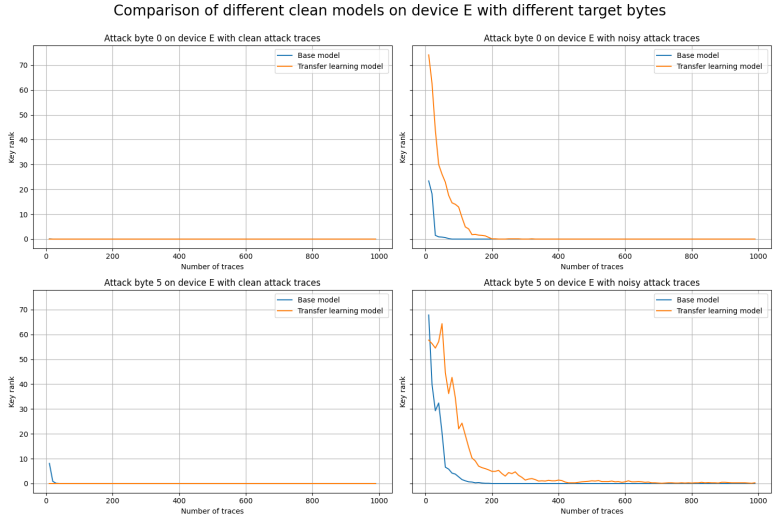
(b) Performance of base model and transfer learning model on byte 0 and 5 for device B



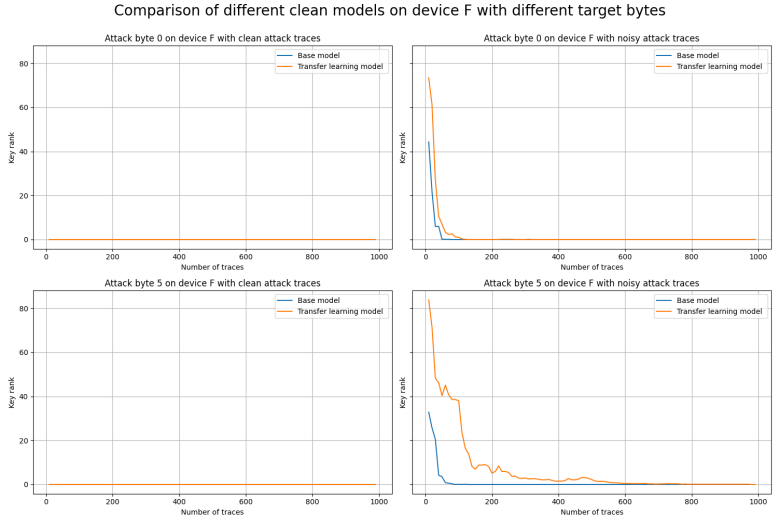
(c) Performance of base model and transfer learning model on byte 0 and 5 for device C



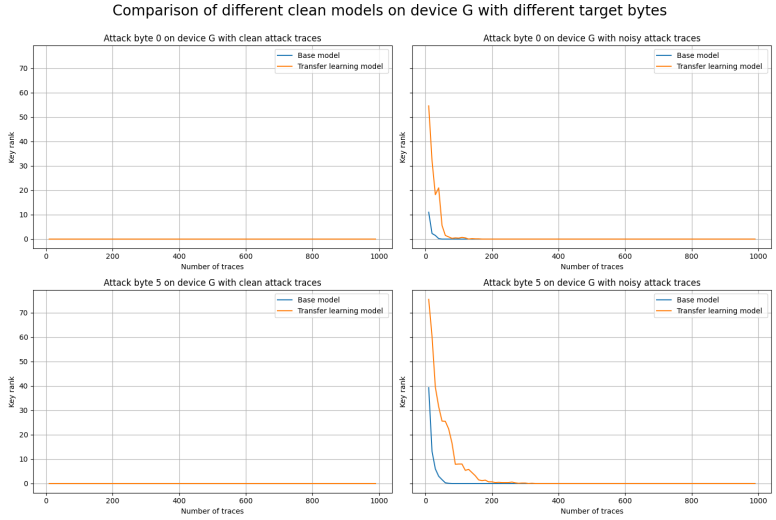
(d) Performance of base model and transfer learning model on byte 0 and 5 for device D



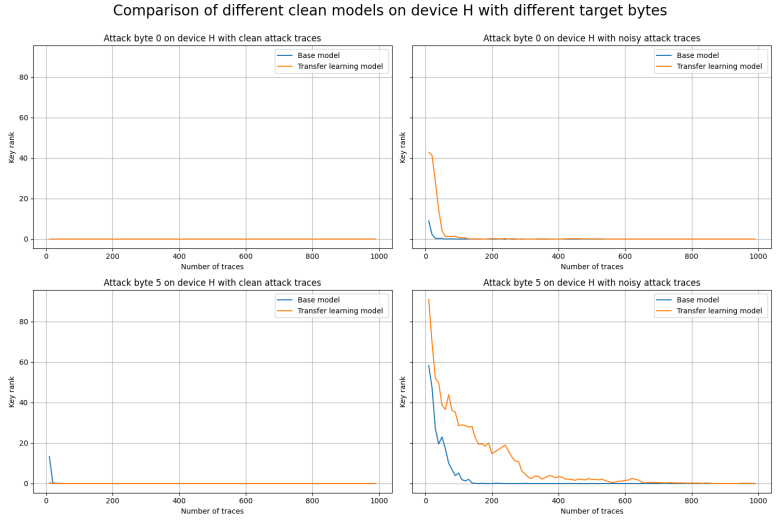
(e) Performance of base model and transfer learning model on byte 0 and 5 for device E



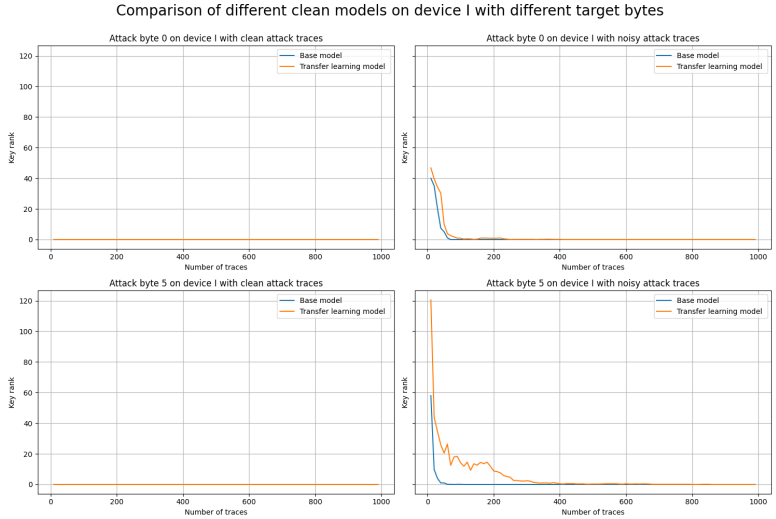
(f) Performance of base model and transfer learning model on byte 0 and 5 for device F



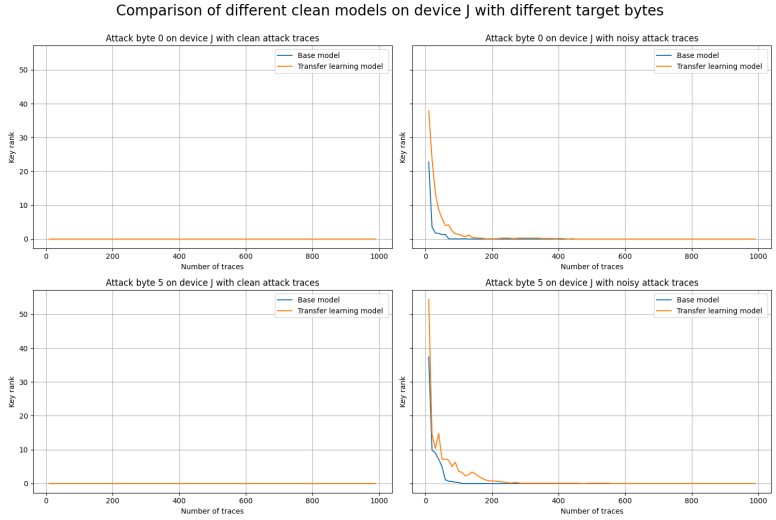
(g) Performance of base model and transfer learning model on byte 0 and 5 for device G



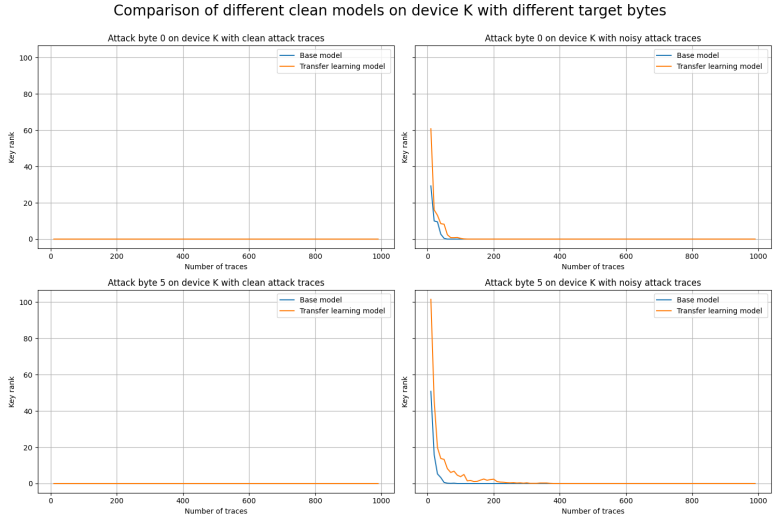
(h) Performance of base model and transfer learning model on byte 0 and 5 for device H



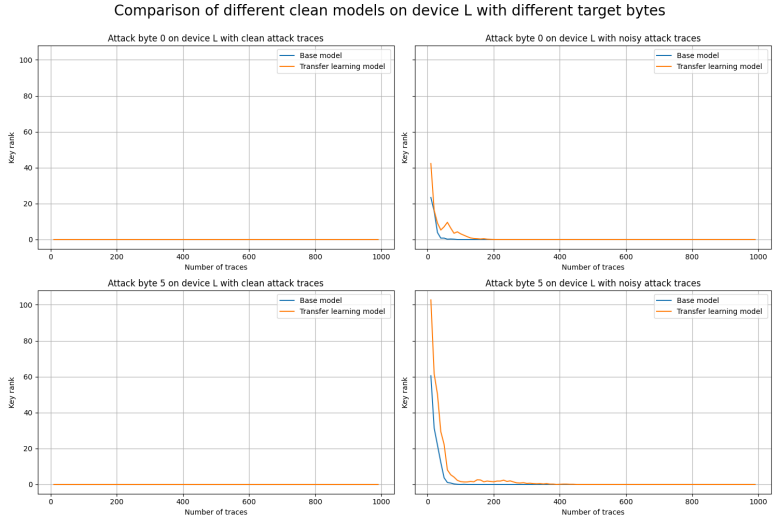
(i) Performance of base model and transfer learning model on byte 0 and 5 for device I



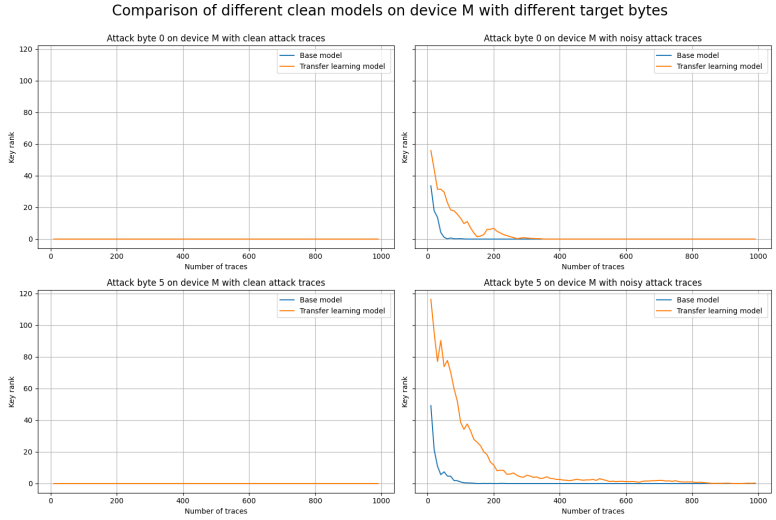
(j) Performance of base model and transfer learning model on byte 0 and 5 for device J



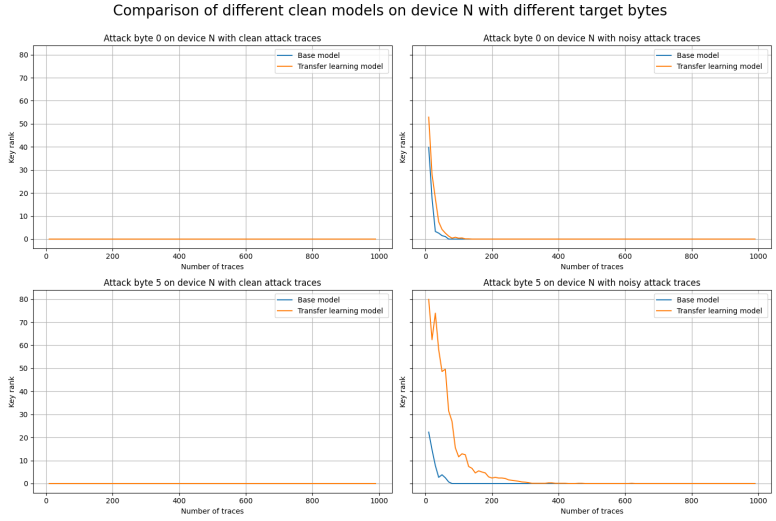
(k) Performance of base model and transfer learning model on byte 0 and 5 for device K



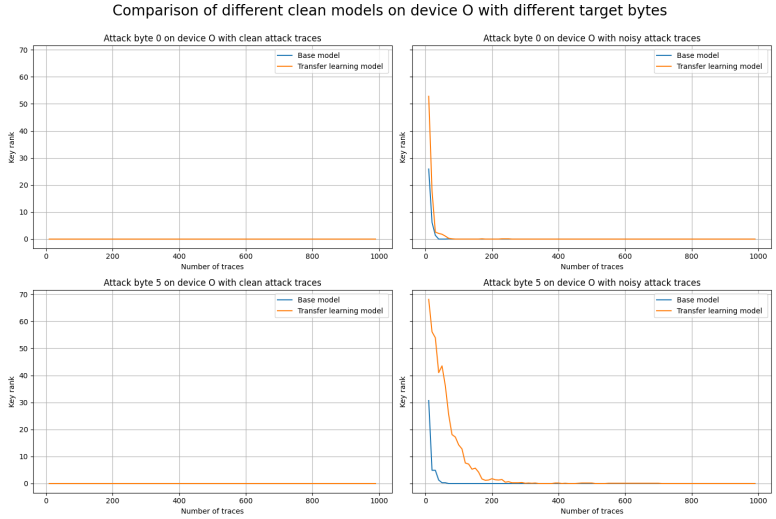
(l) Performance of base model and transfer learning model on byte 0 and 5 for device L



(m) Performance of base model and transfer learning model on byte 0 and 5 for device M



(n) Performance of base model and transfer learning model on byte 0 and 5 for device N

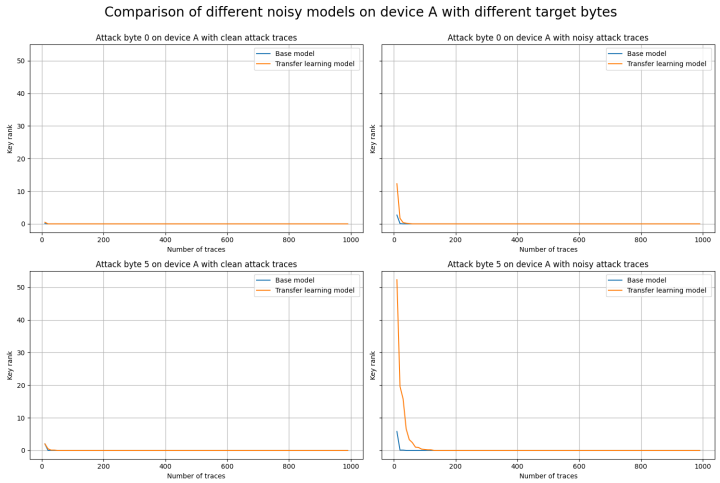


(o) Performance of base model and transfer learning model on byte 0 and 5 for device O

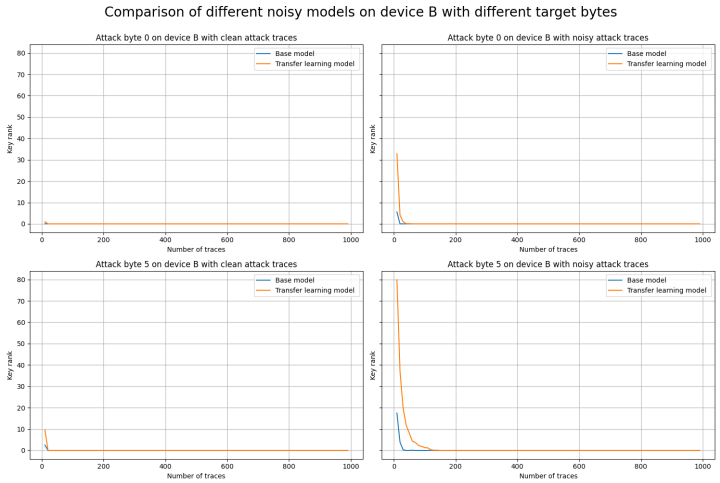
Figure B.1: Performance of base model and transfer learning model with clean base model for each device

Appendix C

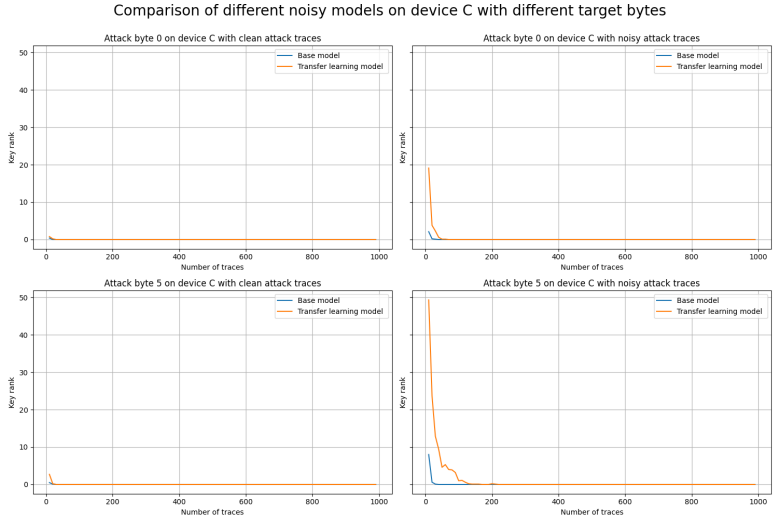
Key ranks for each device for base models trained on traces with simulated noise



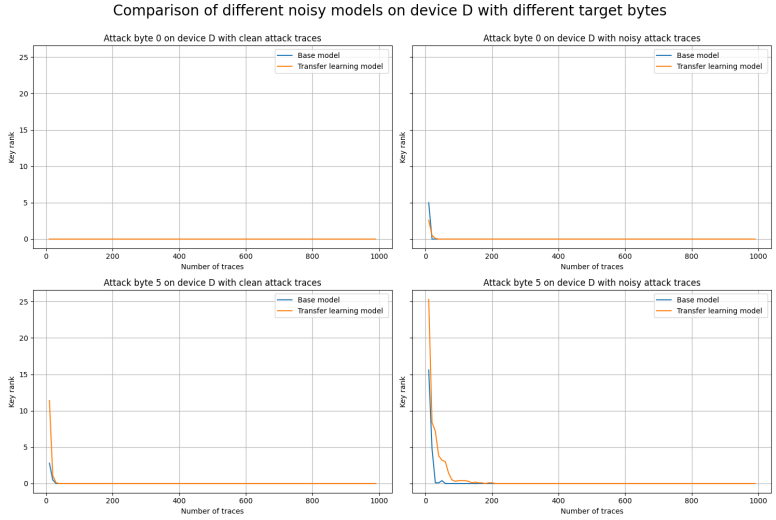
(a) Performance of base model and transfer learning model on byte 0 and 5 for device A



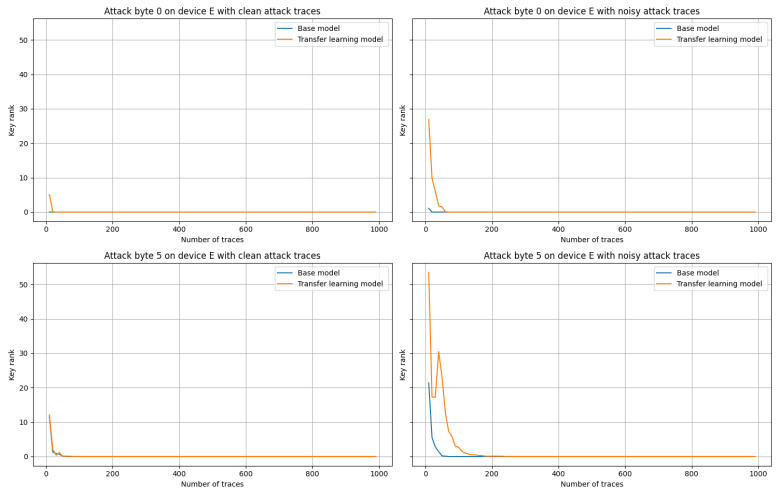
(b) Performance of base model and transfer learning model on byte 0 and 5 for device B



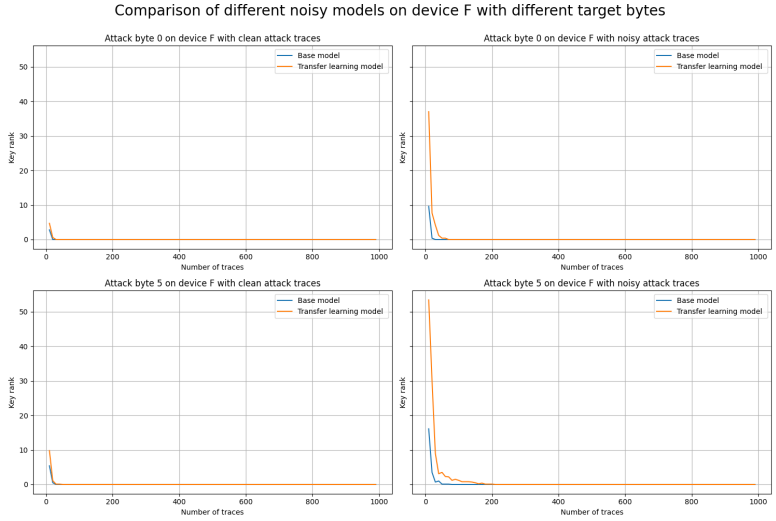
(c) Performance of base model and transfer learning model on byte 0 and 5 for device C



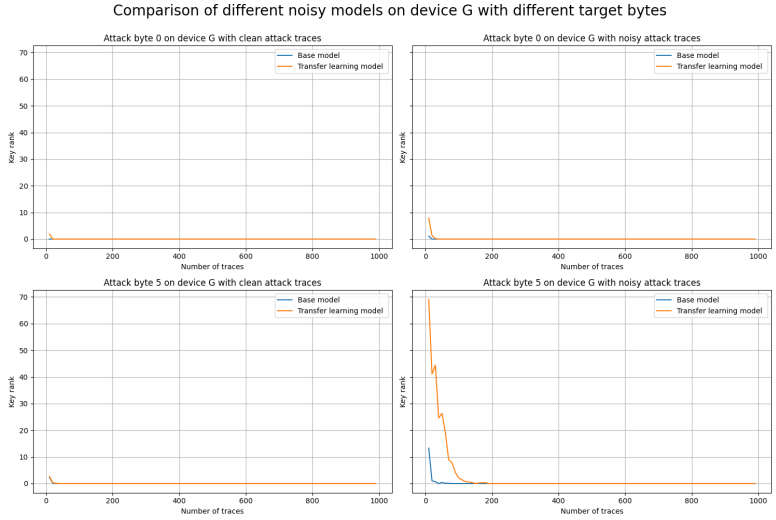
(d) Performance of base model and transfer learning model on byte 0 and 5 for device D



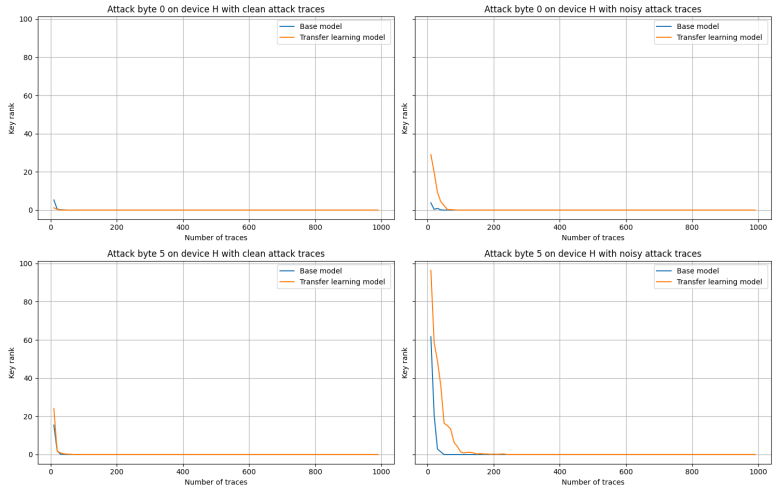
(e) Performance of base model and transfer learning model on byte 0 and 5 for device E



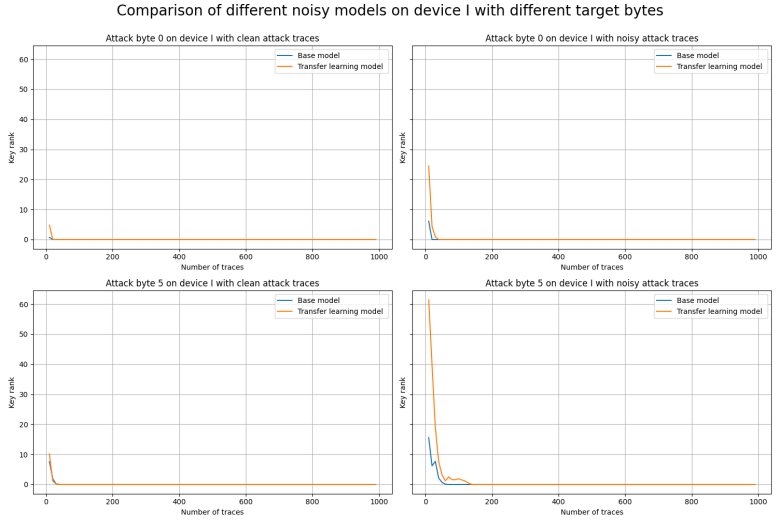
(f) Performance of base model and transfer learning model on byte 0 and 5 for device F



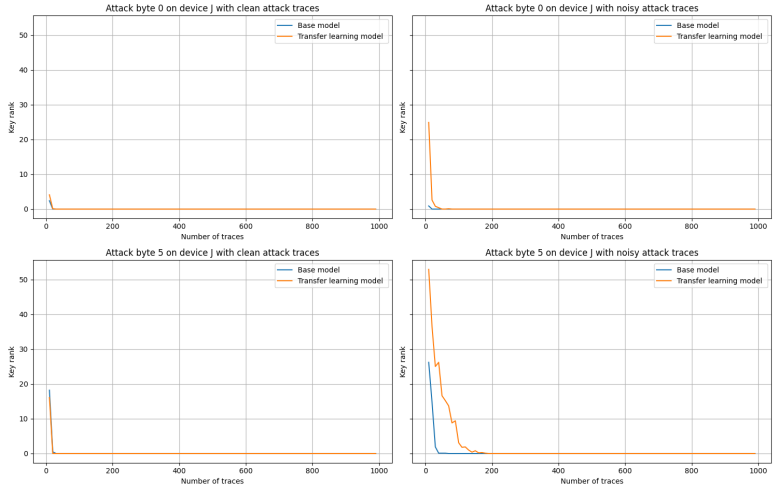
(g) Performance of base model and transfer learning model on byte 0 and 5 for device G



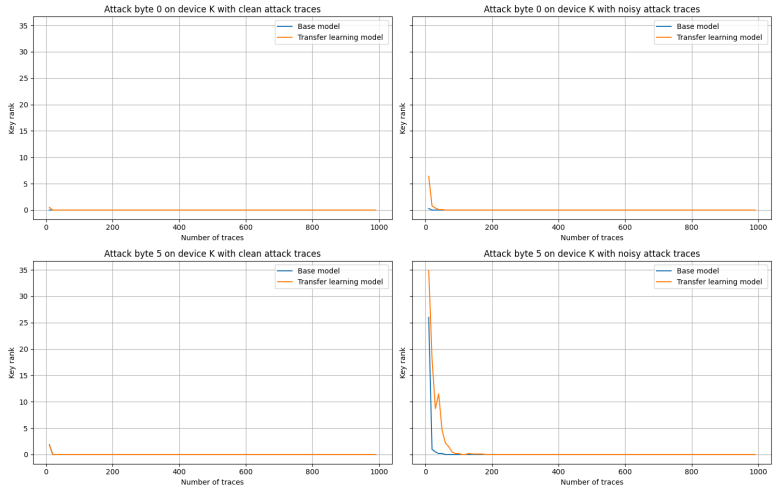
(h) Performance of base model and transfer learning model on byte 0 and 5 for device H



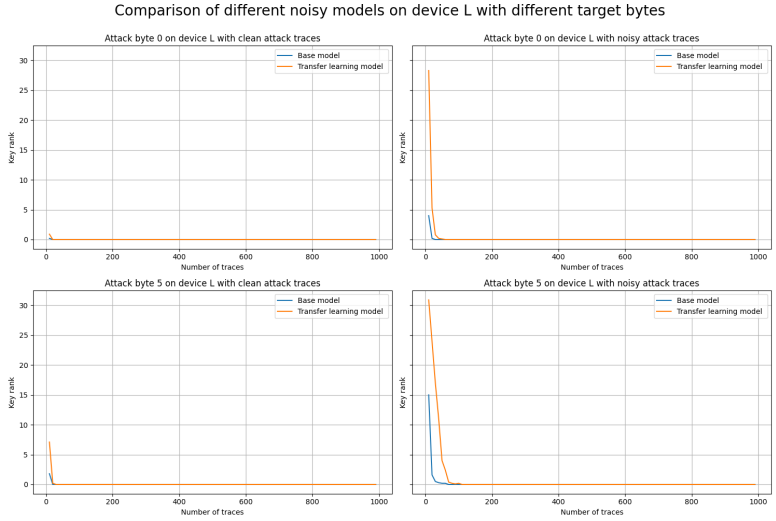
(i) Performance of base model and transfer learning model on byte 0 and 5 for device I



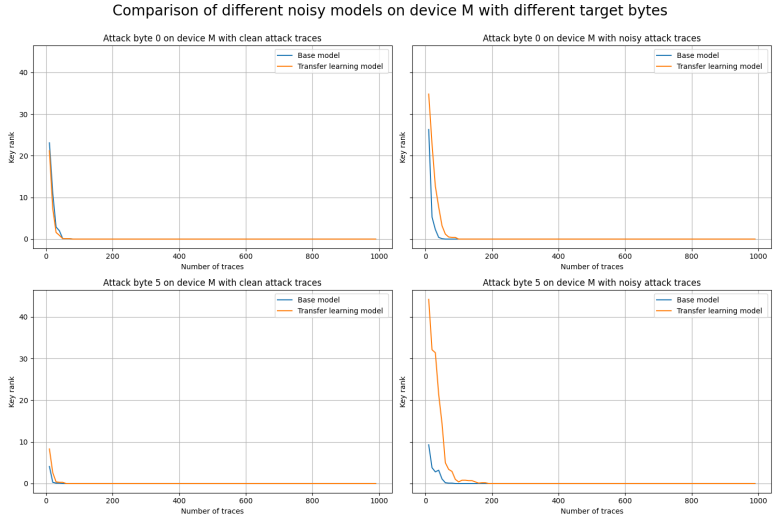
(j) Performance of base model and transfer learning model on byte 0 and 5 for device J



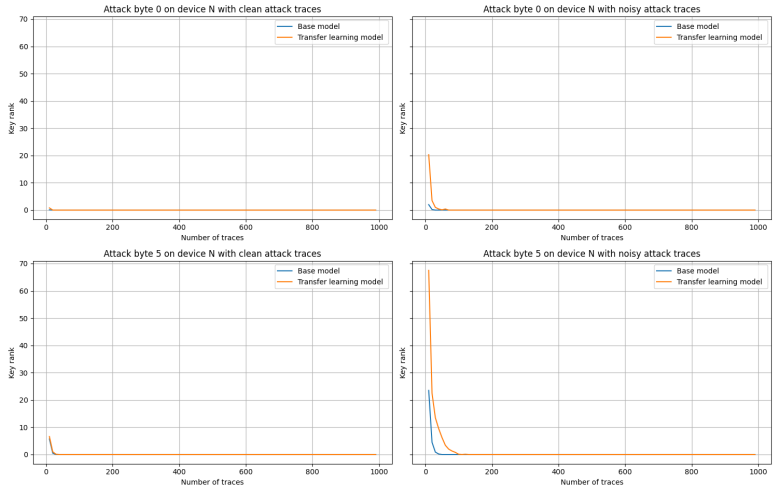
(k) Performance of base model and transfer learning model on byte 0 and 5 for device K



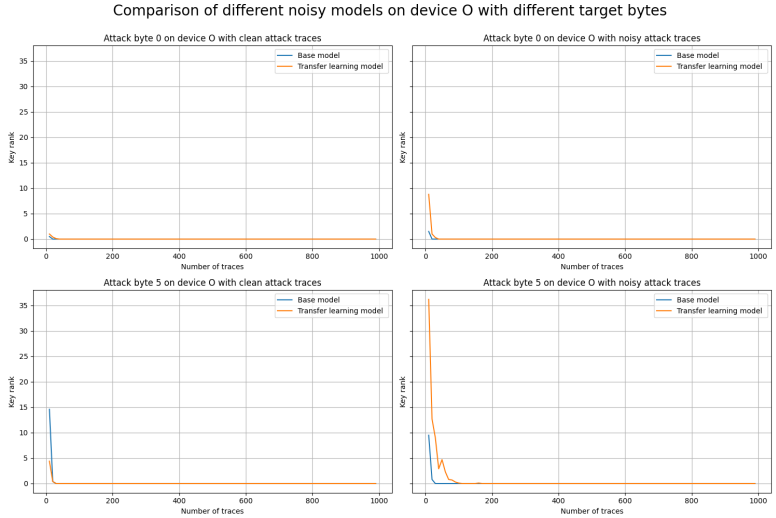
(l) Performance of base model and transfer learning model on byte 0 and 5 for device L



(m) Performance of base model and transfer learning model on byte 0 and 5 for device M



(n) Performance of base model and transfer learning model on byte 0 and 5 for device N



(o) Performance of base model and transfer learning model on byte 0 and 5 for device O

Figure C.1: Performance of base model and transfer learning model with noisy base model for each device