IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 4, pp. 27–56. DOI:10.46586/tches.v2021.i4.27-56

Cross-Device Profiled Side-Channel Attack with Unsupervised Domain Adaptation

Pei Cao, Chi Zhang, Xiangjun Lu and Dawu Gu

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China, {loccs_cp,zcsjtu,luxiangjun,dwgu}@sjtu.edu.cn

Abstract. Deep learning (DL)-based techniques have recently proven to be very successful when applied to profiled side-channel attacks (SCA). In a real-world profiled SCA scenario, attackers gain knowledge about the target device by getting access to a similar device prior to the attack. However, most state-of-the-art literature performs only proof-of-concept attacks, where the traces intended for profiling and attacking are acquired consecutively on the same fully-controlled device. This paper reminds that even a small discrepancy between the profiling and attack traces (regarded as domain discrepancy) can cause a successful single-device attack to completely fail. To address the issue of domain discrepancy, we propose a Cross-Device Profiled Attack (CDPA), which introduces an additional fine-tuning phase after establishing a pre-trained model. The fine-tuning phase is designed to adjust the pre-trained network, such that it can learn a hidden representation that is not only discriminative but also domain-invariant. In order to obtain domain-invariance, we adopt a maximum mean discrepancy (MMD) loss as a constraint term of the classic cross-entropy loss function. We show that the MMD loss can be easily calculated and embedded in a standard convolutional neural network. We evaluate our strategy on both publicly available datasets and multiple devices (eight Atmel XMEGA 8-bit microcontrollers and three SAKURA-G evaluation boards). The results demonstrate that CDPA can improve the performance of the classic DL-based SCA by orders of magnitude, which significantly eliminates the impact of domain discrepancy caused by different devices.

Keywords: Side-channel Attacks · Profiled Attacks · Deep Learning · Cross-device Attacks · Domain Adaptation

1 Introduction

1.1 Overview

Side-channel attack (SCA) has drawn a significant amount of attention since Kocher proposed timing attack [Koc96]. It aims at retrieving the secret values of cryptographic algorithms from a device or a system through the measurement and analysis of physical information.

Among all kinds of SCAs, profiled attacks play an essential role. It is considered one of the most powerful SCAs, at least from the information theory point of view [CRR02]. In such a context, the attacker is able to characterize the device leakage by means of a full-knowledge (plaintexts/ciphertexts and keys) access to a device that is similar to the one under attack. The first profiled attack is the template attack (TA) [CRR02], which builds models with means and covariances. Since then, several works have been done to make the TA more realistic in practice [RO04, CK13]. As profiled SCAs can be formulated as classification problems, the application of machine learning techniques (e.g.,

Received: 2021-04-15 Accepted: 2021-06-15 Published: 2021-08-11

Support Vector Machine [HGM+11, HZ12, BL12] and Random Forest [LBM14, LPB+15]) is inevitably investigated. More recently, deep learning techniques have been introduced as more sophisticated tools to perform the profiled attacks [MPP16] and have shown great success. The SCA community highlights the ability of DL-based SCA since it a) can deal with the high dimensional input better than classical methods like TA, b) naturally caters to masking countermeasures [MPP16, PSB+18, ZBHV20, WAGP20], and c) is robust to misalignment by using specific architectures like Convolutional Neural Networks (CNNs) [CDP17, KPH+19].

Although having the above mentioned advantages, the performance of DL-based profiled attacks may be overestimated due to the gap between the experimental setting and reality. We remark that a major assumption of machine learning is that the training data and testing data must be sampled from the same domain[1]. However, in realistic profiled SCA scenarios, the attack traces must be measured from the target device. Although the target algorithm can be the same, the embodying hardware platform, software implementation, and measurement environment can be different. For example, a random process variation introduced during fabrication, turning on hiding countermeasures on the target device, and even a different level of noise during measurement can result in two different distributions of leakage. Unfortunately, most previous works and publicly available datasets do not consider the domain discrepancy, and simplify the experimental setting using only a single device for both profiling and attacking (the red route in Figure 1). As a result, a classification model trained on the profiling device may not generalize well on the target device under substantial variations in devices, implementations, measurement setups, etc.

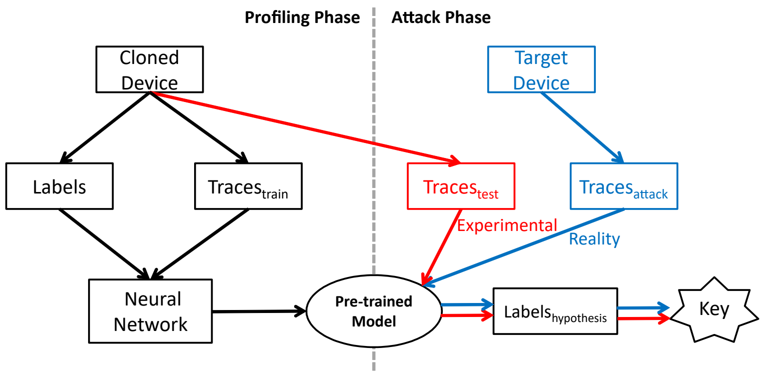


Figure 1: Framework of DL-based profiled attacks.

In this paper, we propose the use of unsupervised domain adaptation as a powerful tool to enable cross-device profiled attacks. Although the unsupervised domain adaptation proved to be successful in transferring the knowledge from a source domain to an unlabeled target domain [LCWJ15, LZWJ16, RMH+19], as far as we are aware, this technique has not been explored in the SCA community to cope with the domain discrepancy between the profiling and attack traces. We demonstrate that a limitation of the classic two-phases profiled attack is it cannot utilize the discrepancy information, which is directly neglected. Therefore, we introduce an additional fine-tuning phase to enhance the pre-trained model. To quantify the domain discrepancy, we adopt the maximum mean discrepancy (MMD), a standard distribution distance metric, as a penalty loss of the classification loss function. We demonstrate the effectiveness of our strategy in dealing with three types of domain discrepancies, i.e., the variations in devices, the addition of countermeasure/noise, and different acquisition settings. The results show that fine-tuning with MMD loss is efficient

[1]A domain consists of two parts: a feature space and a marginal probability distribution [WKW16]. If the two domains are different, they have different feature spaces or probability distributions.

in removing the effect of domain discrepancy in all investigated situations.

1.2 Related Work

The gap between the experimental setting and reality when performing profiled attacks has already been noticed in the past few years but is still an open topic. The authors in [RSV+11] performed a study on 20 different devices, showing that the TA may not work at all when the profiling phase and attack phase are conducted on different devices. Elaabid et al. [EG12] showed that the variations in measurement setup could also lead to worse TA results even using the same device. Recently, in [KKKR18], the authors reported only a 28% success rate on a different keyboard as compared to 100% when profiling and attacking the same keyboard. In [WdHG+20], the authors conducted profiled power analysis on the key loading procedure of multiple DST transponders. They concluded that their cross-device attacks could hardly succeed if only using a single device for profiling. The authors in [DGD+19, GDD+19, BCH+20] reminded us the portability issue still exists and should never be neglected in the context of DL-based SCA. Although solutions to make profiled attacks work on different devices were proposed, e.g., using waveform realignment and acquisition campaigns normalization [EG12], by choice of compression methods [CK14], and using multiple devices for profiling [CK14, DGD+19, GDD+19, BCH+20, WdHG+20], these methods mainly require target-specific preprocessing or are based on a multiple-profiling-devices assumption.

The SCA community recently noticed that transfer learning might be a feasible way to transfer the knowledge learned from the profiling device to the target device [GGH20, TAM20]. However, their focus was on reducing the number of profiling traces. Furthermore, they still used labeled traces acquired from the target domain, which was not a cross-device scenario from the perspective of attackers.

1.3 Our Contributions

Herein, we consider how to transfer the pre-trained model to fit the target device with unsupervised domain adaptation, which, to the best of our knowledge, has not been explored before in the SCA community. Specifically, we introduce a new approach to remove the effect of domain discrepancy between profiling and attack traces, which makes it possible or easier to recover the key of a different device. Our main contributions are as follows:

1. **A cross-device profiled attack (CDPA) strategy**. CDPA is the extension of DL-based profiled attacks, which introduces an additional fine-tuning phase to adjust the pre-trained model for improving the performance when attacking different devices. We emphasize that no labeled attack traces are required since the designed fine-tuning phase focuses on the domain discrepancy instead of the classification task itself.

2. **Introducing a new loss function to DL-based SCA**. With the MMD loss, our network is able to focus on the device discrepancy directly without using multiple profiling devices or task-specific preprocessing. We show that by minimizing the MMD loss and classification loss simultaneously, the fine-tuned model can learn a hidden representation that is not only discriminative but also domain-invariant.

3. **A benchmark of cross-device SCA with satisfying results**. We evaluate the effect of our strategy on eight Atmel XMEGA 8-bit microcontrollers and three SAKURA-G evaluation boards. We show that CDPA can significantly improve the performance of the attacks on different devices, and can even turn an impossible attack into a reality. Besides, we also show the potential of CDPA in removing the effect of adding (simulated) countermeasures/noise and overcoming the

human error (electromagnetic probe placement). These experiments can be reproduced through the following Github repository: https://github.com/CDPA-SCA/Cross-Device-Profiled-Attack.

1.4 Organization

The rest of this paper is organized as follows. In Section 2, we provide some background about the DL-based profiled attacks. In Section 3 we propose a cross-device profiled attack strategy and explore the methodology for removing the effect of domain discrepancy. Section 4 introduces the datasets used in our experiments. Section 5 presents the experimental results on multiple investigated situations. In Section 6, we provide a discussion around hyperparameter selection and its effect on our models. Then we give a brief comparison between CDPA and other promising techniques. Finally, we conclude this paper in Section 7.

2 Background

2.1 Notations

Throughout this paper we use calligraphic letters as $\mathcal{X}$ to denote sets, the corresponding capital letter X to denote random variables (resp. random vectors $\mathbf{X}$), and the lowercase x (resp. $\mathbf{x}$) to denote their realizations. The probability space of a set $\mathcal{X}$ is denoted by $\mathcal{P}(\mathcal{X})$. If $\mathcal{X}$ is discrete, $\mathcal{P}(\mathcal{X})$ corresponds to the set of vectors $[0,1]^{|\mathcal{X}|}$ such that the coordinates sum to 1. Let $\Pr[X]$ denote the distribution of X and $\Pr[X = x]$ the probability when X equals x. We use $\mathbb{E}$ to denote the expected value and the condition might be subscripted by a random variable $\mathbb{E}_X$, or by a probability distribution $\underset{X\sim\Pr[X]}{\mathbb{E}}$ to specify under which probability it is computed.

For profiled attacks, the target sensitive variable is $V = f(P, K)$ where f denotes a cryptographic primitive (e.g., the SubBytes operation), P denotes a known variable (e.g., plaintext or ciphertext) and K denotes a part of the secret key (e.g., a byte) that an attacker tries to recover. Among all the possible value K may take, k^* will denote the right key hypothesis. Side-channel traces will be viewed as discrete realizations of a random vector $\mathbf{x} = x_1, ..., x_D$, with D being the number of features. We use y to denote the label of a trace, which can be performed using the value or hamming weight (HW) of the sensitive variable V. In particular, we denote $\mathcal{D}_s = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^{n_s}$ with n_s labeled traces the source domain measured from the profiling device, and $\mathcal{D}_t = \{(\mathbf{x}_i^t)\}_{i=1}^{n_t}$ the target domain with n_t unlabeled traces measured from the target device.

2.2 Profiled Side-Channel Attacks

As is shown in Figure 1, a profiled attack generally is composed of two phases: a profiling phase and an attack phase. During the profiling phase, the attacker first estimates the distribution:

$$\Pr[\mathbf{X}|Y = y], \tag{1}$$

using the training set $\mathcal{D}_s = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^{n_s}$. For example, one of the most popular ways to estimate the conditional probability is to use a mean vector $\mu_{\mathbf{y}}$ and a covariance matrix $\mathbf{\Sigma}_{\mathbf{y}}$, which is based on the assumption that $(\mathbf{X}|Y = y)$ has a multivariate Gaussian distribution. Then, given a trace $\mathbf{x}_i$, the attacker can compute the likelihood for each possible y using the estimated probability density function. In other words, the attacker eventually gets a model $F(.) : \mathcal{X} \to \mathcal{P}(\mathcal{Y})$, that can be assimilated to a probability mass function (possibly after normalization).

In the attack phase, the attacker tries to recover the fixed unknown key k^* with the trace set $\mathcal{D}_t = \{(\mathbf{x}_i^t)\}_{i=1}^{n_t}$ measured from the target device. Specifically, he can calculate the log-likelihood score over all the attack traces for each $k \in \mathcal{K}$:

$$\mathbf{d}[k] = \sum_{i=1}^{n_t} \log(F(\mathbf{x}_i)[f(p_i, k)]). \tag{2}$$

The attacker then select the subkey k_{guess} leading to the highest log-likelihood score: $k_{guess} = \text{argmax}_{k\in\mathcal{K}}\mathbf{d}[k]$. The attack is successful if $k_{guess} = k^*$.

2.3 Neural Networks

Neural networks are nowadays the privileged tool to address the classification problem. In such a context, a classification task can also be performed in two phases, a training phase and a testing phase. In the training phase, the neural network aims to construct a function $F(.) : \mathbb{R}^D \to \mathbb{R}^{|\mathcal{Y}|}$ that takes input $\mathbf{x} \in \mathbb{R}^D$ and outputs vector $\mathbf{p} \in \mathbb{R}^{|\mathcal{Y}|}$ of scores. To construct the function $F(.)$, a loss function is computed that quantifies the classification error of $F(.)$ over the training batch. Then each trainable parameter is updated to minimize the loss, which is called backward propagation. After training, the classification is done by choosing the label y such that $y = \text{argmax}\,\mathbf{p}[y]$. In general, a neural network consists of three blocks: an input layer, several hidden layers, and an output layer, which are all composed of multiple neurons. There are many kinds of neural networks, and the different behavior of them is mainly affected by how these neurons are connected within (and between) layers. In this paper, we focus on the family of the CNNs because of their potential in breaking cryptographic implementations protected with countermeasures [MPP16, CDP17, PSB$^+$18, ZBHV20].

CNNs use three main types of layers: convolutional layers, pooling layers, and fully-connected layers. Convolutional layers are linear layers that share weights cross space, whose trainable parameters are several small column vectors called convolutional filters. Each filter slides over the trace by some amount of units (called stride) and is expected to extract a kind of characteristic. As inputs go along convolutional layers, higher-level abstract features are expected to be extracted. To avoid complexity explosion due to the increase of convolutional layers, the pooling layers are introduced. The most common pooling functions are the max-pooling and the average-pooling. The max-pooling outputs the maximum value within a window (called pooling filters), while the average-pooling outputs the average value within the pooling filters. Finally, fully-connected blocks are layers where every neuron is connected with all the neurons in the neighborhood layers.

In this paper, we consider simplifying the presentation of CNN by dividing it into two parts: an encoder part and a classification part. The encoder aims to extract high-level features from the input to help the decision-making. To achieve this, the main block of the encoder is a convolutional layer γ followed by an activation function σ. The former locally extracts information from the input, and the latter provides non-linearity to the learned classification function. After several $(\sigma \circ \gamma)$ blocks, a pooling layer δ is added to reduce the complexity of the network. The above block is repeated several times until obtaining abstract features of reasonable size. Finally, the classification part contains some fully-connected layers λ that combine the features in different locations to obtain a global result depends on the entire inputs. To sum up, a common architecture of CNNs can be characterized by the following formula:

$$F = s \circ [\lambda]^{n1} \circ [\delta \circ [\sigma \circ \gamma]^{n2}]^{n3}, \tag{3}$$

where we use s to denote a softmax layer that is composed of $|\mathcal{Y}|$ classes.

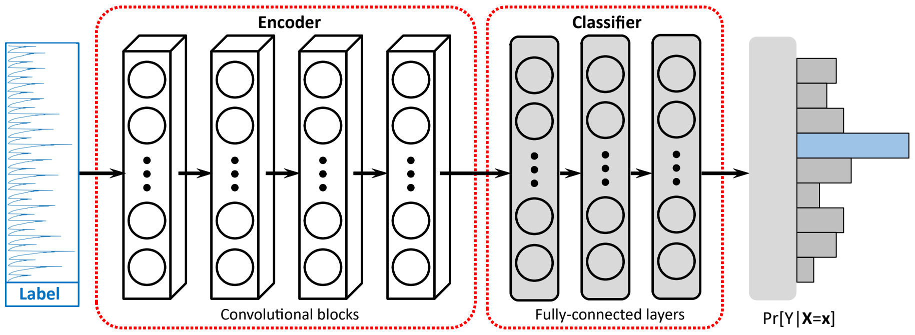


Figure 2: An example scheme of the CNN architecture.

2.4 DL-based Profiled Attacks

The embedding of deep learning to profiled SCA is easy since it is highly related to the profiled attacks in the side-channel context. More specifically, the profiling phase of the profiled SCA corresponds to the training of the neural networks, and the attack phase can be seen as a series of classification tasks. For each input trace, the pre-trained model will output a renormalized score vector $\mathbf{p}$ in such a way that they define a probability distribution $\mathbf{p} \approx \Pr[Y|\mathbf{X} = \mathbf{x}]$. We remark that the computed output does not only provide the most likely label to solve the classification task, but also the likelihood of all the other labels. This form of output allows the attacker to rank the key candidates using multiple traces, as shown in Equation 2. Actually, the whole network Equation 3 could be viewed as an approximation of the probability density function in Equation 1.

Metrics in Profiled Attacks. To evaluate the performance of the profiled attacks, a common option is to use Guessing Entropy (GE) or Success Rate (SR). GE is defined as the average rank of the correct subkey after the attack. More specifically, given n_t attack traces, the attack will output an accumulated score vector $\mathbf{d} = [d_1, d_2, ..., d_{|\mathcal{K}|}]$, each element of which denotes the estimated probability of a candidate subkey. If we sort the score vector to get a $\mathbf{d}'$ in decreasing order of score, the GE can be defined as the average position of d_{k^*} in $\mathbf{d}'$ over multiple experiments. Similarly, the SR is defined as the average empirical probability that $\mathbf{d}'[1] = d_{k^*}$.

2.5 Preprocessing techniques

Data preprocessing is very important when building a network model, which can often determine the training results. In the context of DL-based SCA, the efficiency of preprocessing techniques should never be neglected because they a) make it easier to learn a classification model that generalizes [MBTL13, CK18], and b) can greatly increase the convergence rate of the training process [MOM12]. Here we introduce four preprocessing methods that are most commonly used:

1. **Feature Scaling** readjusts the value of each dimension of the data (these dimensions may be independent of each other), so that the final data vector is within the interval of $[a, b]$. For each dimension, the general formula for scaling is given as:

$$\mathbf{x}' = a + \frac{(\mathbf{x} - \min(\mathbf{x}))(b - a)}{\max(\mathbf{x}) - \min(\mathbf{x})}, \tag{4}$$

   where $\mathbf{x} \in \mathbb{R}^{n_s}$ is an original value, and $\mathbf{x}'$ is the scaled value.

2. **Feature Standardization** aims to make the features of each dimension in the dataset have zero-mean and unit-variance, which is widely used in many machine

learning algorithms. The general method is to determine the mean $\bar{x}$ and standard deviation σ. Then we subtract the mean from each feature and divide the result by its standard deviation:

$$\mathbf{x}' = \frac{\mathbf{x} - \bar{x}}{\sigma}. \tag{5}$$

3. **Horizontal Scaling** is a horizontal version of feature scaling introduced by Wouters et al.[WAGP20]. Unlike the feature scaling that normalizes the side-channel traces per sample, it will be applied on a per trace basis.

4. **Horizontal Standardization** is a horizontal version of feature standardization. It normalizes each trace using the mean and standard deviation calculated per trace on all feature dimensions.

Herein, we note that the extreme values in feature scaling (similarly, the mean and standard deviation in feature standardization) might be improper to transform the attack traces, especially in the context of cross-device SCA [MBTL13]. Furthermore, the feature scaling and feature standardization normalize traces on each dimension of features, which implies that the side-channel traces should be well-aligned. Otherwise, they could confuse and change the original features of side-channel traces. Therefore, we mainly use the horizontal version of normalization methods in the rest of this paper.

3 Cross-Device Profiled Attack

Although deep learning techniques seem to be quite suitable for performing the profiled SCA, the domain discrepancy between the profiling and attack traces is still a bottleneck restricting the application of the profiled attacks in practice. In fact, an implicit hypothesis of deep learning techniques is that the training data must be independent and identically distributed (i.i.d.) with the test data. However, when we adopt deep learning in the context of profiled SCA, this i.i.d. hypothesis is too strong since attack traces are often acquired from a different device without control. In such a context, various settings can easily break the hypothesis and lead to poor performance when we try to attack the target device. For example,

1. **Different Devices**. Although the structural changes introduced during the manufacturing process are relatively small and the devices produced meet the required specifications, no two chips are exactly the same. Even for devices of the same type, the leakage of the side-channel information is inevitably different, which is likely due to random process variations introduced during fabrication and packaging [MT10]. As a result, attacking a different device may cause a successful *single-device-model* attack to completely fail.

2. **Different Implementations**. Although a common assumption of profiled SCA is that the profiling device has the same software (or hardware) implementation as the target device, the attacker cannot take full control of the target device in most situations. Thus, we consider a more difficult scenario where the owner of the target device can turn on hiding countermeasures. Consequently, the target device can be considered a black box with an unknown data distribution, eventually increasing the difficulty in obtaining the key.

3. **Different Settings of Acquisition**. Apart from the above two reasons, the domain discrepancy over multiple measurements occurs because of changes in the measurement setup (e.g., the location of probes, the supply voltage, and the oscilloscope settings) or environmental changes (e.g., temperature [MMR20]).

Keeping these in mind, in Section 3.1, we propose a generic attack strategy aiming at eliminating the effect of domain discrepancy. Section 3.2 elaborates the MMD loss that enables the networks to learn domain-invariant representations. Finally, in Section 3.3, we show how to minimize and embed the MMD in a standard CNN model.

3.1 A Cross-Device Profiled Attack Strategy

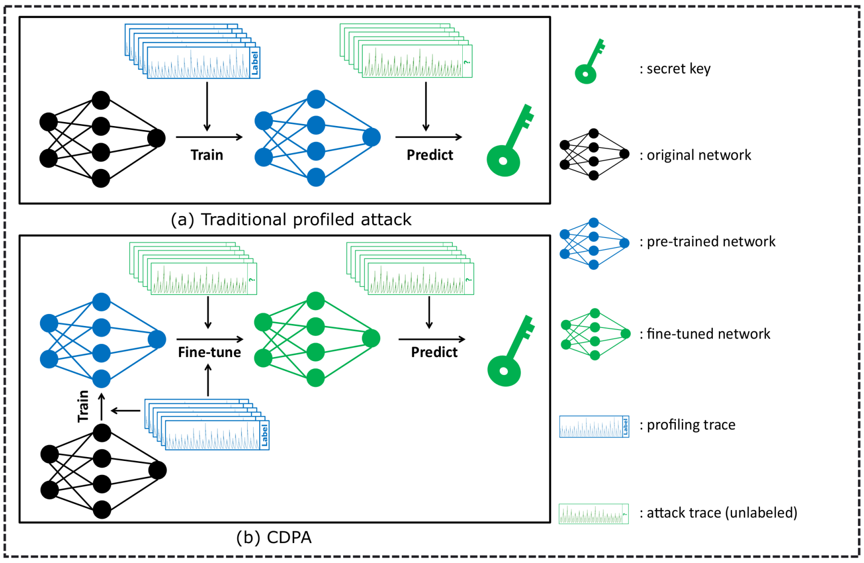


Figure 3: Comparison of traditional profiled attack and CDPA.

As described in Section 2.2, a profiled attack is composed of two phases: a profiling phase and an attack phase. We note that a limitation of the two-phases attack is that it cannot utilize the information brought by domain discrepancy, which is directly neglected. Therefore, we propose extending the classic profiled attacks by introducing an additional fine-tuning phase before the final classification (see Figure 3). Fine-tuning is a widely adopted technique in transfer learning for deep neural networks where a few epochs of training are applied to a pre-trained model's parameters to adapt them to a new task. An implicit assumption behind fine-tuning is that the source and target task should be related, which is definitely true in profiled SCA since the classification task should be the same in the profiling and attack phases. Thus, we could expect that the network parameters are not far from the optimal values for the target device.

One straightforward approach for fine-tuning is to take a pre-trained network and then train (part of) its parameters using the data from the target domain. However, in a realistic SCA scenario, there is no labeled trace measured from the target device. In our strategy, therefore, the inputs of the fine-tuning phase are the original profiling traces with known labels, and a limited number of unlabeled traces measured from the target device. Our network should then capture the discrepancy information of two domains, based on which we can adjust the trainable parameters to minimize the domain discrepancy and the classification error simultaneously. Thus, we can expect to obtain a classification model that is capable of extracting domain-invariant features.

3.2 Methodology for Minimizing the Domain Discrepancy

More formally, the source domain consists of labeled traces $\mathcal{D}_s = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^{n_s}$ measured from the profiling device, and the target has only unlabeled traces $\mathcal{D}_t = \{(\mathbf{x}_i^t)\}_{i=1}^{n_t}$ measured from the target device. The trace $\mathbf{x}_i^*$ belongs to the topological space $\mathcal{X}$. The corresponding label is represented by $y_i^s \in \mathcal{Y}$, where $|\mathcal{Y}|$ is 9 for the HW and 256 for a byte. Then our goal is to train a classifier $F(.)$ that can predict the labels $\{\hat{y}_i^t\}_{i=1}^{n_t}$ of the attack traces, where the data distributions of the source and target domains are different, i.e., $\Pr[\mathbf{X}_s, Y_s] \neq \Pr[\mathbf{X}_t, Y_t]$.

We note that minimizing the domain discrepancy can be considered equivalent to the task of finding a representation that makes the domains appear as similar as possible. In fact, this problem is called (unsupervised) domain adaptation, which is a branch of transfer learning and has been well studied in the last few years. Inspired by these state of the arts [THZ+14, LCWJ15, RMH+19], we introduce the Maximum Mean Discrepancy (MMD) [GBR+12], a standard distribution distance metric, to measure the similarity between the source and target domains in a reproducing kernel Hilbert space (RKHS). We hereafter recall its definition:

Definition 1 (Maximum Mean Discrepancy [GBR+12])**.** Let $\mathbf{X}_s$ and $\mathbf{X}_t$ be random variables defined on a topological space $\mathcal{X}$, with respective Borel probability measures p and q. Let $\mathcal{F}$ be a class of functions $f\colon \mathcal{X} \to \mathbb{R}$. The MMD is defined as:

$$\mathrm{MMD}(\mathcal{F}, p, q) = \sup_{f\in\mathcal{F}} \left(\mathop{\mathbb{E}}_{\mathbf{X}_s\sim p}[f(\mathbf{X}_s)] - \mathop{\mathbb{E}}_{\mathbf{X}_t\sim q}[f(\mathbf{X}_t)] \right). \tag{6}$$

It has been shown that a unit ball $\mathcal{F}$ in a universal RKHS $\mathcal{H}$ is rich enough to distinguish any two distributions, and MMD can be expressed as the distance in $\mathcal{H}$ between their mean embeddings: $\mathrm{MMD}^2(\mathcal{F}, p, q) = \|\mu_p - \mu_q\|_{\mathcal{H}}^2$ [GBR+12]. The MMD can be eventually calculated using kernel methods. Specifically, for a nonlinear mapping $\phi(.)$ associated with the RKHS $\mathcal{H}_{ker}$ and kernel $ker(.)$, where $ker(\mathbf{x}^s, \mathbf{x}^t) = \langle \phi(\mathbf{x}^s), \phi(\mathbf{x}^t) \rangle$, an empirical estimate of the MMD can be obtained as:

$$\begin{aligned}\widehat{\mathrm{MMD}}^2(\mathcal{F}, \mathcal{X}_s, \mathcal{X}_t) = \frac{1}{n_s^2}\sum_{i=1}^{n_s}\sum_{j=1}^{n_s} ker(\mathbf{x}_i^s, \mathbf{x}_j^s) + \frac{1}{n_t^2}\sum_{i=1}^{n_t}\sum_{j=1}^{n_t} ker(\mathbf{x}_i^t, \mathbf{x}_j^t) \\ - \frac{2}{n_s n_t}\sum_{i=1}^{n_s}\sum_{j=1}^{n_t} ker(\mathbf{x}_i^s, \mathbf{x}_j^t).\end{aligned} \tag{7}$$

Having the empirical estimate of MMD, we consider to bound the target error by the source classification error plus the MMD between the source and target domain:

$$\mathcal{L} = \mathcal{L}_C(\mathcal{X}_s, \mathcal{Y}_s) + \lambda \cdot \widehat{\mathrm{MMD}}^2(\mathcal{F}, \mathcal{X}_s^l, \mathcal{X}_t^l), \tag{8}$$

where $\mathcal{L}_C(\mathcal{X}_s, \mathcal{Y}_s)$ denotes the classification loss[2] calculated on the available labeled traces, $\widehat{\mathrm{MMD}}^2(\mathcal{F}, \mathcal{X}_s^l, \mathcal{X}_t^l)$ denotes the distance between the source domain $\mathcal{X}_s^l$ and target domain $\mathcal{X}_t^l$, $\lambda > 0$ is a penalty parameter. Note that $\mathcal{X}_*^l$ is the lth layer hidden representations of side-channel traces. In this way, we can expect the high-level features in the lth layer are both discriminative and domain-invariant (see Figure 4). Note that MMD focuses on the difference in distribution between the learning and attacking datasets, regardless of the labeling information. Therefore, there are no restrictions on the mask/plaintext/key of the attack traces and no need to know whether the labels are identical between the profiling and attack traces. This unsupervised property of MMD exactly fits the realistic profiled SCA scenarios where the labeling information of the attack traces is never known before the attack.

[2]We use cross-entropy loss by default. However, an attacker can also select other loss functions that are specific to the DL-based SCA, such as the cross-entropy rate introduced in [ZZN+20] and the ranking loss introduced in [ZBD+20].

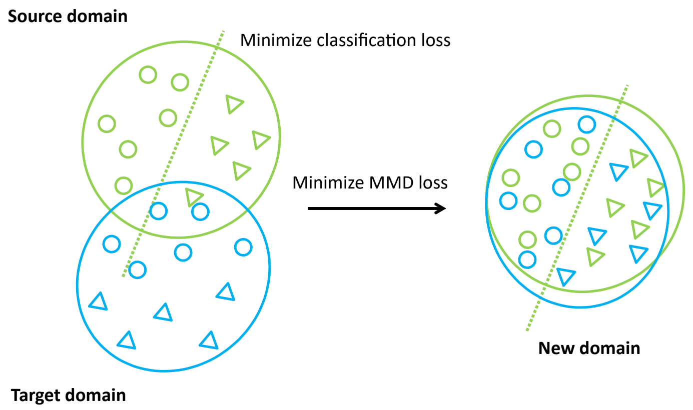


Figure 4: Optimizing an objective that simultaneously minimizes classification error and MMD loss. We use small circles and triangles to denote different labels (unknown in the target domain), and use the dotted line to denote the classifier.

We can notice that the behavior of Equation 8 is very similar to what L1 and L2 regularizations do when training a classification model. However, the main purpose of L1 and L2 regularizations is to prevent overfitting by controlling the complexity of the model, while MMD regularization aims to minimize the domain discrepancy to make the domains appear as similar as possible. As mean embedding matching is sensitive to the kernel choices, instead of using a single kernel function in Equation 7, we consider the multiple kernel variant of MMD (MK-MMD) proposed in [GSS+12], which leads to a principled method for optimal kernel selection. Specifically, the characteristic kernel $ker(.)$ is determined as a convex combination of m radial basis function (RBF) kernels $\{k_j(\mathbf{x},\mathbf{x}') = e^{-\|\mathbf{x}-\mathbf{x}'\|^2/\gamma}\}_{j=1}^{m}$, by varying bandwidth γ between $2^{-\lfloor m/2 \rfloor}\gamma_0$ and $2^{\lfloor m/2 \rfloor}\gamma_0$ with a multiplicative step-size of 2. We set the γ_0 to be the median distance between points in the aggregate sample—the *median heuristic* [GBR+12]. Thus, the kernel is finally denoted as:

$$ker(.) = \sum_{j=1}^{m} \beta_j k_j, \tag{9}$$

where $\sum_{j=1}^{m} \beta_j = 1$, and $\beta_j \geq 0$. In our work, we set $\beta_j = 1/m$ according to [LZWJ16] and it works well in practice.

3.3 Methodology for Embedding MMD in CNN-based SCA

Then the question arises: how to minimize the new loss function in Equation 8 during the fine-tuning phase? For one thing, the original network architecture receives only labeled traces for training, which cannot be directly used to calculate the MMD loss. For another, we have not decided where to calculate the MMD loss in our network.

First, we have to modify the network architecture such that the MMD loss can be easily calculated. Herein, we consider extending the pre-trained network as depicted in Figure 5, which enables us to optimize the classification and MMD loss simultaneously. Our architecture is composed of a source and a target CNN, with shared trainable weights. The trainable weights are initialized to be the same as the pre-trained model. For each fine-tuning iteration, the extended network receives two batches of input traces. One batch of traces is from the labeled source domain, and the other batch is from the unlabeled target domain. The two batches of traces are fed to the source CNN and target CNN, respectively. In particular, the batch of labeled profiling traces is used to compute the classification loss $\mathcal{L}_C$ as before, while the MMD loss is computed over two batches of hidden representations of both the profiling and attack traces. Then, all the trainable parameters of the network are updated by minimizing the total loss (see Equation 8) in the backpropagation algorithm.

Besides, we must determine where to calculate the MMD loss in the network. As is revealed in [YCBL14], the deep features must transform from generic to task-specific as one goes up the layers of a deep CNN. In other words, the transferability of the hidden representation tends to significantly drop in higher layers with increasing domain discrepancy. Therefore, we decide to minimize the MMD loss on the classifier part (fully-connected layers). Note that the encoder part (convolutional blocks) of the network is still trainable during the fine-tuning phase to further adapt to the target domain. We make the encoder part trainable mainly because we expect the convolutional blocks to learn shift-invariant features in case the target domain is not well aligned. In [LCWJ15], authors show that instead of considering a single layer for adaptation, another approach is to sum up the MMD loss in multiple fully-connected layers. Thus, Equation 8 can be rewritten as:

$$\mathcal{L} = \mathcal{L}_C(\mathcal{X}_s, \mathcal{Y}_s) + \lambda \cdot \sum_{l \in \mathcal{S}} \widehat{\mathrm{MMD}}^2(\mathcal{F}, \mathcal{X}_s^l, \mathcal{X}_t^l), \tag{10}$$

where $\mathcal{S}$ is the set of target fully-connected layers. During our tests, we observe that this approach usually leads to better results. Therefore, we adopt Equation 10 as the loss function of the fine-tuning phase in the rest of this paper.

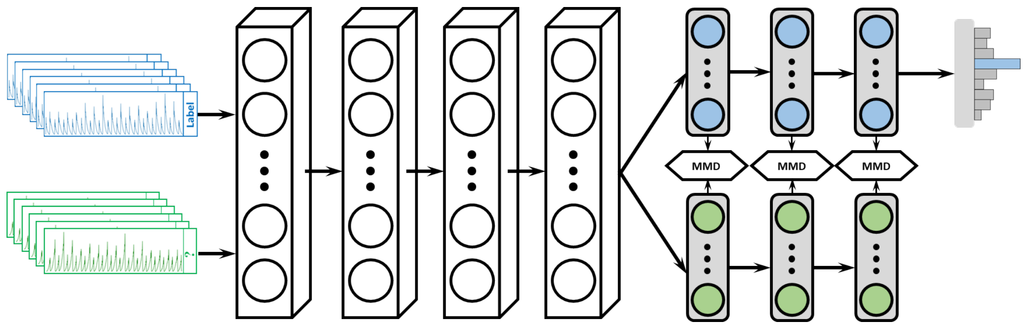


Figure 5: An architecture for cross-device SCA (fine-tuning phase).

To summarize, an end-to-end cross-device SCA consists of the following three steps:

1. The attacker obtains a set of labeled profiling traces from a profiling device. He can train a classification model solely with the profiling traces by minimizing the cross-entropy loss.

2. The attacker then obtains a limited set of unlabeled attack traces from the target device. The attack traces together with the profiling traces are fed to the fine-tuning network, with the new loss function defined in Equation 10. He can minimize the cross-entropy loss with the labeled profiling traces, and minimize the MMD loss with the additional unlabeled attack traces.

3. The attacker finally uses the fine-tuned model for attack instead of the pre-trained model.

4 Datasets

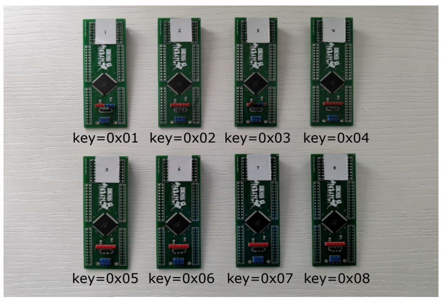
Different types of domain discrepancy must be investigated to evaluate the performance of CDPA. First, we investigate the cross-device scenario with eight Atmel XMEGA 128A1U

8-bit microcontrollers and three SAKURA-G evaluation boards. Based on these devices, we build two datasets[3] (we refer to the datasets as XMEGA and SAKURA_AES hereafter) covering the main types of SCA scenarios. Second, we are curious to see if CDPA can deal with the domain discrepancy caused by the addition of countermeasures/noise. To this end, we use the ASCAD dataset [PSB+18], by simulating two types of countermeasures/noise: Gaussian noise and clock jitters. After the simulation, we train the CNN model on the original dataset but test its performance on the protected/noisy datasets. Finally, we explore the portability issue when considering the electromagnetic (EM) side-channel and probe placing by human operators. Therefore, we build another dataset named XMEGA_EM with the eight XMEGA boards.

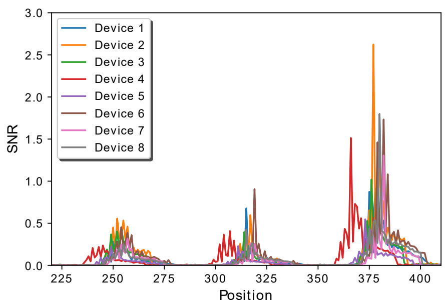
- **XMEGA** provides measurements of an unprotected AES-128 software implementation written in C language. To build a realistic cross-device dataset, we initialize the devices with different secret keys (fixed inside the device, see Figure 6a). To perform the acquisition, we insert a 10 Ω resistor between the microcontroller and GND. Then measuring the voltage drop across the resistor allows side-channel measurement in terms of power consumption. During measurement, the microcontroller is clocked at 2MHz and is connected to a Pico 3203D oscilloscope with a sampling rate of 125MS/s. The power traces are synchronized with a board-generated trigger, and a computer is used to control the whole measurement setup. For each execution, the computer generates a random 16-byte plaintext and transmits it to the microcontroller via UART. Upon receiving the corresponding ciphertext, the software then retrieves the waveform samples from the oscilloscope and saves them to disks. For each device, we acquired 25000 power traces for profiling and 5000 traces for the attack. Each trace consists of 500 points of interest (PoIs), corresponding to the SubBytes operation of the AES-128 algorithm in the first round. To highlight the difference in leakage of different devices, we calculate the SNR characterizing the first byte using 20000 traces labeled by the HW of the Sbox output. As clearly shown in Figure 6b, the leakage differs, and each board has its leakage characteristics. The reason that *Device 4* is shifted apparently in time could be explained by the imprecise clock. Expecting to have 625 points in 10 clock cycles, we only observed 616 points for *Device 4*. We use the HW model in the experiments; thus the leakage model is:

$$Y(k^*) = HW(Sbox[p_i \oplus k^*]), \tag{11}$$

  where p_i is a plaintext byte and we chose $i = 1$. The maximum SNR of this dataset reaches 2.6231.



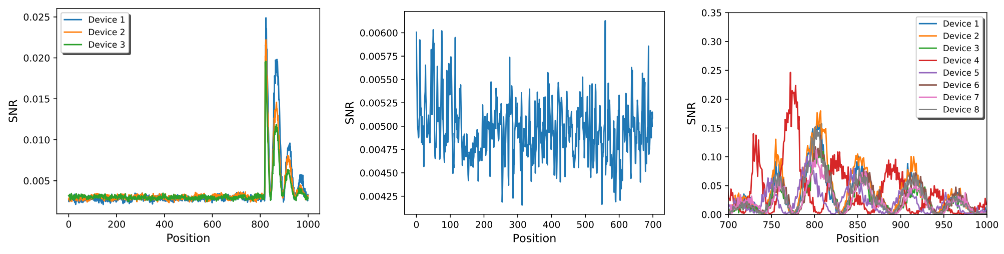
(a) SCA boards labeled with different keys



(b) SNR comparison, XMEGA

Figure 6: Eight XMEGA boards and their difference from the perspective of SNR.

[3]The datasets are available at https://github.com/CDPA-SCA/Cross-Device-Profiled-Attack.



(a) SAKURA_AES **(b)** ASCAD **(c)** XMEGA_EM

Figure 7: SNRs of several investigated datasets.

- **SAKURA_AES** is an unprotected AES-128 implemented on Xilinx Spartan-6 FPGA. We acquire the side-channel traces from three SAKURA-G evaluation boards, which use different secret keys. The AES-128 core is written in a round-based architecture, which takes 11 clock cycles to perform each encryption. Side-channel traces are measured by monitoring power waveforms on the core voltage of the main FPGA. Measurements are sampled using a Teledyne LeCroy Waverunner 610zi oscilloscope with a sampling rate of 500MS/s. A suitable and commonly used leakage model is the Hamming Distance (HD) model, which is related to the register writing in the last round, i.e.,

$$Y(k^*) = \overbrace{Sbox^{-1}[c_2 \oplus k^*]}^{\text{previous register value}} \oplus \overbrace{c_6}^{\text{ciphertext byte}}, \tag{12}$$

  where c_2 and c_6 correspond to the second and sixth bytes of the ciphertext respectively. The relation between c_2 and c_6 is given by the ShiftRows operation of AES. For each device, the profiling set contains 90000 traces and the test set contains 10000 traces, each trace with 1000 features. The measurements are relatively noisy and the measured SNR reaches 0.0248 (see Figure 7a).

- **ASCAD** is introduced in [PSB+18] to provide a benchmark to evaluate deep learning techniques in the context of SCA. The target platform is an 8-bit AVR microcontroller (ATmega8515) with an operating frequency of 4MHz, where a masked AES-128 is implemented. All traces are acquired with a sensor attached to an oscilloscope sampling at 2GS/s. This dataset contains three HDF5 files with different desynchronization levels. Each file contains 50000 traces for profiling and 10000 traces for the attack. Each EM trace consists of 700 PoIs corresponding to the masked Sbox for $i = 3$. Since the mask is unknown, we use the leakage model as:

$$Y(k^*) = Sbox[p_i \oplus k^*]. \tag{13}$$

  The SNR for the ASCAD dataset is around 0.8 under the assumption we know the mask [PSB+18], and it is 0.0061 if the mask is unknown (see Figure 7b).

- **XMEGA_EM** provides measurements of the previously mentioned XMEGA chips that run the unprotected software AES-128 encryption. Instead of measuring power consumption as before, we acquire side-channel traces by measuring the EM radiation emitted from the chips. This dataset is captured using a Langer LF-U 5 near-field probe, each time at a similar position but with human error. The side-channel traces are taken around the target Sbox computation in the first round, with the Teledyne LeCroy Waverunner 610zi oscilloscope sampling at 250MS/s. The dataset has 35000 traces (for each device) where each trace has 1500 features. We also use the HW of the Sbox output as the label. Thus the leakage model is the same as Equation 11. The maximum SNR of this dataset reaches 0.2464 (see Figure 7c).

We summarize the datasets used in our experiments as shown in Table 1.

Table 1: Statistical information and the partition of all considered datasets.

Dataset	# of features	Train & Validation	Fine-tune	Test
XMEGA	500	25000	100	5000
SAKURA_AES	1000	90000	200	10000
ASCAD	700	50000	200	10000
XMEGA_EM	1500	25000	100	10000

5 Experimental Results

5.1 Setttings

All the experiments are implemented in python using the *PyTorch* library, and are run on a GPU server equipped with 128GB RAM and a NVIDIA GeForce RTX 3090 with 24GB memory. Since the focus of this paper is cross-device SCA rather than optimizing the network architectures, we do not dive into the hyperparameter tuning game, but use similar CNN architectures and hyperparameters based on the state-of-the-art results [ZBHV20]. Table 2 summarizes the training parameters used in our experiments. The main difference with previous works is that we introduce a new hyperparameter λ that is defined in Equation 10. The sensitivity of this hyperparameter is discussed in Section 6.2. Throughout the experiments, we use Nt_{GE} to denote the number of traces needed to reach a constant GE of 1. To get a good estimation of Nt_{GE}, we perform the attack 100 times with randomly-selected attack traces and $\bar{N}t_{GE}$ is the average value.

Remark 1. We note that the fine-tuning phase requires a set of unlabeled traces that come from the attack dataset. Therefore, when attacking a different device for the first time, the amount of traces used for fine-tuning should be counted as the cost of the entire attack. Fortunately, this cost is affordable since the fine-tuned model can still outperform the pre-trained model, even taking the fine-tuning cost into consideration. Besides, this cost is one-time since the fine-tuned model can still be used for attacks even if the secret key is changed. Note that the number of traces used for fine-tuning can be further reduced, and there should exist a trade-off between the number of fine-tuning traces and a good estimate of MMD. In the rest of this paper, we report the performance ($\bar{N}t_{GE}$) of the models without counting the traces used for fine-tuning.

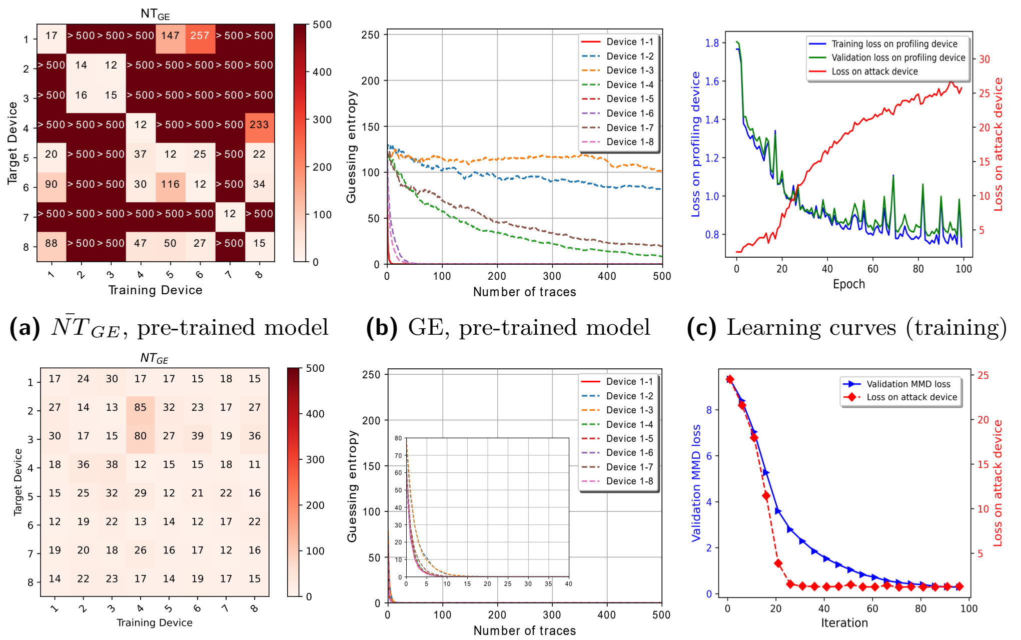
Table 2: Choice of hyperparameters.

Datasets	Optimizer	Learning Rate	Activation Function	Initialization	λ
XMEGA	Adam	0.001	SELU	He uniform	0.1
SAKURA_AES					0.05
ASCAD					0.1
XMEGA_EM					0.1

5.2 Study on Different Devices

5.2.1 XMEGA

We first evaluate our methodology on the XMEGA dataset. We use 20000 traces for training, 5000 traces for validation, 100 traces for fine-tuning, and 5000 traces for attacking. Since our focus is not to optimize the network architecture, we adopt the 9-classes HW model and use a simple CNN structure with three convolutional blocks followed by one fully-connected layer (see Appendix B Table 4). The hyperparameter λ is set to 0.1, and the learning rate is set to 0.001. All traces are preprocessed using horizontal standardization. Finally, we train the model for 100 epochs in the training phase, and another 15 epochs for fine-tuning. We set the batch size to be the number of traces used for fine-tuning. After training or fine-tuning, we save the model that achieves the lowest validation loss. For conciseness, we use *Device x-y* to denote that we train the model on device x and then try to recover the key of device y.



(a) $\bar{NT}_{GE}$, pre-trained model **(b)** GE, pre-trained model **(c)** Learning curves (training)

(d) $\bar{NT}_{GE}$, fine-tuned model **(e)** GE, fine-tuned model **(f)** Learning curves (fine-tune)

Figure 8: Results on XMEGA.

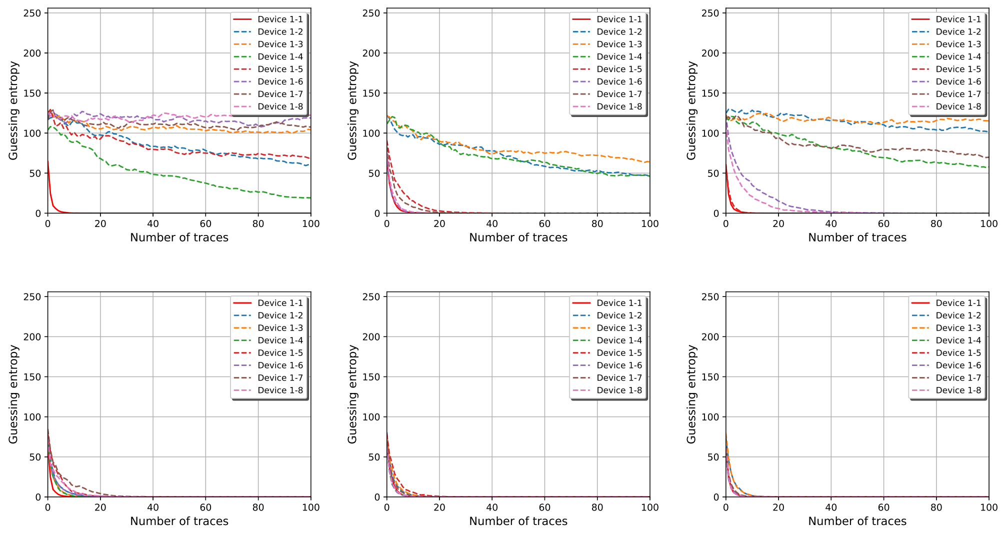
We show the performance of the pre-trained model on different devices in Figure 8a. It can be observed that the devices show different behaviors when they are attacked. Specifically, although the value of $\bar{N}t_{GE}$ for the single-device attack given on the diagonal is very small, it varies widely in the case of cross-device attacks. The matrix in Figure 8a is not perfectly symmetrical, which implies that the difficulties of task *Device x-y* and task *Device y-x* can be different. Then, we exemplarily present the GE curves of the pre-trained model (*Device 1-y*) in Figure 8b. We can observe that the GE for *Device 1-2* and *Device 1-3* is almost random using 500 attack traces, yet *Device 1-4* performs slightly better although the SNR between *Device 1* and *Device 4* is shifted also in time (see Figure 6b). It is not surprising since SNR can only reveal the PoIs and the signal quality, which cannot be optimal to characterize the real differences between devices. We can also infer from the results that, due to the shift-invariant nature of CNN, misalignment may not be the main problem in the cross-device scenarios. Since the bad performance of cross-device attacks may also be explained by overfitting[4] [BCH+20], we draw the learning curves of the training phase (task *Device 1-4*) in Figure 8c. We can see that the validation loss is highly consistent with the training loss when traces are from the same profiling device. In

[4]Here, overfitting means the deep learning model adapts to the training set too well but loses the ability to generalize to the validation and test datasets. A simple way to identify overfitting during the training phase is to compare the loss on the training and validation sets: if the training set loss is much lower than the validation set loss, then the model overfitted.

other words, overfitting is not observed when we use the same device to train and attack. However, the test loss increases apparently when we attack a different device (*Device 4*). From the above results, we could infer that the main reason for the poor performance is that the pre-trained model has learned device-specific features that cannot be safely generalized to other devices.

By applying the CDPA, the impact of device discrepancy is dramatically reduced. All the fine-tuned models could stably recover the key of a different device within 100 attack traces (see Figure 8d and Figure 8e). To further understand how the model learns during fine-tuning, we show the evolution of the test loss (cross-entropy loss on the target device) and the validation MMD loss in Figure 8f. We can observe that minimizing the MMD loss can effectively reduce the test loss on the target device, which confirms the necessity and effectiveness of our methodology. Besides, the test loss seems to converge faster (in 30 iterations) than the MMD loss. In other words, fine-tuning for a small number of iterations could be sufficient for getting a well-performed cross-device model, while further minimizing the MMD loss may not significantly improve the results.

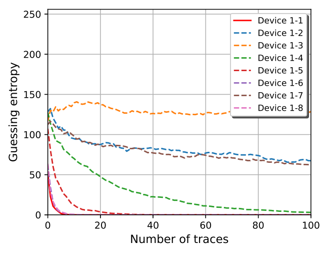
Impact of Preprocessing. To further understand how preprocessing methods affect the performance of the pre-trained and fine-tuned models, we present more results on the XMEGA dataset by varying the preprocessing techniques. As depicted in Figure 9, the portability issue is very obvious when no preprocessing is applied. Although preprocessing improves the pre-trained models in several cases, the effect of device discrepancy still cannot be eradicated. As we expected, CDPA improves the cross-device attacks significantly in all investigated situations.



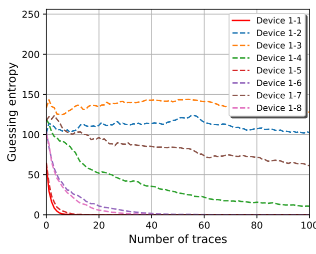
(a) No preprocessing **(b)** Horizontal scaling **(c)** Horizontal standardization

Figure 9: Influence of preprocessing on (top) pre-trained models and (bottom) fine-tuned models.

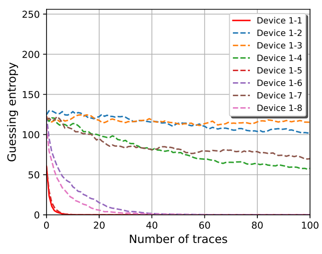
Results with Different Numbers of Profiling Traces. Increasing the amount of training data is efficient to prevent overfitting and can help us to obtain a more precise model [PW17]. Therefore, we investigate whether using more profiling traces can improve the performance of the pre-trained models in cross-device attack scenarios. We test with different numbers of profiling traces, and show the attack results of the pre-trained models in Figure 10. Interestingly, we can observe that increasing the number of training traces does not lead to better generalization when targeting different devices. Similar results were reported in [BCH+20]. This is reasonable since the appended profiling traces are acquired from the same device, which cannot guarantee an improved performance when we test on a target device with a different distribution.



(a) 5000 profiling traces



(b) 10000 profiling traces

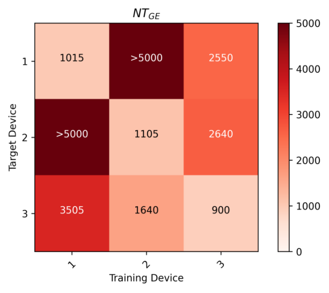


(c) 20000 profiling traces

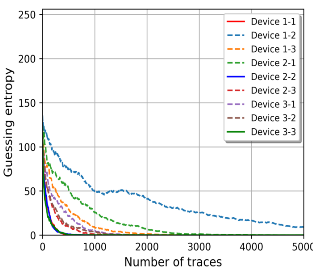
Figure 10: Influence of the number of profiling traces on pre-trained models.

5.2.2 SAKURA_AES

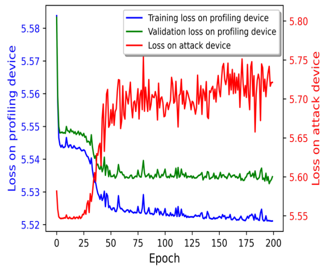
Unlike the above described dataset, the SAKURA_AES dataset provides measurements of an unprotected hardware implementation of AES-128 on FPGA. We use 85000 traces for training, 5000 traces for validation, 200 traces for fine-tuning, and 10000 traces for attacking. As before, the learning rate is set to 0.001, and traces are preprocessed using horizontal standardization. The profiling phase runs for 200 epochs with a batch size of 200. The fine-tuning phase runs for only 15 epochs with the λ set to 0.05. The MMD loss is computed on the flattened layer and the fully-connected layer.



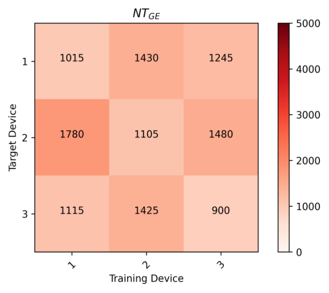
(a) $\bar{NT}_{GE}$, pre-trained model



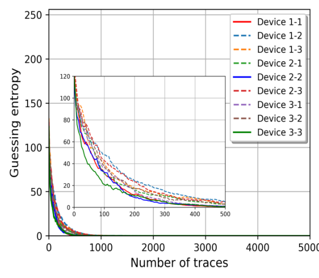
(b) GE, pre-trained model



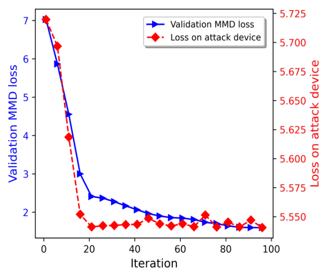
(c) Learning curves (training)



(d) $\bar{NT}_{GE}$, fine-tuned model



(e) GE, fine-tuned model



(f) Learning curves (fine-tune)

Figure 11: Results on SAKURA_AES.

Since the SNR of this dataset is relatively small, our pre-trained models require around 1000 traces to successfully recover the key of the same device (see Figure 11a and Figure 11b). When we apply the pre-trained models to other devices, the required number of attack traces is likely to double. As before, no obvious overfitting is detected on the same device, whereas the test loss (task *Device 1-2*) increases rapidly as the model learns on the profiling traces (see Figure 11c). The results after fine-tuning are shown in Figure 11d and Figure 11e. We can observe that all the cross-device experiments get improved after applying CDPA. Most fine-tuned models achieve almost similar performance as using the same device for attacking. Consequently, CDPA is also suitable and efficient for deep learning-based SCA on hardware implementations.

Remark 2. For the experiments on SAKURA_AES and ASCAD, we adopt batch normalization [IS15] after each convolutional block to make the optimization easier and faster [STIM18]. In general, batch normalization contains two non-trainable weights that get updated during the training phase. These are the variables tracking the mean and variance of the inputs. Whereas, during the fine-tuning phase, the batch normalization layers should be kept frozen (in inference mode). Otherwise, the updates applied to the non-trainable weights will suddenly destroy what the model has learned [AAB$^+$15]. In our experiments, we freeze the batch normalizations using the *model.eval()* method provided by the *PyTorch* library.

5.3 Extended Applications to Other Portability Issues

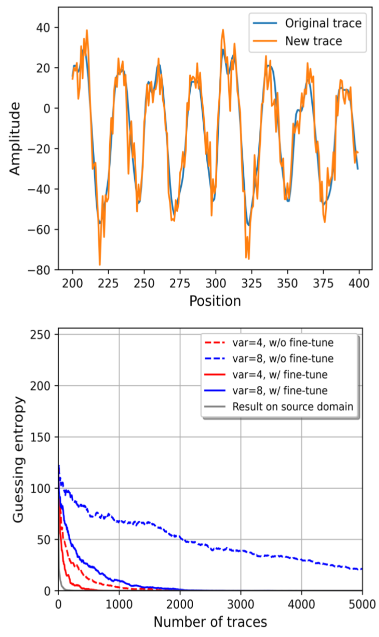
As mentioned in Section 3, the portability issue exists not only in different devices. The variance in implementations or settings of acquisition can also lead to bad attacking performance. To this end, we investigate two other scenarios that are very common in practice. Our first study simulates different implementations by adding artificial countermeasures/noise to the original dataset. After the simulation, we train the CNN model on the original dataset (source domain) and evaluate its performance on the deformed datasets (target domain). These experiments simulate a complex attack scenario that the target device is treated as a black box that can turn on side-channel countermeasures. Finally, we explore the portability issue in the EM analysis, where the measurements are very sensitive to probe placement (position, distance, and orientation).

5.3.1 Addition of Countermeasures/Noise

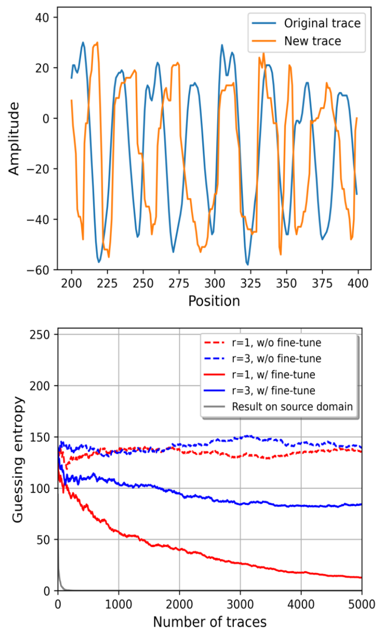
Herein, we consider two types of countermeasures/noise including Gaussian noise and clock jitters. All the experiments are performed based on the ASCAD dataset.

- **Gaussian Noise** is the most common type of noise existing in side-channel measurements. The source of Gaussian noise can come from data buses, transistors, oscilloscopes, or even the work environment. To demonstrate the influence of the addition of noise, we build the target domain by adding a normal-distributed random value $r \sim \mathcal{N}(0, var)$ to each point of the trace. As a result, Gaussian noise distorted the shape of the original traces in the amplitude domain (see Figure 12a (top)).

- **Clock Jitters** is a classical hardware countermeasure implemented by introducing the instability in the clock [CDP17]. Herein, we simulate the clock jitters following the work of [WP20] by randomly adding and removing points with a pre-defined range. Specifically, when scanning each point in the trace, n points will be added to the trace if n is larger than zero. Otherwise, the following n points will be removed, where $n \in [-r, r]$. An example of the zoom-in viewed trace after deformation is given in Figure 12b (top).



(a) Gaussian noise



(b) Clock jitters

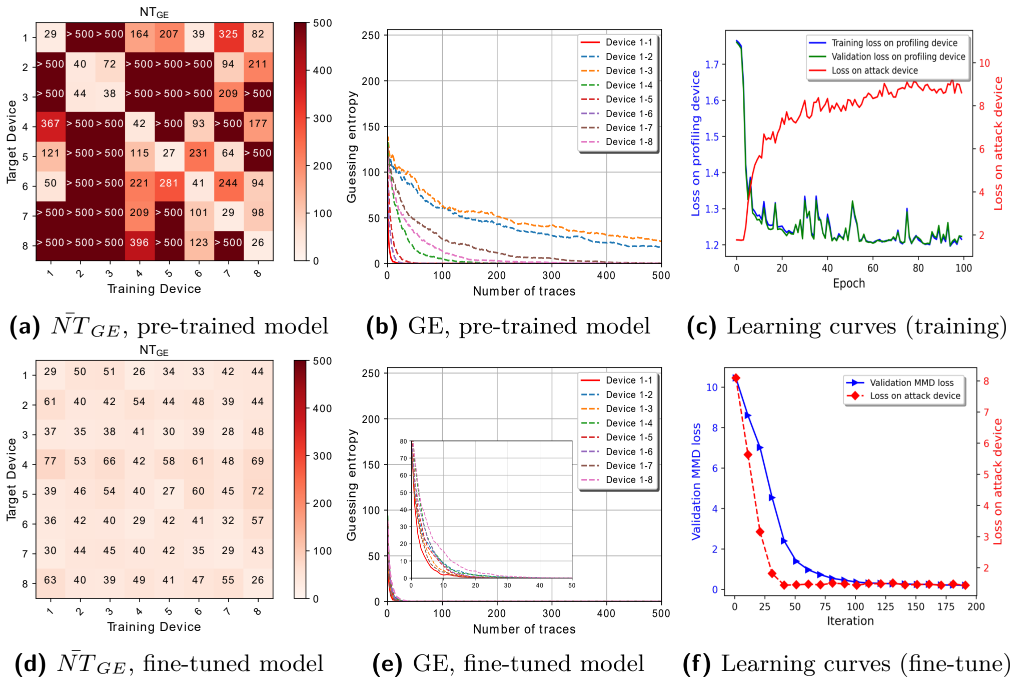
Figure 12: Example traces (top) and attack results (bottom).

We use a similar network architecture as the one used in [ZBHV20] for the ASCAD dataset. Besides, we use 45000 traces for training, 5000 traces for validation, 200 traces for fine-tuning, and 10000 traces for attacking. The profiling phase runs for 100 epochs on the original dataset, while the fine-tuning phase runs for 15 epochs, with a batch size of 200. The attacking results are summarized in Figure 12 (bottom). As we can see, although the pre-trained model performs well ($\bar{N}t_{GE}$ reaches 345) in the original ASCAD dataset, more than 5000 traces are required to reach a GE of 0 when the variance of Gaussian noise is set to 8. After applying the CDPA, the attack performance on the noisy target domain is significantly improved. We can infer from the results that CNN may not generalize well if only clean traces are fed to the network. However, fine-tuning using a small number of (unlabeled) noisy traces can unleash the power of CNNs and drive the network to learn domain-invariant features. The pre-trained model does not work after adding the clock jitters, which is not surprising as too much randomness was introduced. Although we still cannot recover the key within 5000 traces after fine-tuning, the GE curves decrease with more attack traces.

5.3.2 Portability of Electromagnetic Probe Placement

Apart from power analysis, EM-based SCA is becoming increasingly popular due to its non-invasive and spatially flexible nature. Note that EM measurements are very sensitive to probe placement. However, when we consider the realistic profiled attack scenario, the probe must be moved from the profiling device to the target device. Hence, there is always a slight difference in the probe placement caused by human error due to the position distance and orientation.

To investigate the impact of human error, we perform more cross-device experiments on the XMEGA_EM dataset. This dataset is captured from eight different devices with different keys, each time at a similar probe position but with human error. We use the same CNN architecture and training parameters as the XMEGA dataset. The performance



(a) $\bar{NT}_{GE}$, pre-trained model **(b)** GE, pre-trained model **(c)** Learning curves (training)

(d) $\bar{NT}_{GE}$, fine-tuned model **(e)** GE, fine-tuned model **(f)** Learning curves (fine-tune)

Figure 13: Results on XMEGA_EM.

of the pre-trained model is depicted in Figure 13a and Figure 13b. We can observe that the result matrix is very similar to the attack result of the XMEGA dataset, but not exactly the same. This difference is inevitable since the EM traces have different features and signal quality from the power traces. As we expected, the fine-tuned models outperform the pre-trained models significantly, which can stably recover the correct key within 80 traces (see Figure 13d and Figure 13e). In Figure 13f, we see that the evolution of MMD loss is again highly consistent with the test loss (task *Device 1-2*), which confirms our previous results.

5.4 Computation Cost

We present the computation cost of training and fine-tuning in Table 3. The learning time of each epoch is mainly determined by the size of training sets, the batch size, and the length of raw traces. We can observe that the epoch time for fine-tuning is approximately twice that of training. This is reasonable since more traces are processed and an additional MMD loss is calculated in the fine-tuning phase. In addition, the time cost is still affordable. For example, if we run the fine-tuning phase for 15 epochs, this process can be completed within two minutes for all considered datasets.

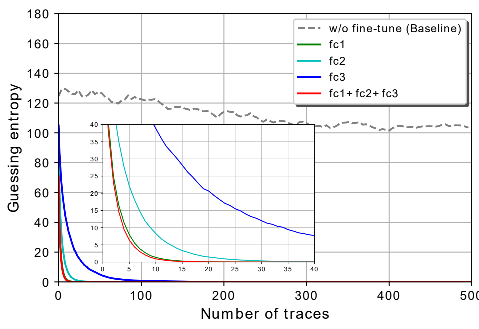
Table 3: Summary of the computation cost.

Dataset	Batch size	Epoch time (Training)	Epoch time (Fine-tuning)
XMEGA	100	0.71 s	1.24 s
SAKURA_AES	200	2.85 s	6.04 s
ASCAD	200	1.47 s	3.52 s
XMEGA_EM	100	0.76 s	1.37 s

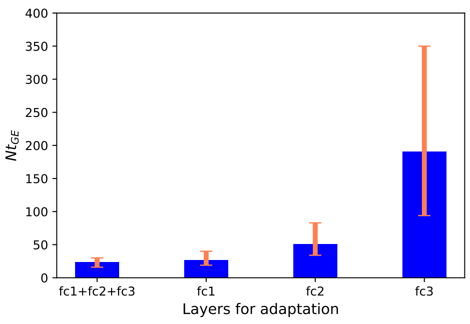
6 Discussion

6.1 Effect of Adaptation Layers

In order to further understand how the location of the adaptation layers affects the output, we conduct a series of experiments on the XMEGA dataset (task *Device1-2*) with different adaptation layers. We use a CNN network whose classifier part has three fully-connected layers (fc1–fc3). We first fine-tune the network using only a single layer, and then compare it with the result of using all three layers. As before, our network is fine-tuned for 15 epochs. The results are shown in Figure 14.



(a) Guessing Entropy



(b) Variation of $\bar{N}t_{GE}$

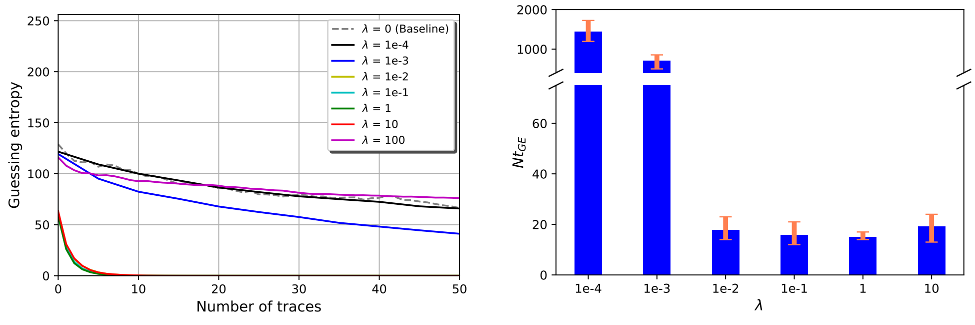
Figure 14: Attacking results of different adaptation layers (average over 20 fine-tuned models).

An obvious observation is that the CDPA still works even a single layer is used for minimizing the MMD loss. Another observation is that the deeper the layer, the more difficult it seems to learn domain-invariant features. This is reasonable since the features obtained in higher layers must depend greatly on the specific dataset, which are not safely transferable to novel domains. Still, using all the layers of the classifier part is a good trade-off, which usually brings better results than using a single adaptation layer.

6.2 Effect of the Penalty Parameter λ

The hyperparameter λ in Equation 10 determines how strongly we would like to confuse the source and target domains. Intuitively, setting the λ too small can cause the MMD regularizer to have no effect on the learned representation, yet setting the λ too large will regularize too heavily and may result in a degenerate representation in which all features are too close together. To further understand the sensitivity of parameter λ, we give an illustration of the variation of the *GE* as $\lambda \in \{0, \text{1e-4}, \text{1e-3}, \text{1e-2}, \text{1e-1}, 1, 10, 100\}$ on XMEGA (task *Device1-4*) in Figure 15. The network is fine-tuned for 15 epochs with a batch size of 200.

We can observe that setting the λ too small ($\lambda = \text{1e-4}$) or too large ($\lambda = 100$) may not improve the attack, which is consistent with our analysis. In other cases of λs, all the fine-tuned models improve the results significantly. Although there is usually a wide range of λ where the pre-trained models get improved, a good empirical choice is to start with a relatively small value (e.g., 1e-2), especially when the SNR of the dataset is small. A smaller value of λ means that the optimizer should put more effort into the tough classification task. If it is observed that the reduction of MMD loss is not significant or too slow, we can gradually increase the value of λ to speed up the fine-tuning process. In practice, we can automatically select the parameter λ on a validation set (consists of source-labeled traces and target-unlabeled traces) by jointly assessing the cross-entropy



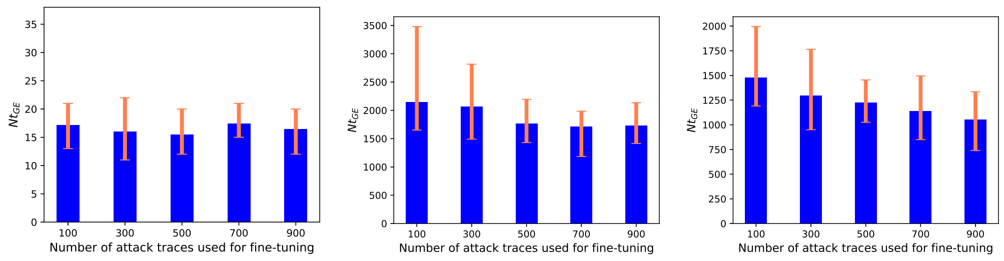
(a) Guessing Entropy **(b)** Variation of $\bar{N}t_{GE}$

Figure 15: Attacking results of different λ (average over 20 fine-tuned models).

loss and MMD loss. For example, a small classification loss and a large MMD loss may indicate that you are using a relatively small λ. On the contrary, a large classification loss and a very small MMD loss may result from a large value of λ.

6.3 Effect of the Number of Traces for Fine-tuning

Although fine-tuning with MMD loss helps us obtain a robust model, we need a set of attack traces to estimate the MMD. Despite the fact acquiring multiple unlabeled traces from the target device is not a strong assumption, it is still meaningful to figure out how many traces are appropriate in practice when we fine-tune the network. Therefore, we conduct a series of experiments with the number of traces varying in {100, 300, 500, 700, 900} with a batch size of 100. Finally, we fine-tune the networks for 15 epochs to ensure that the model can learn the domain discrepancy well. The results on XMEGA (*Device 1-4*), SAKURA_AES (*Device 1-2*), and ASCAD (with Gaussian noise) are depicted in Figure 16. It can be observed that 100 traces (as small as the batch size) are sufficient for the fine-tuning phase. We remark that MMD focuses on the domain discrepancy of the profiling and attack traces instead of the classification task itself. So, the results are not surprising since 100 unlabeled traces can provide sufficient information that is distinguishable from the source domain. From the results on SAKURA_AES and ASCAD, we can conclude that using more traces could lead to a more stable and robust fine-tuned model. This is reasonable since more traces help us to obtain a more precise estimate of MMD. The result on XMEGA seems to be less affected since an unprotected implementation is definitely easier to learn and transfer.



(a) XMEGA **(b)** SAKURA_AES **(c)** ASCAD

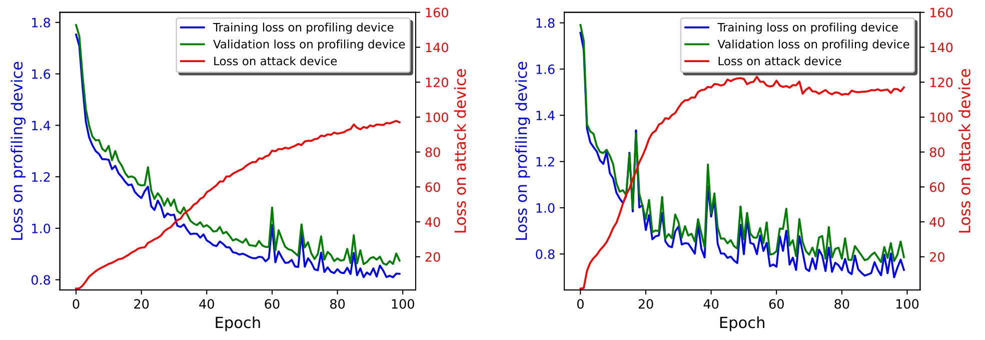
Figure 16: Attacking results of different number of fine-tuning traces (average over 20 fine-tuned models).

6.4 Comparison with Other Promising Techniques

Data Augmentation has been proven successful in enhancing the robustness of CNN models [CDP17, KPH$^+$19]. For example, artificially generating new profiling traces by deforming (shifting) those previously acquired can help the network to learn shift-invariant features. However, data augmentation focuses on enlarging the dataset without considering the real difference between the profiling and attack traces. So, it is unclear what kind of augmentation is optimal to introduce, from the perspective of attackers.

The recently introduced denoising convolutional autoencoder (CAE) is promising to remove noise (e.g., Gaussian noise and desynchronization) of raw traces [WP20]. By considering the variance between different devices as noise, the CAE may be hopeful to remove it. However, the training of CAE requires noisy-clean trace pairs, which is challenging to obtain in realistic settings. Similar to data augmentation, the simulated noise could not be optimal to characterize the differences between different devices.

Since the bad performance of the pre-trained model could also be explained by overfitting, several intuitive techniques that prevent overfitting may also be helpful to improve the performance. These techniques typically include restricting model complexity and early stopping. We, therefore, test two very simple CNN architectures (details in Figure 17) that differ only in the number of fully-connected layers. We investigate whether a less complex architecture could lead to better generalization. As shown in Figure 17, the test loss (on the attack device) increases during training for both CNN architectures. In other words, using a less complex architecture cannot improve the performance of the pre-trained model significantly. Besides, in the cross-device scenario, it is difficult to estimate a proper network setting and to identify overfitting without observing the labeled traces coming from the target device. As we have shown in the learning curves, the validation loss and the test loss may behave in the opposite way. Therefore, observing the validation loss may be insufficient to identify this kind of overfitting, which means early stopping may not be suitable for cross-device scenarios.



(a) CNN with 1 conv. block (8 filters, ks=50) and 1 fully-connected layer (20 neurons).

(b) CNN with 1 conv. block (8 filters, ks=50) and 3 fully-connected layers (20 neurons).

Figure 17: Learning curves for two different groups of CNN hyperparameters on the same dataset (XMEGA).

Note that we are not the first to utilize additional unlabeled traces to build a stronger model. The authors in [PHJ$^+$18] investigate the profiled attack in the semi-supervised learning scenario, where unlabeled traces are classified and then added to the training set to enhance the pre-trained model. They show that semi-supervised learning significantly helps TA and its pooled version. However, semi-supervised learning relies on strong assumptions (e.g., smoothness assumption, cluster assumption, and manifold assumption [vEH20]). If we consider a target domain with a different distribution, the unlabeled traces is very likely to be assigned a wrong guessed label. Consequently, these traces with incorrect labels may not yield an improvement but destroy what the model has learned.

Apart from the above DL techniques, some special tricks, like zero-mean unit-variance normalization [MBTL13] and multi-devices profiling [CK14, DGD$^+$19, GDD$^+$19, BCH$^+$20, WdHG$^+$20] are also helpful to attack a different device. Zero-mean unit-variance normalization is a kind of preprocessing technique but requires the traces to be well-aligned. Multi-devices profiling is currently one of the most popular methods to overcome the portability issue. It assumes that a powerful attacker possesses multiple profiling devices and can capture as many side-channel traces as necessary during the profiling stage. The end goal is to use the model generated during the profiling phase (with multiple devices) to recover the secret key from an unseen device. This method is effective since the network can see more training data with different distributions. Therefore, the more profiling devices used, the more robust the model becomes. However, it still cannot promise that one profiling device is very close to the target one. Besides, in realistic settings, possessing multiple profiling devices cannot always be satisfied. To relaxes the assumption of possessing multiple devices, the proposed CDPA takes advantage of the domain information of the profiling and attack traces. Instead of introducing more labeled traces to the profiling set, CDPA focuses directly on the variance between the profiling and target devices by adopting the MMD loss. The cost of CDPA is a few additional epochs of fine-tuning with a small number of unlabeled attack traces, which is affordable compared with the cost of using multiple profiling devices.

7 Conclusion and Future Work

This paper focuses on addressing the open question of portability in profiled SCA, using transfer learning techniques (specifically, unsupervised domain adaptation). We consider the issue of portability as domain discrepancy, and we propose a new attack strategy called CDPA to eliminate it. CDPA introduces a fine-tuning phase before the traditional attack phase. The kernel idea is to adjust the pre-trained model such that it eventually learns a representation that is not only discriminative but also domain-invariant. To achieve this, we adopt the MMD loss as a penalty of the cross-entropy loss function, which can be easily calculated and embedded in a common CNN architecture.

We evaluate the performance of our strategy on eight XMEGA chips and three SAKURA-G boards with AES-128 implementations. Consequently, CDPA can improve the attack performance by $> 20\times$ and could even turn an impossible attack into a reality. Besides considering different devices, we also explore the ability of the CDPA to remove the impact of adding countermeasures/noise. Our results on the ASCAD dataset show that CNN may not generalize well if only clean traces are fed. However, fine-tuning using a limited number of (unlabeled) desynchronized/noisy traces could unleash the power of CNNs and drive the network to learn domain-invariant features. Finally, we show how portability issues also arise when considering the EM side-channel and probe placing by human operators. Subsequently, we demonstrate how CDPA helps the performance in such a scenario.

For future work, it would be interesting to see how CDPA performs in a cross-family attack scenario, which is a more challenging task. Another direction is to explore other transfer learning techniques that are more appropriate in the context of SCA. We believe that this work will pave the way for the realistic study on cross-device profiled SCA.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No.62072307). We thank the anonymous reviewers and the shepherd for their valuable suggestions on

how to improve this paper.

References

[AAB$^+$15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[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. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020.* The Internet Society, 2020.

[BL12] 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 - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2012.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013.

[CK14] Omar Choudary and Markus G. Kuhn. Template attacks on different devices. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 179–198. Springer, 2014.

[CK18] Marios O. Choudary and Markus G. Kuhn. Efficient, portable template attacks. *IEEE Trans. Inf. Forensics Secur.*, 13(2):490–501, 2018.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DGD$^+$19] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-deepsca: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, page 134. ACM, 2019.

[EG12] M. Abdelaziz Elaabid and Sylvain Guilley. Portability of templates. *J. Cryptogr. Eng.*, 2(1):63–74, 2012.

[GBR$^+$12] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773, 2012.

[GDD$^+$19] Anupam Golder, Debayan Das, Josef Danial, Santosh Ghosh, Shreyas Sen, and Arijit Raychowdhury. Practical approaches toward deep-learning-based cross-device power side-channel attack. *IEEE Trans. Very Large Scale Integr. Syst.*, 27(12):2720–2733, 2019.

[GGH20] Christophe Genevey-Metat, Benoît Gérard, and Annelie Heuser. On what to learn: Train or adapt a deeply learned profile? *IACR Cryptol. ePrint Arch.*, 2020:952, 2020.

[GSS$^+$12] Arthur Gretton, Bharath K. Sriperumbudur, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, and Kenji Fukumizu. Optimal kernel choice for large-scale two-sample tests. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1214–1222, 2012.

[HGM$^+$11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptogr. Eng.*, 1(4):293–302, 2011.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2012.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[KKKR18] K Kim, TH Kim, T Kim, and S Ryu. Aes wireless keyboard: Template attack for eavesdropping. *Black Hat Asia, Singapore*, 2018.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[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 Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):148–179, 2019.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: an approach based on machine learning. *Int. J. Appl. Cryptogr.*, 3(2):97–115, 2014.

[LCWJ15] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 97–105. JMLR.org, 2015.

[LPB$^+$15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.

[LZWJ16] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I. Jordan. Unsupervised domain adaptation with residual transfer networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 136–144, 2016.

[MBTL13] David P. Montminy, Rusty O. Baldwin, Michael A. Temple, and Eric D. Laspe. Improving cross-device attacks using zero-mean unit-variance normalization. *J. Cryptogr. Eng.*, 3(2):99–110, 2013.

[MMR20] Thorben Moos, Amir Moradi, and Bastian Richter. Static power side-channel analysis - an investigation of measurement factors. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(2):376–389, 2020.

[MOM12] Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors. *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*. Springer, 2012.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016.

[MT10] Roel Maes and Pim Tuyls. Process variations for security: Pufs. In Ingrid M. R. Verbauwhede, editor, *Secure Integrated Circuits and Systems*, Integrated Circuits and Systems, pages 125–141. Springer, 2010.

[PCP20] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):337–364, 2020.

[PHJ$^+$18] Stjepan Picek, Annelie Heuser, Alan Jovic, Karlo Knezevic, and Tania Richmond. Improving side-channel analysis through semi-supervised learning. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*, volume 11389 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2018.

[PSB$^+$18] Emmanuel Prouff, Rémi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptol. ePrint Arch.*, 2018:53, 2018.

[PW17] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[RMH$^+$19] Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younès Bennani. *Advances in Domain Adaptation Theory.* Elsevier, 2019.

[RO04] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In Chae Hoon Lim and Moti Yung, editors, *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*, volume 3325 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2004.

[RSV$^+$11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2011.

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 2488–2498, 2018.

[TAM20] Dhruv Thapar, Manaar Alam, and Debdeep Mukhopadhyay. Transca: Cross-family profiled side-channel attacks using transfer learning on deep neural networks. *IACR Cryptol. ePrint Arch.*, 2020:1258, 2020.

[THZ$^+$14] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *CoRR*, abs/1412.3474, 2014.

[vEH20] Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Mach. Learn.*, 109(2):373–440, 2020.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):147–168, 2020.

[WdHG$^+$20] Lennert Wouters, Jan Van den Herrewegen, Flavio D. Garcia, David F. Oswald, Benedikt Gierlichs, and Bart Preneel. Dismantling dst80-based immobiliser systems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):99–127, 2020.

[WKW16] Karl R. Weiss, Taghi M. Khoshgoftaar, and Dingding Wang. A survey of transfer learning. *J. Big Data*, 3:9, 2016.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):389–415, 2020.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3320–3328, 2014.

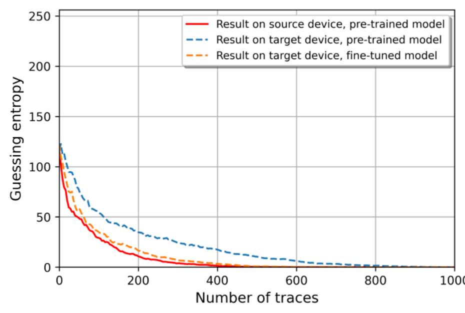
[ZBD$^+$20] Gabriel Zaid, Lilian Bossuet, François Dassance, Amaury Habrard, and Alexandre Venelli. Ranking loss: Maximizing the success rate in deep learning side-channel analysis. *IACR Cryptol. ePrint Arch.*, 2020:872, 2020.

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):1–36, 2020.

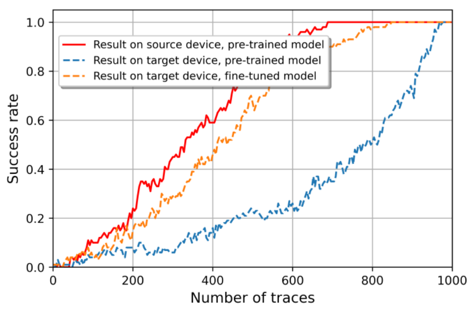
[ZZN$^+$20] Jiajia Zhang, Mengce Zheng, Jiehui Nan, Honggang Hu, and Nenghai Yu. A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):73–96, 2020.

A Additional Results on CHES CTF 2018

We also give the result on the CHES Capture-the-flag (CTF) AES-128 dataset, released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). The traces refer to a masked AES-128 implementation running on 32-bit STM microcontrollers. Each raw trace consists of 650000 samples covering the whole encryption of AES. As the raw trace is too long to be processed, Perin et al. [PCP20] conduct a feature selection and reduce the length to 2200 samples. In our experiments, we use the dataset and the network architecture provided by Perin et al. The profiling set consists of 45000 traces (we use 43000 traces for training, 1000 traces for validation, and 1000 traces for testing). The target set has 5000 traces acquired from a different device with a different fixed key. The profiling phase runs for 15 epochs with a batch size of 200. After training, we use 200 attack traces for fine-tuning (5 epochs) with the λ set to 0.1. The attack results are shown in Figure 18. We can observe that although the device discrepancy exists, it is not as apparent as what we present in other experiments. Again, we see that CDPA improves the performance on the target device. We refer to the source code in our Github repository for more details of this attack.



(a) Guessing entropy



(b) Success rate

Figure 18: Attacking results on CHES CTF 2018.

B The Detailed Configuration of the Models

Table 4: CNN architecture of the XMEGA dataset.

Layer	Filter size	# of filters	Pooling stride	# of neurons
Conv block	1	16	2	-
Conv block	50	32	50	-
Conv block	3	64	2	-
Flatten	-	-	-	-
Fully-connected	-	-	-	20

Table 5: CNN architecture of the SAKURA_AES dataset.

Layer	Filter size	# of filters	Pooling stride	# of neurons
Conv block	1	8	2	-
Batch normalization	-	-	-	-
Conv block	11	16	11	-
Batch normalization	-	-	-	-
Conv block	3	128	3	-
Batch normalization	-	-	-	-
Flatten	-	-	-	-
Fully-connected	-	-	-	2

Table 6: CNN architecture of the ASCAD dataset.

Layer	Filter size	# of filters	Pooling stride	# of neurons
Conv block	1	32	2	-
Batch normalization	-	-	-	-
Conv block	50	64	50	-
Batch normalization	-	-	-	-
Conv block	3	128	2	-
Batch normalization	-	-	-	-
Flatten	-	-	-	-
Fully-connected * 3	-	-	-	20