A Second Look at the Portability of Deep Learning Side-Channel Attacks over EM Traces

Mabon Ninan
University of Cincinnati
Cincinnati, OH, USA
ninanmm@mail.uc.edu

Evan Nimmo
University of Cincinnati
Cincinnati, OH, USA
nimmoem@mail.uc.edu

Shane Reilly
University of Cincinnati
Cincinnati, OH, USA
reillysp@mail.uc.edu

Channing Smith*
College of Charleston
Charleston, SC, USA
smithcs@g.cofc.edu

Wenhai Sun
Purdue University
West Lafayette, IN, USA
whsun@purdue.edu

Boyang Wang
University of Cincinnati
Cincinnati, OH, USA
boyang.wang@uc.edu

John M. Emmert
University of Cincinnati
Cincinnati, OH, USA
john.emmert@uc.edu

ABSTRACT

Deep learning side-channel attacks can recover encryption keys on a target by analyzing power consumption or electromagnetic (EM) signals. However, they are less portable when there are domain shifts between training and test data. While existing studies have shown that pre-processing and unsupervised domain adaptation can enhance the portability of deep learning side-channel attacks given domain shifts over EM traces, the findings are limited to easy targets (e.g. 8-bit microcontrollers).

In this paper, we investigate the portability of deep learning side-channel attacks over EM traces acquired from more challenging targets, including 32-bit microcontrollers and EM traces with random delay. We study domain shifts introduced by the combination of hardware variations, distinct keys, and inconsistent probe locations between two targets. In addition, we perform comparative analyses of multiple existing (and new) pre-processing and unsupervised domain adaptation methods. We conduct a series of comprehensive experiments and derive three main observations. (1) Pre-processing and unsupervised domain adaptation methods can enhance the portability of deep learning side-channel attacks over more challenging targets. (2) The effectiveness of each method, however, varies depending on the target and probe locations in use. In other words, observations of a method on easy targets do not necessarily generalize to challenging targets. (3) None of the methods can constantly outperform others. Moreover, we highlight two types of pitfalls that could lead to over-optimistic attack results in cross-device evaluations. We also contribute a large-scale public dataset (with 3 million EM traces from 9 probe locations over multiple targets) for benchmarking and reproducibility of side-channel attacks tackling domain shifts over EM traces.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**.

KEYWORDS

Side-channel analysis, deep learning

ACM Reference Format:
Mabon Ninan, Evan Nimmo, Shane Reilly, Channing Smith, Wenhai Sun, Boyang Wang, and John M. Emmert. 2023. A Second Look at the Portability of Deep Learning Side-Channel Attacks over EM Traces. In *Proceedings of This is the author's version of the work. It is posted here for your personal use. Not for redistribution. (RAID'24).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3678890.3678900

1 INTRODUCTION

Side-channel attacks can reveal encryption keys on a target (e.g., a microcontroller or FPGA) by analyzing power consumption or electromagnetic (EM) signals [4, 7, 19]. The attack is feasible as correlations exist between power consumption and intermediate results of encryption. Deep learning side-channel attacks train a neural network by acquiring training data from a training target and then recover keys on a test target with the trained neural network. Existing research [1, 25, 30, 34] have shown that deep learning side-channel attacks can outperform traditional side-channel attacks, such as Template Attacks [7], especially when countermeasures (such as masking and random delay) are applied.

On the other hand, deep learning side-channel attacks are less effective or even fail to recover keys when there are domain shifts between training data and test data [2, 10, 15, 38, 54, 55]. These domain shifts can be introduced by various factors between the training target and test target, such as hardware manufacturing variations [2], distinct keys [2], various software settings [47], and

*The work was done when the author was at the University of Cincinnati.

RAID'24, 15.00

ACM ISBN 978-1-4503-9859-6/23/05
https://doi.org/10.1145/3678890.3678900

	DFT	PCA	LDA	MVN	MMD-DA	ADA	On-the-Fly FT	8-bit target	16-bit target	32-bit target	Random delay
JCE13 [28]	×	×	×	√	×	×	×	×	√	×	×
JETC22 [9]	√	√	√	×	×	×	×	√	×	×	×
CHES21 [6]	×	×	×	×	√	×	×	√	×	×	×
Ours	√	√	√	√	√	√	√	√	×	√	√

Table 1: High-level comparison between ours and previous studies on cross-device side-channel attacks over EM traces (√: investigated; ×: not investigated).

inconsistent probe positions [9]. For side-channel attacks over EM signals, domain shifts can be primarily caused by the combination of discrepancies in keys, hardware, and probe locations. Specifically, it is difficult to manually replicate exact probe locations (as well as heights) over the tiny surface of a microcontroller (e.g., 0.7 cm × 0.7 cm) between training data acquisition and test data acquisition. An example of discrepancies on EM signals caused by probe locations is presented in Fig. 1 (in Sec. 3). In the context of side-channel attacks, *portability* indicates the effectiveness of recovering keys in cross-device evaluations given domain shifts between training and test data.

Two recent studies [6, 9] have proposed to leverage pre-processing [9] and unsupervised domain adaptation [6] to enhance the portability of deep learning side-channel attackers over EM traces, where domain shifts are caused by keys, hardware, and probe locations. However, the findings from these existing studies are ***limited to easy targets***, such as 8-bit microcontrollers running AES (Advanced Encryption Standard) with no countermeasures. While there are studies [14, 54] examined domain shifts of EM traces from more challenging targets (e.g., 32-bit microcontrollers), their methods are limited to supervised domain adaptation (i.e., assuming an attacker knows the key even before the attack, which is infeasible for real-world attackers). In addition, ***no large-scale public datasets*** are available for the research community to benchmark, reproduce, and compare different methods addressing domain shifts caused by keys, hardware, and probe locations over EM traces.

In this study, we take a second look at domain shifts between training and test data and investigate whether pre-processing and unsupervised domain adaptation can promote the portability of deep learning side-channel attacks over EM traces. Specifically, we focus on domain shifts that are introduced by discrepancies in keys, hardware, and probe locations. We explore more challenging targets (i.e., cases with lower side-channel leakage), including 32-bit microcontrollers and/or EM traces with random delay[1] We perform comparative analysis of existing (and new) pre-processing and unsupervised domain adaptation methods given these domain shifts. Pre-processing aims to mitigate domain shifts by identifying significant features or transforming features into a different space before passing data to a neural network. It addresses training data and test data *independently*. On the other hand, domain adaptation tackles domain shifts by *jointly* adjusting training data and test data either during the training or after the training.

Our main efforts and findings are summarized below. We also provide a high-level comparison between our work and three existing studies [6, 9, 28] that are most close to ours in Table 1.

- We leverage a customized acquisition setup built upon ChipWhisperer and establish a large-scale dataset of EM traces across 9 probe locations from multiple microcontrollers, including 8-bit AVR XMEGA and 32-bit ARM STM32F3, running unmasked AES-128. In addition, we generate EM traces with random delay. The research community can leverage our dataset to ① reproduce our findings and ② evaluate future methods that can more effectively tackle domain shifts over EM traces across different probe locations.
- We investigate and compare three existing pre-processing methods, including Discrete Fourier Transform (DFT), Principle Component Analysis (PCA), and Linear Discriminant Analysis (LDA). The performance of these methods has been discussed over EM traces from 8-bit microcontrollers in existing research [9] but not over EM traces from 32-bit microcontrollers or EM traces with random delay.
- We examine and compare four domain adaptation methods, including Zero-Mean Unit-Variance Normalization (MVN)[2] [28], Maximum-Mean-Discrepancy Domain Adaptation (MMD-DA) [6], Adversarial Domain Adaptation (ADA) [47], and On-the-Fly Fine Tuning (FT). All the four methods are *unsupervised* — do not require labeled test traces in the domain adaptation phase. We propose On-the-Fly Fine Tuning in this paper while the other three were proposed in previous studies to address domain shifts from less challenging targets [6, 28] or power traces [6, 47].
- We highlight two types of pitfalls, denoted as Type I and Type II, that are often ignored in existing studies. Type I pitfalls applying supervised methods with labeled test traces in cross-device evaluations[3]. Type II pitfalls ignore the number of test traces that is required to learn statistical information but only report attack results based on the number of test traces in the test phase[4]. Carrying either type (or both) of the two pitfalls would derive over-optimistic results in cross-device evaluations.

[1]Our statistic analyses using Normalized Inter-Class Variance in Sec. 5 show that traces from 32-bit microcontrollers and traces with random delays carry much lower side-channel leakage than the ones from 8-bit microcontrollers.

[2]Note that Zero-Mean Unit-Variance Normalization can also be considered as a pre-processing method. We particularly consider it as a simple (feature-based) domain adaptation method depending on how it is implemented and how its attack performance is measured in our study.

[3]Possessing labeled test traces in advance suggests that an attacker knows the key already before the attack. If that is the case, there is no need to perform the attack.

[4]For instance, if a method requires 2,000 test traces to learn feature means and feature variances to adapt test data to a trained neural network, and this neural network can rank the correct key as the top candidate with 100 test traces in the test phase based on this domain adaptation, the number of test traces that is needed to recover the keys should be 2,000 but not 100.

- According to our experimental results, we have three major observations. (I) Pre-processing and unsupervised domain adaptation methods can improve the portability of deep learning side-channel attacks over challenging targets. (II) However, the effectiveness of each method varies depending on the target and probe locations in use. Specifically, observations from easy targets do not necessarily generalize to challenging targets. (III) None of the methods we examine can always outperform others in our evaluation. In some cases, even traditional attacks with Correlation Power Analysis [4] can derive the best attack results.

Reproducibility. Our source code and dataset are publicly available at https://github.com/UCdasec/CrossEM.

2 BACKGROUND

2.1 Side-Channel Attacks

Side-channel attacks can be categorized into *profiling* side-channel attacks and *non-profiling* side-channel attacks. In a profiling side-channel attack, an attacker has control of a training target, where this attacker can choose the encryption key, select plaintexts (i.e., inputs of the encryption), and capture associated traces. A trace is a sequence of samples when a target runs one execution of an encryption algorithm given a plaintext and a key. In addition, this attacker can capture traces and associated plaintexts on a test target but it does not know the (fixed) encryption key on the test target.

The attack consists of two phases, the *training phase* (or *profiling phase*) and the *test phase* (or *attack phase*). In the profiling phase, the attacker trains a profile with labeled traces from the training target. The labels are intermediate results of encryption (or the Hamming Weights of these intermediate results), which can be calculated based on the known key and plaintexts from the training target by following the encryption algorithm. In the attack phase, the attacker takes traces and plaintexts from the test target and aims to recover the unknown key on the test target by leveraging the trained profile. Deep learning side-channel attacks leverage trained neural networks as profiles.

In a non-profiling attack, an attacker does not have access to a training target to build a profile in advance. Instead, it only has unlabeled traces from a test target to perform the attack. *In this study, we primarily focus on profiling side-channel attacks over EM traces but also provide results from non-profiling attacks, such as Correlation Power Analysis (CPA) [4], as baseline results for comparison.*

2.2 Notations, Leakage Model, and Metrics

An EM trace (or *trace* for short) can be presented as a vector $x = (x[0], ..., x[l-1])$, where $x[i]$ is the sample of EM signal from a target at timestamp i and l is the number of samples of a trace. Let $\mathcal{M}$ be the plaintext space and $\mathcal{K}$ be the key space. A trace x is captured when a target runs encryption with plaintext m and key k, where $m \in \mathcal{M}$ and $k \in \mathcal{K}$. We use $z = \varphi(m, k)$ to denote an intermediate value of encryption, where function $\varphi(\cdot)$ is an intermediate step carrying side-channel leakage.

AES-128. We study side-channel attacks on targets running AES-128, where an encryption key has 128 bits. The attack reveals an entire 16-byte key using divide-and-conquer, where one key byte is compromised each time. For the remaining of this paper, we assume key k, plaintext m, or intermediate value z has one byte by following previous studies [30, 34]. Let $k_0^*, k_2^*, ..., k_{255}^*$ be all the possible 256 key candidates. SubBytes of the 1st round of AES is chosen as the leakage step $\varphi(\cdot)$ as in previous studies.

Leakage Model. We apply *Hamming Weight (HW) model* and *Identity (ID) model* to formulate side-channel leakage [1, 34]. The HW model assumes that there are correlations between EM signals and the Hamming weight of an intermediate value. The label of a trace t is $\texttt{HW}(z)$ — the Hamming weight of the intermediate value $z = \varphi(m, k)$. There are 9 possible labels given the HW model. The ID model assumes that there are correlations between EM signals and the intermediate value itself. The label of a trace is the intermediate value z, and there are 256 possible labels.

Evaluation Metrics. Given a profile and a trace, the test phase outputs a score for every possible label. Then, each score is further assigned to corresponding key candidate(s) based on the label, an associated plaintext, and AES encryption algorithm. Every key candidate's scores across all the test traces are aggregated. As there are 256 key candidates given one byte, 256 aggregated scores are obtained and further sorted in a descending order.

We leverage *Key Rank* and *Measurements To Disclosure* (MTD) to measure the effectiveness of side-channel attacks [31, 41]. Key rank r, where $r \in [1, 256]$, is the rank of the aggregated score of the correct key among all the 256 key candidates given a certain number of test traces. A key rank of 1 suggests that the correct key has the highest score and the attacker distinguishes the key correctly. MTD indicates the number of test traces that is needed for the key rank to converge to 1 (i.e., it is distinguishable from others). A lower MTD indicates the attack is more effective.

MTD of Profiling Attacks with Pre-Processing or Domain Adaptation. Note that for profiling attacks with pre-processing or domain adaptation, if *statistical information (e.g., means, variances, losses, etc.) of test traces* are required to adjust a trained neural network or transform test traces before the test phase, *MTD should be N, the minimal number of test traces to learn required statistical information*, such that an attacker can distinguish the correct key from incorrect key candidates within these N test traces in the test phase.

3 METHOD DESCRIPTION

In this section, we briefly describe pre-processing methods and domain adaptation methods that are examined in this study.

3.1 Pre-Processing for Side-Channel Attacks

Existing studies [9, 15, 56] have shown that pre-processing samples and then passing processed data to neural networks offers opportunities to improve attack results, even there are no or minor domain shifts between training and test data. In this study, we investigate three pre-processing methods, including Discrete Fourier Transform, Principal Component Analysis, and Linear Discriminant Analysis, in the context of side-channel attacks.

While these three pre-possessing methods can reduce dimensions of data by transforming data into different spaces, the transformation of each method is fundamentally different. Specifically,

Discrete Fourier Transform can be performed on each trace independently. Principal Component Analysis needs to be applied to a set of unlabeled traces. Linear Discriminant Analysis requires a set of labeled traces to perform dimension reduction.

Discrete Fourier Transform (DFT). By leveraging DFT, an EM trace in the time domain is decomposed as a sum of a series of sine waveforms with different frequencies, amplitudes, and phases. Specifically, given a trace $x = (x[0], ..., x[l-1])$, the DFT will output a sequence of complex numbers $(X[0], ..., X[l-1])$, where $X[j]$ is represented as

$$X[j] = \sum_{n=0}^{l-1} x[n] \cdot e^{-\frac{i2\pi jn}{l}}$$

where $x[n]$ is the sample at timestamp n, l is the number of samples in a trace, and $j \in [0, l-1]$ is the current frequency. As each $X[j]$ is a complex number, its amplitude can be calculated as:

$$amp[j] = \frac{|X[j]|}{l} = \frac{\sqrt{Re(X[j])^2 + Im(X[j])^2}}{l}$$

where $Re(X[j])$ denotes the real part of $X[j]$ and $Im(X[j])$ indicates the imaginary part of $X[j]$.

For side-channel attacks with DFT pre-processing, only amplitudes from the positive frequencies of an EM trace, i.e., $(amp[0], ..., amp[l-1/2])$, will be utilized as an input of a neural network. We leverage Fast Fourier Transform algorithm to perform DFT. It is worth to mention that some existing studies [56] pre-select amplitudes with higher values and discard the ones with lower amplitudes rather than using all the amplitudes from positive frequencies. *We do not believe this extra step is necessary. It is because a high amplitude in the frequency domain does not necessarily indicate that the specific frequency carries high side-channel leakage.*

Principle Component Analysis (PCA). PCA [18] is a classic dimension reduction method, which reduces the number of dimensions of data. Specifically, PCA calculates means over features and obtains a covariance matrix. Next, eigendecomposition is applied to obtain eigenvectors and eigenvalues. A transformation matrix is obtained by sorting and selecting eigenvectors with the top eigenvalues. PCA is *unsupervised* — it does not require labels when performing dimension deduction.

For side-channel attacks with PCA pre-processing, given a set of training traces $\{x_0, ..., x_{n-1}\}$, where $x_i = (x_i[0], ..., x_i[l-1])$, PCA calculates a covariance matrix and its eigenvectors and then outputs a set of traces, $\{y_0, ..., y_{n-1}\}$, where $y_i = (y_i[0], ..., y_i[r-1])$ and $r \ll l$. In other words, the number of traces in a dataset remains the same but the number of dimensions of each trace is significantly reduced. Each trace y_i will be utilized as an input of a neural network. In addition, PCA outputs a transformation matrix, i.e., a $l \times r$ dimensional matrix W.

In this study, we leverage this matrix W to perform PCA over test traces and report MTD as the number of test traces that is required for key rank to converge to 1.

Type II Pitfall with PCA. Since PCA is an unsupervised method, learning a transformation matrix independently over unlabeled test traces in cross-device evaluations is an alternative approach to apply PCA over test traces. When reporting attack results with this alternative approach, ignoring the number of test traces that is needed for learning a (proper) transformation matrix would be a Type II pitfall.

Linear Discriminant Analysis (LDA). LDA [12] is a dimension reduction method, which finds a linear combination of features that can discriminate data from different classes. LDA first computes means over features within each class and calculates a between-class scatter matrix and a within-class scatter matrix. Eigenvectors and eigenvalues are then derived based on the scatter matrices. A transformation matrix is obtained by selecting eigenvectors with the top eigenvalues. As a result, LDA maps data into a lower-dimensional space by maximizing the ratio of between-class variance to within-class variance. In other words, LDA maximizes the distance among class means and minimizes the variances within each class. LDA is *supervised* — it requires labels when performing dimensions deduction.

For side-channel attacks with LDA pre-processing, given a set of training traces $\{x_0, ..., x_{n-1}\}$ and their labels $\{t_0, ..., t_{n-1}\}$, where $x_i = (x_i[0], ..., x_i[l-1])$ and t_i is the label of x_i, LDA outputs a set of traces, $\{y_0, ..., y_{n-1}\}$, where $y_i = (y_i[0], ..., y_i[r-1])$, $r \ll l$, and t_i is the label of y_i. Each trace y_i will be utilized as an input of a neural network. In addition, LDA outputs a transformation matrix, i.e., a $l \times r$ dimensional matrix W.

In this paper, we leverage this matrix W to perform LDA over test traces and report MTD as the number of test traces that is needed for key rank to converge to 1.

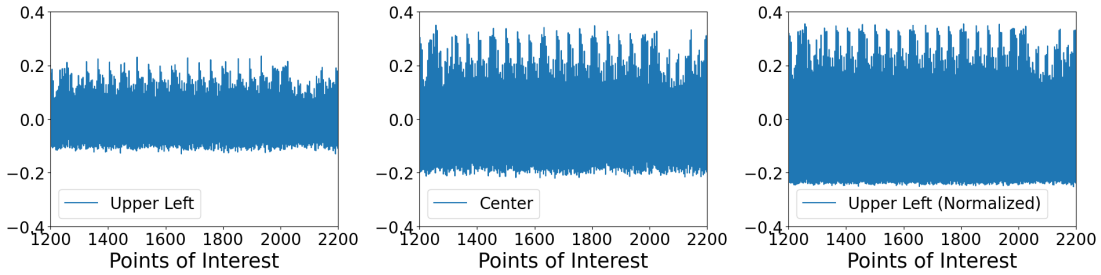
Type I Pitfall with LDA. Since LDA is a supervised method, learning a transformation matrix independently over labeled test traces in cross-device evaluations would be a Type I pitfall as test traces are unlabeled to an attacker before the test/attack phase.

3.2 Domain Adaptation for Side-Channel Attacks

We examine four domain adaptation methods, including Zero-Mean and Unit-Variance Normalization, Maximum-Mean-Discrepancy Domain Adaptation, Adversarial Domain Adaptation, and On-the-Fly Fine Tuning. All of these are unsupervised — do not require labeled test traces.

Zero-Mean and Unit-Variance Normalization (MVN). It is a common normalization technique for signals and was examined as a way of addressing domain shifts for Template Attacks over EM traces in [28]. There are two equivalent ways of implementing MVN as described in [28]. The first way is to normalize data independently within each dataset based on feature means and variances. The second way is to normalize test data based on feature means and variances of training data. In other words, compared to the first way, the second way does not need to know the feature means and variances of test traces. We consider the second way as a domain adaptation method in our study and present it as below.

Specifically, given a set of training traces $\{x_0, ..., x_{n-1}\}$, where $x_i = (x_i[0], ..., x_i[l-1])$, MVN calculates a mean trace $\mu = (\mu[0], ...,$



(a) EM trace (Upper Left). (b) EM trace (Center). (c) Normalized EM Trace (Upper Left).

Figure 1: An example of Zero-Mean and Unit-Variance Normalization (normalizing a trace from upper left based on traces from center, features from 1st round SubBytes of AES only, target: ARM STM32F3).

$\mu[l-1])$ as

$$\mu[j] = \frac{1}{n}\sum_{i=0}^{n-1} x_i[j], \quad for\ 0 \le j \le l-1$$

and a variance trace as $v = (v[0], ..., v[l-1])$

$$v[j] = \frac{1}{n}\sum_{i=0}^{n-1} (x_i[j] - \mu[j])^2, \quad for\ 0 \le j \le l-1$$

Similarly, MVN calculates a mean trace μ' and variance trace v' over a set of test traces $\{x'_0, ..., x'_{m-1}\}$. Each test trace $x'_i = (x'_i[0], ..., x'_i[l-1])$ is then normalized as below

$$x'_i[j] = \frac{x'_i[j] - u'[j]}{\sqrt{v'[j]}} \cdot \sqrt{v[j]} + u[j], \quad for\ 0 \le j \le l-1$$

Given normalized test traces and a trained profile derived from training traces, the profile will output aggregated scores and report key ranks in the test phase. *For a profiling attack with MVN, MTD is reported as N, the minimal number of test traces that is needed to learn feature means and variances of test traces, such that the key rank can converge to 1 within these N traces in the test phase.*

Maximum-Mean-Discrepancy Domain Adaptation (MMD-DA). Cao et al. [6] proposed an unsupervised domain adaptation to address domain shifts for side-channel attacks using Maximum Mean Discrepancy. We denote this method as MMD-DA in this paper. Specifically, MMD-DA first trains a Convolutional Neural Network with labeled training traces. Then, the last few layers of the Convoltutional Neural Network (i.e., fully-connected layers) are fine-tuned by using *labeled training traces and unlabeled test traces* to minimize the Maximum Mean Discrepancy loss between training and test traces.

After fine tuning with MMD, these unlabeled test traces are passed to the tuned neural network to record key ranks. *For a profiling attack with MMD-DA, MTD is reported as N, the minimal number of test traces that is involved in the fine tuning phase, such that the key rank can converge to 1 within these N traces in the test phase.*

Adversarial Domain Adaptation (ADA). ADA [13, 44] trains a domain adversarial network to learn a domain-invariant feature space by leveraging generative adversarial learning [16]. A domain adversarial network consists of three components, including a Feature Extractor, a Domain Discriminator, and a Source Classifier. Each component is a neural network. During the training, Feature Extractor takes *labeled* training data and *unlabeled* test data as inputs and outputs domain-invariant features. In addition, Domain Discriminator aims to distinguish data from two domains while Source Classifier minimizes the loss on predicting the correct labels of training data. The training process is formulated as a min-max game between Feature Extractor and Domain Discriminator. Wang et al. [47] recently leveraged ADA to address domain shifts caused by software discrepancies over power traces in side-channel attacks.

For a profiling attack with ADA, MTD is recorded as N, the minimal number of test traces that is used in the training phase, such that the key rank can converge to 1 within these N traces in the test phase.

Type II Pitfall with MVN, MMD-DA, and ADA. Since each domain adaptation method involves an adaptation/tuning phase and a test phase to distinguish correct keys, reporting only the number of test traces that is needed to recover keys in the test phase as attack results would be a Type II pitfall. It is because successfully recovering keys in the test phase relies on the number of test traces that is needed in the adaptation/tuning phase.

On-the-Fly Fine Tuning (FT). Given a trained neural network, Fine Tuning — tuning the last few layers with labeled test traces — is a common method to address domain shifts. However, directly applying it in the context of cross-device side-channel attacks would result in a Type I pitfall as labels of test traces remain unknown before attacks.

To avoid this pitfall while still being able to leverage Fine Tuning in cross-device side-channel attacks, we extend Fine Tuning with an existing technique named On-the-Fly Labeling [24, 43] (also known as Partition-based Differential Power Analysis [42]). We denote our method as On-the-Fly Fine Tuning. Specifically, given N unlabeled test traces and their plaintexts, guessed labels of these traces are generated based on one key candidate. In essence, it produces a partition of test traces based on the guessed labels generated by this key candidate. Given this partition, the neural network trained based on training traces is fine-tuned based on these test traces and their guessed labels. The accuracy of these N test traces over the fined-tuned neural network is recorded accordingly based on the guessed labels. The partition and fine tuning are repeated for 256 times, one for each key candidate.

Given all the 256 fine-tuned neural networks, the one with the highest accuracy is considered as the one generated by the top key candidate and that key candidate is ranked as 1 (i.e., the correct key is distinguishable from others). In other words, *accuracy* based

on guessed labels serves as a *Distinguisher* in On-the-Fly FT. In addition to accuracy, *loss* during fine tuning can also be utilized as a Distinguisher. On-the-Fly Labeling has been introduce to non-profiling attacks in previous studies [24, 43, 46].

For a profiling attack with On-the-Fly FT, MTD is reported as N, the minimal number of test traces that is involved in the On-the-Fly FT, such that the key rank can converge to 1 based on the selected Distinguisher over these N traces.

A Combination of Type I and II Pitfalls with Fine Tuning. Over-optimistic attack results would be derived if an evaluation applies Fine Tuning with labeled test traces directly and reports MTD as the number of test traces that is needed for the key rank to converge to 1 in the test phase.

It is also worth to mention that *ID model should not be leveraged to assign guessed labels to produce partitions in On-the-Fly Labeling* [43, 46]. This is because the 256 partitions produced by the 256 key candidates would be exactly the same given ID model, and there would be no statistical differences among these 256 partitions (i.e., a Distinguisher would fail to identify the correct key from incorrect ones regardless how this Distinguisher is chosen).

Comparison among MVN, MMD-DA, ADA, and On-the-Fly FT. Compared to MMD-DA and ADA, MVN or On-the-Fly FT only requires an attacker to train a less complicated neural network, e.g., a Convolutional Neural Network (CNN). This can lead to much shorter training time in the evaluation as training a CNN is less time-consuming than training in ADA or MMD-DA[5]. In addition, MVN or On-the-Fly FT only requires an attacker to train a profile in advance with labeled training traces (i.e., without the need of involving unlabeled test traces in the training phase). This can offer better scalability in the evaluation. For instance, if a lower number of test traces is not sufficient to reveal the key, MVN or On-the-Fly FT can simply increase the number of test traces without retraining a profile while ADA or MMD-DA needs to retrain the profile.

Compared to MVN, On-the-Fly FT does not require normalizing test traces. On the other hand, as a tradeoff, On-the-Fly FT needs to fine-tune a neural network over test traces for 256 times based on the partitions of test traces, which leads to longer attack time. A more detailed comparison in terms of attack time among the four methods can be found in Sec. 5.

4 DATA COLLECTION AND DATASETS

We establish a customized EM data collection setup built upon ChipWhisperer and acquire a large-scale dataset of EM traces from microcontrollers running unmasked AES-128. ChipWhisperer is a popular hardware toolset, which is primarily designed for collecting power traces to perform side-channel attacks. While ChipWhisperer offers the capability of acquiring EM traces (e.g., manually holding CW505 Planar H-Field Probe on the top of a target), this setup does not provide opportunities to capture relatively stable EM traces, especially in a large scale.

Our EM Data Collection Setup. Besides CW-Lite, CW308 UFO Board, and a target microcontroller from ChipWhisperer Level-1 Kit, our setup also consists of additional hardware, including a Langer EMV RF 7-4 Near-Field Probe, a Langer EMV PA 306 Amplifier, a Keysight 3D Probe Holder, and a Riscure XYZ stage.

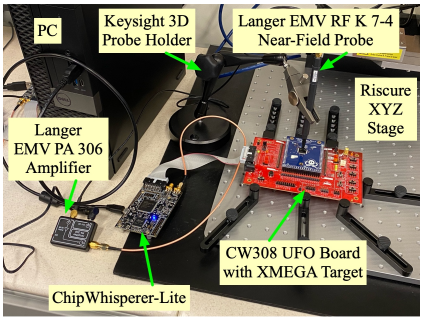


Figure 2: EM data collection setup

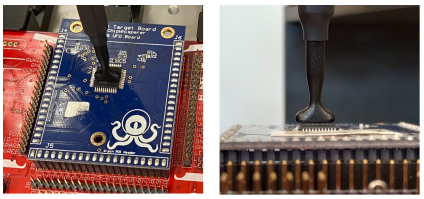


Figure 3: The EM probe positioning on top of the target.

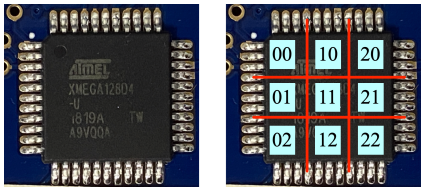


Figure 4: AVR XMEGA (left) and the cells of 9 EM probe locations (right) over it.

As presented in Fig. 2, we first leverage Riscure XYZ stage to fasten ChipWhisperer CW308 UFO Board. A target microcontroller, an AVR XMEGA (8-bit) or an ARM STM32F3 (32-bit), is mounted on ChipWhisperer CW308 UFO Board. Next, we place a Langer EMV RF K 7-4 Near-Field Probe around 5 millimeters above the center of the target (as shown in Fig. 3) and utilize a Keysight 3D Probe Holder to hold the probe. The probe captures EM traces from the target and passes EM traces to ChipWhisperer Lite (and eventually a PC) through the Langer EMV PA 306 Amplifier. We use the default sampling rate for a target as the one used in ChipWhisperer's power trace collection[6]. We set the gain of the scope of ChipWhisperer as 60 (i.e., `scope.gain.gain = 60`).

Given a target, we evenly divide its surface into 9 cells (as shown in Fig. 4) and we refer these cells as {C00, C01, C02, C10, C11, C12, C20, C21, C22} respectively. We acquire EM traces from each of these cells by adjusting the position of probe such that the probe is roughly around the center of each cell. Two XMEGA targets (denoted as X1 and X2) and two STM32F3 targets (denoted as S1 and S2) are involved in our data collection. The surface sizes of XMEGA and STM32F3 are similar (i.e., both around 0.7cm × 0.7cm).

[5]For instance, training a CNN takes about 0.5 hours while training in ADA takes about 3 hours in our evaluation.

[6]The sampling rate of XMEGA is 29.34 MHz and the sampling rate of STM32 is 29.52 MHz. We run `scope.clock.adc_freq` command in ChipWhisperer script to obtain these information.

Table 2: Our CrossEM Dataset (3 million EM traces, 66 GBs). Each subset contains 150,000 EM traces, 150,000 plaintexts, and a key. Each trace consists of 5,000 samples.

X1_K1_C11	X2_K2_C00	X2_K2_C01	X2_K2_C02
	X2_K2_C10	X2_K2_C11	X2_K2_C12
	X2_K2_C20	X2_K2_C21	X2_K2_C22
S1_K1_C11	S2_K3_C00	S2_K3_C01	S2_K3_C02
	S2_K3_C10	S2_K3_C11	S2_K3_C12
	S2_K3_C20	S2_K3_C21	S2_K3_C22

Table 3: Encryption Keys Used in Our Dataset

K1	0x2b7e 1516 28ae d2a6 abf7 1588 09cf 4f3c
K2	0xaa80 d8a7 84d3 3f5c 0b90 a985 208e ff4a
K3	0xd2d5 0168 8283 9143 969e e9a2 53a7 52e1

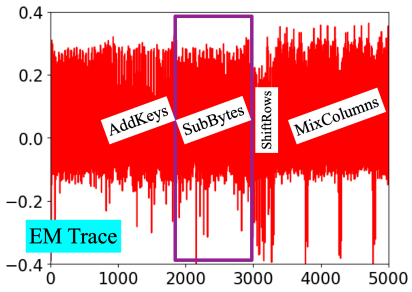


Figure 5: An EM trace of AES-128 on XMEGA. It contains 5,000 samples from the beginning of AES-128 to almost the halfway of MixColumn of the 1st round of AES-128.

We always use different keys across two targets of the same type. We utilize TinyAES, an unmasked C implementation of AES-128 from ChipWhisperer repository, as the encryption algorithm running on a target. This implementation is designed for small code size and low memory usage, and it is widely used for embedded systems, such as microcontrollers. We utilize cross-compilers (`avg-gcc` for XMEGA and `arm-none-eabi-gcc` for STM32F3) to compile the C code and produce binaries. We use `O0` optimization when we compile C code.

Our CrossEM Dataset. Our dataset, denoted as CrossEM, consists of 20 subsets with a total of 3 million EM traces (66 GBs). Specifically, we collect a subset of 150,000 EM traces of AES-128 running on X1 with probe over cell C11. Next, we collect one subset of 150,000 EM traces of AES-128 running on X2 with the probe over each cell. We repeat the same process with S1 and S2. Each EM trace consists of 5,000 samples. Each subset contains EM traces, associated plaintexts, and an encryption key. The plaintexts and keys are randomly generated. We use the following format to name each subset, `Target_Key_ProbeLocation`. Alternatively, we also use the format `Target_ProbeLocation` for simplicity. A summary of our dataset is highlighted in Table. 2. An example of an EM trace collected from XMEGA running AES-128 is presented in Fig. 5. In addition, we also generate subsets with random delays by shifting samples in every trace with a random value of r, where $r \in [0, 20]$.

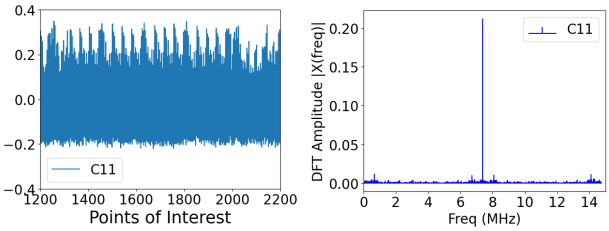
Points of Interest. We use *Points of Interest (POI)* to indicate samples associated with the leakage step in a trace. For example,

Table 4: Hyperparameters of CNN [1]

Conv 1	filters: 64; kernel size: 11; stride: 2; Relu
Conv 2	filters: 128; kernel size: 11; stride: 2; Relu
Conv 3	filters: 256; kernel size: 11; stride: 2; Relu
Conv 4~5	filters: 512; kernel size: 11; stride: 2; Relu
AvgPool 1~5	pooling size: 2; stride: 2
Dense 1~2	No. of neurons: 4096; Relu
Output	No. of neurons: 9 (HW) or 256 (ID); Softmax

Table 5: Hyperparameters of MLP [1]

Dense 1~5	No. of neurons: 200; ReLU
Output	No. of neurons: 9 (HW) or 256 (ID); Softmax



(a) Trace in the time domain. **(b) Trace in the frequency domain.**

Figure 6: An EM trace of AES-128 (1st round of SubBytes only) on STM32F3 before and after applying DFT.

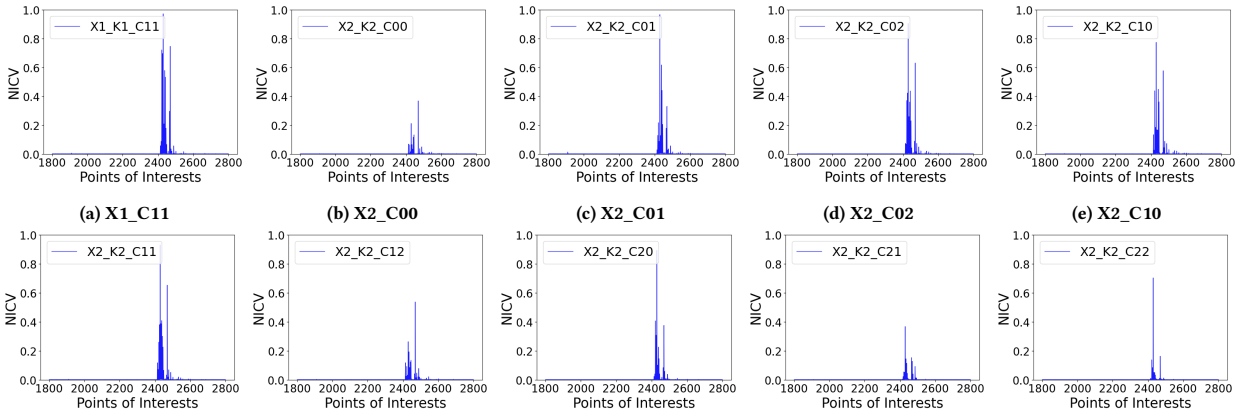
$POI = [100, 400]$ suggests that measurements from timestamp 100 to timestamp 400 of a trace are associated with the leakage step. For our dataset, we utilize the SubBytes of the first round of AES-128 as the leakage step. Specifically, we select $POI = [1800, 2800]$ for XMEGA and $POI = [1200, 2200]$ for STM32. The POIs are identified in advance by manually analyzing the pattern of EM traces (with and without the leakage step in C code) as mentioned in [46, 47]. When we train a neural network, we only use samples within POI of a trace as inputs and samples outside POI are skipped.

5 EVALUATION

5.1 Evaluation Settings

Details of Neural Networks. We examine two neural networks, including a Convolutional Neural Network (CNN) and a Multi-Layer Perceptron (MLP), for side-channel attacks in our evaluation. Both architectures derived promising attack results over the well-known ASCAD dataset [1]. The CNN includes 5 convolutional layers, 5 pooling layers, 2 fully connected layers, and 1 output layer. The hyperparameters of this CNN are outlined in Table 4. The MLP consists of 5 dense layers and 1 output layer. The hyperparameters of this MLP are presented in Table 5. We implement neural networks using TensorFlow 2.4.1 with CUDNN 11.11.

Experiment Details of Pre-possessing Methods. For Discrete Fourier Transform, samples within the POI are utilized as inputs to DFT, where $l = 1,000$. Each output from DFT consists of $l/2 = 500$ amplitudes from the positive frequencies. An example of an EM trace (samples within POI only) on STM32F3 in the time domain and its processed format in the frequency domain are presented in Fig. 6. Note that the frequency with the high peak in the frequency domain Fig. 6 is the clock frequency (7.38 MHz) of the target. It



(a) X1_C11 (b) X2_C00 (c) X2_C01 (d) X2_C02 (e) X2_C10

(f) X2_C11 (g) X2_C10 (h) X2_C20 (i) X2_C21 (j) X2_C22

Figure 7: NICVs over 10 subsets from XMEGA (no delays).

does not indicate that high side-channel leakage happens at that frequency. For PCA/LDA, given a trace of 1,000 POI samples, we transform it into a trace of 256 features using PCA/LDA.

Experiment Details of Domain Adaptation We implement MVN and On-the-Fly FT using Python. We always fine tune the last two layers given a CNN in On-the-Fly FT. For ADA, we leverage the PyTorch source code from [47] where we use the CNN (except the last layer) as Feature Extractor. For MMD-DA, we leverage the PyTorch source code from [6] where its CNN has a different architecture than the one we use from ASCAD dataset. We denote it as CNN* in the tables. CNN* consists of 3 convolutional layers (with SELU activation), 3 average pooling layers, 2 fully connected layers, and 1 output layer.

Experiment Setting. We conduct almost all the experiments on a desktop with Ubuntu 22.04, an Intel i9-14900K CPU, 128 GB memory, and a NVIDIA 4080 GPU. In addition, a server (with Ubuntu 22.04, an AMD EPYC 7742 64-Core processor, 512 GB memory and 4 NVIDIA A100 GPUs) is also utilized to run some large-scale experiments (e.g., On-the-Fly FT) more efficiently. We randomly select **3rd byte** of an AES key[7] and report attack results on it with either the HW model or ID model, depending on which model performs better. Specifically, we report results from the ID model for all the methods (except MMD-AD and On-the-Fly FT) as our preliminary results show that ID model outperforms HW model in these methods. For MMD-AD, we record results from the HW model as it outperforms the ID model in our preliminary results. For On-the-Fly FT, we use the HW model only as the ID model is not applicable.

When we train a neural network, we use 150 epochs. Given a subset, we allocate 135,000 traces for training, 5,000 traces for validation, and 10,000 traces for testing. The reported key ranks and MTDs are average results by running tests 20 times over test traces and randomly shuffling test traces each time. When we report results from domain adaptation methods, given 135,000 training traces from the training target, at most 10,000 test traces from the test target will be used for domain adaptation. We use divide and conquer to find MTDs in these methods.

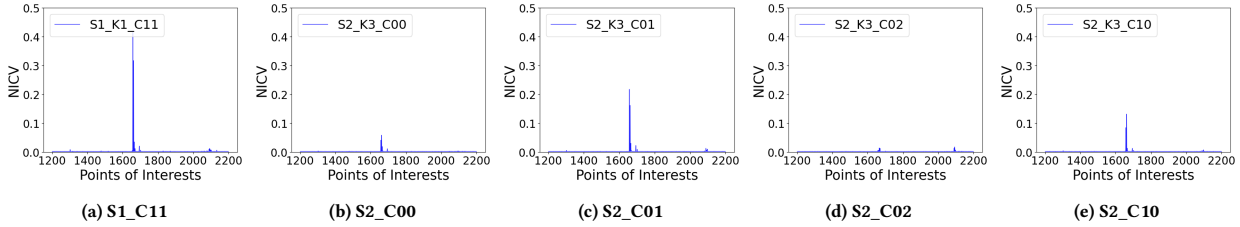
[7]The evaluation of existing studies typically focuses on one byte. If one byte can be revealed, then it is sufficient to show the entire key (all the 16 bytes) can be recovered. There is no obvious difference in terms of which specific byte is chosen for the target.

5.2 Experiments

Experiment 1: Measuring NICV over Our Dataset. We first perform statics analysis, specifically Normalized Inner Class Variance (NICV) [3], to show that side-channel leakage presents but may not be obvious, especially for challenging targets. Specifically, given a subset of EM traces in our dataset, where each trace has l samples, NICV does not recover keys but can output an NICV value α for every timestamp i, where $\alpha \in [0, 1]$ and $i \in [0, l-1]$. A higher NICV value indicates that the correlation between samples from the associated timestamp and intermediate values is higher — side-channel leakage is higher.

As shown in Fig. 7, we observe that all the 10 subsets from XMEGA carry obvious side-channel leakage (i.e., peaks with high NICV values). This suggests that keys can be easily recovered from these subsets. In Fig. 8, we observe that over half of the subsets from STM32F3 carry relatively obvious peaks with reasonably high NICV values. However, the NICV values are much lower than the ones obtained from XMEGA. In addition, subsets from some locations (e.g., C02 and C22) on STM32F3 do not show obvious side-channel leakage. Overall, it demonstrates that STM32F3 is a more challenging target (i.e., carrying less side-channel leakage) than XMEGA. This is expected as STM32F3 (32-bit) takes fewer instruction cycles to execute a function with a wider data bus than XMEGA (8-bit), which leads to less side-channel leakage in general. We also find that it will be challenging to recover keys over subsets with random delay as only extremely low NICV values (e.g., < 0.01 on XMEGA and < 0.003 on STM32F3) are reported in Fig. 9 and Fig. 10. This is also expected as the executions of the leakage function (i.e., 1st round of SubBytes) are unsynchronized across traces due to random delay.

Experiment 2: Side-Channel Attacks on XMEGA (No Delay). In this experiment, we investigate the results of deep learning side-channel attacks over EM traces (no delay) from XMEGA. We first report baseline results from the same device scenario, where we use traces from X1_C11 as the training and test data. Next, we present results from cross-device scenarios, where train traces are



(a) S1_C11 (b) S2_C00 (c) S2_C01 (d) S2_C02 (e) S2_C10

(f) S2_C11 (g) S2_C12 (h) S2_C20 (i) S2_C21 (j) S2_C22

Figure 8: NICVs over 10 subsets from STM32F3 (no delays).

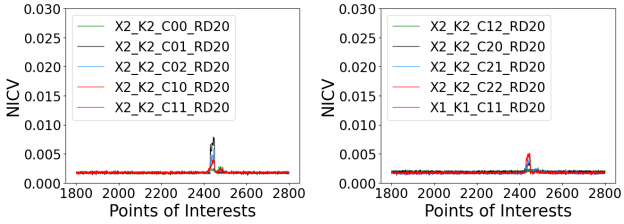


Figure 9: NICVs over 10 subsets from XMEGA (with delay)

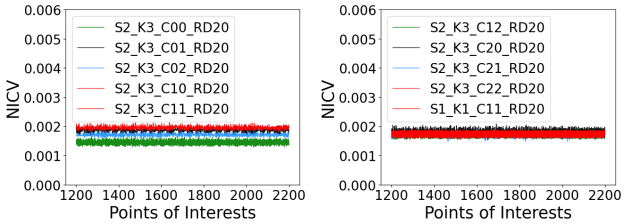
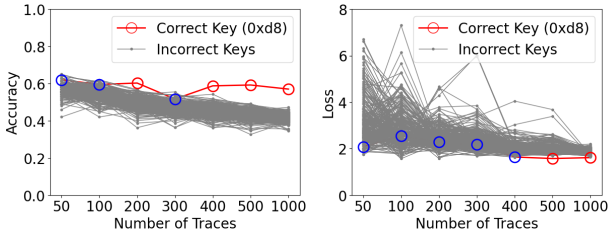


Figure 10: NICVs over 10 subsets from STM32F3 (with delay)



(a) Accuracy as Distinguisher (b) Loss as Distinguisher

Figure 11: Comparison between accuracy and loss on serving as the Distingusigher in On-the-Fly FT (Train: X1_C11, Tune: X2_C11). The correct key is distinguishable starting from 400 test traces with accuracy and from 500 test traces with loss.

from X1_C11 but test traces are from X2 with different keys and locations. Attack results from CPA and Template Attacks over test traces are also reported as baseline for comparison. We select 100 POIs for Template Attacks using the *sum of difference* [7]. Choosing a greater number of POIs for Template Attacks is also possible but requires significant computational overheads. In Fig. 11, we also demonstrate that accuracy serves as a more effective Distinguisher than loss in On-the-Fly FT, where a smaller number of test traces is needed to identify the correct key. Therefore, we use accuracy as the Distinguisher of On-the-Fly FT in all our experiments.

As shown in Table 6, a neural network can easily recover keys with 1 test trace in the same device scenario on XMEGA. We have several main observations from cross-device scenarios. ① The attack performance of deep learning side-channel attacks drops when there are discrepancies between the training and test data, where a CNN or MLP requires a greater number of test traces to recover keys or even fails to recover keys (e.g., X2_C00, X2_C12, and X2_C22) on XMEGA. ② All the three pre-processing methods, including DFT, PCA, and LDA, can improve the attack results compared to neural networks without pre-processing. However, this is only for some subsets not all the subsets from X2. For instance, CNN still performs the best over X2_C02 even compared to the ones obtained from pre-processing or domain adaptation. ③ Among the three pre-processing methods, not a single method can outperform the others over all the 9 locations from X2. ④ MVN overall performs the best compared to the other three domain adaptation methods and can derive better results than CNN over some subsets (e.g., X2_C10, X2_C11, and X2_C20). ⑤ ADA and On-the-Fly FT can derive better results than CNN and MLP but are less effective than CPA over some subsets (e.g., X2_C00 and X2_C21).

Experiment 3: Side-Channel Attacks on STM32F3 (No Delay). In this experiment, we investigate the results of deep learning side-channel attacks over EM traces (no delay) from STM32F3. Similar as the last experiment, we first report baseline results from the same-device scenario using S1_C11 and then report cross-device scenarios by using the same trained neural network but different subsets from S2 as test data. Attack results from CPA over test traces are also reported for comparison.

As presented in Table 7, a CNN can easily recover keys with 21 test trace in the same device scenario on STM32F3 while MLP fails to recover keys (unless LDA is applied). *For cross-device scenarios, we*

Table 6: Attack Results in MTD (XMEGA, without Delay, Train: X1_C11, best results across different methods over each subset are highlighted with bold font and ⊥ indicates failing to recover keys within 10,000 test traces)

	Same	Cross								
Method \ Test Data	X1_C11	X2_C00	X2_C01	X2_C02	X2_C10	X2_C11	X2_C12	X2_C20	X2_C21	X2_C22
CPA	90	**500**	100	100	300	100	**200**	200	**200**	500
Template (ID)	14	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
CNN (ID)	2	⊥	10	**6**	26	220	⊥	27	⊥	⊥
CNN (ID) + DFT	3	⊥	23	⊥	⊥	181	⊥	⊥	⊥	**152**
CNN (ID) + PCA	**1**	⊥	196	63	1,072	98	⊥	723	⊥	⊥
CNN (ID) + LDA	**1**	⊥	**8**	9	1,643	23	9,995	95	⊥	⊥
MLP (ID)	**1**	⊥	282	263	⊥	⊥	⊥	27	⊥	⊥
MLP (ID) + DFT	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + PCA	**1**	⊥	⊥	1,164	⊥	3,257	⊥	3,734	⊥	⊥
MLP (ID) + LDA	**1**	⊥	217	126	5,812	88	⊥	⊥	⊥	⊥
CNN (ID) + MVN	NA	⊥	10	10	**20**	**10**	⊥	**10**	10,000	⊥
CNN (ID) + ADA	NA	2,000	500	1,000	500	500	⊥	500	1,500	10,000
CNN* (HW) + MMD-DA	NA	⊥	250	250	250	250	⊥	250	⊥	⊥
CNN (HW) + On-the-Fly FT	NA	⊥	300	100	400	200	⊥	1,000	5,000	1,000

Table 7: Attack Results in MTD (STM32F3, without Delay, Train: S1_C11, best results across different methods over each subset are highlighted with bold font and ⊥ indicates failing to recover keys within 10,000 test traces)

	Same	Cross								
Method \ Test Data	S1_C11	S2_C00	S2_C01	S2_C02	S2_C10	S2_C11	S2_C12	S2_C20	S2_C21	S2_C22
CPA	3,000	10,300	2,500	**1,600**	3,300	3,100	3,700	**900**	7,200	**2,900**
Template (ID)	20	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
CNN (ID)	21	**1,640**	113	⊥	697	49	1,198	⊥	⊥	⊥
CNN (ID) + DFT	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
CNN (ID) + PCA	37	⊥	⊥	⊥	⊥	8,230	⊥	⊥	⊥	⊥
CNN (ID) + LDA	**15**	5,067	431	⊥	1,994	146	455	⊥	⊥	⊥
MLP (ID)	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + DFT/PCA	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + LDA	17	⊥	6,875	⊥	⊥	154	**57**	⊥	⊥	⊥
CNN (ID) + MVN	NA	2,000	**30**	⊥	**30**	**20**	2,000	⊥	⊥	⊥
CNN (ID) + ADA	NA	⊥	1,500	⊥	1,000	1,500	1,000	10,000	⊥	⊥
CNN* (HW) + MMD-DA	NA	6,000	2,000	⊥	⊥	250	⊥	⊥	**5,000**	8,000
CNN (HW) + On-the-Fly FT	NA	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

find that the observations on XMEGA from the previous experiment do not always apply to subsets from STM32F3. More specifically, ① compared to the observations from XMEGA, the domain shifts are more severe from STM32F3, where the attacks recover keys with a lower number of subsets. ② Among the three pre-processing methods, only LDA with MLP can achieve better results that CNN. However, it is limited to one subset only (S2_C12). ③ Among the four domain adaptation methods, MVN can achieve better results than CNN over 3 subsets (S2_C01, S2_C10, and S2_C11). MMD-DA can offer the best result over one subset (S2_C21). ADA can derive better results than CNN but is less effective than CPA over one subset (e.g., S2_C20). On-the-Fly FT fails to recover keys over all the 9 subsets.

Experiment 4: Side-Channel Attacks on XMEGA (with Delay). In this experiment, we compare the results of deep learning side-channel attacks over EM traces (with delay) from XMEGA in Table 8. The observations in cross-device evaluations are relatively consistent with the ones from XMEGA without delay except the following: ① CPA or Template Attacks do not work anymore as traces are misaligned due to delay; ② ADA, MMD-DA, and On-the-Fly FT can outperform other methods over one subset respectively; ③ There is one subset (X2_C00) that none of the methods can recover keys.

Experiment 5: Side-Channel Attacks on STM32F3 (with Delay). In this experiment, we compare the results of deep learning side-channel attacks over EM traces (with delay) from STM32F3 in

Table 8: Attack Results in MTD (XMEGA, with Delay, Train: X1_C11, best results across different methods over each subset are highlighted with bold font and ⊥ indicates failing to recover keys within 10,000 test traces)

	Same	Cross								
Test Data / Method	X1_C11	X2_C00	X2_C01	X2_C02	X2_C10	X2_C11	X2_C12	X2_C20	X2_C21	X2_C22
CPA or Template (ID)	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
CNN (ID)	2	⊥	**6**	**18**	83	**38**	⊥	64	⊥	⊥
CNN (ID) + DFT	7	⊥	82	75	37	61	⊥	1,133	⊥	**59**
CNN (ID) + PCA	3	⊥	4,267	⊥	4,887	8,585	⊥	⊥	⊥	⊥
CNN (ID) + LDA	⊥	⊥	⊥	⊥	⊥	9,984	⊥	⊥	⊥	⊥
MLP (ID)	**1**	⊥	⊥	⊥	⊥	1,669	⊥	⊥	⊥	⊥
MLP (ID) + DFT	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + PCA	**1**	⊥	5,614	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + LDA	4	⊥	53	365	1,075	1,162	⊥	212	5,871	⊥
CNN (ID) + MVN	NA	⊥	40	20	**20**	200	⊥	**50**	⊥	200
CNN (ID) + ADA	NA	⊥	500	500	500	500	⊥	500	**1,000**	⊥
CNN* (HW) + MMD-DA	NA	⊥	250	250	250	250	**5,000**	⊥	⊥	⊥
CNN (HW) + On-the-Fly FT	NA	⊥	500	300	3,500	400	⊥	2,000	**1,000**	10,000

Table 9: Attack Results in MTD (STM32F3, with Delay, Train: S1_C11, best results across different methods over each subset are highlighted with bold font and ⊥ indicates failing to recover keys within 10,000 test traces)

	Same	Cross								
Test Data / Method	S1_C11	S2_C00	S2_C01	S2_C02	S2_C10	S2_C11	S2_C12	S2_C20	S2_C21	S2_C22
CPA or Template (ID)	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
CNN (ID)	**24**	3,006	3,006	⊥	**639**	821	**1,590**	⊥	⊥	⊥
CNN (ID) + DFT/PCA/LDA	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID)	1,166	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + DFT	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + PCA	169	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
MLP (ID) + LDA	37	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
CNN (ID) + MVN	NA	**2,000**	**1,000**	⊥	2,000	**30**	2,000	⊥	⊥	⊥
CNN (ID) + ADA	NA	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
CNN* (HW) + MMD-DA	NA	6,000	2,000	⊥	⊥	250	⊥	⊥	**5,000**	**9,000**
CNN (HW) + On-the-Fly FT	NA	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Table 9. We have the following observations in cross-device evaluations. ① CPA or Template Attacks do not work anymore. ② CNN outperforms the ones with pre-processing or domain adaptation over two subsets (S2_C10 and S2_C12). ③ All the three pre-processing methods fail to recover keys. ④ MVN and MMD-DA can outperform other methods respectively over some subsets. ⑤ There are two subsets (S2_C02 and S2_C20) that none of the methods can recover keys.

To summarize, we have the following overall findings regarding the portability of deep-learning side-channel attacks over EM traces from the results presented from Experiment 2 to Experiment 5:

Ⅰ Pre-processing and unsupervised domain adaptation methods can improve the portability of deep learning side-channel attacks on challenging targets. Ⅱ The effectiveness of each method varies depending on the target and probe locations in use. Specifically, observations from easy targets do not necessarily generalize to more challenging targets. Ⅲ None of the methods can constantly outperform others in our evaluation. In some cases, even traditional attacks with CPA can derive the best attack results.

Given domain shifts between training and test EM traces in general, we suggest that a fair evaluation should experimentally explore all these pre-processing and domain adaptation methods to determine which method can achieve the best attack results. On the other hand, we recommend exploring other methods first before performing On-the-Fly FT as it is time consuming in the attack phase compared to other methods. The computation overhead of each domain adaptation method on our desktop (with a NVIDIA 4080 GPU) is presented in Table. 10 as a reference for comparison.

Table 10: Computational overhead among domain adaptation methods (given 10,000 test traces and a CNN trained with 135,000 training traces)

Method	MVN	ADA	MMD-DA	On-the-Fly FT
Time (hours)	0.3	0.5	0.4	28

Table 11: Comparison of Attack Results in MTD (No pitfalls v.s. with pitfalls, XMEGA, without Delays, Train: X1_C11, over-optimistic results due to pitfalls are highlighted in red)

Method \ Test Data	X2_C00	X2_C01	X2_C02	X2_C10	X2_C11	X2_C12	X2_C20	X2_C21	X2_C22
CNN (ID)	⊥	10	**6**	26	220	⊥	27	⊥	⊥
CNN*(HW) + MMD-DA	⊥	250	250	250	250	⊥	250	⊥	⊥
CNN*(HW) + MMD-DA + Type II	⊥	45	24	34	45	⊥	150	⊥	⊥
CNN (HW) + On-the-Fly FT	⊥	300	100	400	200	⊥	1,000	5,000	1,000
CNN (HW) + FT + Type I & II	2,541	11	12	88	18	558	61	182	417

Pre-processing methods all perform efficiently in our experiments and we skip their detailed running time.

Experiment 6: Examples of Over-Optimistic Results from Type I and II Pitfalls. In this experiment, we provide examples reporting over-optimistic results when there are Type I and/or Type II pitfalls in cross-device evaluations. Specifically, we examine two cases, including MMD-DA with Type II and Fine-Tuning with Type I and II, and we compare these over-optimistic results with the ones we obtain from previous experiments.

For MMD-DA with Type II, we assume that an attacker performs MMD-DA with 135,000 training traces and (at most) 10,000 unlabeled test traces in the domain adaptation phase. then it reports the MTD as the number of test traces when key rank converges to 1 in the test phase but ignore the number of test traces used in the domain adaptation phase (Type II). For Fine Tuning with Type I and II, we assume that an attacker performs fine tuning directly on a trained CNN with 10,000 labeled test traces (Type I) and reports the MTD as the number of test traces when key rank converges to 1 in the test phase (Type II). As illustrated in Table 11, over-optimistic attack results would be derived in each method. For instance, one would have a wrong conclusion that Fine Tuning is the best approach to overcome domain shifts in the context of side-channel attacks over EM traces due to these pitfalls. It is worth mentioning that several studies [14, 54] applying Fine Tuning with Type I and II pitfalls when addressing domain shifts in side-channel attacks. Their conclusions, as a consequence, are obviously over-optimistic.

Experiment 7: Quantifying Domain Shifts. In this experiment, we quantify domain shifts between training traces and test traces and investigate whether the degree of domain shifts correlates with attack performance (e.g., does a more severe domain shift indicate that it is more challenging to recover keys?). Specifically, we quantify domain shifts by measuring the Euclidean distance d between mean trace $\mu = (\mu[0], ..., \mu[l-1])$ of training traces and mean trace $\mu' = (\mu'[0], ..., \mu'[l-1])$ of test traces, where

$$d = \sqrt{\sum_{j=0}^{l-1} (\mu[j] - \mu'[j])^2}$$

We measure the Euclidean distance between training traces of S1_C11 and test traces of every subset from S2. We summarize the distances in Fig. 12a. In addition, we also summarize MTDs obtained from CNN (ID model) and MTDs derived from the best results among the domain adaptation methods we use in Experiment 3 over these 9 probe locations. From Fig. 12a, we observe that ① Euclidean distance can serve as an indicator of attack performance

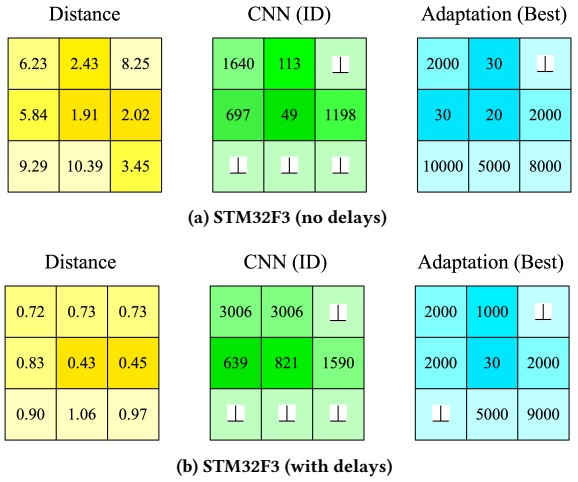


Figure 12: Distance between training traces (S1_C11) and test traces (S2_CXX), where each cell XX ∈ {00, 01, 10, 10, 11, 12, 20, 21, 22} is consistent with the one defined in Fig. 4. Darker color indicates a smaller distance or MTD.

in general, where a small distance likely results in a small MTD and a great distance likely end up with a great MTD (or even fail to recover keys); ② However, there is no linear correlation between Euclidean distance and attack performance in MTD. We have consistent observations from Fig. 12b based on distance and results from traces with delay on STM32F3.

6 DISCUSSIONS AND LIMITATIONS

Given the observation that none of the existing methods can constantly outperform others in this paper, we believe that more efforts are needed from the research community to develop new methods, especially new unsupervised methods (i.e., methods that do not require labeled test traces), to more effectively address domain shifts of EM traces caused by probe locations. We believe that our EM acquisition setup, dataset, and source code serve as stepping stones for future work to expand our findings and demonstrate the effectiveness of new methods in this line of research.

Future research should also further explore results from more challenging targets, such as FPGAs and masked AES, given domain shifts of EM traces caused by probe locations. Specifically, although recovering keys on these targets (given traces from the same probe location) is feasible with deep learning from as shown in recent research [27], it requires more comprehensive neural network architectures and learning processes. For instance, recovering keys

from a masked AES implementation on STM32 from ASCADv2 dataset requires a comprehensive ResNet rather than a CNN [27]. It remains unknown whether existing unsupervised domain adaptation methods can be integrated into these complex neural networks to tackle domain shifts of EM traces.

As mentioned, we observe that the effectiveness of each method varies depending on the target and probe location in use. We believe that it is mainly because the placements of the data bus and the Arithmetic Logic Unit (ALU) are different across different targets due to distinct hardware designs. Both data exchange on the data bus and executions inside the ALU are the major factors that contribute to the change of power consumption (as well as EM emanations) when calculating intermediate results of encryption on a target [26]. A probe location that is closer to the data bus and ALU, in general, can capture EM traces with higher leakage.

In our current acquisition setup, we manually place a probe over one of the 3 by 3 cells on the tiny surface of a microcontroller. A more fine-grained grid, e.g., 4 by 4 cells or 10 by 10 cells, can be achieved but (preferably) with the need of additional equipment to accurately place a probe over the top of each cell and move a probe at a millimeter scale. For instance, this can done by leveraging Riscure's XYZ table, its software, and its high precision EM probe. An alternative way is to build a customized setup by integrating a 3D printer into the EM acquisition of ChipWhisperer as described in [8]. These setups require either expensive cost on equipment or significant engineering efforts.

While we do not specifically perform evaluation over EM traces from other implementations of unmasked AES, we expect that, given a different unmasked implementation, our observations between training data and test data across different probe positions will still apply as the encryption algorithm as well as the executions of the leakage step (e.g., 1st round of SubBytes) remain the same.

There are other factors that can contribute to domain shifts over EM traces. For instance, utilizing different types of probes (e.g., Langer v.s., Tekbox) between training acquisition and test acquisition would affect the measurements of EM signals, and therefore lead to domain shifts for side-channel attacks. While we try our best to report the most effective attack results of each method based on the implementations we have, we believe that it is still possible to further improve the performance of each method. For instance, neural network architecture search is a promising approach to find the best CNN model to perform side-channel attacks [37]. On the other hand, how to perform neural network architecture search such that the model can overcome domain shifts, specifically for side-channel attacks, remains unknown.

Besides pre-processing and domain adaptation, existing research [2, 9] has shown that multi-domain training (e.g., training with traces from multiple targets or locations) is also an effective approach to mitigate domain shifts. However, it requires collecting large-scale datasets from multiple acquisitions in advance and may not be scalable considering all the factors (hardware, keys, software, acquisition setups, etc.) that could lead domain shifts.

7 RELATED WORK

Deep Learning Side-Channel Attacks. Maghrebi et al. [25] first demonstrated that CNNs can outperform Template Attacks. Cagli et al. [5] suggested that CNNs can effectively recover keys over traces with random delay. Benadjila et al. [1] leveraged MLPs and CNNs in side-channel attacks and demonstrate that CNNs can defeat masking. They also introduce ASCAD dataset, which has been prominently used for benchmarking and comparison of deep learning side-channel attacks (same-device evaluations). Besides portability, other important aspects of deep learning side-channel attacks, such as explainability [45], synthetic trace generation [53], imbalance of data (with HW model) [33], hyperparameter tuning [29, 37, 52], attacking multiple bytes simultaneously [49], and model compression [22, 32], have been examined. Large Language Models [20] and Transformer Networks [17] are recently introduced to improve the effectiveness of side-channel attacks over traces with countermeasures. Li et al. [22] investigated structure pruning to reduce the size of neural networks for side-channel attacks.

Machine learning side-channel attacks can recover keys from other encryption algorithms, such as Kyber KEM (a post-quantum public-key encryption) [35], Ascon (a lightweight authenticated cipher) [36], and SNOW-V (a stream cipher in 5G) [39]. Some recent research [23, 41] utilized deep learning for *pre-silicon* side-channel attacks over power traces simulated at the RTL (Register-Transfer Level) or netlist level to identify leakage during the design stage. It is also feasible to perform non-profiling attacks with deep neural networks [21, 40, 43, 50, 51]. Two comprehensive surveys on deep learning side-channel attacks can be found in [30, 34].

Portability of Profiling Side-Channel Attacks. Many existing studies [2, 9–11, 14, 15, 38, 47, 48, 54–56] have investigated the portability of side-channel attacks when there are domain shifts between training and test data. Bhasin et al. [2] proposed to train neural networks with power traces from multiple targets to mitigate domain shifts between training and test traces in side-channel attacks. Danial et al. [9] trained neural networks with EM traces from multiple probe locations and also utilized pre-processing, including DFT, PCA, or LDA to address domain shifts. Rioja et al. [38] proposed a similarity method based on Dynamic Time Warping to quantify domain shifts. Wang et al. [47] examined domain shifts caused by inconsistent software settings between training and test targets (e.g., O0 vs. O1 optimization). Yu et al. [54] utilized meta-transfer learning to address domain shifts. Genevey-Metat et al. [14] examined fine tuning to address domain shifts over EM and power traces from STM32 microcontrollers. However, both of the two methods [14, 54] require labeled test traces (i.e., Type 1 pitfalls), which is impractical for real-world attackers in cross-device evaluations. Yu et al. [55] proposed to leverage a pre-trained neural network to produce synthetic labels for test traces and then fine tune the pre-trained neural network based on these test traces with synthetic labels for cross-device evaluations. However, it does not consider domain shifts caused by probe positions.

8 CONCLUSION

We investigate the portability of deep learning side-channel attacks over EM traces given domain shifts caused by hardware, keys, and probe locations. Moreover, we conduct comparative analysis over a set of pre-processing and unsupervised domain adaptation methods. Our findings suggest that these methods can improve the portability of deep-learning side-channel attacks on challenging targets. However, the effectiveness of each method varies depending

on the target and probe locations. A large-scale and comprehensive dataset of EM traces is established for the research community to reproduce and expand our findings.

ACKNOWLEDGEMENTS

The authors thank the anonymous shepherd and reviewers for their comments and suggestions. This work was partially supported by National Science Foundation (CNS-2150086, CNS-2225160, CNS-2238680) and CHEST – NSF IUCRC Center for Hardware and Embedded System Security and Trust (CNS-1916722).

REFERENCES

[1] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas. 2020. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering* 10, 2 (2020).

[2] S. Bhasin, A. Chattopadhyay, A. Heuser, D. Jap, S. Picek, and R. R. Shrivastwa. 2020. Mind the Portability: A Warriors Guide through Realistic Profiled Side-channel Analysis. In *Proc. of NDSS'20.*

[3] S. Bhasin, J. Danger, S. Guilley, and Z. Najm. 2014. NICV: Normalized inter-class variance for detection of side-channel leakage. In *2014 International Symposium on Electromagnetic Compatibility.*

[4] E. Brier, C. Clavier, and F. Olivier. 2004. Correlation Power Analysis with a Leakage Model. In *Proc. of CHES'04.*

[5] E. Cagli, C. Dumas, and E. Prouff. 2017. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures. In *Proc. of CHES'17.*

[6] P. Cao, C. Zhang, X. Lu, and D. Gu. 2021. Cross-Device Profiled Side-Channel Attack with Unsupervised Domain Adaptation. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 4 (2021), 27 – 56.

[7] S. Chari, J. R. Rao, and P. Rohatgi. 2002. Template Attacks. In *Proc. of Cryptographic Hardware and Embeeded Systems (CHES 2002).*

[8] J. Danial, D. Das, S. Ghosh, A. Raychowdhury, and S. Sen. 2020. SCNIFFER: Low-Cost, Automated, Efficient Electromagnetic Side-Channel Sniffing. *IEEE Access* (2020).

[9] J. Danial, D. Das, A. Golder, S. Ghosh, A. Raychowdhury, and S. Sen. 2022. EM-X-DL: Efficient Cross-device Deep Learning Side-channel Attack with Noisy EM Signatures. *ACM Journal on Emerging Technologies in Computing Systems* 18, 1 (2022), 1–17.

[10] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen. 2019. X-DeepSCA: Cross-Device Deep Learning Side Channel Attack. In *Proc. of 56th ACM/IEEE Design Automation Conference (DAC'19).*

[11] M. Elaabid and S. Guilley. 2012. Portability of templates. *Journal of Cryptographic Engineering* 2 (2012), 63–74.

[12] R. A. Fisher. 1936. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics* (1936).

[13] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. 2016. Domain-Adversarial Tranining of Neural Networks. *Journal of Machine Learning Research* (2016).

[14] C. Genevey-Metat, A. Heuser, and B. Gerard. 2021. Train or Adapt a Deeply Learned Profile?. In *Proc. of International Conference on Cryptology and Information Security in Latin America (Latin Crypt'21).*

[15] A. Golder, D. Das, J. Danial, S. Ghosh, S. Sen, and A. Raychowdhury. 2019. Practical Approaches Towards Deep-Learning Based Cross-Device Power Side Channel Attack. *IEEE. Trans. on Very Large-Scale Integration (VLSI) Systems* 27, 12 (2019).

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative Adversarial Networks. In *Proc. of the International Conference on Nerual Information Processing Systems (NIPS 2014).*

[17] S. Hajra, S. Chowdhury, and D. Mukhopadhyay. 2024. EstraNet: An Efficient Shift-Invariant Transformer Network for Side-Channel Analysis. *TCHES* (2024).

[18] I.T. Jolliffe. 2002. Principal Componenet Analysis. *Springer Series in Statistics* (2002).

[19] P. Kocher, J. Jaffe, and B. Jun. 1999. Differential Power Analysis. In *Proc. of CRYPTO'99.*

[20] P. Kulkarni, V. Verneuil, S. Picek, and L. Batina. [n. d.]. Order vs. Chaos: A Language Model Approach for Side-channel Attacks. ([n. d.]). https://eprint.iacr.org/2023/1615.pdf.

[21] D. Kwon, H. Kim, and S. Hong. 2021. Non-Profiled Deep Learning-based Side-Channel Preprocessing with Autoencoders. *IEEE Access* (2021).

[22] H. Li, M. Ninan, B. Wang, and J. M. Emmert. 2024. TinyPower: Side-Channel Attacks with Tiny Neural Networks. In *Proc. of IEEE HOST'24.*

[23] L. Lin, D. Zhu, J. Wen, H. Chen, Y. Lu, N. Cheng, C. Chow, H. Shrivastav, C. W. Chen, K. Monta, and M. Nagata. 2021. Multiphysics Simulation of EM Side-Channels from Silicon Backside with ML-based Auto-POI Identification. In *IEEE HOST'21.*

[24] X. Lu, C. Zhang, and D. Gu. 2021. Attention - Based Non-Profiled Side-Channel Attack. In *Proc. of 2021 Asian Hardware Oriented Security and Trust Symposium (AsianHost).*

[25] H. Maghrebi, T. Portigliatti, and E. Proff. 2016. Breaking cryptographic implementations using deep learning techniques. In *Proc. of International Conference on Security, Privacy and Applied Cryptography Engineering (SPACE'16).*

[26] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer New York.

[27] Loic Masure and Remi Strullu. 2023. Side-channel analysis against ANSSI's protected AES implementation on ARM: end-to-end attacks tih multi-task learning. *Journal of Cryptographic Engineering* 13 (2023), 129–147.

[28] D. P. Montminy, R. O. Baldwin, M. A. Temple, and E. D. Laspe. 2013. Improving cross-device attacks using zero-mean unit-variance normalization. *Journal of Cryptographic Engineering* (2013).

[29] S. Nouraniboosjin and F. Ganji. [n. d.]. Too Hot To Be True: Temperature Calibration for Higher Confidence in NN-assisted Side-channel Analysis. ([n. d.]). https://eprint.iacr.org/2024/071.pdf.

[30] M. Panoff, H. Yu, H. Shan, and Y. Jin. 2022. A Review and Comparison of AI-enhanced Side Channel Analysis. *J. Emerg. Technol. Comput. Syst.* (2022).

[31] K. Papagiannopoulos, O. Glamocanin, M. Azouaoui, D. Ros, F. Regazzoni, and M. Stojilovic. 2023. The Side-channel Metrics Cheat Sheet. *ACM Computing Surveys* 55, 10 (2023).

[32] G. Perin, L. Wu, and S. Picek. 2022. Gambling for Success: The Lottery Ticket Hypothesis in Deep Learning-Based Side-Channel Analysis. *Artificial Intelligence for Cybersecurity (Springer)* (2022).

[33] S. Picek, A. Heuser, A. Jovic, and F. Regazzoni. 2019. The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 1 (2019), 209–237.

[34] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina. 2023. SoK: Deep Learning-based Physical Side-channel Analysis. *ACM Computing Surveys* 55, 11 (2023).

[35] P. Ravi, D. Jap, S. Bhasin, and A. Chattopadhyay. [n. d.]. Machine Learning based Blind Side-Channel Attacks on PQC-based KEMs - A case Study of Kyber KEM. ([n. d.]). https://eprint.iacr.org/2024/169.pdf.

[36] A. Rezaeezade, A. Basurto-Becerra, L. Weissbart, and G. Perin. [n. d.]. One for All, All for Ascon: Ensemble-based Deep Learning Side-channel Analysis. ([n. d.]). https://eprint.iacr.org/2023/1922.pdf.

[37] J. Rijsdijk, L. Wu, G. Perin, and S. Picek. 2021. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021).

[38] U. Rioja, L. Batina, and I. Armendariz. 2020. When Similarities Among Devices are Taken for Granted: Another Look at Portability. In *Proc. of AFRICACRYPT 2020.* 337 – 357.

[39] H. Saurabh, A. Golder, S. S. Titti, S. Kundu, C. Li, A. Karmakar, and D. Das. [n. d.]. SNOW-SCA: ML-assisted Side-Channel Attack on SNOW-V. ([n. d.]). https://eprint.iacr.org/2024/428.pdf.

[40] I. Savu, M. Krcek, G. Perin, L. Wu, and S. Picek. [n. d.]. The Need for MORE: Unsupervised Side-channel Analysis with Single Network Training and Multi-output Regression. ([n. d.]). https://eprint.iacr.org/2023/1681.pdf.

[41] D. Shanmugam and P. Schaumont. 2023. Improving Side-Channel Leakage Assessment Using Pre-Silicon Leakage. In *International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE).*

[42] F. Standaert, B. Gierlichs, and I. Verbauwhede. 2008. Partition v.s. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices. In *Information Security and Cryptology – ICISC 2008.*

[43] B. Timon. 2019. Non-Profiled Deep Learning-based Side-Channel Attacks with Sensitivity Analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 2 (2019), 107–131.

[44] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. 2017. Adversarial Discriminative Domain Adaptation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[45] D. van der Valk, S. Picek, and S. Bhasin. 2020. Kilroy Was Here: The First Step Towards Explainability of Neural Networks in Profiled Side-Channel Analysis. In *International Workshop on Constructive Side-Channel Analysis and Secure Design.*

[46] C. Wang, J. Dani, S. Reilly, A. Brownfield, B. Wang, and J. M. Emmert. 2023. TripletPower: Deep-Learning Side-Channel Attacks over Few Traces. In *Proc. of IEEE HOST'23.*

[47] C. Wang, M. Ninan, S. Reilly, J. Ward, W. Hawkins, B. Wang, and J. M. Emmert. 2023. Portability of Deep-Learning Side-Channel Attacks against Software Discrepancies. In *Proc. ACM WiSec'23.*

[48] H. Wang, M. Brisfors, S. Forsmark, and E. Dubrova. 2019. How Diversity Affects Deep-Learning Side-Channel Attacks. In *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC).*

[49] L. Wu, A. Ali-Pour, A. Rezaeezade, G. Perin, and S. Picek. [n. d.]. Breaking Free: Leakage Model-free Deep Learning-based Side-channel Analysis. ([n. d.]). https://eprint.iacr.org/2023/1110.pdf.

[50] L. Wu, G. Perin, and S. Picek. [n. d.]. Hiding in Plain Sight: Non-profiling Deep Learning-based Side-channel Analysis with Plantext/Ciphertext. ([n. d.]). https://eprint.iacr.org/2023/209.pdf.
[51] L. Wu, S. Tiran, G. Perin, and S. Picek. [n. d.]. An End-to-end Plaintext-based Side-channel Collision Attack without Trace Segmentation. ([n. d.]). https://eprint.iacr.org/2023/1109.pdf.
[52] T. Yap, S. Bhasin, and L. Weissbart. [n. d.]. Train Wisely: Multifidelity Bayesian Optimization Hyperparameter Tuning in Side-Channel Analysis. ([n. d.]). https://eprint.iacr.org/2024/170.pdf.
[53] T. Yap and D. Jap. [n. d.]. Creating from Noise: Trace Generations Using Diffusion Model for Side-Channel Attack. ([n. d.]). https://eprint.iacr.org/2024/167.pdf.
[54] H. Yu, H. Shan, M. Panoff, and Y. Jin. 2021. Cross-Device Profiled Side-Channel Attacks using Meta-Transfer Learning. In *Proc. of the 58th ACM/IEEE Design Automation Conference (DAC'21).*
[55] H. Yu, S. Wang, H. Shan, M. Panoff, M. Lee, K. Yang, and Y. Jin. 2023. Dual-Leak: Deep Unsupervised Active Learning for Cross-Device Profiled Side-Channel Leakage Analysis. In *Proc. of IEEE HOST'23.*
[56] F. Zhang, B. Shao, G. Xu, B. Yang, Z. Yang, Z. Qin, and K. Ren. 2020. From Homogeneous to Heterogeneous: Leveraging Deep Learning based Power Anlysis across Devices. In *Proc. of 57th ACM/IEEE Design Automation Conference (DAC'20).*