TripletPower: Deep-Learning Side-Channel Attacks over Few Traces

Chenggang Wang, Jimmy Dani, Shane Reilly, Austen Brownfield, Boyang Wang, John M. Emmert
University of Cincinnati
{*wang2c9, danijy, reillysp, brownfaw*}*@mail.uc.edu, boyang.wang@uc.edu, john.emmert@uc.edu*

***Abstract*—Deep learning has been utilized as a promising technique in side-channel attacks. However, to recover keys successfully, deep-learning side-channel attacks often require *thousands of training traces*, which could be challenging for an attacker to obtain in the real world. This paper proposes a new deep-learning side-channel attack which only *requires hundreds of training traces*. Our proposed method, referred to as TripletPower, trains a *triplet network*, which learns a robust embedding for side-channel attacks with few traces. We demonstrate the advantage of our method in profiling attacks over power traces collected from AVR XMEGA and ARM STM32 microcontrollers using ChipWhisperer. Specifically, experimental results show that our method only needs as low as 250 training traces to train a classifier successfully recovering keys of unmasked AES on XMEGA (or STM32) while a Convolutional Neural Network needs at least 4,000 training traces in profiling attacks. In addition, we extend our method to non-profiling attacks with on-the-fly labeling. Experimental results suggest that our method can effectively recover keys of unmasked AES on XMEGA with only 525 unlabeled power traces in non-profiling attacks. Our method is also effective over power traces collected from masked AES and traces generated with random delay.**

I. Introduction

Side-channel attacks [1]–[4] can reveal encryption keys on a target device, e.g., a microcontroller or FPGA, by examining correlations between power consumption and intermediate outputs of an encryption algorithm. Studies [5]–[19] have demonstrated that machine learning, particularly deep learning, can offer new advantages compared to traditional approaches, such as Correlation Power Analysis [3]. For instance, deep-learning side-channel attacks can recover keys when masking or random delays are applied [9], [10].

However, one primary limitation of existing studies is that a significant number of training traces is required to train a classifier successfully recovering keys. Specifically, to break a key of AES (Advanced Encryption Standard) running on a microcontroller, a Convolutional Neural Network (CNN) often requires at least *several thousands* of training traces [10]. While obtaining thousands of training traces from a target is feasible in a lab environment, it could be challenging for a real-world attacker. For instance, a real-world attacker may not have sufficient attack windows to capture thousands of training traces from a target.

In this paper, we propose *TripletPower*, a new deep-learning side-channel attack that requires fewer training traces to train a classifier to successfully recover keys. Specifically, our method learns an embedding by leveraging *triplet networks* [20] and trains a classifier built upon the embedding. A triplet network consists of three sub-networks in parallel, where each sub-network is a deep neural network. It requires less training data and learns a more robust embedding such that data from the same classes are close while data from different classes are apart in the embedded space. These features are utilized in this paper to distinguish power consumption of intermediate outputs of AES, and therefore, recover keys using a classifier trained with few traces.

The main contributions of this paper are summarized below:

- We propose TriplePower, a method that requires *only hundreds of training traces* to train a classifier successfully recovering keys in side-channel attacks. We demonstrate that our method can recover AES keys in profiling attacks. Moreover, our method can be extended to non-profiling attacks to recover keys.
- For a profiling attack with TriplePower, we first learn an embedding with triplet networks and then attach a k-nn (k-nearest-neighbor) to the end of the embedding to form a classifier performing side-channel attacks. We conduct extensive evaluations in both *same-device* profiling attacks and *cross-device* profiling attacks over power traces collected from microcontrollers using ChipWhisperer [21]. We examine *power traces* of unmasked AES on AVR XMEGA (8-bit RISC) and ARM STM32 (32-bit Cortex-M4). Experimental results show that our method requires a much lower number of training traces than a CNN. For instance, a CNN requires at least 4,000 training traces to train a classifier while our method needs as low as 250 traces to train a classifier recovering keys of unmasked AES on XMEGA. Our method is also effective against masked AES and random delay on XMEGA.
- For a non-profiling attack with TripletPower, we extend our method in the profiling attack with an existing technique, named *on-the-fly labeling* [18], [22] (also known as *partition-based Differential Power Analysis* [23]). Specifically, given a set of power traces without labels (as the key remains unknown to an attacker) in a non-profiling attack, our method first produces 256 labeled sets of power traces, trains 256 embeddings with triplet networks, and forms 256 classifiers accordingly. Each labeled set of power traces, its embedding, and its classifier are derived

based on each guess key byte value from 0 to 255 (`0x00` to `0xff` in hex). We leverage *attack accuracy* as the *Distinguisher* of on-the-fly labeling, where the attack accuracy of the classifier obtained from the correct guess key byte value distinctly outperforms the attack accuracy of other classifiers trained from incorrect guesses key byte values. Our experimental results show that TripletPower can recover keys with as low as 525 traces (500 for training and 25 for testing) over unmasked AES on XMEGA and 700 traces (500 for training and 200 for testing) over unmasked AES on STM32 in non-profiling attacks. Our method also recovers keys over traces from masked AES and random delay on XMEGA.

Reproducibility. The source code and datasets of our study can be found at [24].

We would like to acknowledge that Wu et al. [25] *first* proposed to utilize triplet networks in the context of side-channel attacks. The work was published recently by CHES'22. Different from our work, Wu et al. leveraged triplet networks to learn a robust embedding for template attacks. Their primary motivation is to reduce training time by running one-epoch training over triplet networks while our goal is to reduce the number of training traces. Wu et al. also proposed a new metric named Hybrid Distance, which can improve the quality of the embedding. Despite the differences, both [25] and our study indicate that triplet networks, or essentially *deep metric learning* [26], offer new opportunities for deep-learning side-channel attacks. Our work was conducted in parallel and we were not aware of the study published in [25] at the time of our paper submission.

II. Background

A. Machine-Learning Profiling Attacks.

System and Threat Model. As shown in Fig 1, the system model of machine-learning profiling side-channel attacks includes two targets, including a *training target* and a *test target*. Both of the targets run the same encryption algorithm. There is an *adversary* — either a security analyzer or malicious attacker — who aims to recover an *unknown but fixed* key on the test target. This adversary does not have control of the test target but can passively capture its power consumption and associated plaintexts and/or ciphertexts.

On the other hand, this adversary has complete control of the training target to assist her to learn a profile (e.g., a classifier) to recover the unknown key on the test target. Specifically, this adversary knows the key on the training target and can change it if needed. In addition, this adversary can choose plaintexts and collect a significant amount of power traces when the training target runs an encryption algorithm.

The attack consists of two phases, the *training phase* (a.k.a., *profiling phase*) and the *test phase* (a.k.a., *attack phase*). In the training phase, this adversary trains a classifier leveraging machine learning algorithms with (labeled) power traces collected from the training target. In the test phase, the adversary collects (unlabeled) power traces from the test target and aims to recover the key on the test target with the trained classifier.

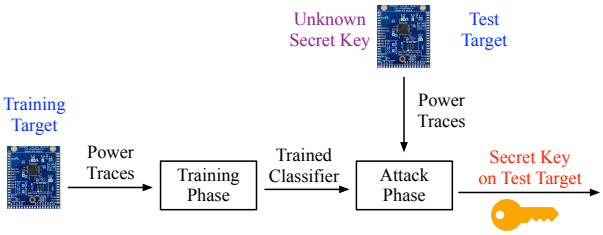


Fig. 1: The system model of profiling side-channel attacks.

Same-Device v.s. Cross-Device. The evaluation of a profiling attack can be carried out in two settings, including the *same-device setting* and *cross-device setting*. We examine both same-device and cross-device profiling attacks in this paper.

In the same-device setting, the training target and test target are exactly the same. In other words, training data and test data are collected from the same target running the same key. This setting can simplify the setup and lower hardware expense of data acquisition. On the other hand, the same-device setting ignores discrepancies caused by hardware imperfections, different keys, and different setups between two targets.

In the cross-device setting, the training target and test target are still the same type but not identical. In addition, the training target and test target use two different keys. It requires additional hardware and more time in data acquisition. Several recent studies [27]–[32] highlight the importance of evaluating deep-learning profiling attacks in the cross-device setting.

B. Notations and Leakage Model

A power trace t is a one-dimensional time series data $t = (t[1], ..., t[l])$, where $t[i]$ is the measurement of power consumption at time i and l is the number of measurements in a power trace. A set T consisting of N power traces is denoted as $T = (t_1, ..., t_N)$. A power trace t contains the power consumption of a target when it runs encryption with a plaintext $m \in \mathcal{M}$ and key $k \in \mathcal{K}$, where m is the input and k is the key of an encryption algorithm. $\mathcal{M}$ is the plaintext space and $\mathcal{K}$ is the key space. Given plaintext m and key k, an intermediate output of the encryption algorithm is denoted as $z = \varphi(m, k)$, where function $\varphi(\cdot)$ is a leakage step.

We focus on attacks on AES, where a side-channel attack operates at the *byte level* and reveals the entire key *byte by byte*. For example, to compromise all the 16 bytes of one AES-128 key, the attack will repeat 16 rounds on the same set of power traces to reveal one byte in each round. In the j-th round, where $j \in [1, 16]$, the label of a power trace is obtained based on the j-th byte of the intermediate output.

Following the conventions of existing literature, *the description of side-channel attacks in the rest of this paper will focus on one byte*, where we assume key k, plaintext m or intermediate output z only has one byte. We use $k_1^*, k_2^*, ..., k_{256}^*$ to denote all the possible 256 key values of one byte. We use SubBytes (i.e., S-box) of the 1st round of AES as the leakage step $\varphi(\cdot)$ for computing intermediate outputs. In other

words, we assume that the power consumption of the output of SubBytes can leak the information of a key.

We leverage **Hamming Weight (HW) model** [10], [33] to formulate the leakage, where the power consumption of two intermediate outputs are considered distinguishable if the Hamming weights of the two intermediate outputs are different. As one byte has 8 bits, there are 9 unique Hamming weights, i.e., $\{0, 1, ..., 8\}$. Each power trace t can be labeled with, $\texttt{HW}(z = \varphi(m, k))$, the Hamming weight of the intermediate output generated by plaintext m and key k. We use $\texttt{HW}(\cdot)$ to denote the function of calculating Hamming weights. Besides Hamming Weight model, other leakage models, such as the *identity model* and the *least/most significant bit model*, can also be used to formulate side-channel leakage [10].

C. Evaluation Metrics

The performance of machine-learning side-channel attacks can be measured with two metrics, *attack accuracy* and *key rank* (a.k.a. *guessing entropy* [33]). Attack accuracy is the same concept as accuracy of a classifier in machine learning. Key rank is the rank of the correct key among all the possible keys based on aggregated scores over a set of power traces. Key rank is considered as a *primary metric* while attack accuracy is *secondary*. This is because attack accuracy in side-channel attacks can be much lower (e.g., 30%) compared to other areas (e.g., over 95% in image recognition), but the attack can still recover keys effectively when it is measured with key rank.

In the training phase, given a training dataset $D_{train} = \{T, M, k\}$, which consists of power traces $T = (t_1, ..., t_N)$, plaintexts $M = (m_1, ..., m_N)$ and key k, where m_i is associated with t_i, an attacker computes intermediate outputs $Z = (z_1, ..., z_N)$, where $z_i = \varphi(m_i, k)$, and assigns $h_i = \texttt{HW}(z_i)$ as the label of power trace t_i. An attacker trains a classifier F with $(T, H) = \{(t_1, h_1), ..., (t_N, h_N)\}$.

In the attack phase, there is a test dataset $D_{test} = \{T', M', k'\}$, which consists of power traces $T' = (t'_1, ..., t'_{N'})$, plaintexts $M' = (m'_1, ..., m'_{N'})$ and key k', where m'_i is associated with t'_i. Given power trace t'_i, an attacker first obtains a score for each Hamming weight from classifier F. Next, each possible intermediate output obtains its score based on its Hamming weight. Then, each possible key obtains its score based on its associated intermediate output, plaintext m'_i, and function $\varphi(\cdot)$.

Attack Accuracy. Given a power trace t'_i, classifier F outputs a *HW score vector* $(s_i[0], ..., s_i[8])$, where $s_i[j]$ is the score for Hamming weight j. If $s_i[g]$ is the highest score among $(s_i[0], ..., s_i[8])$ and the Hamming weight of intermediate output $z'_i = \varphi(m'_i, k')$ is g, the prediction of classifier F is considered correct for power trace t'_i. ***Attack accuracy*** is computed as x/N', where x is the number of power traces obtaining the correct Hamming weight among N' power traces.

Key Rank. Given a HW score vector $(s_i[0], ..., s_i[8])$ obtained from trace t'_i, an attacker obtains a *key score vector* $(r_i[k^*_1], ..., r_i[k^*_{256}])$ by calculating

$$r_i[k^*_g] = s_i[j], \quad \text{if} \quad \texttt{HW}(\varphi(m'_i, k^*_g)) == j \tag{1}$$

for $1 \le g \le |\mathcal{K}| = 256$, where $r_i[k^*_g]$ is the score of possible key k^*_g based on trace t'_i and plaintext m'.

The *aggregated key score vector* $(r[k^*_1], ..., r[k^*_{256}])$ over N' power traces are computed as

$$r[k^*_j] = \sum_{i=1}^{N'} r_i[k^*_j], \ \textit{for} \ 1 \le j \le 256 \tag{2}$$

The aggregated key scores $(r[k^*_1], ..., r[k^*_{256}])$ are further sorted in descending order. ***Key rank*** is denoted as w, where $w \in [0, 255]$, if key k' is ranked as the $(w+1)$-th key among all the 256 possible keys based on the aggregated scores. A key rank of 0 over N' traces suggests that an attacker can recover key k' with N' traces. If key rank converges to 0 with a lower number of traces, it indicates that an attack is more effective.

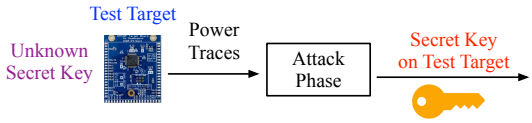


Fig. 2: The system model of non-profiling side-channel attacks.

D. Machine-Learning Non-Profiling Attacks

In a non-profiling attack, an attacker can only obtain a set of power traces and associated plaintexts from a test target but it does not know the key (in Fig. 2). In other words, there are no labeled traces. The goal of this attacker is to recover an unknown but fixed key on the test target. A non-profiling attack is more challenging than a profiling attack as the attacker no longer has access to a training target.

There are different ways to perform non-profiling attacks. One effective approach is *on-the-fly labeling* [18], [22] (also known as *partition-based Differential Power Analysis* [23]), which can transform a non-profiling attack into 256 profiling attacks by enumerating all the possible 256 keys. In each instance of a profiling attack, a classifier is trained accordingly. The training (or/and the testing) of all the 256 classifiers is considered as *the attack phase* of the non-profiling attack.

Specifically, given power trace set $T = (t_1, ..., t_N)$, its associated plaintexts $M = (m_1, ..., m_N)$, and all the 256 possible keys $k^*_1, ..., k^*_{256}$, the attacker generates label sets $H[k^*_1], ..., H[k^*_{256}]$ where label set $H[k^*_i] = (h_1[k^*_i], ..., h_N[k^*_i])$ is computed based on guess key k^*_i and $h_j[k^*_i] = \texttt{HW}(\varphi(m_j, k^*_i))$ is the Hamming weight of guess intermediate output based on plaintext m_j and guess key k^*_i. $h_j[k^*_i]$ is also the guess label of trace t_j given guess key k^*_i.

As a result, on-the-fly labeling produces 256 labeled datasets $(T, H[k^*_1]), (T, H[k^*_2]), ..., (T, H[k^*_{256}])$, where the power traces remain the same but the labels are different due to each guess key. Given each labeled dataset $(T, H[k^*_i])$, an attacker trains a classifier $F[k^*_i]$. A metric, such as the sensitivity of a neural network (if a neural network is utilized

as a classifier) [18], is utilized as a *Distinguisher* to reveal the correct key. Other metrics, such as Mutual Information, can be used as the Distinguisher in traditional partition-based Differential Power Analysis [23].

We use HW model to partition unlabeled traces in on-the-fly labeling in this study. It is worth mentioning that *the identity model should not be used* with on-the-fly labeling as all the partitions of unlabeled traces generated by the identity model would remain the same regardless of which guess key it is [23]. When all the partitions remain the same for all the guess keys, the Distinguisher fails to distinguish the correct key from incorrect ones.

III. The Design of TripletPower

A. Background on Triplet Networks

A triplet network [20] contains three parallel sub-networks sharing identical weights and hyperparameters. An input of a triplet network is referred to as a *triplet*. It consists of an anchor sample A_S, a positive sample P_S, and a negative sample N_S. Triplets are selected from a set of samples, either randomly or based on certain mining strategy [34]. Note that an anchor sample, a positive sample, or a negative sample is a power trace in the context of side-channel attacks.

Training a triplet network with triplets can obtain an embedding, such that the distance between anchors and positive samples is smaller than the distance between anchors and negative samples in an embedded space. One advantage of triplet networks is that the training, in general, requires less data compared to other neural networks.

Triplet loss is utilized to measure the loss during the training [20]. Given a triplet (A_S, P_S, N_S), triplet loss is defined as

$$\begin{aligned} \mathcal{L}(A_S, P_S, N_S) &= \max(d_{AP} - d_{AS} + \alpha, 0) \\ d_{AP} &= ||f(A_S) - f(P_S)||^2 \\ d_{AS} &= ||f(A_S) - f(N_S)||^2 \end{aligned}$$

where $f(\cdot)$ is the embedding, α is a margin between positive and negative samples. The training aims to *minimize* the triplet loss. Cosine distance can be utilized [20] to measure distances between anchors and positive samples (or negative samples).

Since the training of a triplet network only obtains an embedding, an additional training for a classifier is needed in order to perform classifications. Specifically, after the training of a triplet network, a trained sub-network is extracted and utilized as the embedding. A classifier, such as k-nn or SVM (Support Vector Machine), is attached to the sub-network to form the whole classifier. The parameters of k-nn or SVM are trained using additional samples while hyperparameters and weights of the sub-network are frozen. Once trained, the whole classifier (consisting of the embedding and k-nn) can be used to perform classifications.

B. Profiling Attacks with TripletPower

We first describe how to perform profiling side-channel attacks with triplet networks. Specifically, given a number of N training power traces, our method TripletPower first labels each power trace by calculating the Hamming weight of the intermediate output based on the associated plaintext and key as mentioned in Sec. II. Next, it takes *at most* U traces per class (per Hamming weight) to mine triplets and train a triplet network. We refer U as the *Per-Class Threshold* in this paper.

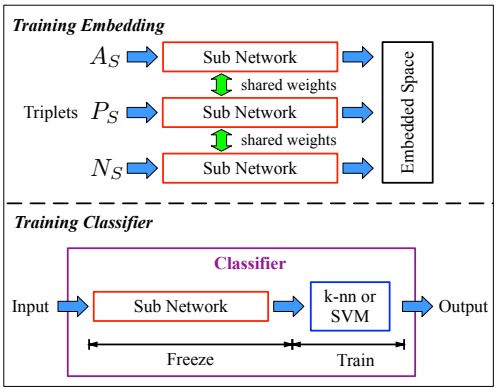


Fig. 3: Overview of a triplet network

There are *two reasons* that we set this threshold U during the training of a triplet network. **First**, given a number of N training traces, there are approximately 71% (i.e., $\frac{C_8^3+C_8^4+C_8^5}{256}$) of these traces will be labeled with Hamming weight 3, 4, or 5. We refer to Hamming weight 3, 4, and 5 as *dominating classes*. If we include a much greater number of traces from these dominating classes compared to the number of traces selected for other classes (e.g., HW=0 or HW=8), we could end up with many easy triplets generated from these dominating classes. An easy triplet is a triplet where the distance between an anchor sample and a positive sample is already less than the distance between an anchor sample and a negative sample in the input space. Easy triplets do not help a triplet network effectively improve the embedding, and therefore, should be avoided. **Second**, setting the per-class threshold can also reduce the mining time and training time of a triplet network, and therefore, optimize the overall training time of our method.

Once a triplet network is trained, our method extracts the sub-network and attaches a k-nn to the end of it to form a classifier. We train the classifier by updating the weights of k-nn but freezing weights in the sub-network. Once the classifier is trained, TripletPower takes a number of V test power traces to calculate the accuracy and key rank as defined in Sec. II.

C. Non-Profiling Attacks with TripletPower.

Our method can also be extended to perform non-profiling attacks by leveraging on-the-fly labeling. Specifically, given a set of power traces $T = (t_1, ..., t_N, ..., t_{N+V})$, its associated plaintexts $M = (m_1, ..., m_N, ..., m_{N+V})$, our method first produces 256 labeled datasets $(T, H[k_1^*]), (T, H[k_2^*]), ..., (T, H[k_{256}^*])$ by following on-the-fly labeling, where $H[k_i^*] = (h_1[k_i^*], ..., h_{N+V}[k_i^*])$ and $h_j[k_i^*]$ is the guess label of trace t_j based on guess key k_i^*. Given each labeled dataset $(T, H[k_i^*])$, TripletPower leverages the first N traces $(t_1, ..., t_N)$ and their guess labels to train a classifier $F_{k_i^*}$, where it trains a triplet network with at most

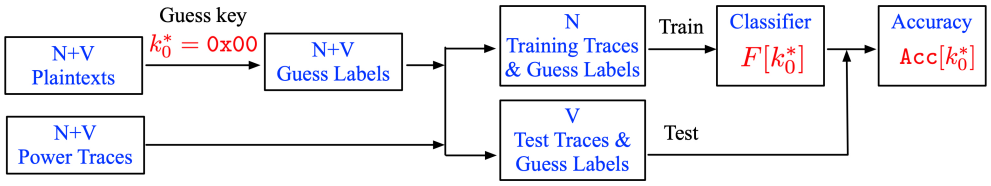


Fig. 4: Highlight of non-profiling attacks with TripletPower: The steps for training and testing one classifier $F_{k_0^*}$ based on guess key $k_0^* = \mathtt{0x00}$. The steps above are repeated for 256 iterations and each iteration uses a different guess key but with the same $N + V$ power traces and $N + V$ plaintexts.

U traces per class and train classifier $F[k_i^*]$ with all the N traces as in a profiling attack. Next, the remaining V traces $(t_{N+1}, ..., t_{N+V})$ and their guess labels are utilized in testing to measure the attack accuracy of each classifier $F[k_i^*]$. Fig. 4 shows the steps for one classifier $F[k_0^*]$.

TripletPower leverages the attack accuracy of a classifier during the testing as the Distinguisher. The intuition is that if the guess key is correct, the embedding of a trained triplet network can distinguish traces across different classes, and therefore, lead to better performance for the classifier during the testing. On the other hand, if the guess keys are incorrect, the embedding of a trained triplet network cannot distinguish traces across different classes, and therefore, lead to poor performance during attacks.

Specifically, given test accuracy $\mathtt{Acc}[k_1^*], ..., \mathtt{Acc}[k_{256}^*]$ obtained from all the 256 classifiers and the unknown key k' on the test target, where $\mathtt{Acc}[k_i^*]$, for $1 \leq i \leq 256$, is the testing accuracy of classifier $F[k_i^*]$ over traces $(t_{N+1}, ..., t_{N+V})$ and associated guess labels derived from guess key k_i^*, the non-profiling attack with TripletPower is considered successful if

$$k' = \arg\max_{k_i^*}(\mathtt{Acc}[k_i^*]) \tag{3}$$

In other words, if the test accuracy derived from the classifier trained with the correct guess key achieves the highest accuracy, the non-profiling attack is successful.

IV. Evaluation

A. Data Collection

We examine side-channel attacks by analyzing the power consumption of a target running AES-128 encryption. Specifically, we utilize ChipWhisperer Level 1 Kit as capture boards to collect power traces from multiple targets, including XMEGA (8-bit RISC) microcontrollers and STM32F3 (32-bit Cortex-M4) microcontrollers. Both XMEGA and STM32F3 are widely adopted in embedded systems and IoT devices. ChipWhisperer Level 1 Kit offers a sampling rate of 105 million measurements per second [21].

As shown in Fig. 5, we leverage two computers (PC1 and PC2) and two Chipwhisperer Level 1 Kits (CL1 and CL2) to collect power traces. PC1 is equipped with Intel i5 CPU, 16GB RAM, and USB 3.0; PC2 is equipped with Intel i7 CPU, 32GB RAM, and USB 3.0. Both of them run Ubuntu 18.04.

Two XMEGA targets (X1 and X2) and two STM32F3 targets (S1 and S2) are examined. We always use PC1 and CL1 to collect data from X1 (and S1) and we always use PC2 and CL2 to obtain data from X2 (and S2). For ease of description, we will only keep the name of a target when we refer to a dataset. For the name of each dataset we collect, we use the following convention to describe it

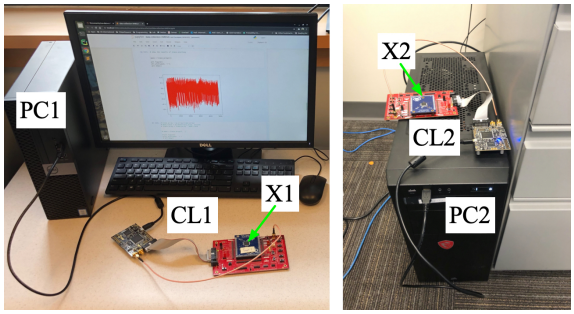


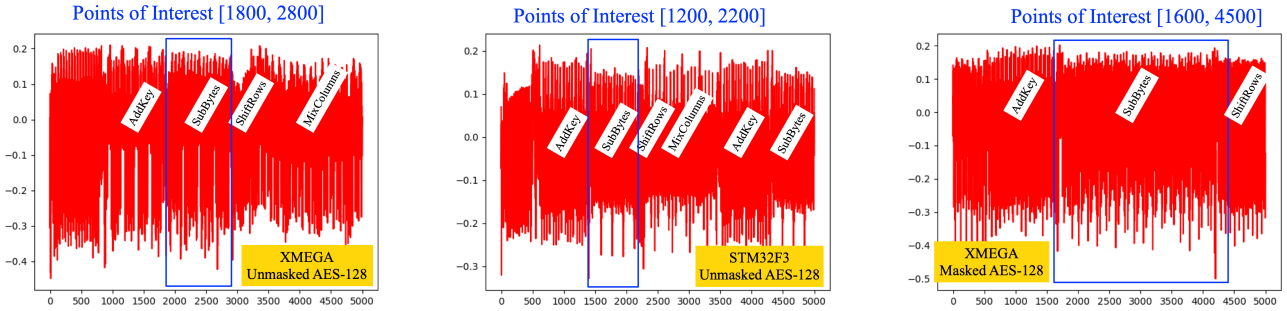
Fig. 5: The data collection setup

`Target_Key_MaskedInfo_NoOfTraces`

where `Target` indicates which target it is, `Key` suggests which key was used, `NoOfTraces` shows how many traces are included, `MaskedInfo={U, M}` suggests whether it is collected from unmasked AES (`U`) or masked AES (`M`). For instance, `X1_K1_M_20k` indicates that a dataset was collected from X1 running masked AES using secret key K1 and this dataset includes 20,000 traces.

We collect power traces from unmasked AES-128 on XMEGA and STM32F3 and masked AES-128 on XMEGA. For unmasked AES, we use tinyAES (written in C) provided by ChipWhisperer APIs [35]. For masked AES on XEMGA, we use the implementation (version 1) from [36], which is written in assembly. How to compile this assembly code for ChipWhisperer can be found on our GitHub repository [24]. Each (raw) power trace includes a sequence of power measurements when a target runs AES encryption.

Points of Interest. We use *Points of Interest* to denote the corresponding power measurements associated with SubBytes of the first round of AES-128. For unmasked AES-128 on XMEGA, we use the default offset (offset=0) in ChipWhisperer APIs during the data collection and select [1800, 2800] as the Points of Interest as shown in Fig. 6. For unmasked AES-128 on STM32F3, we select [1200, 2200] as the Points of Interest. For masked AES-128 on XMEGA, we are not able to observe power measurements associated with SubBytes of the



(a) XMEGA, unmasked AES (b) STM32, unmasked AES (c) XMEGA, masked AES (offset 17,500)

Fig. 6: Power Pattern of AES-128 on Microcontrollers.

TABLE I: Our TP Dataset

XMEGA, Unmasked	STM32, Unmasked	XMEGA, Masked	XMEGA, Unmasked, Random Delay
X1_K0_U_200k	S1_K0_U_200k	X1_K0_M_200k	X1_K0_U_Delay_200k
X1_K0_U_20k	S1_K0_U_20k	X1_K0_M_20k	X1_K0_U_Delay_20k
X2_K1_U_20k	S2_K2_U_20k	X2_K3_M_20k	X2_K1_U_Delay_20k

TABLE II: Keys used in TP Dataset

K0	0x2b, 7e, 15, 16, 28, ae, d2, a6, ab, f7, 15, 88, 09, cf, 4f, 3c
K1	0x95, 19, 7f, 66, b0, 6d, 6e, 21, 67, 8a, fa, 8f, 87, a9, 64, e5
K2	0x10, 6c, 38, 41, d5, e8, a5, 6e, 22, b1, c9, 9f, cc, 4b, 25, f8
K3	0xa6, 9f, c8, 50, 7d, f6, 60, 03, ce, 44, 45, b1, 70, 47, f0, 50

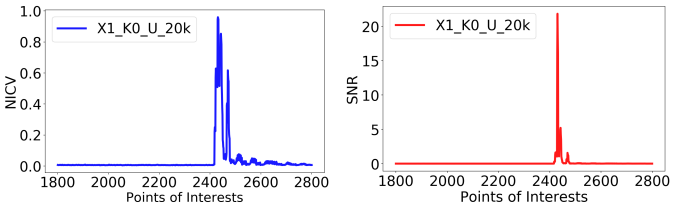


Fig. 7: NICV and SNR of X1_K0_U_20k (3rd byte)

first round with offset 0. Therefore, we set offset=17,500 and select [1600, 4500] as the Points of Interest.

We identify Points of Interest in advance, where we use consecutive dummy operations, e.g., 20∼30 NOPs (No Operations), before and after SubBytes of the first round of AES in C or assembly code, to create a visible gap in power consumption. We adjust the offset incrementally in order to observe the gap if needed. The Points of Interest are further validated with Normalized Inter-Class Variance (NICV) [37] and Signal to Noise Ratio (SNR) [38] to ensure leakage points (i.e., high peaks) are included. The NICV and SNR of dataset X1_K0_U_20k over 3rd byte are shown in Fig. 7. For all our side-channel attacks, given a (raw) power trace, we extract the Points of Interest to form one input to a classifier.

TP Dataset. We collect a large-scale power trace dataset, denoted as **TripletPower (TP) dataset**. It contains 960,000 power traces in total with an overall size of 43 GBs. The entire TP dataset consists of 12 datasets using different targets and keys. Different keys and messages were generated randomly and recorded. Each dataset consists of tuples, where each tuple is recorded in the form of (trace, plaintext, key).

TABLE III: Hyperparameters of Neural Networks

	CNN	Sub-Network (Ours)
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**; **softmax**	No. of neurons: **256**; **Relu**

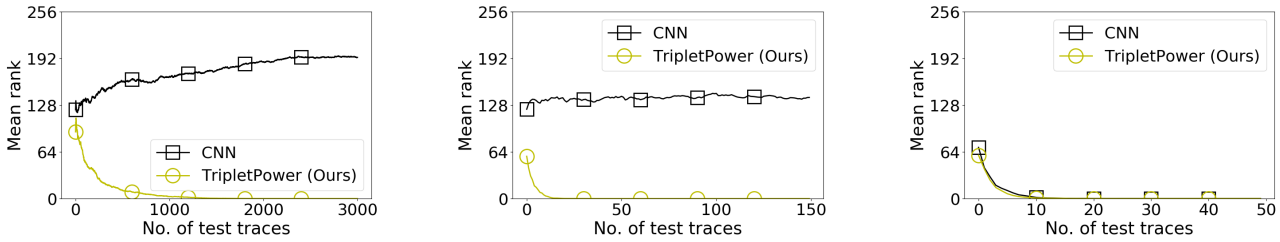
The number of tuples in a dataset is the same as the number of traces. The list of our datasets is summarized in Table I. The keys involved in TP dataset are presented in Table II.

We also generate power traces by simulating random delays. For instance, we generate dataset X1_K0_U_Delay_200k based on dataset X1_K0_U_200k by shifting Points of Interest of each power trace with a delay randomly selected from [0, 10]. We use the same approach to generate the other two datasets with random delays.

B. Experiments

Experiment Setting. We implement our method with Python 3.6. Specifically, we utilize Scikit-learn and Tensorflow 2.3 to implement machine learning algorithms. The experiments are conducted on a desktop with Ubuntu 18.04, Intel Core i7 CPU, NVIDIA Titan RTX GPU, and 64GB RAM. For all of our experiments, we always report attack results on the **3rd byte** of an AES key and we use Hamming Weight model to formulate the leakage.

Architectures of Neural Networks. For CNN, we use the CNN from the ASCAD paper [10]. It consists of 5 blocks and 2



(a) No. of training traces $N = 250$ (b) No. of training traces $N = 2,000$ (c) No. of training traces $N = 4,000$

Fig. 8: Comparison of Key Ranks between CNN and TripletPower (**XMEGA, unmasked AES, 3rd byte**).

fully connected layers and an output layer. Each block consists of 1 convolutional layer and 1 average pooling layer. Detailed hyperparameters of this CNN can be found in Table III. For TripletPower, we choose the same CNN as a sub-network of a triplet network. We slightly modify the output layer of the CNN as the sub-network serves as embedding, not classification. For k-nn classifier in TripletPower, we set $k = 10$.

Experiment 1. Comparison between CNN and TripletPower (Profiling, Same-Device). We compare the attack performance of CNN and TripletPower in **same-device profiling side-channel attacks**, where the training and test data are collected from the same target with the same key. We first examine unmasked AES on XMEGA and STEM32.

For unmasked AES on XMEGA, we use `X1_K0_U_200k` for training and `X1_K0_U_20k` for testing. Given the training dataset, we randomly choose N traces for training, where $N = \{125, 250, 500, 1k, 2k, 4k, 8k, 16k\}$. Given each N, we train a CNN with 100 epochs. For a fair comparison, we use the same N traces to train our method TripletPower. Specifically, we first train a triplet network for 100 epochs by setting per-class threshold as $U = 300$. We defer the discussion on the impact of per-class threshold in a later experiment. We use offline *semi-hard* mining to select triplets. Once we train a triplet network, we use all the N traces to train a k-nn classifier.

As shown in Table IV, for unmasked AES on XEMGA, when the number of training traces N is 125, neither CNN or TripletPower can recover the key, i.e., the key rank does not converge to 0. When we increase N, both CNN and TripletPower are able to perform better. However, TripletPower outperforms CNN when N remains relatively low. Note that we leverage key rank (or parameter R, the number of test traces that key rank converges to 0) in the attack phase as the primary metric to compare the attack performance of CNN and TripletPower. Attack accuracy is utilized as a secondary metric in the comparison. Key ranks are always reported on average by running the evaluation five times, where the order of test traces is randomly shuffled every time.

Specifically, when $250 \leq N \leq 2,000$, TripletPower can recover keys while CNN cannot. For instance, given $N = 1,000$, TripletPower can recover the key within 123 test traces in the attack phase while CNN cannot. The detailed key ranks for some of the values of N are presented in Fig. 8. As a necessary tradeoff, TripletPower requires a longer training time than CNN given the same N. CNN can eventually outperform TriplePower when there are 4,000 or more training traces.

TABLE IV: Comparison of CNN and TripletPower, **Same-Device** Profiling Attacks, Hamming Weight Model, **XMEGA**, **Unmasked AES**, **3rd byte**, Train: `X1_K0_U_200k`, Test: `X1_K0_U_20k`, N: No. of Training Traces, R: No. of Test Traces Key Rank Converging to 0 (– indicates that key rank does not converge to 0), TT: Training Time (seconds).

N	CNN			TripletPower		
	ACC	R (traces)	TT (sec)	ACC	R (traces)	TT (sec)
125	21.69%	–	8	17.20%	–	102
250	28.18%	–	11	24.12%	**2,983**	333
500	28.18%	–	17	30.21%	**978**	1,311
1,000	28.18%	–	30	43.97%	**123**	3,730
2,000	28.18%	–	56	53.89%	**32**	9,763
4,000	41.16%	**30**	130	57.29%	33	12,388
8,000	53.48%	**16**	299	58.57%	26	14,827
16,000	62.91%	**16**	524	55.01%	27	17,322

TABLE V: Comparison of CNN and TripletPower, **Same-Device** Profiling Attacks, Hamming Weight Model, **STM32F3**, **Unmasked AES**, **3rd byte**, Train: `S1_K0_U_200k`, Test: `S1_K0_U_20k`, N: No. of Training Traces, R: No. of Test Traces Key Rank Converging to 0 (– indicates that key rank does not converge to 0), TT: Training Time (seconds).

N	CNN			TripletPower		
	ACC	R (traces)	TT (sec)	ACC	R (traces)	TT (sec)
125	21.20%	–	8	25.88%	–	100
250	27.50%	–	11	25.96%	**594**	321
500	27.50%	–	17	40.72%	**78**	1,255
1,000	27.29%	–	28	56.87%	**43**	5,040
2,000	27.29%	–	54	55.78%	**52**	9,454
4,000	27.29%	–	105	49.97%	**36**	12,445
8,000	50.26%	**23**	260	47.75%	24	14,927
16,000	64.42%	**9**	502	51.39%	26	18,724

It is worth mentioning that a higher accuracy does not necessarily lead to a better performance in key rank, especially when accuracy is at a similar level. For instance, as shown in Table IV, given $N = 250$, CNN achieves 28.18% accuracy while TripletPower achieves a lower accuracy of 24.14%. However, key rank shows that TriplePower can recover the key with 2,983 traces while CNN cannot.

We have consistent observations between CNN and TripletPower from unmasked AES on STM32 (Table V), masked

TABLE VI: Comparison of CNN and TripletPower, **Same-Device** Profiling Attacks, Hamming Weight Model, **XMEGA**, **Masked AES**, **3rd byte**, Train: `X1_K0_M_200k`, Test: `X1_K0_M_20k`, N: No. of Training Traces, R: No. of Test Traces Key Rank Converging to 0 (– indicates that key rank does not converge to 0), TT: Training Time (seconds).

N	CNN			TripletPower		
	ACC	R (traces)	TT (sec)	ACC	R (traces)	TT (sec)
125	22.13%	–	14	34.56%	**171**	243
250	26.98%	–	28	32.19%	**42**	916
500	26.98%	–	34	65.39%	**45**	3,668
1,000	26.98%	–	61	88.65%	**9**	11,734
2,000	29.79%	488	131	93.46%	**7**	22,886
4,000	57.56%	20	375	97.02%	**5**	30,748
8,000	90.58%	7	705	98.71%	**6**	47,312
16,000	98.26%	6	1,217	98.32%	**5**	50,423

TABLE VII: Comparison of CNN and TripletPower, **Same-Device** Profiling Attacks, Hamming Weight Model, **XMEGA**, **Random Delay**, **Unmasked AES**, **3rd byte**, Train: `X1_K0_U_Delay_200k`, Test: `X1_K0_U_Delay_20k`, N: No. of Training Traces, R: No. of Test Traces Key Rank Converging to 0 (– indicates that key rank does not converge to 0).

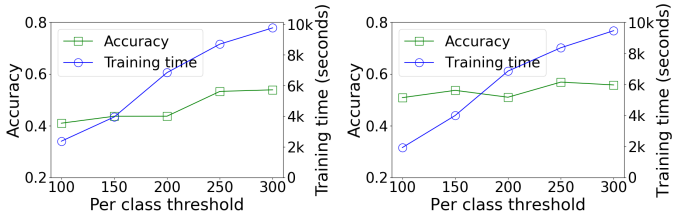
N	CNN		TripletPower	
	ACC	R (traces)	ACC	R (traces)
250	28.18%	–	22.62%	–
500	28.18%	–	25.37%	**3,745**
1,000	24.71%	–	26.55%	**1,427**
2,000	28.08%	–	39.64%	**123**
4,000	28.31%	–	43.27%	**54**
8,000	47.87%	**19**	42.32%	63

AES on XMEGA (Table VI), and unmasked AES on XMEGA with random delay (Table VII). We skip the training time for unmasked AES on XMEGA with random delay as it is the same as the ones without random delays reported in Table IV.

Observation 1: *The results suggest that TripletPower can achieve better attack performance when there are few training traces in same-device profiling attacks. In addition, it can also defeat countermeasures, including masking and random delays. On the other hand, when N is sufficiently large to train a CNN successfully recovering a key, there is no need to utilize TriplePower as it requires much longer training time and achieves lower attack performance.*

Experiment 2. The Impact of Per-Class Threshold for TripletPower. To examine the impact of per-class threshold on the performance of our method, we select the number of training traces as $N = 2,000$ and we choose per-class threshold $U = \{100, 150, 200, 250, 300\}$. Given each N and U, we retrain TripletPower with `X1_K0_U_200k` as training dataset and test with `X1_K0_U_20k` as test dataset.

As shown in Fig. 9a, when we increase per-class threshold U, the training time of TripletPower increases significantly. On the other hand, accuracy increases much slower or remains similar. For instance, when $U = 300$, it takes close to 10,000 seconds to train TripletPower and offers 53.89% accuracy.



(a) XMEGA (b) STM32

Fig. 9: Training time and attack accuracy of TripletPower given $N = 2,000$, unmasked AES, 3rd byte.

However, when $U = 150$, the training time can be reduced to around 4,000 seconds while TripletPower can achieve 43.69% accuracy, which is still sufficient for recovering keys. We also examine unmasked AES on STM32 with `S1_K0_U_200k` as training dataset and `S1_K0_U_20k` as test dataset. The observation is even more obvious in Fig. 9b.

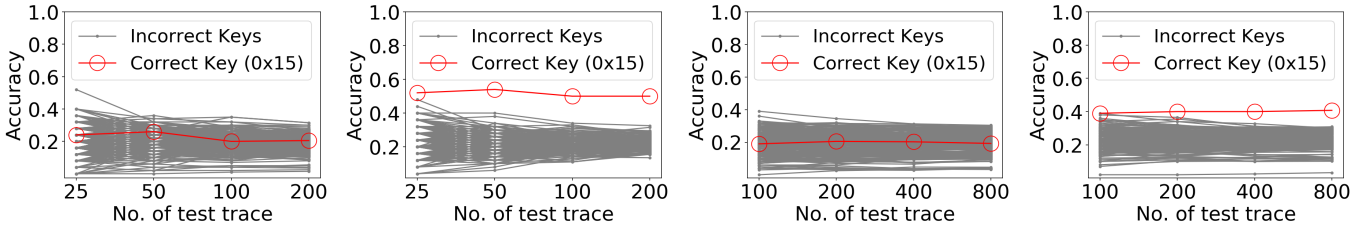
Observation 2: *Given N, selecting a greater value of U can help improving the attack performance of TripletPower but the tradeoff between training time and attack performance should be kept in mind to avoid unnecessarily long training time.*

Experiment 3. Comparison between CNN and TripletPower (Profiling, Cross-Device). We compare the attack performance of CNN and TripletPower in **cross-device profiling side-channel attacks**, where the training and test data are collected from different targets with different keys. We examine unmasked AES on XMEGA and STEM32, respectively.

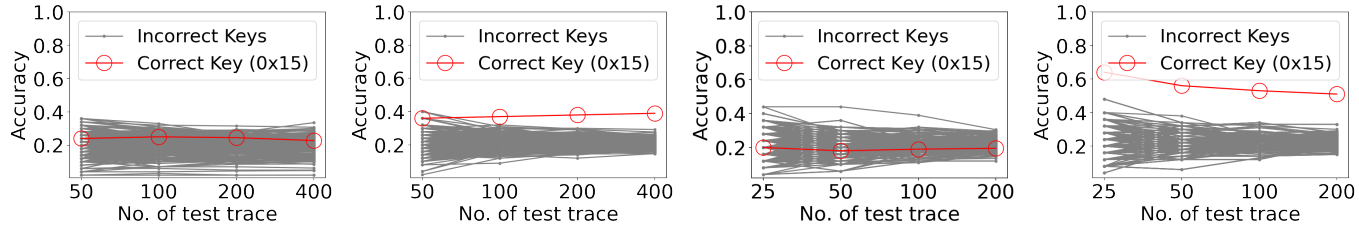
For unmasked AES on XMEGA, we use `X1_K0_U_200k` as the training dataset and `X2_K1_U_20k` as the test dataset. As we use the same dataset for training in Experiment 1, we directly leverage the trained classifiers obtained in Experiment 1 for both CNN and TripletPower. Other details of the experiment remain the same as the ones mentioned in Experiment 1. As described in Table VIII, we have consistent observation, where TriplePower can outperform CNN in terms of the number of traces key rank converging to 0 given $250 \leq N \leq 2,000$.

In addition, we observe that both CNN and TripletPower suffer minor attack performance drops (in both attack accuracy and key rank) when we compare the results between same-device profiling attacks in Table IV and cross-device profiling attacks in Table VIII. For instance, given $N = 500$, our method can recover the key with 978 traces in the same-device setting but requires 1,478 traces in the cross-device setting. Given $N = 4,000$, CNN can recover keys with 30 traces in the same-device setting but needs 43 traces in the cross-device setting. This is expected as previous studies [27]–[32] have shown that discrepancies between training and test data in the cross-device setting can lead to performance drops. We also observe the same from unmasked AES on STM32 (in Table IX) when we use `S1_K0_U_200k` as training dataset and `S2_K2_U_20k` as test dataset.

Observation 3: *The results of this experiment suggest that our method outperforms CNN even in cross-device profiling attacks when there are few training traces.*



(a) No. of training traces $N = 250$, XMEGA, unmasked, 3rd byte (b) No. of training traces $N = 500$, XMEGA, unmasked, 3rd byte (c) No. of training traces $N = 250$, STM32, unmasked, 3rd byte (d) No. of training traces $N = 500$, STM32, unmasked, 3rd byte



(e) No. of training traces $N = 125$, XMEGA, masked, 3rd byte (f) No. of training traces $N = 250$, XMEGA, masked, 3rd byte (g) No. of training traces $N = 1,000$, XMEGA, random delay, 3rd byte (h) No. of training traces $N = 2,000$, XMEGA, random delay, 3rd byte

Fig. 10: Attack results of non-profiling attacks: unmasked AES on XMEGA (a-b); unmasked AES on STM32 (c-d); masked AES on XMEGA (e-f); unmasked AES with random delay on XMEGA (g-h).

TABLE VIII: Comparison of CNN and TripletPower, **Cross-Device** Profiling Attacks, Hamming Weight Model, **XMEGA**, **Unmasked AES**, **3rd byte** Train: `X1_K0_U_200k`, Test: `X2_K1_U_20k`, N: No. of Training Traces, R: No. of Test Traces Key Rank Converging to 0.

N	CNN		TripletPower	
	ACC	R (traces)	ACC	R (traces)
125	21.93%	–	8.65%	–
250	27.52%	–	22.98%	**4,988**
500	27.52%	–	29.54%	**1,478**
1,000	27.52%	–	42.46%	**122**
2,000	27.52%	–	56.80%	**49**
4,000	40.95%	**43**	55.84%	45
8,000	50.42%	**23**	56.13%	39
16,000	63.01%	**13**	53.05%	34

TABLE IX: Comparison of CNN and TripletPower, **Cross-Device** Profiling Attacks, Hamming Weight Model, **STM32F3**, **Unmasked AES**, **3rd byte**, Train: `S1_K0_U_200k`, Test: `S2_K2_U_20k`, N: No. of Training Traces, R: No. of Test Traces Key Rank Converging to 0.

N	CNN		TripletPower	
	ACC	R (traces)	ACC	R (traces)
125	21.61%	–	26.08%	–
250	27.54%	–	26.44%	**695**
500	27.54%	–	40.97%	**89**
1,000	27.54%	–	57.31%	**58**
2,000	27.54%	–	56.03%	**60**
4,000	27.54%	–	49.79%	**39**
8,000	50.19%	**17**	47.56%	41
16,000	64.71%	**12**	51.41%	34

Experiment 4. Attack Performance of TripletPower in Non-Profiling Attacks. We demonstrate that our method is effective in **non-profiling side-channel attacks**. We examine unmasked AES on XMEGA and STM32F3, masked AES on XMEGA, and unmasked AES with random delay on XMEGA.

For unmasked AES on XMEGA, we use `X1_K0_U_20k` as the dataset from a test target in a non-profiling attack. We still attack the 3rd byte. According to the results from our profiling attacks, we select the number of training traces as $N = \{250, 500\}$, and perform on-the-fly labeling to train 256 triplet networks and 256 classifiers. Then, we select the number of test traces as $V = \{25, 50, 100, 200\}$. We leverage the attack accuracy as the Distinguisher. As shown in Fig. 10a and Fig. 10b, when $N = 250$, our method is not able to distinguish the classifier generated by the correct key and classifiers trained by incorrect keys. On the other hand, when we increase the number of training traces to $N = 500$, the classifier trained from the correct key has the highest accuracy and is distinguishable from other classifiers when the number of test traces $V \geq 25$. In other words, our method can recover keys in non-profiling attacks over unmasked AES on XMEGA with as low as 525 traces (500 training + 25 test).

In addition, we use `S1_K0_U_20k`, `X1_K0_M_20k`, and `X1_K0_U_Delay_20k` respectively as the dataset from a test target for unmasked AES on STM32, masked AES on XMEGA, and unmasked AES with random delay on XMEGA in non-profiling attacks. Our results show that our method only needs 700 traces (500 training + 200 test, in Fig. 10d), 350 traces (250 training + 100 test, in Fig. 10f), and 2,025 traces (2,000 training + 25 test, in Fig. 10h) respectively, to distinguish the correct key from incorrect ones.

For the attacks over traces without random delays, as the number of training traces N remains relatively small (e.g., $N \leq 500$), training 256 triplet networks in one non-profiling attack completes around $\frac{1311 \times 256}{3600 \times 24} \approx 3.9$ days (according to the training time reported in Table IV). This is still feasible with

one GPU machine. For the attack over traces with random delays from XMEGA, $N = 2000$ is sufficient to distinguish the correct key. However, training 256 triplet networks given $N = 2000$ with a single GPU machine would be extremely time-consuming (around $\frac{9763\times256}{3600\times24} \approx 29$ days). We optimize the running time by reducing the number of epochs from 100 to 50, reducing Points of Interest to $[2100, 2600]$, and utilizing 4 GPU machines (including 2 local machines with a Titan RTX per machine and 2 Amazon EC2 instances with an NVIDIA Tesla V100 GPU per instance) running in parallel. With those optimizations, we are able to complete one non-profiling attack within 4 days for random delays on XMEGA.

Observation 4: *Our results suggest that our method is effective in non-profiling attacks, where traces are unlabeled.*

V. Related Work

We briefly discuss existing studies that are mostly related to this paper. Comprehensive surveys on machine-learning side-channel attacks can be found in [33], [39], [40].

Machine-Learning Profiling Attacks (Same-Device Setting). Several studies [5], [8], [41], [42] utilized traditional machine learning algorithms, such as SVM or Random Forest, to perform side-channel attacks. Cagli et al. [9] showed that CNNs can defeat jittering (i.e., random delays). Benadjila et al. [10] examined profiling attacks with Multi-Layer Perceptrons and CNNs and released ASCAD dataset, which is one of the largest datasets for side-channel analysis. They also demonstrated that CNNs can defeat masking. Maghrebi et al. [7] demonstrated deep learning can outperform Template Attacks [2]. Van der Valk et al. [43] examined the explainability of neural networks in side-channel attacks. Picek et al. [11] studied data imbalance when performing side-channel attacks with Hamming Weight model. Rijsdijk et al. [44] proposed to leverage reinforcement learning to automatically tune hyperparameters. Perin et al. [45] utilized ensemble learning to improve the generalization of neural networks.

Wang et al. [14] proposed to utilize Conditional Generative Adversarial Networks to produce augmented training traces when there are limited training traces, and then train CNNs with both original traces and augmented traces. However, these augmented traces are generated only in the data domain but do not happen on targets in the real world. In other words, augmented traces do not necessarily follow the same side-channel leakage distribution, which could lead to suboptimial attack performance. This is a general limitation of data augmentation. In addition, there are no results in [14] suggesting the method can be effective in non-profiling attacks.

Machine-Learning Profiling Attacks (Cross-Device Setting). Several studies [27]–[32], [46]–[49] investigated machine-learning side-channel attacks in the cross-device scenario. Das. et al. [32] proposed multi-device training – utilizing power traces collected from multiple devices to train a neural network. In their extended study [31], pre-processing techniques, including Principal Component Analysis and Dynamic Time Warping, were utilized in addition to multi-device training. The authors further proposed an algorithm for optimal selection of multiple training devices in [28].

Zhang et al. [48] proposed to leverage Fast Fourier Transform to pre-process power traces and locate Points of Interest to tackle discrepancies across different types of microcontrollers. Bhasin et al. also [27] proposed to train a neural network with traces from multiple devices. Yu et al. [30] utilized meta-transfer learning to address side-channel attacks across different chip models (STM32F0 v.s. STMF32F1) or different channels (power v.s. EM signals). Rioja et al. [29] investigated how to quantify discrepancies of power traces across devices with Dynamic Time Warping. Cao et al. [17] utilized transfer learning built upon unsupervised domain adaptation to overcome discrepancies in the cross-device scenario. Specifically, they introduced Maximum Mean Discrepancy as a part of the loss function to fine-tune the last few layers of a neural network without the need of labeled traces from a test target.

Deep-Learning Non-Profiling Attacks. Some studies [12], [18], [22], [50] address non-profiling attacks using deep learning, where all the traces are from a single target but unlabeled. The author in [18] proposed *differential deep learning analysis*, which labels a set of power traces with 256 possible keys, trains 256 deep neural networks, and leverages sensitivity analysis over a neural network during the training as the Distinguisher to reveal the correct key.

A recent study [51] shows that leveraging pre-processing (e.g., scaling and t-test) and combining multiple Distinguishers (e.g., Correlation Power Analysis and Mutual Information Analysis) can also reduce the number of unlabeled traces in non-profiling attacks. For instance, the proposed approach can recover keys of unmasked AES-128 on an 8-bit microcontroller within 50 traces. We believe that applying pre-processing in [51] may further improve the performance of our method. We will leave it as future work.

VI. Conclusion

We propose a new deep-learning side-channel attack, which requires only hundreds of training traces to recover AES encryption keys. We conduct comprehensive experiments and demonstrate the advantage of our method over CNNs in same-device profiling attacks, cross-device profiling attacks, and non-profiling attacks. We also demonstrate this proposed method can defeat masking and random delays.

Acknowledgements

The authors thank the anonymous reviewers for their comments and suggestions. The authors also thank the anonymous reviewers from WiSec'22 and RAID'22 for pointing out the limitations in the previous versions of this study. This work was partially supported by National Science Foundation (CNS-1947913, CNS-2150086), and NSF IUCRC Center for Hardware and Embedded System Security and Trust (CHEST) under Grant 1018660.

REFERENCES

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

[2] S. Chari, J. R. Rao, and P. Rohatgi, "Template Attacks," in *Proc. of Cryptographic Hardware and Embeeded Systems (CHES 2002)*, 2022.

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

[4] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual Information Analysis," *Cryptographic Hardware and Embedded Systems (CHES 2008)*, 2008.

[5] G. Hospodar, B. Gierlichs, E. D. Mulder, I. Verbauwhede, and J. Vandewalle, "Machine Learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, no. 4, pp. 293–302, 2011.

[6] L. Lerman, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked AES," *Journal of Cryptographic Engineering*, 2015.

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

[8] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens, "Side-Channel Analysis and Machine Learning: A Practical Perspective," in *Proc. of 2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

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

[10] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database," *Journal of Cryptographic Engineering*, vol. 10, no. 2, 2020.

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

[12] K. Ramezanpour, P. Ampadu, and W. Diehl, "SCAUL: Power Side-Channel Analysis with Unsupervised Learning," *IEEE Transactions on Computers*, 2020.

[13] L. Wu, G. Perin, and S. Picek, "I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-Channel Analysis," https://eprint.iacr.org/2020/1293.

[14] P. Wang, P. Chen, Z. Luo, G. Dong, M. Zheng, N. Yu, and H. Hu, "Enhancing the Performance of Practical Profiling Side-Chanel Attacks Using Conditional Generative Adversarial Networks," https://arxiv.org/abs/2007.05285.

[15] G. Zaid, L. Bossuet, F. Dassanace, A. Habrard, and A. Venelli, "Ranking Loss: Maximizing the Success Rate in Deep Learning Side-Channel Analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 1, pp. 25–55, 2021.

[16] X. X. L. Zhang, J. Fan, Z. Wang, and S. Wang, "Multilabel Deep Learning-Based Side-Channel Attack," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[17] P. Cao, C. Zhang, X. Lu, and D. Gu, "Cross-Device Profiled Side-Channel Attack with Unsupervised Domain Adaptation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 4, pp. 27 – 56, 2021.

[18] B. Timon, "Non-Profiled Deep Learning-based Side-Channel Attacks with Sensitivity Analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 107–131, 2019.

[19] M. Kerkhof, L. Wu, G. Perin, and S. Picek, "No (Good) Loss no Gain: Systematic Evaluation of Loss functions in Deep Learning-based Side-channel Analysis," https://eprint.iacr.org/2021/1091.

[20] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[21] [Online]. Available: https://www.newae.com/chipwhisperer

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

[23] F. Standaert, B. Gierlichs, and I. Verbauwhede, "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*, 2008.

[24] [Online]. Available: https://github.com/UCdasec/TripletPower

[25] L. Wu, G. Perin, and S. Picek, "The Best of Two Worlds: Deep Learning-assisted Template Attack," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 3, pp. 413–437, 2022.

[26] E. Hoffer and N. Ailon, "Deep Metric Learning Using Triplet Network," in *International Workshop on Similarity-Based Pattern Recognition*, 2015.

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

[28] J. Danial, D. Das, A. Golder, S. Ghosh, A. Raychowdhury, and S. Sen, "EM-X-DL: Efficient Cross-device Deep Learning Side-channel Attack with Noisy EM Signatures," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 18, no. 1, pp. 1–17, 2022.

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

[30] H. Yu, H. Shan, M. Panoff, and Y. Jin, "Cross-Device Profiled Side-Channel Attacks using Meta-Transfer Learning," in *Proc. of the 58th ACM/IEEE Design Automation Conference (DAC'21)*, 2021.

[31] A. Golder, D. Das, J. Danial, S. Ghosh, S. Sen, and A. Raychowdhury, "Practical Approaches Towards Deep-Learning Based Cross-Device Power Side Channel Attack," *IEEE. Trans. on Very Large-Scale Integration (VLSI) Systems*, vol. 27, no. 12, 2019.

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

[33] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina, "Sok: Deep Learning-based Physical Side-channel Analysis," https://eprint.iacr.org/2021/1092.

[34] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning," in *Proc. of ACM CCS'19*, 2019.

[35] [Online]. Available: https://github.com/newaetech/chipwhisperer

[36] [Online]. Available: https://github.com/ANSSI-FR/secAES-ATmega8515

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

[38] V. Immler, R. Specht, and F. Unterstein, "Your Rails Cannot Hide From Localized EM: How Dual-Rail Logic Fails on FPGAs," https://eprint.iacr.org/2017/608.pdf.

[39] B. Hettwer, S. Gehrer, and T. Guneysu, "Applications of machine learning techniques in side-channel attacks: a survey," *Journal of Cryptographic Engineering*, vol. 10, pp. 135–162, 2020.

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

[41] T. Bartkewitz and K. Lemke-Rust, "Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines," in *Proc. of International Conference on Smart Card Research and Advanced Applications (CARDIS 2012)*, 2012.

[42] L. Lerman, G. Bontempi, and O. Markowitch, "Power analysis attack: an approach based on machine learning," *International Journal of Applied Cryptography*, vol. 3, no. 2, pp. 97–115, 2014.

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

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

[45] G. Perin, L. Chmielewski, and S. Picek, "Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020.

[46] M. Elaabid and S. Guilley, "Protability of templates," *Journal of Cryptographic Engineering*, vol. 2, pp. 63–74, 2012.

[47] H. Wang, M. Brisfors, S. Forsmark, and E. Dubrova, "How Diversity Affects Deep-Learning Side-Channel Attacks," in *2019 IEEE Nordic Cir-*

cuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC), 2019.

[48] F. Zhang, B. Shao, G. Xu, B. Yang, Z. Yang, Z. Qin, and K. Ren, "From Homogeneous to Heterogeneous: Leveraging Deep Learning based Power Anlysis across Devices," in *Proc. of 57th ACM/IEEE Design Automation Conference (DAC'20)*, 2020.

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

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

[51] S. Seckiner and S. Kose, "Preprocessing of the Physical Leakage Information to Combine Side-Channel Distinguishers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 12, 2021.