USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Breaking the Blindfold: Deep Learning-based Blind Side-channel Analysis

Azade Rezaeezade, *Delft University of Technology and Digital Security Group, Radboud University;* Trevor Yap, Dirmanto Jap, and Shivam Bhasin, *Temasek Laboratories and National integrated Centre For Evaluation, Nanyang Technological University;* Stjepan Picek, *Digital Security Group, Radboud University and University of Zagreb Faculty of Electrical Engineering and Computing*

https://www.usenix.org/conference/usenixsecurity25/presentation/rezaeezade

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

Breaking the Blindfold: Deep Learning-based Blind Side-channel Analysis

Azade Rezaeezade[1, 4, *], Trevor Yap[2,3,*], Dirmanto Jap[2,3], Shivam Bhasin[2,3], *Stjepan Picek*[4, 5]
[1]*Delft University of Technology, The Netherlands.*
[2]*Temasek Laboratories, Nanyang Technological University, Singapore.*
[3]*National integrated Centre For Evaluation, Nanyang Technological University, Singapore.*
[4]*Digital Security Group, Radboud University, The Netherlands.*
[5]*University of Zagreb Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia.*
Email: {azade.rezaeezade, stjepan.picek}@ru.nl, {trevor.yap, djap, sbhasin}@ntu.edu.sg

Abstract

Physical side-channel analysis (SCA) operates on the foundational assumption of access to known plaintext or ciphertext. However, this assumption can be easily invalidated in various scenarios, ranging from common encryption modes like Offset CodeBook (OCB) to complex hardware implementations, where such data may be inaccessible. Blind SCA addresses this challenge by operating without the knowledge of plaintext or ciphertext. Unfortunately, prior such approaches have shown limited success in practical settings.

This paper introduces the Deep Learning-based Blind Side-channel Analysis (DL-BSCA) framework, leveraging deep neural networks to recover secret keys in blind SCA settings. In addition, we propose a novel labeling method, Multi-point Cluster-based (MC) labeling, accounting for dependencies between leakage variables by exploiting multiple sample points for each variable, improving the accuracy of trace labeling. We validate our approach across four datasets, including symmetric key algorithms (AES and ASCON) and a post-quantum cryptography algorithm, Kyber, with platforms ranging from high-leakage 8-bit AVR XMEGA to noisy 32-bit ARM STM32F4. Notably, previous methods failed to recover the key on the same datasets. We demonstrate the first successful blind SCA on a desynchronization countermeasure enabled by DL-BSCA and MC labeling. All experiments are validated with real-world SCA measurements, highlighting the practicality and effectiveness of our approach.

1 Introduction

While standard cryptographic algorithms are considered theoretically secure, they remain vulnerable to physical attacks, such as side-channel attacks (SCA) [23]. In physical SCA, an adversary observing physical leakages, such as power or electromagnetic emanations, can exploit those to learn sensitive information like the secret key. SCA typically involves two scenarios: non-profiled and profiled [46]. Profiled attacks, such as template attacks [5, 46], stochastic attacks [11, 43], and more recently, deep learning-based methods [26, 31], have demonstrated remarkable effectiveness when a clone device is available. There, a training dataset is obtained from a clone device to build a leakage profile or model. Traces from the target device (with an unknown key) but known plaintext or ciphertext are then compared to this profiled model to extract information about the secret key. In contrast, non-profiled attacks like Correlation Power Analysis (CPA) [19] directly compute statistical dependencies between SCA traces and an intermediate value generated using known plaintext/ciphertext to recover the key without requiring a profiling phase.

Physical SCAs are practical and demonstrated in real-world settings [40]. Furthermore, deep learning-based SCAs are becoming increasingly relevant and powerful [41]. Early works on deep learning-based SCA focused on both profiled [25, 31, 36] and non-profiled scenarios [6, 42], while later studies extended this to more challenging settings, such as weakly supervised attacks [49], leakage model-flexible attacks [50], and collision attacks [45, 51]. Moreover, the Federal Office for Information Security (BSI) in Germany recently published a document describing the requirements for machine learning-based SCA [9].[1]

A critical assumption underlying both profiled and non-profiled attacks is that attackers have access to known data, such as plaintext or ciphertext. This dependency on known input or output is foundational to most SCA techniques, but is not always valid in real-world settings where access to such data may be restricted. For example, common encryption modes like Offset CodeBook (OCB) or XTS [1] limit access to the input/output of the encryption block. Another example arises during key derivation in the EMV payment scheme, where the ciphertext cannot be recovered and only a small portion of the plaintext is known. To fully recover the session key in this context, one must use the blind setting, without access to either full plaintext or ciphertext [35].These

[*]These authors contributed equally to this work

[1]This document is a part of the AIS 46 document (information regarding the evaluation of cryptographic algorithms and additional information for the evaluation of random number generators).

Ref.	Board	Target
[53]	8-bit AVR ATMega	AES-128
[13]	8-bit AVR ATMega	AES-128
[35]	32-bit STM32F3	Kyber (profiled)
This work	8-bit AVR XMEGA	AES-128
This work	8-bit AVR XMEGA	Protected AES-128 (desynchronized)
This work	32-bit STM32F3	Kyber
This work	32-bit STM32F4	ASCON

Table 1: Blind SCA works with practical demonstrations.

constraints present a unique challenge for adversaries, leading to the emergence of techniques targeting implementations where input/output data is unknown—a scenario referred to as **blind SCA [13, 53]**.

In blind SCA, attackers rely exclusively on side-channel traces, captured as the device processes confidential data tied to the secret key, without access to plaintext or ciphertext. This significantly complicates the attack process, making blind SCA a compelling focus for advancing the field of side-channel research. Despite some advancements in the last few years, most blind SCA research remains confined to simulated environments. Practical demonstrations are typically limited to platforms such as the 8-bit AVR microcontroller [13, 53] targeting AES-128 with no side-channel countermeasures. These platforms are known for their exceptionally high signal-to-noise ratio (SNR), which makes them *less* representative of real-world conditions. Attempts to extend blind SCA to countermeasures like masking have been limited to simulated environments, as in [13]. While Ravi et al. [35] reported blind SCA targeting Kyber on 32-bit STM32F3, it assumes a strong adversary with access to a clone device to train classifiers related to precise knowledge of secret inputs and trace leakage samples. Furthermore, all practical implementations tested to date have not been hardened against SCA.

Contributions. This paper addresses key limitations in existing blind SCA techniques by employing a deep learning-based approach. We demonstrate blind SCA across a broad range of platforms, cryptographic algorithms, and desynchronization countermeasures, validating all attacks with real-world measurements and significantly advancing the practical applicability of blind SCA. This is compared with prior works in Table 1. More precisely, blind SCA's main challenge lies in inferring labels for each measurement, as it does not rely on known plaintext or ciphertext. We model this as a *deep learning problem with noisy labels* and show that deep neural networks (DNNs) effectively identify the underlying distribution of these measurements, outperforming traditional techniques in blind SCA performance.

To emphasize the real-world applicability, we consider various cryptographic algorithms: Advanced Encryption Standard (AES), a widely used encryption standard; ASCON, the CAESAR competition winner and NIST's choice for lightweight cryptography; and Kyber, the post-quantum key encapsulation mechanism selected by NIST. Moreover, we deploy the implementations on various platforms, including an ARM Cortex-M4 chip. We emphasize that ARM Cortex-M represents the highest market share among all ARM products, leading with a market share of USD 6.0 billion in 2023, projected to grow to USD 10.5 billion by 2032, reflecting its dominance in embedded applications and low-power devices [16]. Finally, ARM Cortex-M4 is the preferred platform for the NIST post-quantum cryptography competition and the associated public pqm4 library [22].

The main contributions of this work are as follows:

1. We consider blind SCA as a deep learning problem with noisy labels, formalized under the proposed **Deep Learning-based Blind Side-channel Attacks (DL-BSCA) framework.**
2. We introduce an efficient **unsupervised labeling scheme** called Multi-point Cluster-based (MC) labeling for identifying labels from side-channel traces. This is the **first multivariate labeling technique** proposed within the context of blind SCA.
3. We validate our approach on three devices and four datasets using real measurements. Unlike prior work, which focused on low-noise simulations or 8-bit AVR ATMEGA platforms, **we demonstrate that blind SCA is practical on various platforms**. Notably, previous methods failed to recover the key on the same datasets.
4. We present the **first successful blind SCA on a desynchronization countermeasure**, exploiting the MC labeling technique to utilize multiple leakage samples, also known as points of interest (PoIs).

The most up-to-date source code is available here.

2 Background

2.1 Side-channel Attacks

Side-channel attacks (SCAs) exploit unintended physical leakages from cryptographic devices, such as power consumption, electromagnetic radiation, or timing variations, to extract sensitive information like secret keys. Unlike traditional cryptanalysis, which relies on weaknesses in cryptographic algorithms, SCAs target the implementation of these algorithms. SCAs are classified into profiled and non-profiled attacks [46]. Profiled SCAs require the attacker to have access to a clone device to model the side-channel leakage behavior. The attacker collects side-channel traces from this device while processing known plaintext or ciphertext, building a profile of how the device behaves under specific conditions. This profile is then used to correlate side-channel measurements from the target device to the secret key or other sensitive data. Techniques ranging from template attacks [46] to deep learning-based methods [26] are often employed in profiled SCAs. Non-profiled SCAs, on the other hand, exploit statisti-

cal relationships between the side-channel measurements and the cryptographic operations related to known plaintext or ciphertext without needing a model of the system. As evident, both profiled and non-profiled SCAs require access to input plaintext or output ciphertext.

2.2 Blind Side-channel Attacks

In a blind SCA scenario, the attackers lack access to any known data (e.g., plaintext/ciphertext) and can only use the target device's side-channel measurements to deduce the key. This poses a significantly harder task than usual SCAs. We focus on the work by Linge et al. [53] and Claiver et al. [13] as we consider the same scenario. Both [53] and [13] consider the blind SCA framework. This framework comprises three phases, which are discussed below. We illustrate the attack flow of the blind SCA framework in Figure 1.

Phase 1: Pre-computing theoretical joint distribution. Suppose m is a public variable (plaintext or ciphertext) while y is the sensitive intermediate variable (e.g., $y = Sbox(m \oplus k^*)$ where $Sbox$ is a Substitution Box, and k^* is the secret key). Assuming that the leakage is following the Hamming weight (HW) model, the key observation is that the joint distributions $(HW(m), HW(y))$ of the public variable and the sensitive intermediate variables are distinct for every secret key k^*. Therefore, in this phase, the theoretical joint distribution, denoted as (h_m^*, h_y^*), is computed for all key candidates. This is done by iterating through all the keys and m to count the number of times $(HW(m), HW(y))$ tuples appear. Then, we normalize the frequencies to obtain a probability distribution, $Pr((h_m^*, h_y^*)|k)$, for given hypothesis k. This can be precomputed beforehand since it is independent of the measurements collected from the device. For a clearer understanding of the theoretical joint distribution calculation for different targeted algorithms in this work, please refer to Appendix A.

Phase 2: Labeling traces to obtain empirical distribution. The goal of this phase is to acquire the empirical distribution. We further divide it into two substeps for clarity.

1. **PoIs Selection:** To attain the empirical distribution, one first needs to identify a suitable PoI that represents $HW(m)$ and another suitable PoI that represents $HW(y)$. Both [53] and [13] use one PoI for each targeted variable, i.e., m and y. Both works assume that the adversary can precisely locate the PoIs related to targeted intermediate variables. Consequently, finding precise PoIs is crucial as these points are used to estimate the actual Hamming weights (i.e., $HW(m)$ and $HW(y)$) and obtain the empirical joint distribution (denoted as (h_m, h_y)). We follow the assumption of prior works that the adversary can precisely locate the PoIs.[2] We achieve this by assuming that the adversary uses correlation between the measurements and the Hamming weight of the targeted variable to locate the PoIs. Accordingly, we select sample points that exhibit the highest correlation. Note that this is the only part where the public variable is required, and it is a non-blind setting. It is worth mentioning that even with this assumption in selecting PoIs, classical techniques fail to recover the secret key (see Tables 2 to 5). For interested readers regarding classical PoIs selection in a non-blind setting, see Appendix E.

2. **Labeling Traces:** Next, one needs to obtain the empirical distribution using the selected PoIs. This is achieved by labeling the traces. Two labeling methods are proposed in [53] and [13]. We recall both labeling methods: Slicing labeling [53] and Variance Analysis (VA) labeling [13].
   - **Slicing labeling [53]:** Linge et al. [53] decided on the Hamming weight value, (h_m, h_y), based on the amplitude of the measurements at selected PoI, which they call slicing, thus *Slicing* labeling. The underlying assumption is that if the amplitude of the consumed power at the considered PoI is small, then the Hamming weight of the corresponding intermediate value is small.
     Let N be the number of traces to be labeled and let l_v denote the amplitude of the consumed power at the selected PoI that reflects $HW(v)$ where $v = m$ or $v = y$. They first sort the traces according to their amplitude values, l_v, in ascending order. Then, it is reasonable to assign the smallest values to the Hamming weight $h_v = 0$, then the next smallest values to $h_v = 1$, and so on. If the targeted variable has $\mathcal{B}$ bits, its corresponding Hamming weight can take values between 0 (when all bits are 0) and $\mathcal{B}$ (when all bits are 1). To assign the correct number of traces to each Hamming weight, they fragmented the traces based on the distribution of the Hamming weight. The proportion of the different Hamming weight from 0 to $\mathcal{B}$ can be calculated using $\binom{\mathcal{B}}{h_v}$. With the assumption of a uniform distribution for cryptographic data, among the N traces, theoretically, $(\frac{N}{2^{\mathcal{B}}}\binom{\mathcal{B}}{h_v})$ of them should have the Hamming weight equal to h_v, $0 \leq h_v \leq \mathcal{B}$. The technique is applied on both l_m and l_y separately, providing the labels (h_m, h_y) for each trace.
   - **VA labeling [13]:** Similar to [53], Clavier and Reynaud [13] also assume the knowledge of two suitable PoIs for $HW(m)$ and $HW(y)$. They proposed

[2] [13] used the highest peak from the traces' variances along with reasonable assumptions about the implementation to locate the PoIs. However, based on our initial experiments, extracting the key was not successful using this method.

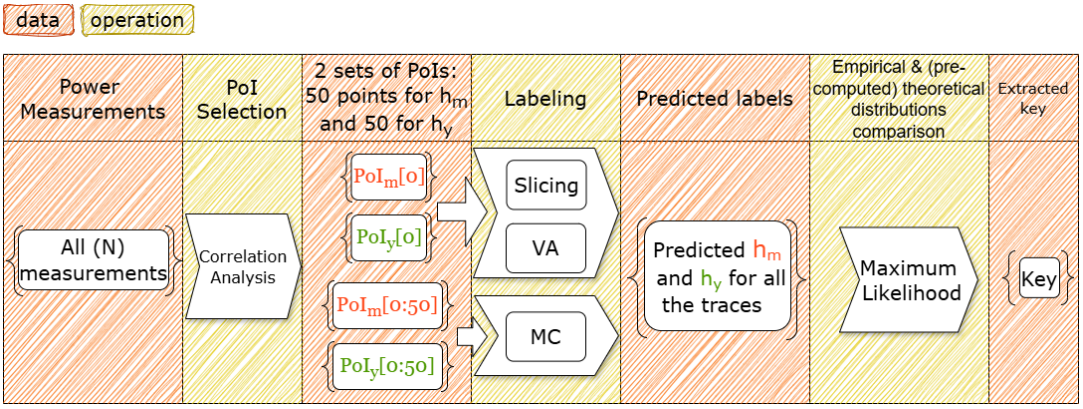


Figure 1: Blind Side-channel Analysis Framework.

two linear regression methodologies to label the traces instead of using slicing to label their traces. However, the first method requires knowledge of the intermediate byte values, which is not practical because of the plaintext/ciphertext inaccessibility. Thus, we only recall the second linear regression known as *Variance Analysis* (VA) labeling. First, the authors assume the noisy leakage of the sample point as $l_v = \alpha_v HW(v) + \beta_v + \varepsilon_v$ where α_v, β_v are constants to be determined, ε_v is the Gaussian noise, and v is the sensitive variable to be considered (i.e., $v = m$ or $v = y$). Hence, the variance can be written as $Var(l_v) = \alpha^2 Var(HW(v)) + Var(\varepsilon_v)$. $HW(v)$ can be considered as binomial distribution $B(\mathcal{B}, p)$, with $\mathcal{B}$ being the number of bits and p being the probability of having value 1. Hence, the variance can be calculated as $\mathcal{B}p(1-p)$. Since the distribution of bits 0 and 1 is uniform, $p = 0.5$, and as such, $Var(HW(v)) = 0.25\mathcal{B}$. For an 8-bit implementation, $Var(HW(v)) = 2$. Thus, we can obtain α_v and β_v as follows:

$$\begin{aligned} \alpha_v &= \sqrt{(Var(l_v) - Var(\varepsilon_v))/(0.25\mathcal{B})}, \\ \beta_v &= \mathbb{E}(l_v) - 4\alpha_v. \end{aligned} \tag{1}$$

Then, the estimated Hamming weight will be $h_v = \frac{l_v - \beta_v}{\alpha_v}$, from which the labels for the traces (h_m, h_y) are obtained.

Notably, these labeling methods are not designed to handle more than one PoI for each variable.

Phase 3: Comparing the empirical distribution with the theoretical joint distribution. Various methodologies were proposed to compare an empirical distribution with its theoretical joint distribution. The authors in [53] considered different metric-based comparisons like χ^2 distance, inner product, or harmonic mean. Le Bouder proposed to use the maximum likelihood criterion to compare instead [24]. The authors in [13] compared these techniques and found that the maximum likelihood criterion obtained better results. Thus, we recall the maximum likelihood criterion next.

Let (h_m^*, h_y^*) denote the true Hamming weight tuple of (m, y), while the recovered Hamming weight tuple using labeling technique is $(h_m, h_y) = (h_m^* + \varepsilon_m, h_y^* + \varepsilon_y)$ where ε_m and ε_y are Gaussian noise with standard deviations σ_m and σ_y, respectively. Based on the Bayes formula, the probability of the key k given a single observation (h_m, h_y) is given as

$$Pr(k|(h_m, h_y)) = \frac{Pr((h_m, h_y)|k) \cdot Pr(k)}{Pr((h_m, h_y))}. \tag{2}$$

The denominator $Pr((h_m, h_y))$ is a normalization term independent of the key, so we can ignore it. Moreover, $Pr(k)$ is assumed to be uniformly distributed. The probability of the key given the set of observation $((h_m, h_y)_i)_{i=1,\ldots N}$ is denoted as

$$\begin{aligned} &Pr(k|((h_m, h_y)_i)_{i=1,\ldots,N}) \\ &\quad = Pr((h_m, h_y)_N|k) \cdot Pr(k|((h_m, h_y)_i)_{i=1,\ldots,N-1}). \end{aligned} \tag{3}$$

Then, by the law of total probability, we can rewrite

$$\begin{aligned} &Pr((h_m, h_y)_i|k) \\ &\quad = \sum_{h_m^*, h_y^*} Pr((h_m, h_y)_i|(h_m^*, h_y^*)) \cdot Pr((h_m^*, h_y^*)|k) \end{aligned} \tag{4}$$

for each i. For the second term, we use the theoretical joint distribution from before, while for the first term, we can rewrite it as the probability of its noise, which we assume follows a Gaussian distribution:

$$\begin{aligned} &Pr((h_m, h_y)_i|(h_m^*, h_y^*)) \\ &\quad = Pr(\varepsilon_{m,i} = h_m^* - h_{m,i}) \cdot Pr(\varepsilon_{y,i} = h_y^* - h_{y,i}) \\ &\quad = \left(\frac{1}{\sigma_m\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_{m,i} - h_m^*}{\sigma_m}\right)^2}\right) \cdot \left(\frac{1}{\sigma_y\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_{y,i} - h_y^*}{\sigma_y}\right)^2}\right). \end{aligned} \tag{5}$$

The values of $(h_m, h_y)_i$ are obtained by applying the labeling techniques, and are subsequently used as empirical labels for i^{th} trace in the computation of Eq. (5).

Attack Metrics: Using these equations, we can consider $Pr(k|((h_m,h_y)_i)_{i=1,\ldots,N})$ as the score. The key candidate with the highest probability across N traces is considered the most likely key. Formally, we sort the score in descending values and classify them into a key guessing vector $\boldsymbol{G} = (G_0,\ldots,G_{|\mathcal{K}|-1})$ where $\mathcal{K}$ is the key space. Therefore, the key candidate corresponding to G_0 is the most likely key candidate while the $G_{|\mathcal{K}|-1}$ is the least likely key candidate. The index in $\boldsymbol{G}$ is the key rank. In a typical attack, the average key ranks over multiple experiments are computed, and the top o key candidates are enumerated to obtain the correct key. Therefore, we denote the guessing entropy GE as the average rank of the correct key over multiple experiments (we average the guessing entropy over 100 experiments in Section 4 for the reported results). When $GE \leq o$ for a fixed number of traces, the secret key is ranked in the top o key candidates on average. GE is a standard metric used in various SCA standardizations like ISO/IEC17825:2024.

When the attack guesses the secret key as the best key candidate over all the experiments, i.e., $GE = 0$, $NTGE$ denotes the smallest number of traces required to achieve $GE = 0$.

In the following, we focus on the attack's ability to recover the secret key when $GE \leq 10$. Specifically, having $GE \leq 10$ indicates that the attack consistently identifies the correct key within the top 10 candidates.[3] Throughout the three phases of blind SCA, other than the PoIs selection, the value of the public variable m is never used.

2.3 Gaussian Mixture Model

Here, we recall the well-known clustering technique called Gaussian Mixture Model (GMM) [8], which will be used in our proposed labeling technique. The GMM clustering technique is a probabilistic method that assumes the data can be represented as a combination of several Gaussian distributions. This assumption is often valid for side-channel measurements because the noise in these measurements can typically be approximated by a Gaussian distribution [46].

To apply the GMM clustering method, we must first define the number of clusters the model should generate. In our case, this is straightforward because we know the possible Hamming weights that the values of m and y can take for various cryptographic algorithms. The GMM clustering begins by initializing the parameters for each cluster, which include the mean, covariance, and mixing coefficients.[4] Once initialized, the model uses the Expectation-Maximization (EM) algorithm to refine these parameters iteratively.

The EM algorithm operates in two main steps.

1. **Expectation Step:** Compute the probability that each data point belongs to each Gaussian distribution, given the current estimates of the model's parameters.
2. **Maximization Step:** Update the parameters (mean, covariance, and mixing coefficients) to maximize the expected likelihood of the data, given these probability estimates.

We repeat both steps until the model converges or until a specified number of iterations is reached, providing a set of parameters for the GMM clustering method to generate clusters. We will use this approach in our proposed labeling method described in Section 3.3.

3 Deep Learning-based Blind Side-channel Attack

3.1 Threat Model

We follow the same threat model as blind SCA presented in both [53] and [13]. The adversary has no prior knowledge about the data being processed (like plaintexts or ciphertexts) and must solely rely on SCA measurements to infer the secret key. This presents a considerably more challenging scenario compared to traditional SCAs, where each SCA trace has a known plaintext or ciphertext associated with each trace. Additionally, similar to the blind SCA framework, we assume the adversary has knowledge of the most informative PoIs. The relevant PoIs are obtained by analyzing the correlation between the measurements and the actual Hamming weight of the target variables.

3.2 Methodology

Within the blind SCA framework, the main problem is to label the traces (Phase 2 of Figure 1). If the traces are labeled correctly, the attack will be successful. But if there are too many mislabeled traces, the attack will fail. Classical blind SCA approaches typically rely on the assumption that larger Hamming weights lead to higher power consumption and use this to assign labels. In practice, though, such relationships are often obscured in raw SCA traces. This is reflected by the poor performance of classical methods (shown later in Tables 2–5). By contrast, DNNs have been shown to generalize unseen data, even with mislabeled data [2, 29, 44]. To harness the capabilities of DNNs in blind SCA, we propose the Deep Learning-based Blind Side-channel Analysis (DL-BSCA) framework. Our framework consists of the same three phases outlined in Section 2.2. Phases 1 and 3 remain unchanged; the modification has been introduced in Phase 2. We depict the attack flow of DL-BSCA in Figure 2. To label traces and obtain the empirical distribution, we divide Phase 2 into three substeps:

(a) PoI Selection.

[3] Note that while $GE \leq 10$ renders the attack practical for recovering a targeted portion of the key, it still necessitates brute-force or key enumeration techniques for complete key recovery.

[4] Mixing coefficients, denoted by π_l, represent the contribution of each Gaussian distribution l to the overall mixture: $p(\mathbf{x}) = \sum_{k=l}^{L} \pi_l \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)$. Since we work with more than one PoI, we use multivariate Gaussian distributions.

(b) Labeling a subset of the traces and training the DNN with it.

(c) Predict the labels of remaining traces using DNN to obtain empirical distribution.

Unlike the blind SCA framework, which consists of one set of traces, DL-BSCA splits the traces into two sets of traces. One set of traces is labeled using a certain labeling technique to train the DNN, which we denote as *training traces*, while the other set of traces is passed through the trained DNN to obtain the empirical distribution, which we call *attack traces*. We denote N_t and N_a as the number of training traces and attack traces used.

Since the labeling technique may result in many mislabeled traces, the DL-BSCA can be viewed as training a DNN with noisy labels. The issue of "noisy labels" is well-recognized in the deep learning community [20, 44, 57].[5] An interesting observation for DNNs is that they tend to learn simpler patterns first and memorize instances that do not show the straightforward relation between input features and labels later in training [2]. This behavior implies that the network can capture the core data patterns from correctly labeled data early in training, even with noisy labels.

The problem of training DNNs with noisy labels is also not completely new for SCA. In [29], Perin et al. proposed an iterative framework to improve the percentage of correct labels using accurately labeled traces slightly better than a random guess within the context of a horizontal attack on public datasets.[6] However, in blind SCA, determining the correct measurement labels is highly challenging and often impractical.[7] Therefore, the labeling technique we use is critical, as it determines the number of mislabeled traces within the dataset used for training the DNN. In this context, both Slicing labeling and VA labeling, which were previously proposed, can be applied. In the following, we also propose a new labeling method called MC labeling.

3.3 Multi-point Cluster-based Labeling

As mentioned above, the labeling technique is a key aspect of the framework. Unlike Slicing labeling and VA labeling, which consider only one PoI for $HW(m)$ and $HW(y)$ each, we use a set of PoIs to represent $HW(m)$ and $HW(y)$ better when labeling. By selecting multiple PoIs, we can better account for noise introduced by the environment and device, as each PoI reflects the same value for the target variable. We take 50 PoIs for $HW(m)$ and another 50 PoIs for $HW(y)$. These points are selected as the top 50 sample points with the highest correlation to $HW(m)$ and $HW(y)$, respectively. Our experiments show that using 50 PoIs for each variable is suitable as it provides enough information for accurate clustering despite the noise (see Section 5). At the same time, 50 PoIs is a manageable number, allowing us to consolidate all the information from the points and assign a unique label to each trace. We refer to the selected 50 PoIs for $HW(m)$ as PoI_m and the other selected 50 PoIs for $HW(y)$ as PoI_y. We truncate the traces into traces with 100 sample points in total (i.e., $|PoI_m| + |PoI_y| = 100$, where $|PoI_\ell|$ denotes the number of sample points in the set PoI_ℓ for $\ell = m, y$). The MC labeling is executed in two stages: **Produce Clusters** and **Provide Labels**.

Produce Clusters. In this step, the truncated traces with 100 sample points are given to a clustering technique. Our technique clusters the traces considering all the target variables at once. The number of clusters is specified with the number of bits for a target variable, $\mathcal{B}$, and the number of variables considered, n_b. Therefore, the number of clusters equals $(\mathcal{B}+1)^{n_b}$. In other words, instead of clustering the traces based on the sample points from PoI_m and PoI_y separately, we cluster the traces considering both sets of sample points from PoI_m and PoI_y together and cluster the traces with the same $HW(y)$ and $HW(m)$ into one cluster. Figure 3 illustrates the cluster generation step in the MC labeling process.

For example, in AES, we group traces into 81 clusters and consider both $HW(y)$ and $HW(m)$ simultaneously. This is to capture the interaction between m and y and provide a more comprehensive view of the data, which is particularly useful when the variables are interdependent (in the AES example, $y = Sbox(m \oplus k)$ is a function of m and the secret key). If we cluster the traces based on the sample points from PoI_m and PoI_y separately, we will lose information on the relation between m and y. Therefore, we consider both PoI_m and PoI_y together (a total of 100 sample points) when applying the clustering technique.

In our work, we use GMM as the clustering technique since we found it the most successful compared to other clustering techniques. We use the *scikit-learn* library when applying the GMM for clustering. Then, we use the *predict* function to obtain the clusters of traces.

Provide Labels. The clustering processes group traces based on similarities, but the clusters still lack labels. Thus, we must map each cluster $\mathcal{C}$ to an associated label, $(Y_{\mathcal{C}}^{(m)}, Y_{\mathcal{C}}^{(y)})$. $(Y_{\mathcal{C}}^{(m)}, Y_{\mathcal{C}}^{(y)})$ will represent the label (h_m, h_y).

We provide the label of each cluster based on the steps:

1. We first compute the 'center' of each cluster. Let $CT_{\mathcal{C}} \in \mathbb{R}^{|PoI_m|+|PoI_y|}$ be the center of cluster $\mathcal{C}$. $CT_{\mathcal{C}}$ can be attained by averaging each PoI of all the traces within the

[5] Noisy labels can originate from various sources, such as the complexity of determining accurate labels, non-expert labeling, or even adversarial manipulation.

[6] We conducted various experiments using the iterative framework proposed by Perin et al. [29] within the context of DL-BSCA. However, it yielded suboptimal results, likely due to the significantly higher number of mislabeled traces compared to the setup in [29].

[7] The accurate labeling when using Slicing labeling and VA labeling on CW datasets is only around 2%, while the setting in [29] has a dataset with 52% accuracy.

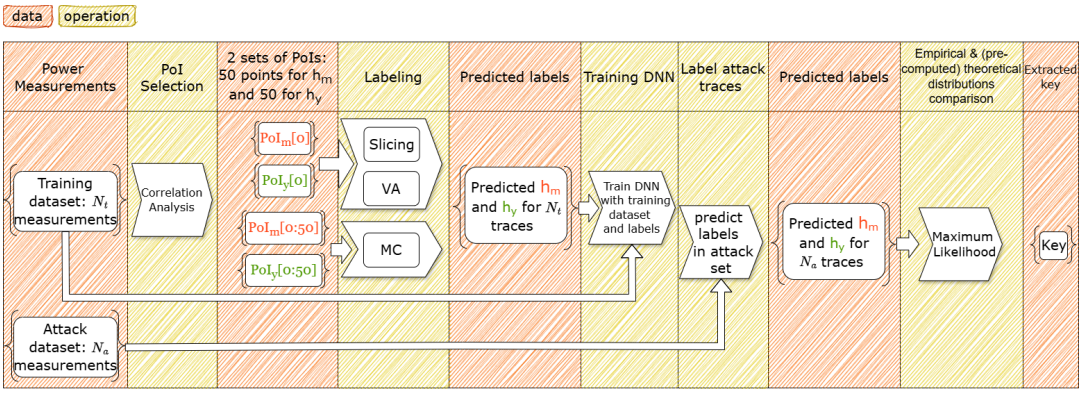


Figure 2: DL-BSCA Framework.

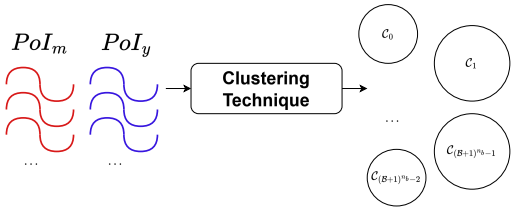


Figure 3: Pictorial illustration of the MC labeling step to produce clusters.

cluster $\mathcal{C}$. Formally, we have

$$CT_{\mathcal{C}}[i] = \frac{1}{N_{\mathcal{C}}} \sum_{t=0}^{N_{\mathcal{C}}-1} trace_{\mathcal{C}}[t,i] \tag{6}$$

where $trace_{\mathcal{C}}[t,i]$ is the i^{th} sample point of the t^{th} trace from the cluster $\mathcal{C}$ and $N_{\mathcal{C}}$ is the total number of traces in cluster $\mathcal{C}$.

2. We split the PoIs of $CT_{\mathcal{C}}$ based on PoI_m and PoI_y, denoted $CT_{\mathcal{C}}^{PoI_m}$ and $CT_{\mathcal{C}}^{PoI_y}$, respectively. Now, suppose there are M different clusters. We define

$$\begin{aligned} CT^{PoI_m}[i] &= \{CT_{\mathcal{C}_j}^{PoI_m}[i] | j \in \{0,\ldots,M\}\} \text{ and} \\ CT^{PoI_y}[i] &= \{CT_{\mathcal{C}_j}^{PoI_y}[i] | j \in \{0,\ldots,M\}\} \end{aligned} \tag{7}$$

where $CT_{\mathcal{C}_j}^{PoI_\ell}[i]$ denote the i^{th} sample point within the PoI_ℓ portion of the center trace $CT_{\mathcal{C}_j}$ for $\ell = m$ or y.

We describe the methodology for CT^{PoI_m} to obtain $Y_{\mathcal{C}_j}^{(m)}$ for all clusters $\mathcal{C}_j$. The technique can be analogously applied to CT^{PoI_y} to attain $Y_{\mathcal{C}_j}^{(y)}$ for all clusters $\mathcal{C}_j$.

3. For each sample point i in PoI_m, we label $CT_{\mathcal{C}_j}^{PoI_m}[i]$ of all $\mathcal{C}_j$ when applying the slicing labeling on the $CT^{PoI_m}[i]$. This yields a collection of possible labels for a $CT_{\mathcal{C}_j}^{PoI_m}$, one from each sample point. More precisely, for each $\mathcal{C}_j$, we have $|PoI_m|$ number of possible labels. This is depicted on the left side of Figure 4.

4. To obtain one label for the cluster, we apply the weighted majority voting to obtain the overall label $Y_{\mathcal{C}_j}^{(m)}$ for all clusters $\mathcal{C}_j$. From the collection of possible labels, we apply the weighted majority voting as follows:

$$Y_{\mathcal{C}_j}^{m} = \arg max_{h \in \{0,\ldots,\mathcal{B}\}} \left(\frac{|CT_{\mathcal{C}_j}^{PoI_{m,h}}|}{\binom{\mathcal{B}}{h}} \right) \tag{8}$$

with $\mathcal{B}$ being the number of bits for a target variable, h being the corresponding Hamming weight, and $|CT_{\mathcal{C}_j}^{PoI_{m,h}}|$ being the number of sample points in $CT_{\mathcal{C}_j}^{PoI_m}$ labeled as the Hamming weight h through slicing labeling. We consider the weighted version because the nature of the Hamming weight and the slicing labeling causes an imbalance within the collection of labels. Indeed, the proportion of appearance of, e.g., Hamming weights 4 and 5 is higher than the Hamming weights 0 and 8 for AES [33]. To compensate for that, we assign more weight to the extreme values of Hamming weights 0 and 8. Thus, we prevent the more occurring Hamming weights from dominating the decision-making process. The weights corresponding to each Hamming weight are calculated using $\binom{\mathcal{B}}{h}^{-1}$.

5. We repeat the same steps 3 and 4 for $CT_{\mathcal{C}_j}^{PoI_y}$ to obtain $Y_{\mathcal{C}_j}^{(y)}$. Lastly, we set all the traces within the cluster $\mathcal{C}_j$ to the same label $(h_m, h_y) = (Y_{\mathcal{C}_j}^{(m)}, Y_{\mathcal{C}_j}^{(y)})$.

We illustrate steps 3 and 4 in Figure 4.

The MC labeling introduced here predicts the Hamming weight for two variables, h_m and h_y (e.g., AES and Kyber in Section 4). However, the MC labeling technique can be generalized to predicts the Hamming weight for three or more variables, as shown for ASCON in Section 4.4. The proposed MC labeling can be used directly with the previously proposed blind SCA [13, 53] to label all the traces and obtain the empirical distribution. Alternatively, MC labeling can be combined with DL-BSCA, where MC labeling is used to label

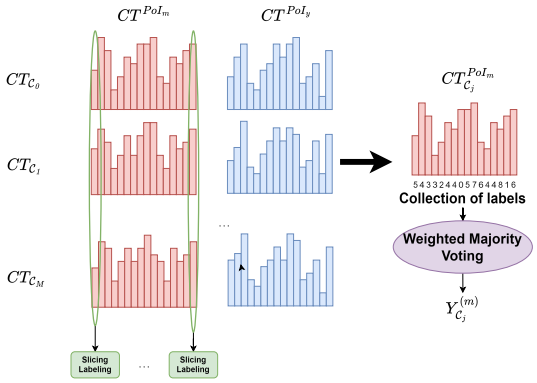


Figure 4: Pictorial illustration of MC labeling steps 3 and 4 to provide label $Y_C^{(m)}$.

the traces for training the DNN.

4 Experimental Setup and Results

4.1 Neural Networks and Hyperparameter Search Space

As the hyperparameters of the neural networks will affect the performance, we randomly sample different hyperparameters from the range listed in Table 8 in Section C. These ranges are chosen based on the reported ranges in the previous works [18, 30, 56]. It has been shown that using regularization techniques helps to maintain the generalization performance of neural networks in the presence of noisy labels [2] and deep learning-based SCA [38]. Thus, we report the results in two scenarios, one with dropout and one without dropout regularization.

We randomly sampled 100 different models to find the best network for each dataset considered for our experiments. We consider two types of architectures: CNNs and MLPs, which have been widely and successfully used in deep learning-based SCA [31, 34, 39, 54]. When training the DNNs, we use categorical cross-entropy. We note that when the label is stored as a tuple, i.e., (h_m, h_y), transformation is required to facilitate a single-input and single-output DNN for training. To address this, we convert the tuple to a single value via $(\mathcal{B}+1)*h_m+h_y$.

4.2 ChipWhisperer

ChipWhisperer (CW) dataset provides measurements of an unprotected AES software implementation running on an 8-bit XMEGA mounted on a ChipWhisperer CW308 UFO board [15]. This dataset has been used in previous works [52, 55].

Attack Point. For AES, the attack point for the DL-BSCA is the Sbox output of the first round, as it represents the non-linear part of the algorithm. Consequently, the two interesting variables to build the joint distribution in the AES primitive are the plaintext, m, and the Sbox output, $y = Sbox(m \oplus k^*)$. Using these two variables, we estimate the joint distribution of $(HW(m), HW(y))$ to carry out the attack on AES.

Dataset. The dataset consists of 10,000 traces with fixed key. Each trace includes 5,000 sample points. We use N_t =8,000 traces for labeling and training the DNN and N_a =2,000 for the attack. The reported values for the guessing entropy are the average values over repeating the attack 100 times using 1,700 random traces from the attack set.[8] As for the classical blind SCA, we use $N = 8,000$ traces to label and attack.

Experimental Results for the CW Dataset. First, we applied the method from Linge et al. [53] and Clavier and Reynaud [13] on the CW dataset and could not recover the secret key despite the relatively high SNR (see Figure 5). This shows previous attacks are not necessarily practical even with higher SNR. We also applied MC labeling on its own without using DNN and observed that the secret key could not be retrieved.

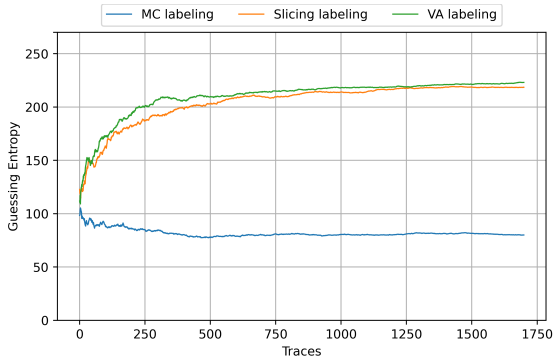


Figure 5: Guessing Entropy on CW dataset using classical blind SCA framework without DNN.

Then, we employ the DL-BSCA framework with various architectures with or without dropout and different labeling techniques. Table 2 provides the overall $NTGE$ and GE obtained in the various tested scenarios. Figures 6 and 7 illustrate GE for MLP and CNN, respectively. We observe that when using the DL-BSCA framework, GE converges below 10 for all cases except for the CNN with MC labeling. This shows the effectiveness of DL-BSCA compared to previous works. In fact, for each labeling technique, when used together with a DNN, it allows us to achieve $GE = 0$ for at least one scenario (DNN + {Slicing, VA, MC + Dropout}). The best results

[8] Guessing entropy measures the average rank of the key across multiple attacks. Since the attack set has limited traces, we randomly select a portion of these traces for each attack iteration. This approach helps make the results less dependent on specific traces and more reflective of realistic attack scenarios.

Table 2: Performance for the CW dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or $NTGE$ when $GE = 0$).

	Without DNN	CNN	MLP
Linge et al. [53]	$GE = 218$	–	–
Clavier & Reynaud [13]	$GE = 223$	–	–
MC labeling	$GE = 79$	–	–
DNN + Slicing	-	$GE = 7.11$	1475
DNN + Slicing + Dropout	-	$GE = 6.14$	$GE = 0.14$
DNN + VA	-	$GE = 7.13$	901
DNN + VA + Dropout	-	$GE = 7.24$	$GE = 8.21$
DNN + MC	-	$GE = 3.05$	$GE = 15.05$
DNN + MC + Dropout	-	$GE = 1$	1455

are obtained when we apply DL-BSCA with VA labeling; we achieve $NTGE$ value of 901.

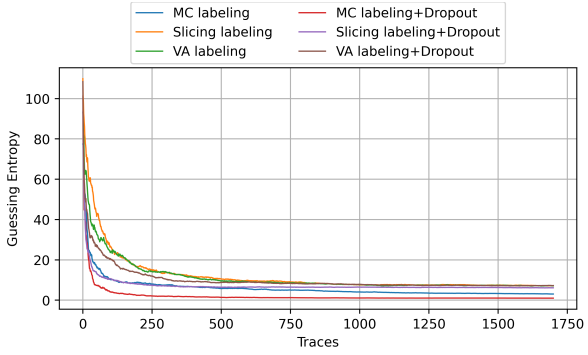


Figure 6: Guessing Entropy for the CW dataset with CNN.

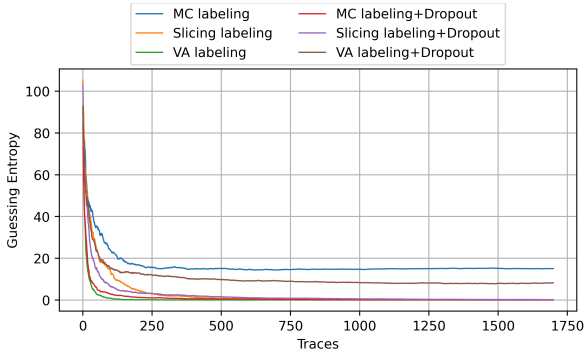


Figure 7: Guessing Entropy for the CW dataset with MLP.

4.3 Kyber

CRYSTAL-Kyber is the standard for Key Encapsulation Mechanism (KEM), which NIST selected for Post Quantum Cryptographic (PQC) applications. We briefly introduce key parameters and components of the Kyber algorithm, which are essential for understanding our attack. For more information, please refer to [3]. The Kyber KEM has three procedures for a full key exchange between two parties. The first party generates the secret key and public key pair, (pk, sk), using the $KeyGen()$ procedure. The secret key is a k-dimensional vector of polynomials, which are elements of the ring $\mathbb{Z}_q[X]/(X^n+1)$. Kyber's security level is therefore determined by parameters k, n, and q. In this work, we focus on security level 3, also known as Kyber768, where the parameters are set as $k = 3$, $n = 256$, and $q = 3329$. Subsequently, the second party employs the public key to encrypt a message through the procedure referred to as $Encapsulation(pk)$. Finally, the first party utilizes its secret key to decrypt the received ciphertext and extract the original message via $Decapsulation(sk, c)$. In this work, we focus on recovering the secret key from the $Decapsulation(sk, c)$ procedure through our proposed DL-BSCA framework. Algorithm 1 shows the decryption steps in the $Decapsulation(sk, c)$ procedure of Kyber. We target the operation in line 4 of the algorithm, highlighted in red.

Algorithm 1 Kyber.CPAPKE.Dec(sk, c): decryption

Input: Secret key $sk \in \mathbb{B}^{12 \cdot k \cdot n/8}$
Input: Ciphertext $c \in \mathbb{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$
Output: Message $m \in \mathbb{B}^{32}$

1: $\mathbf{u} := \text{Decompress}_q(\text{Decode}_{d_u}(c), d_u)$
2: $\mathbf{v} := \text{Decompress}_q(\text{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$
3: $\hat{\mathbf{s}} := \text{Decode}_{12}(sk)$
4: $m := \text{Encode}_1\left(\text{Compress}_q\left(\mathbf{v} - \text{NTT}^{-1}\left(\hat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u})\right), 1\right)\right)$
5: **return** m

Attack Point. In line 4 of Algorithm 1, the secret key in the Number Theoretic Transform (NTT) domain, $\hat{\mathbf{s}}^T$, is multiplied with the ciphertext $\mathbf{u}$ in the NTT domain, to decrypt the message. The NTT is a specialized variant of the Discrete Fourier Transform that operates over finite fields. Transferring the polynomials into the NTT domain provides an efficient way of multiplying them (linear time instead of quadratic time complexity). When converting the polynomials to the NTT domain, the roots of unity of the polynomials are needed. Using NTT transform, a polynomial of degree 255 converts into 128 polynomials of degree one.[9] Eq. (9) shows the expression of the polynomial $a(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, with $n = 256$, in the NTT domain. As one can see, each degree one polynomial in the NTT domain has two coefficients.

$$\text{NTT}(a) = a_0 + a_1 x, a_2 + a_3 x, \ldots, a_{254} + a_{255} x. \quad (9)$$

The forms of the ciphertext u and secret key s are the same as Eq. (9) in the NTT domain. In the case of Kyber, with

[9] This is called an incomplete NTT.

Table 3: Performance for the Kyber dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or $NTGE$ when $GE = 0$).

	Without DNN	CNN	MLP
Linge et al. [53]	$GE = 1242$	–	–
Clavier & Reynaud [13]	$GE = 2415$	–	–
MC labeling	$GE = 1146$	–	–
DNN + Slicing	–	$GE = 165$	$GE = 255$
DNN + Slicing + Dropout	–	$GE = 57$	$GE = 58$
DNN + VA	–	$GE = 2369$	$GE = 1372$
DNN + VA + Dropout	–	$GE = 2356$	$GE = 1143$
DNN + MC	–	$GE = 2.01$	$GE = 9.2$
DNN + MC + Dropout	–	$GE = 8.04$	$GE = 16.2$

the incomplete transform of the polynomials to the NTT domain, we need to use pair-pointwise multiplication to compute the polynomial multiplication. With pair-pointwise multiplication, the coefficients of the deciphered message can be obtained using the coefficients of u and s as follows:

$$\begin{aligned} m_0 &= s_0u_1 + s_1u_0 \\ m_1 &= s_0u_0 + s_1u_1\zeta, \end{aligned} \tag{10}$$

where s_0 and s_1, and u_0 and u_1 are coefficients of first two polynomials of s and u in the NTT domain, respectively. ζ is the root of unity corresponding to the first two polynomials.

We consider the attack point for Kyber with $m = u_0$ and $y = s_0u_0$ (highlighted in red in Eq. (10)). These points are chosen because s_0u_0 involves multiplying the secret key coefficients with those of different ciphertexts, and u_0 is the public variable. The theoretical joint distribution (h^*_m, h^*_y) for two Kyber secret key coefficients differs (see Appendix B). Thus, we can compare the empirical joint distribution of $m = u_0$ and $y = s_0u_0$ with the theoretical ones to recover the secret key coefficient, s_0.

Dataset. The dataset is similar to the one from [35]. It is based on the reference implementation of Kyber KEM taken from *pqm4* [21] library. This library is a benchmarking and testing framework for PQC schemes on the 32-bit ARM Cortex-M4 microcontroller. The library is also a NIST-recommended optimization target for embedded software implementations. The traces are captured from an STM32F3 microcontroller running at 7.372 MHz when using a ChipWhisperer CW308 setup [15]. Measurements are collected with a Lecroy 610Zi oscilloscope at a sampling rate of 500×10^6 samples per second. We captured 100,000 traces with the fixed secret key. In the original dataset, each trace involves 50,000 time samples. To reduce the number of samples in the dataset, we used the window resampling technique from [31], which was shown to be effective in previous works. The final traces after window resampling have 10,000 samples each. We use N_t =80,000 traces for labeling and training the neural network, and the remaining N_a =20,000 traces are used to compute the empirical distribution. The reported values for the guessing entropy are the average values over repeating the attack 100 times using 5,000 random traces from the attack set. With regards to the classical blind SCA, we use $N = 80,000$ traces to label and attack.

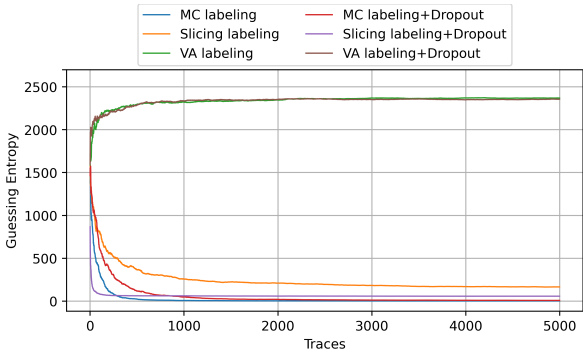


Figure 8: Guessing Entropy for the Kyber dataset using CNN.

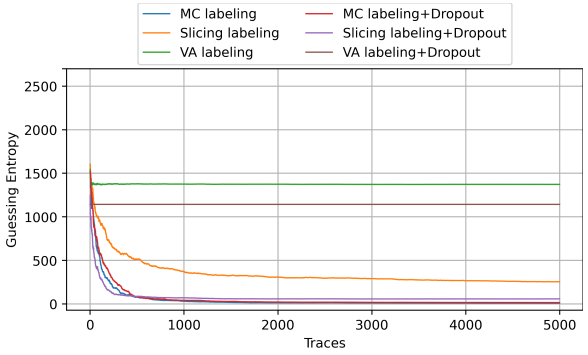


Figure 9: Guessing Entropy for the Kyber dataset using MLP.

Experimental Results on Kyber. We first tried the classical blind SCA with various labeling techniques without using DNN. We could not recover any of the keys in these settings (see Table 3). Next, we run experiments with various scenarios of DL-BSCA and record the performance in Table 3. Figures 8 and 9 provide the performance results for CNN and MLP, respectively. Only MC labeling with DNN could bring GE to values smaller than 10. This shows that with the Kyber dataset, MC labeling results in excellent performance.

4.4 ASCON

ASCON is both CEASAR and NIST lightweight cryptography competition winner, currently being standardized for broad public use [17]. It is an authenticated encryption algorithm based on sponge construction. ASCON encrypts a message to maintain its confidentiality while also providing integrity by

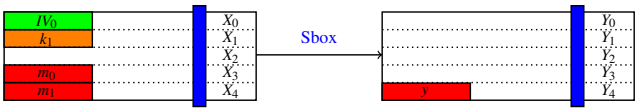


Figure 10: ASCON Substitution Layer: Sbox operation takes in 5 bits input in a column-wise manner, one bit from each word X_i and outputs 5 bits output with one bit from each word Y_i (highlighted in blue). The green block corresponds to the 8 bits of IV used to compute the leakage. The attack points used in our DL-BSCA are highlighted in red. Each red block corresponds to 8 bits in the state. The orange block k_1 corresponds to the 8 bits key we are trying to recover.

attaching a tag to the encrypted message and associated data. ASCON has four inputs: plaintext P, associated data A, nonce N, and a key K. It then outputs the authenticated ciphertext C and an authentication tag T. The algorithm has four operation phases. They are *initialization*, *associated data processing*, *plaintext processing* during encryption (resp. *ciphertext processing* during decryption), and *finalization*. The input of the initialization phase is a 320 bits state that can be written as five 64 bit words X_0 to X_4 with X_0 being the initialization value constant IV, X_1 and X_2 consisting of the 128 bits secret key k, and lastly, X_3 and X_4 being another 128 bits fresh nonce N. The permutation round function of ASCON consists of three parts: addition of the round constants, application of a 5-bit nonlinear Sbox in a column-wise manner (see blue component in Figure 10), and a linear diffusion layer. Figure 10 provides the visualization of the first round substitution and where the variables are located.

This work considers the Sbox output of the first round of the permutation as the attack point. Similar to [37], we consider the leakage from the first 8 bits of Y_4. After substituting the IV, key, and nonce, we have:

$$y = k_1 \& (255 \oplus IV_0 \oplus m_0) \oplus m_0 \oplus m_1, \tag{11}$$

where IV_0 is the first 8 bits constants from X_0 block (in green block of Figure 10), while m_0, m_1 are nonces from the X_3 and X_4 block in the input. Lastly, k_1 is the 8 bit key we are trying to recover (orange block in Figure 10 on the X_1 block). As seen from eq 11, for ASCON, there are three variables for the attack point instead of two. Therefore, we consider both parts of the nonce and the Sbox output in the joint distribution, i.e., $(HW(m_0), HW(m_1), HW(y))$, for both theoretical and empirical.

Dataset. The ASCON dataset is a public dataset obtained from [48]. Traces are captured using a ChipWhisperer Lite board and an 8-bit precision oscilloscope connected to the STM32F4 target. The target microcontroller is a 32-bit platform operating at a default clock frequency of 7.37 MHz. The traces are recorded to include power samples only from the first round of the initialization permutation of the unprotected software implementations. The dataset consists of 200,000 traces, each containing 772 samples. The last 100,000 traces are collected using fixed keys, and we only use them. The N_t =80,000 traces are used as training, while the remaining N_a =20,000 are used as attack traces. The reported values for the guessing entropy are the average values over repeating the attack 100 times using 5,000 traces randomly picked from the attack set. For methods using the classical blind SCA framework, we use the $N = 80,000$ traces.

Table 4: Performance for the ASCON dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or $NTGE$ when $GE = 0$).

	Without DNN	CNN	MLP
Linge et al. [53]	$GE = 77$	–	–
Clavier & Reynaud [13]	$GE = 77$	–	–
MC	$GE = 45$	–	–
DNN + Slicing	–	$GE = 81$	$GE = 49$
DNN + Slicing + Dropout	–	$GE = 12.5$	$GE = 51$
DNN + VA	–	$GE = 48$	$GE = 54$
DNN + VA + Dropout	–	$GE = 11.5$	$GE = 18$
DNN + MC (50 PoIs each)	–	$GE = 2$	$GE = 39$
DNN + MC (50 PoIs each) + Dropout	–	$GE = 19.5$	$GE = 32$
DNN + MC (7 PoIs each)	–	$GE = 5.7$	$GE = 3.5$
DNN + MC (7 PoIs each) + Dropout	–	$GE = 4$	$GE = 0.64$

Experimental Results for ASCON. Applying DL-BSCA on ASCON implementation is interesting because we need to extend the framework for three variables: both nonce m_0 and m_1, and the Sbox output y_4. As mentioned earlier, the theoretical and empirical joint distributions should also include HW of these three variables, and the labeling method should obtain all three variables, i.e. $(h_{m_0}, h_{m_1}, h_{y_4})$. Slicing labeling and VA labeling are extended by applying their technique on the additional variable. As for MC labeling, since these three variables are in bytes (8 bits), the number of clusters that the GMM clustering technique generates is $(\mathcal{B}+1)^3 = (8+1)^3 = 729$. The difference in the case of three variables compared to two variables is that we provide PoIs for all three variables at once to GMM clustering technique, and the CT_C is divided into three portions $CT_C^{PoI_{m_0}}$, $CT_C^{PoI_{m_1}}$, and $CT_C^{PoI_{y_4}}$ after clustering. The rest of the technique remains the same as described earlier in Section 3.3.

We applied both methodologies from [53] and [13] but were unable to recover the secret key practically (see Table 4). Thus, we will focus exclusively on scenarios using DL-BSCA. We run experiments for the various settings of DL-BSCA and show the results in Table 4, and Figures 11 and 12. We could not recover the secret key except for CNN combined with MC labeling when using DL-BSCA. This poor performance could be due to the leakage of y, m_0, and m_1 overlapping in multiple sample points, resulting in many more mislabeled traces (see Figure 18 in Appendix D). This will result in very

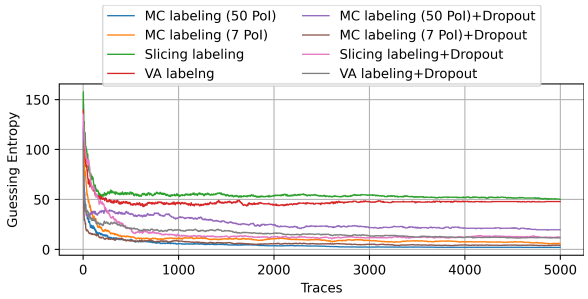


Figure 11: Guessing Entropy for ASCON dataset using CNN.

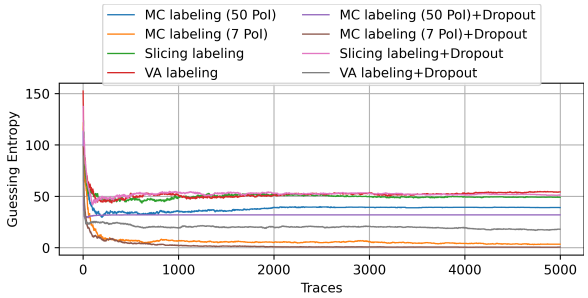


Figure 12: Guessing Entropy for ASCON dataset using MLP.

similar labels between y, m_0 and m_1 when labeling all their $CT_{C_j}^{PoI_\ell}[i]$ in step 4 of the MC labeling. Therefore, it resulted in more mislabeled traces. To confirm this, we select 7 sample points for each leakage with the highest correlation to their corresponding leakages with minor overlap. We observe that GE decreases to under 10 for both CNN and MLP with and without dropout when we use MC labeling. This observation shows that one should consider non-overlapping PoIs when using MC labeling with DL-BSCA for key recovery. Overall, the performance improves significantly when using the DL-BSCA framework instead of just the BSCA framework.

Targeting Countermeasures: Desynchronized CW Dataset. Previous blind SCA approaches selected a single PoI for each variable. Naturally, a hiding countermeasure like desynchronization, which hampers the alignment of the traces, prevents an adversary from selecting a single relevant PoI. Thus, previous blind SCA methods were never applied to desynchronization (or other hiding) countermeasures. The proposed DL-BSCA framework considers 50 PoIs per variable. Moreover, the ability of DNN to handle desynchronization is already demonstrated in prior works [10, 47, 54].

Our desynchronized dataset is derived by applying random desynchronization of up to 10 samples (in either direction) to the CW dataset. We use 8,000 training traces and 2,000 attack traces. The results are averaged over 100 experiments using 1,700 random attack traces. Then, we run the experiments for the different settings with DL-BSCA. The results are provided in Table 5, and Figures 13 and 14. As before,

Table 5: Performance for the desynchronized CW dataset. We highlight successful attacks in blue (i.e., either $GE \leq 10$ or $NTGE$ when $GE = 0$).

	Without DNN	CNN	MLP
Linge et al. [53]	$GE = 67$	–	–
Clavier & Reynaud [13]	$GE = 208$	–	–
MC labeling	$GE = 55$	–	–
DNN + Slicing	–	$GE = 45$	$GE = 20.5$
DNN + Slicing + Dropout	–	$GE = 41$	$GE = 19.5$
DNN + VA	–	$GE = 186$	$GE = 209$
DNN + VA + Dropout	–	$GE = 198$	$GE = 128$
DNN + MC	–	$GE = 2.4$	$GE = 3.5$
DNN + MC + Dropout	–	$GE = 2.1$	$GE = 1.41$

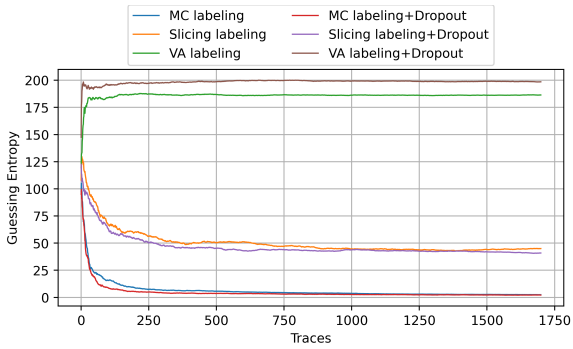


Figure 13: Guessing Entropy for the CW dataset protected with desynchronization using CNN models.

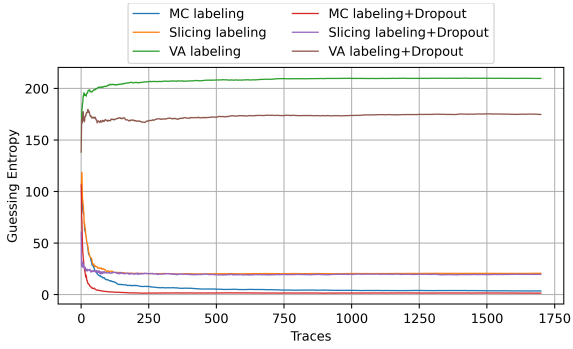


Figure 14: Guessing Entropy for the CW dataset protected with desynchronization using MLP models.

without using DNN, we cannot recover the secret key. Furthermore, with both Slicing and VA labeling, $GE > 10$. We can obtain $GE < 4$ in scenarios when using DL-BSCA with MC labeling. This shows the effectiveness of MC labeling with the DL-BSCA framework in recovering the secret key, even when desynchronization is used to protect the dataset. In fact, this is the first time that a viable attack on a desynchronized target has been demonstrated in the realm of blind SCA, which was previously considered impossible.

5 Discussion

Accuracy. We investigate how the proportion of mislabeled data used to train the DNN affects its performance in key recovery. As MLP and CNN yield similar results, we present MLP results here and CNN results in Appendix F. We begin with completely random training labels and gradually replace a portion of them with the correct labels. Using this set of traces, we train 100 MLPs and compute the average *GE* of the top 10 best-performing MLPs. The results are depicted as a grey line in Figure 15. The trained MLPs obtained an average $GE \leq 10$ when the training labels are above approximately 15% correctly labeled. Next, we label the training traces using the various labeling techniques, achieving an accuracy of 1.2% with MC labeling, 3.3% with Slicing labeling, and 2.7% with VA labeling. Similarly, for each labeling method, we train 100 MLPs and compute their average *GE* for the top 10 best-performing MLPs. The results are plotted in Figure 15 with green for MC labeling, red for Slicing labeling, and blue for VA labeling. Notably, only the MC labeling achieved an average GE of 8, despite a low training accuracy of 1.2%. This discrepancy highlights the inadequacy of accuracy as a metric for evaluating side-channel attack effectiveness, a finding consistent with work by Picek et al. [33] and subsequent research [27, 34]. The reason is that accuracy assesses per-trace performance, whereas GE reflects the cumulative ability to recover the secret key.

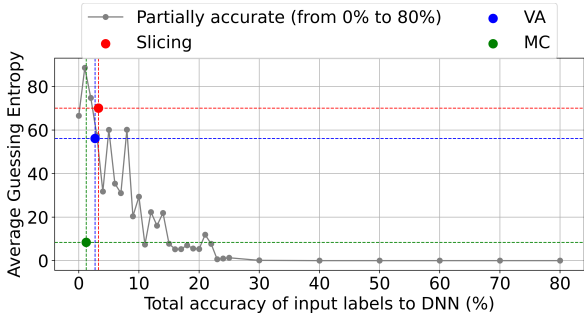


Figure 15: Average Guessing Entropy vs Accuracy within the training data. The average *GE* is computed over the top 10 best-performing MLPs.

We further examine the accuracy for each class for h_m and h_y when applying various labeling techniques on the training traces. As shown in Figure 16 for h_m (for h_y, see Figure 20, Appendix F), with Slicing or VA labeling, the accuracy for marginal classes ($HW(v) = 0, 1, 7$ and 8 for $v = m, y$) is negligible. This is expected given the binomial distribution of Hamming weights, leading to a lower frequency of these marginal classes than the central classes $(3, 4, 5)$. In contrast, MC labeling provides a more balanced representation across all classes, including the marginal ones. The presence of even a few correct examples in less frequent classes enables the DNN to learn and generalize more effectively to unseen data.

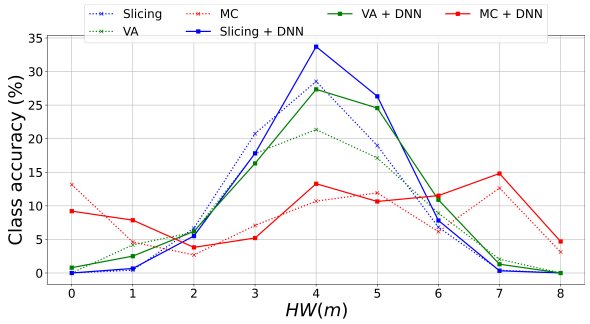


Figure 16: The effect of using MLP on the class accuracy of h_m. The dotted line is the class accuracy provided by the labeling technique on the training traces, and the solid line shows how the accuracy for each class changes after training the DNN and using it to evaluate the same training traces.

Tuning the number of PoIs. Next, we evaluate how employing multiple PoIs can enhance the performance of DL-BSCA. For the CW dataset[10], we compare scenarios using MC labeling with a varying number of PoIs to generate training labels for DNN. For each scenario, we train 100 different models and take the top 10 best-performing DNNs to compute the average *GE*. Table 6 summarizes the average *GE* for CNN and MLP.

Table 6: Average GE for MC labeling for DL-BSCA with different numbers of PoIs.

Number of PoIs (per variable)	1	10	20	50	70	100	200
AVG_GE (CNN)	49	79	40	3	27	62	67
AVG_GE (MLP)	52	60	58	7	18	69	73

Tuning the number of PoIs used is crucial for successful key recovery. Using too few PoIs may fail to capture sufficient information about the targeted variables, as seen in the cases with 1, 10, and 20 PoIs. Conversely, too many PoIs, such as 70 to 200, introduce irrelevant points and deteriorate performance. In our experiments, 50 PoIs gave the best results, but this number is dataset- and model-dependent.

Limitations

Selection of PoIs in Blind Setting. One of the limitations is that PoIs' knowledge is a prerequisite for our and all prior works. Existing methods for PoI selection working in non-blind settings have been shown to be effective [7, 55, 58]. In our work, the PoIs are selected based on their correlation between the measurements and the actual Hamming weight of the target variable. In this step, the public variable m is

[10]The results in other datasets are aligned with the reported results here. As such, we only present results for CW Dataset.

required (a non-blind setting). PoI selection in full blind selection is an open challenge, and we leave it as future work.

Masking. Previous works analyze masking in a weak setting [13,14]. Moreover, all blind SCA on masking are reported in a simulated setting. Naive application of the DL-BSCA framework to masked implementations is not feasible. The randomized shares obscure the original leakage values, rendering it impossible to produce informative labels essential for training the DNN within DL-BSCA. Therefore, attacking the masked implementation in a practical setting remains an open problem that is left for future work.

6 Related Work

The blind SCA framework was first introduced by Linge et al. [53], which also proposed the first labeling technique, slicing. Later, Clavier and Reynaud [13] proposed another labeling technique called VA labeling that leveraged variance analysis for Hamming weight estimation. This attack was practically demonstrated on an 8-bit AVR microcontroller (Arduino Uno) with a high signal-to-noise ratio running an unprotected AES implementation.

Blind SCAs are also explored on popular SCA countermeasures, called masking, in [13] and [14]. Clavier and Reynaud [13] assume the same mask is used for the input and output bytes, whereas [14] extends the attack to scenarios where masks are reused across rounds but applied uniformly to all bytes within a single round. However, these settings represent weak masking schemes, as mask reuse is generally considered poor practice in secure implementations. Both studies were validated only in simulated environments, highlighting the complexity of attacking protected implementations with blind SCA (although [13] includes a practical demonstration, it relies on known plaintext, making it incompatible with the blind SCA). Moreover, all the aforementioned blind SCA approaches only exploit one single PoI. As a result, these methods are not inherently effective against hiding countermeasures such as desynchronization, which introduce additional challenges by obscuring the leakage's temporal alignment.

Recently, Ravi et al. [35] demonstrated blind SCA on a newly standardized PQC algorithm, Kyber, making it the first blind SCA on a public key cryptosystem. The attack targets the decapsulation procedure, where the secret key is manipulated. The work was validated on the STM32F3 platform but required access to a clone device. Access to a clone device was required to train Random Forest classifiers related to precise knowledge of secret inputs and precise PoI selection. In other words, although the attacker only requires side-channel traces without knowledge of input ciphertexts during the attack phase, the adversary still needs knowledge of secret and known inputs from the clone device to train the Random Forest classifier, which may not be a realistic assumption in practice. Separately, blind SCAs have also been investigated for another cryptographic scheme, namely, authenticated encryption with associated data (AEAD). The works [4, 28] proposed theoretical blind SCA targeting the LFSR-based counter on Elephant and Sparkle. Both of these are validated solely in a simulated environment.

Table 7: Comparison of proposed results on different datasets/devices with prior works. We highlight successful attacks in ✓ and failed attacks in ×.

	CW	CW (desync)	Kyber	ASCON
Linge et al. [53]	×	×	×	×
Clavier & Reynaud [13]	×	×	×	×
MLP + MC	×	✓	✓	✓
MLP + MC + Dropout	✓	✓	×	✓
CNN + MC	✓	✓	✓	✓
CNN + MC + Dropout	✓	✓	✓	✓

In contrast, the methods proposed in this paper address the major limitations of existing blind SCA by leveraging deep learning. Table 7 compares our work with [13, 53].[11] Our approach successfully demonstrates blind SCA on various platforms, cryptographic algorithms, and countermeasures, validated through real-world experiments. This significantly enhances the practical applicability of blind SCA.

7 Conclusions and Future Work

This work proposes DL-BSCA, a novel framework for deep learning-based blind side-channel analysis. DL-BSCA harnesses the power of deep learning to effectively handle noisy datasets, addressing key challenges in blind SCA. Unlike prior works focusing on simulated environments or high-leakage devices, we demonstrate successful attacks across diverse platforms, validating our approach on four datasets of real measurements from symmetric key and post-quantum cryptography algorithms. Another key innovation in our work is the MC labeling method, which improves trace labeling by considering dependencies between secret and public variables. This approach outperforms prior techniques, particularly in scenarios where leakage points do not overlap, as shown by our results on AES and Kyber datasets. Finally, we report the first successful blind SCA on hiding countermeasures like desynchronization, showcasing the versatility of the DL-BSCA framework.

Despite these advances, there are open challenges. Previous works considered weak masking and experimented with simulated traces only. Existing studies, including ours, assume that adversaries can locate PoIs. Addressing these limitations would significantly broaden the applicability of blind SCA, paving the way for more robust evaluations of cryptographic implementations.

[11] Other works are excluded from comparison as they were either simulations only or fully profiled.

8 Ethical Considerations

This paper proposes a new deep learning-based blind side-channel attack framework that enables breaking protected cryptographic implementations even when no access to plaintext/ciphertext is available. Our objective is to identify vulnerabilities to improve the security of cryptographic implementations rather than exploiting weaknesses. We do not use live systems or violate terms of service, and to the best of our knowledge, we follow all laws. Our research does not contain elements that could potentially negatively impact team members. The results of our research are shared with relevant evaluation labs.

9 Open Science Policy

Our research results are available to the public. All used datasets are publicly available. The codes and datasets we used to conduct this work can be found using this Version DOI link. One can find the latest update under this Concept DOI: 10.5281.

Acknowledgment

This work was supported in part by the Netherlands Organization for Scientific Research NWO project DISTANT (CS.019) and NWO project PROACT (NWA.1215.18.014). It is also supported in parts by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity Research & Development Programme (Development of Secured Components & Systems in Emerging Technologies through Hardware & Software Evaluation NRF-NCR25-DeSNTU-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the view of National Research Foundation, Singapore and Cyber Security Agency of Singapore. We wish to thank Prasanna Ravi from NTU, Singapore for the initial discussion that started the project.

References

[1] Ieee standard for cryptographic protection of data on block-oriented storage devices. *IEEE Std 1619-2007*, pages 1–40, 2008.

[2] Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron C. Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 233–242. PMLR, 2017.

[3] Roberto Avanzi, Joppe W. Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber (version 3.0): Algorithm specifications and supporting documentation (October 1, 2020). *Submission to the NIST post-quantum project*, 2020.

[4] Awaleh Houssein Meraneh and Christophe Clavier and Hélène Le Bouder and Julien Maillard and Gaël Thomas. Blind Side Channel on the Elephant LFSR. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the 19th International Conference on Security and Cryptography, SECRYPT 2022, Lisbon, Portugal, July 11-13, 2022*, pages 25–34. SCITEPRESS, 2022.

[5] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In *International Conference on Smart Card Research and Advanced Applications*, pages 263–276. Springer, 2012.

[6] Benjamin Timon. Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):107–131, 2019.

[7] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage. *IACR Cryptol. ePrint Arch.*, page 717, 2013.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[9] Germany BSI. Guidelines for Evaluating Machine-Learning based Side-Channel Attack Resistance Part of AIS 46, 2024.

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

[11] Marios O. Choudary and Markus G. Kuhn. Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits. In Marc Joye and Amir Moradi, editors, *Smart Card*

Research and Advanced Applications, pages 85–103, Cham, 2015. Springer International Publishing.

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

[13] Christophe Clavier and Léo Reynaud. Improved Blind Side-Channel Analysis by Exploitation of Joint Distributions of Leakages. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 24–44. Springer, 2017.

[14] Christophe Clavier, Léo Reynaud, and Antoine Wurcker. Quadrivariate improved blind side-channel analysis on boolean masked aes. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 153–167, Cham, 2018. Springer International Publishing.

[15] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 243–260. Springer, 2014.

[16] Aarti Dhapte. ARM Microcontroller Market Research Report By Microcontroller Architecture, 2025.

[17] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34:1–42, 2021.

[18] Prouff Emmanuel, Strullu Remi, Benadjila Ryad, Cagli Eleonora, and Dumas Cecile. Study of deep learning techniques for side-channel analysis and introduction to ascad database. *CoRR*, 53:1–45, 2018.

[19] Eric Brier and Christophe Clavier and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004 - 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[20] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8536–8546, 2018.

[21] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[22] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM cortex-m4. *IACR Cryptol. ePrint Arch.*, page 844, 2019.

[23] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[24] Hélène Le Bouder. *UN FORMALISME UNIFIANT LES ATTAQUES PHYSIQUES SUR CIRCUITS CRYTOGRAPHIQUES ET SON EXPLOITATION AFIN DE COMPARER ET RECHERCHER DE NOUVELLES ATTAQUES*. Theses, Ecole Nationale Supérieure des Mines de Saint-Etienne, October 2014.

[25] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay Attention to Raw Traces: A Deep Learning Architecture for End-to-End Profiling Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):235–274, 2021.

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

[27] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A Comprehensive Study of Deep Learning for Side-Channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[28] Modou Sarry and Hélène Le Bouder and Eïd Maaloouf and Gaël Thomas. Blind Side Channel Analysis Against AEAD with a Belief Propagation Approach. In Shivam Bhasin and Thomas Roche, editors, *Smart Card Research and Advanced Applications - 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14-16, 2023, Revised Selected Papers*, volume 14530 of *Lecture Notes in Computer Science*, pages 127–147. Springer, 2023.

[29] Guilherme Perin, Lukasz Chmielewski, Lejla Batina, and Stjepan Picek. Keep it unsupervised: Horizontal attacks meet deep learning. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):343–372, 2021.

[30] Guilherme Perin and Stjepan Picek. On the influence of optimizers in deep learning-based side-channel analysis. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 615–636. Springer, 2020.

[31] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring Feature Selection Scenarios for Deep Learning-based Side-channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):828–861, 2022.

[32] Stjepan Picek, Annelie Heuser, Alan Jovic, and Lejla Batina. A systematic evaluation of profiling through focused feature selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2802–2815, 2019.

[33] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):209–237, 2019.

[34] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep Learning-based Physical Side-channel Analysis. *ACM Comput. Surv.*, 55(11):227:1–227:35, 2023.

[35] Prasanna Ravi, Dirmanto Jap, Shivam Bhasin, and Anupam Chattopadhyay. Machine learning based blind side-channel attacks on pqc-based kems-a case study of kyber kem. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 01–07. IEEE, 2023.

[36] Francesco Regazzoni, Shivam Bhasin, Amir Alipour, Ihab Alshaer, Furkan Aydin, Aydin Aysu, Vincent Beroulle, Giorgio Di Natale, Paul D. Franzon, David Hély, Naofumi Homma, Akira Ito, Dirmanto Jap, Priyank Kashyap, Ilia Polian, Seetal Potluri, Rei Ueno, Elena-Ioana Vatajelu, and Ville Yli-Mäyry. Machine learning and hardware security: Challenges and opportunities -invited talk-. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*, pages 141:1–141:6. IEEE, 2020.

[37] Azade Rezaeezade, Abraham Basurto-Becerra, Léo Weissbart, and Guilherme Perin. One for all, all for ascon: Ensemble-based deep learning side-channel analysis. In Martin Andreoni, editor, *Applied Cryptography and Network Security Workshops*, pages 139–157, Cham, 2024. Springer Nature Switzerland.

[38] Azade Rezaeezade and Lejla Batina. Regularizers to the rescue: fighting overfitting in deep learning-based side-channel analysis. *J. Cryptogr. Eng.*, 14(4):609–629, 2024.

[39] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):677–707, 2021.

[40] Thomas Roche. EUCLEAK Side-Channel Attack on the YubiKey 5 Series (Revealing and Breaking Infineon ECDSA Implementation on the Way), 2024.

[41] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to titan. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 231–248. USENIX Association, August 2021.

[42] Ioana Savu, Marina Krček, Guilherme Perin, Lichao Wu, and Stjepan Picek. *The Need for MORE: Unsupervised Side-Channel Analysis with Single Network Training and Multi-output Regression*, page 113–132. Springer Nature Switzerland, 2024.

[43] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 30–46. Springer, 2005.

[44] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning From Noisy Labels With Deep Neural Networks: A Survey. *IEEE Trans. Neural Networks Learn. Syst.*, 34(11):8135–8153, 2023.

[45] Marvin Staib and Amir Moradi. Deep Learning Side-Channel Collision Attack. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):422–444, 2023.

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

[47] Suvadeep Hajra and Siddhartha Chowdhury and Debdeep Mukhopadhyay. EstraNet: An Efficient Shift-Invariant Transformer Network for Side-Channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):336–374, 2024.

[48] Léo Weissbart and Stjepan Picek. Lightweight but not easy: Side-channel analysis of the ascon authenticated cipher on a 32-bit microcontroller. *IACR Cryptol. ePrint Arch.*, page 1598, 2023.

[49] Lichao Wu, Guilherme Perin, and Stjepan Picek. Weakly profiling side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024:707–730, 2024.

[50] Lichao Wu, Azade Rezaeezade, Amir Alipour, Guilherme Perin, and Stjepan Picek. Leakage Model-flexible Deep Learning-based Side-channel Analysis. *IACR Commun. Cryptol.*, 1(3):41, 2024.

[51] Lichao Wu, Sébastien Tiran, Guilherme Perin, and Stjepan Picek. Plaintext-based Side-channel Collision Attack. *IACR Commun. Cryptol.*, 1(3):20, 2024.

[52] Lichao Wu, Léo Weissbart, Marina Krček, Huimin Li, Guilherme Perin, Lejla Batina, and Stjepan Picek. Label correlation in deep learning-based side-channel analysis. *Trans. Info. For. Sec.*, 18:3849–3861, January 2023.

[53] Yanis Linge and Cécile Dumas and Sophie Lambert-Lacroix. Using the Joint Distributions of a Cryptographic Function in Side Channel Analysis. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2014.

[54] Trevor Yap, Shivam Bhasin, and Léo Weissbart. Train Wisely: Multifidelity Bayesian Optimization Hyperparameter Tuning in Deep Learning-Based Side-Channel Analysis. In Maria Eichlseder and Sébastien Gambs, editors, *Selected Areas in Cryptography - SAC 2024 - 31st International Conference, Montreal, QC, Canada, August 28-30, 2024, Revised Selected Papers, Part II*, volume 15517 of *Lecture Notes in Computer Science*, pages 294–315. Springer, 2024.

[55] Trevor Yap, Stjepan Picek, and Shivam Bhasin. OccPoIs: Points of Interest Based on Neural Network's Key Recovery in Side-Channel Analysis Through Occlusion. In Sourav Mukhopadhyay and Pantelimon Stanica, editors, *Progress in Cryptology - INDOCRYPT 2024 - 25th International Conference on Cryptology in India, Chennai, India, December 18-21, 2024, Proceedings, Part II*, volume 15496 of *Lecture Notes in Computer Science*, pages 3–28. Springer, 2024.

[56] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for Efficient CNN Architectures in Profiling Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):1–36, 2020.

[57] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8792–8802, 2018.

[58] Yingxian Zheng, Yongbin Zhou, Zhenmei Yu, Chengyu Hu, and Hailong Zhang. How to Compare Selections of Points of Interest for Side-Channel Distinguishers in Practice? In Lucas Chi Kwong Hui, S. H. Qing, Elaine Shi, and Siu-Ming Yiu, editors, *Information and Communications Security - 16th International Conference, ICICS 2014, Hong Kong, China, December 16-17, 2014, Revised Selected Papers*, volume 8958 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2014.

A Algorithms for Joint Distribution Calculation

Algorithms 2 to 4 show how to calculate the theoretical joint distributions for three cryptographic algorithms attacked in this work. The difference in these algorithms stems from the difference in the targeted variables and the nature of the algorithms. It even results in different dimensions (three dimensions) in the case of ASCON (see Algorithm 4).

B Visualization of theoretical joint distribution for Kyber

The theoretical joint distribution $(HW(m), HW(y))$ for two Kyber secret key coefficients differs. For example, the theoretical joint distribution for the secret key coefficient $s_0 = 52$ differs significantly from the theoretical joint distribution for $s_0 = 2056$ as shown in Figure 17. This means we can compare the empirical joint distribution of $m = u_0$ and $y = s_0 u_0$ with the theoretical ones to recover the secret key coefficient, s_0.

Algorithm 2 $(HW(m), HW(y))$ Joint Distribution Calculation for AES

1: **for** fixed k, $k \in \{0,\ldots,255\}$ **do**
2: **for** each m, $m \in \{0,\ldots,255\}$ **do**
3: Calculate $y = Sbox(k \oplus m)$
4: Calculate $HW(m)$ and $HW(y)$
5: Record occurrence of $(HW(m), HW(y))$ tuple
6: **end for**
7: Count the frequency of each tuple $(HW(m), HW(y))$
8: Divide by the total number of observations
9: Save values obtained in line 8 as expected theoretical joint distribution while using key k to be used later
10: **end for**

Algorithm 3 $(HW(u_0), HW(w_0))$ Joint Distribution Calculation for Kyber

1: **for** fixed s_0, $s_0 \in \{0,\ldots,q\}$ **(With** $q = 3329$**) do**
2: **for** each u_0, $u_0 \in \{-\frac{q}{2},\ldots,\frac{q}{2}\}$ **do**
3: Calculate $w_0 = \text{reduced}(u_0.s_0)$
4: Calculate $HW(u_0)$ and $HW(w_0)$
5: Record occurrence of $(HW(u_0), HW(w_0))$ tuple
6: **end for**
7: Count the frequency of each tuple $(HW(u_0), HW(w_0))$
8: Divide by the total number of observations
9: Save values obtained in line 8 as expected theoretical joint distribution while using key s_0 to be used later
10: **end for**

Algorithm 4 $(HW(m_1), HW(m_2), HW(y)$ Joint Distribution Calculation for Ascon

1: Set the initial vector $iv = [128, 64, 12, 6, 0, 0, 0, 0]$
2: **for** for each fixed k, $k \in \{0,\ldots,255\}$ **do**
3: **for** each m_1, $m_1 \in \{0,\ldots,255\}$ **do**
4: **for** each m_2, $m_2 \in \{0,\ldots,255\}$ **do**
5: Calculate $y = k_1 \& (255 \oplus IV_0 \oplus m_0) \oplus m_0 \oplus m_1$
6: Calculate $HW(m_1)$, $HW(m_2)$, $HW(y)$
7: Record occurrence of triple $(HW(m_1), HW(m_2), HW(y)$
8: **end for**
9: **end for**
10: Count the frequency of each triple $(HW(m_1), HW(m_2), HW(y)$
11: Divide by the total number of observations
12: Save values obtained in line 8 as expected theoretical joint distribution while using key k to be used later
13: **end for**

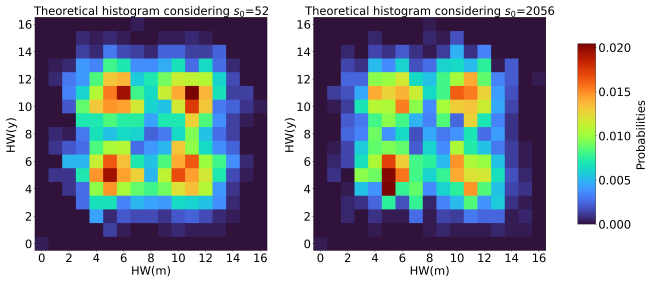


Figure 17: Two theoretical joint distribution of $(HW(m), HW(y))$ considering two different secret key in Kyber.

Table 8: Hyperparameters search space for MLP and CNN.

Hyperparameters	Range
	Dense layers
Number of neurons	[10, 30, 50, 70, 90, 120, 150, 200, 250, 300, 400, 500]
Number of layers	[2, 8], step = 1
	Convolution layers
Number of layers	[2,4], step = 1
Number of neurons	[50, 100, 150, 200, 300, 400, 500]
Number of kernels	[4, 20], step = 2
First layer's filter size	[4, 8, 12, 16, 24]
i^{th} layer filter size	$((i-1)^{th} filter\_size)^2$
Pooling	"Average", "Max"
stride	[2, 10], step = 2
Pooling size and stride	[4, 10], step = 2
	Learning hyperparameters
Weight initialization	"random_uniform", "he_uniform", "glorot_uniform"
Activation function	"relu", "selu", "elu", "tanh"
Batch size	[128, 256, 512]
Learning rate	$[1e^{-3}, 5e^{-4}, 1e^{-4}, 5e^{-5}, 1e^{-5}]$
Optimizer	Adam
Dropout rate	0.5
Epochs	ChipWhisperer: 100, Kyber and Ascon: 25

C Hyperparameter Ranges

D Correlation Analysis for ASCON

Figure 18 shows the correlation analysis of the variables involved in the attack with the traces.

As can be seen, some points with high correlation for y are overlapped with m_0 or m_1. Examples are points around samples 200 and 500. Since the MC algorithm uses all the PoIs to decide about the HW of all three variables simultaneously, this overlapping can confuse the clustering technique, resulting in more mislabeled traces.

E Classical PoIs Selection Tools in Non-Blind Settings

Regarding PoI selection in non-blind side-channel analysis (SCA), early studies proposed the use of Sum of Squared Differences (SOSD) and Sum of Squared T-differences (SOST)

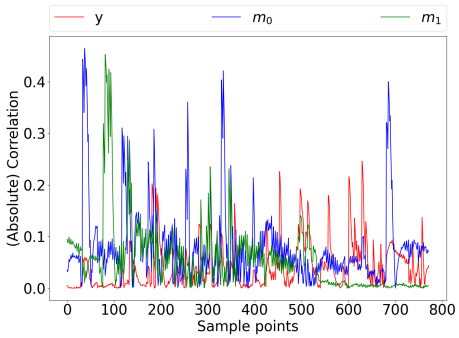


Figure 18: Correlation of various leakage for the ASCON dataset.

to improve the effectiveness of template attacks [12]. Zheng et al. extended this line of work by comparing SOSD and SOST with other techniques, such as Pearson Correlation, and concluded that Pearson Correlation generally offers superior performance [58]. Bhasin et al. [7] later introduced another classical method, the Normalized Inter-Class Variance (NICV), which is closely related to Pearson Correlation and is capable of detecting leakages without relying on a public variable. However, NICV becomes ineffective when the cryptographic primitive is protected by higher-order masking. Picek et al. showed that most feature selection techniques work well for non-blind side-channel analysis, especially if the number of points to be selected is not too small [32]. Thus, these techniques (among others) are effective in finding PoIs, yet selecting PoIs for a blind setting is still an open question.

F Further Experiments on Accuracy

Results related to the accuracy of labeling methods and MLP are provided in Section 5. We present the experimental results for CNN in Figure 19.

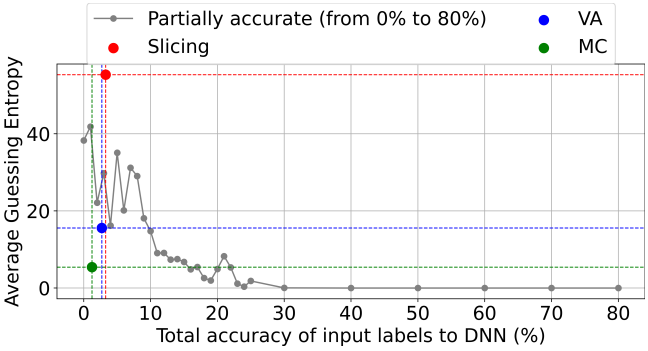


Figure 19: Average Guessing Entropy vs Accuracy within the training data. The average *GE* is computed for the top 10 best-performing CNNs.

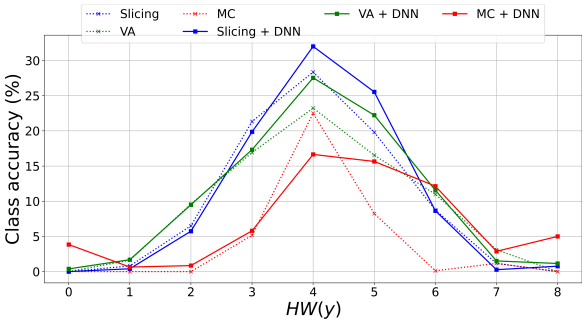
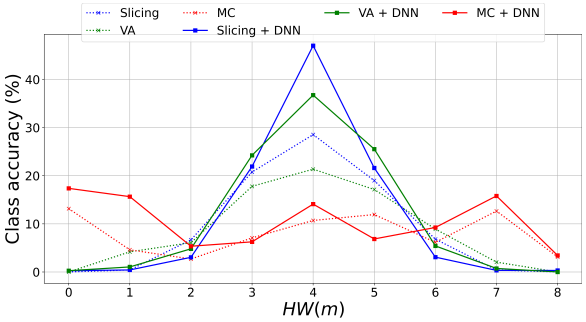
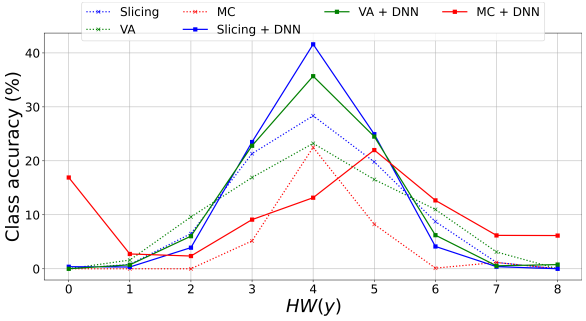


Figure 20: The effect of using MLP on the class accuracy of h_y. The dotted line is the class accuracy provided by the labeling technique on the training traces, and the solid line shows how the accuracy for each class changes after training the DNN and using it to evaluate the same training traces.

Similarly, the results for the accuracy of each class when using MLP are discussed in Section 5. We present the accuracy of each class for h_y when using MLP in Figure 20 and for CNN in Figure 21.



(a) h_m



(b) h_y

Figure 21: The effect of using CNN on the class accuracy of h_m and h_y. The dotted line is the class accuracy provided by the labeling technique on the training traces, and the solid line shows how the accuracy for each class changes after training the DNN and using it to evaluate the same training traces.