HierNet: A Hierarchical Deep Learning Model for SCA on Long Traces

Suvadeep Hajra* and Debdeep Mukhopadhyay*
*Indian Institute of Technology Kharagpur, India

***Abstract*—Side-channel analysis (SCA) compromises the security of cryptographic devices by exploiting various side-channel leakages such as power consumption, electromagnetic (EM) emanations, or timing variations, posing a practical threat to the security and privacy of modern digital systems. In power or EM SCA, statistical or machine learning methods are employed to extract secret information from power/EM traces. In many practical scenarios, raw power/EM traces can span hundreds of thousands of features, with relevant leakages occurring over only a few small segments. Consequently, existing SCAs often select a small number of features before launching the attack, making their success highly dependent on the feasibility of feature selection. However, feature selection may not always be possible, such as in the presence of countermeasures like masking or jitters.**

Several recent works have employed deep learning (DL) methods to conduct SCA on long raw traces, thereby reducing dependence on feature selection steps. However, these methods often perform poorly against various jitter-based countermeasures. While some of these methods have shown high robustness to several jitter-based countermeasures on relatively shorter traces, we demonstrate in this work that their performance deteriorates as trace lengths increase. To mitigate these limitations of the existing models, we develop a hierarchical DL model for SCA on long traces that is effective against masking, random delay and clock jitter countermeasures. The proposed model, HierNet, extracts information from long traces using a two-level information assimilation process. At the base level, a DL model with shift-invariance is employed to extract information from smaller trace segments. Subsequently, a top-level DL model integrates the outputs of the base model to generate the final output. HierNet has been experimentally evaluated against various combinations of masking, random delay, and clock jitter countermeasures, using traces up to $250K$ features long. The results have been compared with three existing SCA benchmark models. They demonstrate HierNet's superiority in several scenarios, such as on long traces or against clock jitter countermeasures, showcasing the ability of HierNet to reach the guessing entropy 1 using fewer than or close to 10 attack traces while the benchmark models fail to do the same using as many as $5K$ attack traces. Additionally, HierNet exhibits significantly better performance in low-training-data scenarios.

***Index Terms*—SCA, Deep Learning, Shift-invariance, Transformer Network**

I. Introduction

In modern days, cryptographic systems serve as the foundation of security and privacy in digital systems. However, these cryptographic systems can be vulnerable to attacks that exploit unintentional information leakage from physical implementations. Side-Channel Analysis (SCA) [1] exploits these unintended channels of information leakage, such as power consumption [2], electromagnetic emanations (EM) [3], or timing variations [1], arising from the physical implementation of cryptographic systems to compromise their security. In this work, we concentrate on power/EM SCA.

A CMOS device's power consumption, or EM depends on the data being processed within the device. This characteristic is leveraged by SCAs to extract the secret key used in a cryptographic device. SCAs involve the collection of power/EM profiles (commonly referred to as power/EM traces) from one or more devices during their execution and recovering the secret key by analyzing these traces using statistical techniques or machine learning methods. Template attack [4] and stochastic attack [5] are two classical approaches to SCAs. More recently, various machine learning (ML) techniques [6], including deep learning (DL) [7], are introduced in SCA.

In many practical scenarios, the raw power/EM traces can be more than $100K$ features long in which the leakages of the relevant cryptographic operations occur over a few small segments (known as Points-of-Interest or PoIs). However, most of the existing works in SCA literature conduct attacks on some pre-selected high signal-to-noise ratio (SNR) features or on a pre-selected attack window containing the PoIs. For example, classical SCA like template and stochastic attacks select a few (in the order of 100) high SNR features prior to the actual attack. Similarly, most of the ML and DL-based methods are investigated on some pre-selected attack window of size in the order of $1K$. However, in implementations protected by masking countermeasures [8], such feature selection requires the knowledge of additional intermediate variables of the cryptographic implementation, which might not always be available. Moreover, even with such knowledge, the feature selection might not be possible in the presence of jitter-based countermeasures (like random delay [9], clock jitter [10], and shuffling [11]). Therefore, the success of the most of the existing SCAs depends on the feasibility of feature selection.

Several deep learning (DL) based research efforts [12]–[16] have focused on improving the scalability and effectiveness of DL models for long attack windows or raw traces, thereby reducing their sensitivity to the attack window selection. Those models have demonstrated good performance even without selecting any smaller attack window. Moreover, several works, such as [13], [14], have demonstrated significantly improved performance when attacking long traces compared to attacking a smaller, selected attack window. However, the performance of most of those methods vastly deteriorates in the presence of countermeasures like random delay and clock jitter. For example, the investigation of [14] reveals that their methods might require more than 30 times more traces for the success-

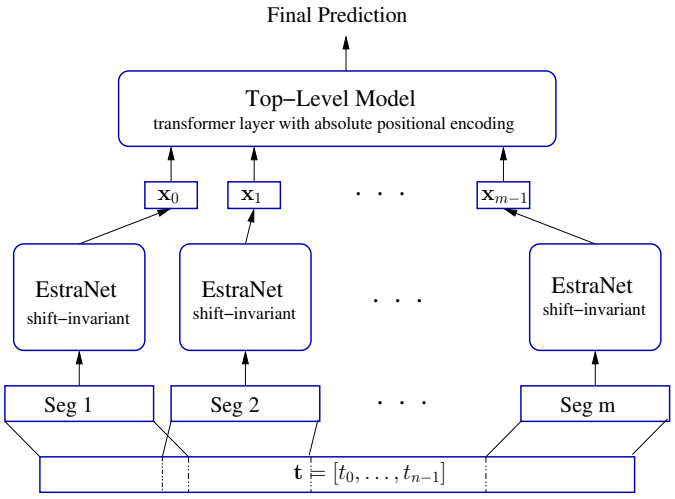


Fig. 1. HierNet Architecture. All parameters except the last layer of the base models are shared.

ful attack (to reach the guessing entropy 1) when subjected to desynchronized traces as compared to synchronized traces. Similarly, the results of [16] illustrate that the DL models from [13] may fail to reach the guessing entropy 1 even when using up to $5K$ traces if the traces are desynchronized using clock jitter effect. In contrast, these models require only around 10 traces to achieve the same result in the absence of clock jitter. In [17], Bursztein et al. have not shown the success of their DL model, GPAM, against jitter-based countermeasures, though they have demonstrated its success on traces with lengths more than $10M$.

Recently, [16] introduced a Transformer Network-based DL model for SCA on long traces, which has demonstrated state-of-the-art results against random delay and clock jitter countermeasures along with the masking countermeasure. In this work, we closely analyze EstraNet and show that its performance gradually deteriorates as trace lengths increase. To address this limitation, we introduce HierNet, a hierarchical DL model for SCA. Unlike EstraNet, which attempts to assimilate information from the entire trace using a stack of monolithic layers, HierNet employs a two-level information assimilation approach. HierNet uses an EstraNet model to process smaller trace segments at the lower level. A second-level TN model then aggregates the outputs from EstraNet across all segments to produce the final result, as illustrated in Figure 1. HierNet retains EstraNet's resilience against masking, random delay, and clock jitter countermeasures while effectively addressing its limitations with longer traces.

Contributions:

The contributions of the work are the following:

- EstraNet, the shift-invariant DL model recently introduced in [16], demonstrated the state-of-the-art performance against various combinations of masking, random delay and clock jitter countermeasures. The model also has linear time and memory cost, making it computationally scalable to long traces. We investigate the attack efficacy of EstraNet on longer traces. We identify that the performance of EstraNet degrades with increasing attack window size or trace length, necessitating a) more effort in hyper-parameter tuning, b) potential overfitting issues, and c) diminished performance.
- We introduce HierNet, a hierarchical DL model for SCA on long traces. HierNet employs a two-level hierarchical DL architecture. At the first level, it utilizes an EstraNet model to assimilate information from smaller trace segments. At the top level, another TN model aggregates the outputs of EstraNet from all segments. This top-level TN model incorporates a novel self-attention mechanism with absolute positional encoding.
- We experimentally evaluate HierNet against various combinations of masking, random delay, and clock jitter countermeasures using traces with lengths upto $250K$ features, comparing its performance with three benchmark DL models. The results show that HierNet can significantly outperform the benchmark models, particularly against clock jitter countermeasures, or on longer traces. HierNet can reach the guessing entropy 1 using less than or close to ten attack traces, whereas the benchmark models struggle to reach the same even using $5K$ attack traces in these scenarios. HierNet has also demonstrated signficantly better performance on low training data scenarios.

The paper is organized as follows. Section II introduces the notations and provides background information. In Section III, we briefly describe the EstraNet model. Section IV investigates the effectiveness of EstraNet on long traces. In Section V, we introduce the proposed HierNet architecture. Section VI presents the empirical evaluation of HierNet and compares it with three benchmark models. In Section VII, we discuss the limitations of our empirical evaluation and explore future directions for the work. Finally, we conclude the paper in Section VIII.

II. Preliminaries

A. Notations

The notational conventions used in the paper are as follows. A letter in the capital (like X), the corresponding small letter (like x), and the corresponding calligraphic letter (like $\mathcal{X}$) are respectively used to represent a random variable, an instantiation, and the domain of the random variable. Similarly, a capital letter in bold (like $\mathbf{X}$) and the corresponding small letter in bold (like $\mathbf{x}$) are respectively used to represent a random vector and its instantiation. We use a capital letter in Roman style (like M) to represent a matrix. The i-th element of a vector $\mathbf{x}$ is represented by $\mathbf{x}[i]$. Similarly, the element of i-th row and j-th column of a matrix M is represented by $\mathrm{M}[i, j]$. The notations $\mathbb{P}[\cdot]$ and $\mathbb{E}[\cdot]$ are respectively used to represent a random variable's probability distribution function and expectation.

B. Side-Channel Analysis

The power consumption and electromagnetic (EM) emissions of a semiconductor device are influenced by the values manipulated within the device. SCA takes advantage of this behavior in semiconductor devices to extract information about

the sensitive variables in a cryptographic implementation, revealing the device's secret key. To achieve this, in an SCA, an attacker gains control of the target device, also referred to as the Device Under Test (DUT). Then he/she collects power or EM measurements, known as traces, by repeatedly executing the encryption/decryption algorithm with various plaintexts/ciphertexts. Subsequently, the attacker employs statistical tests to deduce the device's secret key from the traces.

SCA can be categorized into two types: profiling SCA and non-profiling SCA. In profiling SCA, the attacker possesses a clone of the DUT under control. With this clone device, he/she can create a profile of the DUT's power consumption or EM emission characteristics and employs this profile to carry out the attack on the DUT. On the other hand, in non-profiling SCA, the attacker lacks a clone device and, therefore, cannot build a power/EM profile a priory. Instead, he/she attempts to recover the secret key solely from the traces of the DUT. This paper considers profiling SCA.

C. Deep Learning based Profiling SCA

Like any profiling SCA, DL-based profiling SCA also involves a two-phase process. In the initial phase, known as the profiling phase, the attacker configures a predefined key in the clone device and acquires a large number of traces by executing encryption (or decryption) operations on well-known plaintexts (or ciphertexts) using the device. For each trace, the attacker calculates the value of an intermediate secret variable denoted as $Z = F(X, K)$. Here, X represents a component of the random plaintext (or ciphertext), K represents a component of the known key, and $F(\cdot, \cdot)$ denotes a cryptographic primitive. Then, the attacker uses the traces and the values of the intermediate variable Z to train a DL model to predict Z from the $\mathbf{L}$, where $\mathbf{L}$ represents a trace. More specifically, the DL model takes a trace $\mathbf{L}$ as its input and outputs a probability distribution over the values of Z. Let $f(\cdot; \theta^*)$ denote the trained DL model, with θ^* denoting the model parameters. The output of the DL model for a trace, $\mathbf{l}$, can be given as $\mathbf{p} = f(\mathbf{l}; \theta^*)$ where $\mathbf{p} \in \mathbb{R}^{|\mathcal{Z}|}$ with $\mathbf{p}[j]$, for $j = 0, \cdots, |\mathcal{Z}| - 1$, denoting the predicted probability of the intermediate variable $Z = j$.

In the attack phase, attacker collect multiple traces $\{\tilde{\mathbf{l}}^i\}_{i=0}^{T_a-1}$ by executing the DUT. Let $\{\tilde{p}^i\}_{i=0}^{T_a-1}$ be the corresponding plaintexts (or ciphertexts). Then, he/she uses the set of attack trace-plaintext pairs $\{(\tilde{\mathbf{l}}^i, \tilde{p}^i)\}_{i=0}^{T_a-1}$ to compute the score of each key $k \in \mathcal{K}$ as

$$\hat{\delta}_k = \sum_{i=0}^{T_a-1} \log \mathbf{p}^i[F(\tilde{p}^i, k)] \tag{1}$$

where $\mathbf{p}^i = f(\tilde{\mathbf{l}}^i; \theta^*)$ is the output of the DL model for the i-th trace. The key with the highest score, i.e., $\hat{k} = \text{argmax}_k\, \hat{\delta}_k$, is considered as the predicted key. The prediction is said to be correct if $\hat{k} = k^*$ holds. Alternatively, the rank of the correct key within the list of all possible keys, sorted based on their respective scores $\hat{\delta}_k$, serves as a metric to determine the success level of the attack. The rank of the correct key averaged over multiple repetitions of the attack, referred to as guessing entropy, is a widely used metric to evaluate the success of an attack.

D. Transformer Network

A Transformer Network (TN) is a type of DL model composed of multiple transformer layers. The input trace is fed into the first transformer layer, and the output of each layer serves as the input for the subsequent layer. The final output of the last transformer layer is considered the output of the TN model.

Each transformer layer consists of a self-attention layer and a position-wise feed-forward layer. It takes a sequence $\mathrm{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}]^T \in \mathbb{R}^{n \times d}$ of n feature vectors as input and outputs another sequence $\mathrm{Y} = [\mathbf{y}_0, \ldots, \mathbf{y}_{n-1}]^T \in \mathbb{R}^{n \times d}$ where n is the sequence length (trace length in the context of SCA) and d is the dimension of the feature vectors. The output Y is given by

$$\hat{\mathrm{Y}} = f_{SA}(\mathrm{X}) + \mathrm{X}$$
$$\mathrm{Y} = f_{PFF}(\hat{\mathrm{Y}}) + \hat{\mathrm{Y}}$$

where $f_{SA} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$ is the function representing the self-attention layer and $f_{PFF} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$, is the function representing the position-wise feed-forward layer. The position-wise feed-forward layer independently applies a non-linear transformation to each feature vector, increasing the non-linearity (and, thus, its representation power) of the TN model. On the other hand, the self-attention layer captures the inter-dependencies among different features [15], [16], [18]. It is worth mentioning that the number of parameters in both the self-attention and position-wise feed-forward layer is independent of the sequence length n. Therefore, it can scale to a very long sequence without being prone to overfitting.

The self-attention layer is a key component of a Transformer Network (TN). Different variations of this layer lead to vastly different types of TN models, each varying significantly in performance and computational cost. In the next part of this section, we briefly describe the self-attention layer and some of its variations, setting the groundwork for the following discussion.

1) Self-attention Layer: Let the sequence $\mathrm{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}]^T \in \mathbb{R}^{n \times d}$ of n feature vectors is the input of a self-attention layer and $\hat{\mathrm{Y}} = [\hat{\mathbf{y}}_0, \ldots, \hat{\mathbf{y}}_{n-1}]^T \in \mathbb{R}^{n \times d}$ be the corresponding output sequence. Then, in a self-attention layer, each output feature vector $\hat{\mathbf{y}}_i$, for $i = 0, \ldots, n-1$, is computed as

$$\hat{\mathbf{y}}_i = \sum_{j=0}^{n-1} \frac{k(f(\mathbf{x}_i), g(\mathbf{x}_j))}{\sum_{l=0}^{n-1} k(f(\mathbf{x}_i), g(\mathbf{x}_l))} h(\mathbf{x}_j) \tag{2}$$

where $f(\mathbf{x}_i), g(\mathbf{x}_i),$ and $h(\mathbf{x}_i)$ are some vector valued functions of input feature $\mathbf{x}_i$, and $k : \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \mapsto \mathbb{R}^+$ is called the kernel function. The kernel function outputs a high score, $k(f(\mathbf{x}_i), g(\mathbf{x}_j))$, if it finds the features $\mathbf{x}_i$, and $\mathbf{x}_j$ to be related and a low score otherwise. In summary, in self-attention, each output feature, $\hat{\mathbf{y}}_i$, is computed by taking the weighted sum of the input features $h(\mathbf{x}_0), \ldots, h(\mathbf{x}_{n-1})$ where the weight for the j-th input feature is given by the normalized score $k(f(\mathbf{x}_i), g(\mathbf{x}_j)) / \sum_{l=0}^{n-1} k(f(\mathbf{x}_i), g(\mathbf{x}_l))$. Therefore, if the i-th feature is related to (or dependent on) the j-th feature, the weight of the j-th input feature in the i-th output feature can be significantly greater than weights of the other features,

increasing the component of the j-th feature (in the input) in the i-th feature (in the output). In that way, self-attention layer can combine the information of the i and j-th features, capturing the dependency between the input features.

One of the most common kernel used in TN literature is obtained by setting the kernel as $k(f(\mathbf{x}_i), g(\mathbf{x}_j)) = \exp(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k})$, where $\langle\cdot,\cdot\rangle$ represents the dot product between two vectors and d_k is the dimension of the outputs of $f(\cdot)$ and $g(\cdot)$. The above kernel leads to following self-attention known as softmax self-attention:

$$\hat{\mathbf{y}}_i = \sum_{j=0}^{n-1} \frac{\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1}\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_l)\rangle/\sqrt{d_k}\right)} h(\mathbf{x}_j) \tag{3}$$

$$= \sum_{j=0}^{n-1} \text{softmax}\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right) h(\mathbf{x}_j) \tag{4}$$

In the self-attention mechanism of the form of Eq. (2), the attention score from feature $\mathbf{x}_i$ to feature $\mathbf{x}_j$ does not depend on their positions, i or j, in the input sequence. Therefore, the actual order of the features in the input sequence does not affect the attention scores. In other words, permuting the input sequence also permutes the output sequence in the same way.

In many tasks, we want the attention a feature gives to another feature to depend on their absolute positions or the distance between them. For instance, in the presence of random delay or clock jitter countermeasures, the distances between the informative features remain almost the same across all traces. These properties have been exploited in the TN models proposed in [16], [18] by making the self-attention shift-invariant. This approach is described as follows:

Relative Positional Encoding: To make the self-attention shift-invariant, one can incorporate relative positional encoding into kernel function by generalizing it to $k_r(f(\mathbf{x}_i), g(\mathbf{x}_j), e(i-j))$, where $e(i-j)$ is a vector space encoding of the relative distance $i-j$. Therefore, the output of the resultant self-attention takes the form

$$\hat{\mathbf{y}}_i = \sum_{j=0}^{n-1} \frac{k_r(f(\mathbf{x}_i), g(\mathbf{x}_j), e(i-j))}{\sum_{l=0}^{n-1} k_r(f(\mathbf{x}_i), g(\mathbf{x}_l), e(i-l))} h(\mathbf{x}_j) \tag{5}$$

With relative positional encoding, the kernel function returns a high score not only based on the similarity between $\mathbf{x}_i$, and $\mathbf{x}_j$, but also considering their relative distance $i-j$.

Another scenario where incorporating position information into the attention scores is beneficial is when there is no jitter-based countermeasure in the implementation. In the absence of such countermeasures, the positions of the informative features in the input sequence remain consistent across all traces. To exploit this property, one approach is to make the attention scores dependent on the absolute positions of the features. In this work, we achieve this by incorporating absolute positional encoding into the self-attention layer. We introduce the self-attention with absolute positional encoding in Section V-B.

In the next section, we briefly describe the architecture of EstraNet and then explore its limitations with longer traces.

III. EstraNet Model

We first provide a brief overview of the EstraNet architecture, followed by its key component, the GaussiP attention.

A. The Overall Architecture

The overall EstraNet architecture is shown in Figure 2a. Being a transformer network, EstraNet consists of multiple EstraNet layers stacked one after another. The output sequence of the top EstraNet layer (the sequence $\{\mathbf{y}_0, \ldots, \mathbf{y}_{m-1}\}$, where each $\mathbf{y}_i \in \mathbb{R}^d$) is reduced to a single d-dimensional vector $\bar{\mathbf{y}}$ using a softmax attention layer. The output $\bar{\mathbf{y}}$ is then fed to a classification layer for the final classification.

Like a transformer layer, each EstraNet layer consists of a self-attention layer and a position-wise feed-forward layer (shown in Figure 2b). However, instead of vanilla self-attention, the EstraNet layer uses a novel self-attention named GaussiP attention. The standard layer normalization operation of the vanilla transformer layer is also replaced by a novel layer-centering operation in EstraNet layer.

B. GaussiP Attention

The GaussiP attention is similar to self-attention with relative positional encoding given in Eq. (5). However, it uses a form of Gaussian kernel over the relative distance $i-j$ and special encoding to facilitate a) information flow from one feature to a distant feature, and b) controllable sparsity of the attention scores. Moreover, it uses a closed-form approximation for the denominator in Eq. (5). More precisely, the GaussiP attention's kernel takes the form:

$$k_{GPA}(i-j) = \exp\left(-\frac{||\mathbf{i}-\mathbf{j}||_2^2}{2}\right)$$

$$= \exp\left(-\frac{\beta_2^2 s_p^2 (i-j-c_p n)^2 ||\mathrm{W}_p\mathbf{p}||_2^2}{2}\right) \tag{6}$$

$$= \exp\left(-\frac{\hat{\beta}_2^2 s_p^2 (i/n-j/n-c_p)^2 ||\mathrm{W}_p\mathbf{p}||_2^2}{2}\right), \tag{7}$$

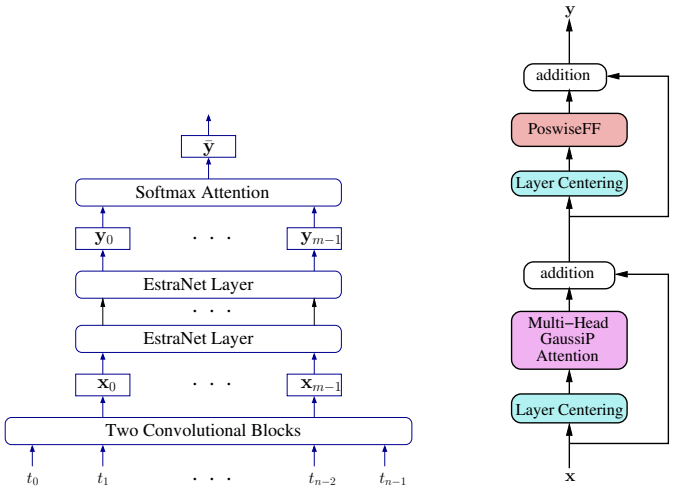
where $\mathbf{i} = \beta_2 s_p \mathrm{W}_p(\mathbf{b}+i\mathbf{p})$ and $\mathbf{j} = \beta_2 s_p \mathrm{W}_p(\mathbf{b}+j\mathbf{p}+c_p n\mathbf{p})$ be the encoding of the positional indices i and j, with $s_p \in \mathbb{R}^+$, $c_p \in (0,1)$, $\mathrm{W}_p \in \mathbb{R}^{d_e \times d}$, $\mathbf{p} \in \mathbb{R}^d$ being the parameters and $\hat{\beta}_2 = n\beta_2 > 0$ being a hyper-parameter of the model. To make the time and memory complexity of the attention linear with respect to the sequence length n, the above kernel is approximately factorized using the d_p-dimensional Fourier feature map defined as

$$\phi_{\text{fr}}(\mathbf{x}) = \frac{1}{\sqrt{d_p}}[\sin(\mathbf{w}_0^T\mathbf{x}), \ldots, \sin(\mathbf{w}_{d_p-1}^T\mathbf{x}), \cos(\mathbf{w}_0^T\mathbf{x}), \ldots, \cos(\mathbf{w}_{d_p-1}^T\mathbf{x})]^T, \tag{8}$$

where $\mathbf{w}_0, \ldots, \mathbf{w}_{d_p-1}$ are the i.i.d. (independent and identically distributed) samples from d_e dimensional Gaussian distribution with zero mean and identity covariance matrix ensuring

$$k_{GPA}(i-j) \approx \phi_{fr}(\mathbf{i})^T \phi_{fr}(\mathbf{j}) \tag{9}$$

to hold. Furthermore, the GaussiP attention approximates the term $\sum_{l=0}^{n-1} k_{GPA}(i-l)$ in the closed form $n/\hat{\beta}_2 s_p ||\mathrm{W}_p\mathbf{p}||_2$. Therefore, the output feature vectors $\hat{\mathbf{y}}_i$s, of the resultant GaussiP attention,



(a) EstraNet Architecture. (b) EstraNet Layer.

Fig. 2. Schematic of EstraNet [16] Architecture.

$$\hat{\mathbf{y}}_i^T = \frac{\sum_{j=0}^{n-1} k_{GPA}\,(i-j)\,h(\mathbf{x}_j)^T}{\sum_{l=0}^{n-1} k_{GPA}\,(i-l)}, \quad \text{following Eq. (5)}, \tag{10}$$

$$\approx \frac{\hat{\beta}_2 s_p ||\mathrm{W}_p\mathbf{p}||_2}{n} \sum_{j=0}^{n-1} \left(\phi_{fr}(\mathbf{i})^T \phi_{fr}(\mathbf{j})\right) h(\mathbf{x}_j)^T$$

$$\approx \frac{\hat{\beta}_2 s_p ||\mathrm{W}_p\mathbf{p}||_2}{n} \left[\phi_{\text{fr}}(\mathbf{i})^T \sum_{j=0}^{n-1} \phi_{\text{fr}}(\mathbf{j}) h(\mathbf{x}_j)^T\right], \tag{11}$$

for all $i = 0, \ldots, n-1$, can be computed in linear time and memory with respect to n.

One significant advantage of the GaussiP attention is that the i-th output feature puts high attention around $i - j - c_p n = 0$ or $j = i - c_p n$, allowing information flow from features near $i - c_p n$ to feature i. Since $i - c_p n$ can be far away from i (by setting $c_p \in (0, 1)$ properly), GaussiP attention allows information flow from one region of the traces to a distant region, enabling integration of the leakages of different secret shares of a masking protected implementation to form the unmasked secret. Another advantage of the GaussiP attention is that by setting $\hat{\beta}_2$ appropriately or learning a proper value of s_p, the sharpness and sparseness of the attention scores can be adjusted. More precisely, when $\hat{\beta}_2 s_p$ is small, the GaussiP attention puts high attention on a larger trace segment. On the other hand, the high attention scores can be made to concentrate on a very small region of the traces by making $\hat{\beta}_2 s_p$ large. This property of GaussiP attention makes it highly effective for SCA on long traces, where each PoI spans over a very small part of the entire trace.

C. Multi-head GaussiP Attention

In implementations protected by masking countermeasures, the intermediate secret variable is split into multiple shares such that any proper subset of the shares is independent of the unmasked secret. Therefore, to successfully attack such implementations, one needs to combine the information from the leakages of all the shares. However, in many masked implementations (e.g., software implementations), different secret shares leak at different regions (which may be long distances apart from each other) of the traces. Thus, to attack such implementations, the DL models must be able to combine the leakages from distant regions.

Though the vanilla GaussiP attention allows information flow from features near $i - c_p n$ to feature i for all i, such flow of information may be too sparse to be able to combine the leakages from the distant regions. In EstraNet, Hajra et al. solve this problem using multi-head GaussiP attention. More precisely, an h-head GaussiP attention, where h is a positive integer, computes a hd_v-dimensional output by concatenating the outputs of h parallel GaussiP attentions. The hd_v-dimensional output is then projected back to d-dimensional output $\hat{\mathbf{y}}$ using a $d \times hd_v$-dimensional projection matrix W_o. The c_p parameter of each of the h GaussiP attentions is initialized to different values, facilitating information flow from h different regions, $\left(i - c_p^{(0)} n\right), \ldots, \left(i - c_p^{(h-1)} n\right)$, to the i-th feature.

IV. Performance Degradation of EstraNet with Increasing Trace Lengths

A. Limitation of GaussiP Attention

The performance of GaussiP attention deteriorates drastically as the size of the attack window increases gradually. To illustrate the point, we recall from Eq. (10) that an output feature of the GaussiP attention is computed as

$$\hat{\mathbf{y}}_i \approx \sum_{j=0}^{n-1} \frac{k_{GPA}(i-j)}{\sum_{l=0}^{n-1} k_{GPA}(i-l)} h(\mathbf{x}_j) = \sum_{j=0}^{n-1} w_j h(\mathbf{x}_j) \tag{12}$$

where $w_j \triangleq k_{GPA}(i-j)/\sum_{l=0}^{n-1} k_{GPA}(i-l)$ is the weight for the j-th input feature vector $\mathbf{x}_j$. Note that since k_{GPA} is a Gaussian kernel, it puts high attention to a small contiguous segment, say $\mathcal{H} = [a, b]$, of the input sequence where $0 \leq a < b < n$. Within this high-attention segment $\mathcal{H}$, a smaller segment, say $\mathcal{I} \subseteq \mathcal{H}$, corresponds to some high SNR features. Thus, we can re-write Eq. (12) as

$$\hat{\mathbf{y}}_i \approx \sum_{j \in \mathcal{I}} w_j h(\mathbf{x}_j) + \sum_{j \in \mathcal{H} \setminus \mathcal{I}} w_j h(\mathbf{x}_j) + \sum_{j \notin \mathcal{H}} w_j h(\mathbf{x}_j) \tag{13}$$

When $\hat{\beta}_2$ hyper-parameter of k_{GPA} remains constant, the number of high attention features, i.e., $|\mathcal{H}|$, increases almost linearly with the increase of the attack window size n (as explained in Appendix A). However, among this $\mathcal{H}$ features, the high SNR features $\mathcal{I}$ remain almost the same. Consequently, the norm of the signal part $||\sum_{j \in \mathcal{I}} w_j h(\mathbf{x}_j)||_2$ within $\hat{\mathbf{y}}_i$ remains almost the same while the norm of the noise part, i.e., $||\sum_{j \in \mathcal{H} \setminus \mathcal{I}} w_j h(\mathbf{x}_j) + \sum_{j \notin \mathcal{H}} w_j h(\mathbf{x}_j)||_2$ increases with the attack window size n, reducing the effective SNR of $\hat{\mathbf{y}}_i$. One way to alleviate the above problem is to increase the value of $\hat{\beta}_2$ with the increase in the attack window size, making the size of high SNR features $|\mathcal{H}|$ close to the number of high SNR features $|\mathcal{I}|$. However, increasing the $\hat{\beta}_2$ hyper-parameter with the increase in the attack window size, the propagation of information through the EstraNet layers becomes too sparse, making the model untrainable. Alternatively, one can increase

TABLE I
THE T_{GE1} (LOWER IS BETTER) VALUES OBTAINED FROM THE ESTRANET MODEL ON TRACES WITH INCREASING TRACE LENGTH. FOR EACH EXPERIMENT, THE MODEL HAS BEEN TRAINED THRICE. THE TABLE SHOWS THE T_{GE1} FROM ALL THREE TRAINING RUNS.

Dataset	Window size $10K$			Full-length Traces			Double-length Traces		
	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3
ASCADf	13	9	15	6	13	30	$> 5K$	$> 5K$	$> 5K$
ASCADr	4	6	5	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$
CHES20	11	4	7	5	6	5	$> 5K$	$> 5K$	$> 5K$

both $\hat{\beta}_2$ and the number of heads, h, of the multi-head GaussiP attention. The larger value of $\hat{\beta}_2$ helps to prevent the deterioration of the effective SNR at the outputs of the GaussiP attention, and the larger value of h increases the flow of information through the EstraNet layers, increasing the model's ability to combine leakages from distant PoIs. However, the approach has several drawbacks:

1) EstraNet might require significantly more tuning of several hyper-parameters like $\hat{\beta}_2$ and h, increasing the compute cost of the training significantly.
2) Increasing the hyper-parameter h increases the model's number of parameters, increasing the model's training and inference cost. Moreover, having more parameters increases the risk of overfitting during training, significantly increasing the required training traces to train the model effectively.
3) Since the GaussiP attention is only an approximation of the Gaussian kernel, the weights, w_js, for the features in the set $\{j : j \notin \mathcal{H}\}$ in Eq. (13) are not exactly zeros, but some small values. When the size of the attack window is very large and the attentions are focused on very small regions (by setting $\hat{\beta}_2$ to a high value), the set $\{j : j \notin \mathcal{H}\}$ becomes very large, increasing the component $\sum_{j \notin \mathcal{H}} w_j h(\mathbf{x}_j)$ in Eq. (13) to a significant value. That, in turn, reduces the effective SNR of the GaussiP attention's outputs, making the training of EstraNet more difficult for the longer attack window sizes.

Next section experimentally verifies the above arguments.

B. Experimental Evaluation of EstraNet on Longer Traces

To experimentally verify the above arguments, we investigate the effectiveness of EstraNet with increasing trace lengths. Towards that goal, we trained and evaluated EstraNet on three datasets used in the previous chapter: ASCAD fixed key (ASCADf), ASCAD random key (ASCADr), and CHES CTF 2020 (CHES20) datasets. These datasets respectively contain $100K$, $250K$, and $62.5K$ features. We conducted three sets of experiments on each of three datasets. In the first set, we selected an attack window of $10K$ from each dataset. In the second set, we trained and evaluated the model on the full-length traces. In the third set, we doubled the length of each trace by concatenating the trace with itself once, and then performed experiments on these double-length traces. The results are summarized in Table I. As in the previous chapter, each experiment is repeated three times. The number of attack traces required to reach guessing entropy 1 (T_{GE1}) in the three training runs of each experiment are shown in the table. From the table, it can be observed that for the attack window of $10K$, EstraNet requires close to 10 attack traces to reach the guessing entropy 1 across all three datasets. When attacks are performed on the full-length traces, EstraNet fails to reach the guessing entropy 1 using as many as $5K$ traces in one dataset (ASCADr dataset), although it performs reasonably well on the remaining two datasets. However, when attacks are performed on the double-length traces, EstraNet is unable to reach the guessing entropy 1 using $5K$ attack traces in any of the three datasets.

Following the arguments of the previous section, we tune $\hat{\beta}_2$ and H hyper-parameters to improve EstraNet's performance on longer traces. Towards that goal, we trained it on the ASCADf dataset using double-length traces, varying combinations of $\hat{\beta}_2$ and H while keeping other hyper-parameters constant. The results are summarized in Table II. From the table, it can be observed that the best performance from EstraNet is achieved with $\hat{\beta}_2 = 300$ and $h = 16$. For this combination of hyper-parameters, the model can reach T_{GE1} significantly below $5K$ in only one out of the three training runs. In the other two training runs, however, it failed to reduce the T_{GE1} below $5K$. This indicates that even with substantial hyper-parameter tuning, EstraNet may not consistently achieve good performance on longer traces

It is worth noting that although EstraNet struggles on longer traces, it performs well with smaller attack windows. However, in practical applications, such as those involving implementations protected by masking or jitter-based countermeasures, selecting sufficiently smaller attack windows from full-length traces may not be feasible. In such scenarios, applying EstraNet could be challenging.

In the next section, we introduce HierNet, a hierarchical TN model for SCA. HierNet inherits EstraNet's effectiveness against masking, random delay, and clock jitter countermeasures while addressing its limitations on longer traces, as shown in Table III).

V. HIERNET: A HIERARCHICAL DL MODEL FOR SCA

The core idea behind HierNet is to process a long trace in segmented manner using EstraNet models and then combine the output of all segments using a top-level DL model (as shown in Figure 1). This approach has several advantages. First, EstraNet may not effectively work on the entire trace but can perform very well on relatively smaller segments. Therefore, applying EstraNet in a segmented manner retains its effectiveness against countermeasures like masking, random delay, and clock jitter while addressing its limitations with increasing trace length. Second, the top-level DL model can

TABLE II
THE T_{GE1} (LOWER IS BETTER) VALUES OBTAINED FROM THE ESTRANET MODEL ON THE ASCADF DATASET WITH DOUBLE-LENGTH TRACES FOR VARIOUS COMBINATIONS OF H AND $\hat{\beta}_2$. FOR EACH HYPER-PARAMETER, THE MODEL HAS BEEN TRAINED THRICE. THE TABLE SHOWS THE T_{GE1} FROM ALL THREE TRAINING RUNS.

	$h = 8$			$h = 12$			$h = 16$		
	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3
$\hat{\beta}_2 = 150$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$
$\hat{\beta}_2 = 300$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	32	$> 5K$	$> 5K$	$> 5K$	7

TABLE III
THE T_{GE1} (LOWER IS BETTER) VALUES OBTAINED FROM THE HIERNET MODEL ON TRACES WITH INCREASING TRACE LENGTH.

Dataset	Full-length Traces			Double-length Traces		
	run 1	run 2	run 3	run 1	run 2	run 3
ASCADf	2	2	2	2	2	2
ASCADr	3	2	3	3	5	5
CHES20	2	2	2	3	3	3

be designed differently from EstraNet, considering different criteria. For instance, the shift-invariance of EstraNet helps mitigate the effects of random delay and clock jitter countermeasures, as demonstrated in the previous chapter. However, when a trace passes through the EstraNet models at the bottom level of HierNet during the forward pass, the effects of those countermeasures get nullified. Therefore, the top-level DL model might exploit this fact by being sensitive to the absolute positions of the informative features in its input sequence. In fact, we design our top-level DL model to be sensitive to the absolute ordering of the input sequence by incorporating absolute positional encoding into the model.

We summarize the approach used in HierNet more precisely. In HierNet, the input trace $\mathbf{t}$ is first segmented into multiple relatively smaller overlapping segments, denoted as $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{m-1}$. Each segment $\mathbf{s}_i$ is independently processed by EstraNet models, producing outputs $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{m-1}$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents the output of EstraNet applied to segment $\mathbf{s}_i$. Next, a top-level DL model (a single layer TN with absolute positional encoding) generates the final output by considering the sequence $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{m-1}\}$ as its input. Several design factors in HierNet are crucial, including the segmentation strategy, the design of the top-level model, and parameter sharing among EstraNet models for different segments. We discuss these factors in detail below.

A. Segmentation

HierNet inherits the robustness of EstraNet against random delay and clock jitter countermeasures by employing EstraNet models to process each trace segment. However, segmenting the input trace in HierNet can potentially compromise this robustness. For example, misalignments in traces may cause some PoI near a segment boundary to oscillate between segments. In one trace, the PoI might belong to one segment, while in another trace, it could be shifted to a different one, resulting in the PoI not being processed in any segment. Such issue can be mitigated by making the segment length reasonably large. Additionally, we ensure significant overlap between consecutive segments. Specifically, if one segment ends at feature t, the next segment starts from feature $t - o$, ensuring that the span from $t-o$ to t belongs to both segments (as illustrated in Figure 1). It's important to note that if the displacements are bounded by length o, each PoI will always remain within one of the two consecutive segments. However, if displacements exceed o, increasing the value of o (a hyper-parameter in HierNet) can enhance HierNet's robustness against larger desynchronizations, albeit at the cost of increased computational overhead.

B. Designing the Top-level DL Model

When designing the top-level DL model, we primarily consider three criteria. First, we aim to ensure that the model's parameterization remains consistent across different trace lengths and segmentations. The number of segments, denoted as m, can vary depending on the trace length and chosen segment length, thereby affecting the input length to the top-level model. Therefore, we select a model whose parameterization is independent of the input length. Second, the model should effectively capture long-distance dependencies to combine leakages from all input elements using a minimal number of layers. Finally, we require the model to easily incorporate absolute positional encoding. Based on these criteria, we selected a single-layer TN model as the top-level DL model, as it meets all three requirements. This model includes a self-attention layer and a position-wise feed-forward layer, as described in Section II-D. We used a softmax self-attention layer and incorporated absolute positional encoding into the self-attention operation. A brief description of this operation is provided in the following section.

1) Self-attention with Absolute Positional Encoding: In the presence of random delay or clock jitter countermeasure, the displacements of PoIs across the traces are likely to remain bounded within a segment. Therefore, they will not affect the order of the input sequence $\{\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}\}$ of the top-level DL model. More precisely, if the leakage related to some intermediate variable appears in the i-th segment for some trace, its position is unlikely to change for the other traces. In other words, the indices of the informative features of the input sequence $\{\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}\}$ will remain the same across all traces. To exploit such phenomenon, we incorporate the absolute positional encoding in the self-attention layer of the top-level TN.

In brief, in vanilla self-attention (the softmax self-attention in Eq. 4), the i-th output feature $\hat{\mathbf{y}}_i$, $i = 0, 1, \ldots, m-1$, is computed as

$$\hat{\mathbf{y}}_i = \sum_{j=0}^{m-1} p_{ij} h(\mathbf{x}_j)$$

with p_{ij}, the attention weight from the i-th feature to the j-th feature, is given as

$$p_{ij} = \text{softmax}\left(\frac{\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle}{\sqrt{d_k}}\right) = \frac{\exp\left(\langle h(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1}\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_l)\rangle/\sqrt{d_k}\right)},$$

where $f(\cdot), g(\cdot)$ and $h(\cdot)$ are three vector valued functions of the input feature $\mathbf{x}_i$ $(i = 0, \ldots, m-1)$, $\langle \cdot, \cdot \rangle$ denotes the dot product between two vectors and d_k be the dimension of the outputs of f (or g). We incorporate the absolute positional encoding into the computation of attention probabilities as follows:

$$p^a_{ij} = (1-\gamma)\cdot \text{softmax}\left(\frac{\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle}{\sqrt{d_k}}\right) + \gamma \cdot \text{softmax}\left(\frac{\langle \mathbf{i}, \mathbf{j}\rangle}{\sqrt{d_k}}\right) \quad (14)$$

$$= (1-\gamma)\frac{\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1}\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_l)\rangle/\sqrt{d_k}\right)} + \gamma\frac{\exp\left(\langle \mathbf{i}, \mathbf{j}\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1}\exp\left(\langle \mathbf{i}, \mathbf{l}\rangle/\sqrt{d_k}\right)}, \quad (15)$$

where $\mathbf{0}, \mathbf{1}, \ldots, \mathbf{m-1}$ are m distinct vectors and $\gamma \in [0, 1]$ is a hyper-parameter. The values of vectors $\mathbf{0}, \mathbf{1}, \ldots, \mathbf{m-1}$ are learned along with other parameters of the model during training. Note that the first component of the R.H.S. of Eq. 14 or 15 computes the attention probability based on the similarity (or inter-dependency) between the two features, $\mathbf{x}_i$ and $\mathbf{x}_j$. On the other hand, the second component computes the probability based on their absolute positions or indices i, j. The hyper-parameters γ controls the relative weights of the two components in the final attention probability p^a_{ij}.

C. Parameter Sharing among the EstraNet Models

To keep the total number of parameters of HierNet small and to prevent the model from overfitting, the parameters of all the EstraNet models are shared among themselves. However, during our initial experiments, we observed that sharing the parameters of the last softmax layer of the EstraNet models resulted in a deceleration of the training convergence for HierNet. We hypothesize that the parameter-sharing of the softmax attention layer was forcing each segment's output towards a unified representation, thereby slowing the training progress. Consequently, we opted not to share the parameters of the last layer of the EstraNet models.

The overall architecture of HierNet is shown in Figure 1. In the next section, we experimentally evaluate HierNet.

VI. Experimental Results

HierNet has been evaluated using three SCA datasets of masked implementations. The evaluation is structured into two distinct parts. In the first part, we conduct the evaluation on long traces with the added random delay effect. In the second part, the evaluation is carried out against the added effect of clock jitter countermeasure on limited-size attack windows. The performance of HierNet is compared with three benchmark DL models.

A. Datasets Used

To carry out the experimental evaluation, we used two datasets of first-order masked AES implementation, namely ASCAD Fixed Key[1] (ASCADf) and ASCAD Random Key[2] (ASCADr) datasets, and a second-order masked implementation of Clyde-128 Tweakable Block Cipher, namely CHES 2020 CTF dataset[3] (CHES20). The first two datasets were collected from an 8-bit ATMega8515 microcontroller, whereas the third dataset was collected from ARM Cortex-M0 microcontroller. Following the previous literature, we target the output of the third sbox of the first round for the first two datasets and the 17th column from the right of the state matrix after the first subbyte operation for the third dataset. The other details of the datasets are given in Appendix B. The statistics of the datasets are summarized in Table IV.

TABLE IV
Statistics of the Datasets

	ASCADf	ASCADr	CHES20
# of Profiling traces	50K	200K	200K
# of Validation traces	5K	10K	10K
# of Attack traces	5K	10K	10K
Trace Length	100K	250K	62.5K
Mask order	1	1	2

B. Benchmark Models

We compare the performance of HierNet with three benchmark models, referred to as EffCNN [19], PolyCNN [12], and LSTMNet [13]. The EffCNN models are CNN models with a flattening layer, whereas the PolyCNN model is a CNN model with a global average-pooling layer. On the other hand, LSTMNet uses softmax attention after a bi-directional LSTM layer to accumulate information spread over a long trace. Further details of the models can be found in Appendix C. Since EstraNet is already found to perform poorly on longer traces in Section IV-B, we do not compare it with HierNet in this section.

C. Details of Hyper-parameters and Training

HierNet uses EstraNet model as the first-level DL model in its architecture. The hyper-parameters related to EstraNet in HierNet are adopted from [16], with a few exceptions. Notably, following the recommendation of [16], the $\hat{\beta}_2$ hyper-parameter of the EstraNet model has been tuned for each dataset. In the top-level DL model, we use absolute positional encoding with γ hyper-parameter set to 1. We set the number of heads of the self-attention layers to 8, though setting it to some other value like 1 also provides similar results. HierNet introduces two new hyper-parameters: segment length and the overlap length between two consecutive segments. Across all datasets,

[1]https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key

[2]https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key

[3]https://ctf.spook.dev/

TABLE V

THE T_{GE1} VALUES (LOWER IS BETTER) FOR DIFFERENT DL MODELS AGAINST THE RANDOM DELAY COUNTERMEASURE. THE ATTACKS HAVE BEEN PERFORMED ON THE FULL-LENGTH TRACES (I.E., TRACES WITH LENGTHS OF $100K$, $250K$, AND $62.5K$ FOR THE ASCADF, ASCADR, AND CHES20 DATASETS, RESPECTIVELY). THE COLUMNS LABELED *Best*, *Med.*, AND *Avg.* RESPECTIVELY DISPLAY THE BEST, MEDIAN, AND AVERAGE T_{GE1} VALUES COMPUTED FROM THREE INDEPENDENT TRAINING TRIALS FOR EACH MODEL. ENTRIES MARKED WITH '-' INDICATE THAT THE AVERAGE VALUE IS UNAVAILABLE BECAUSE THE MODEL FAILED TO REACH GUESSING ENTROPY 1 USING $5K$ ATTACK TRACES IN SOME OF THE TRAINING RUNS.

Dataset	Model	Attack Desync 600			Attack Desync 1000		
		Best	Med.	Avg.	Best	Med.	Avg.
ASCADf	PolyCNN	2626	$> 5K$	–	2102	$> 5K$	–
	EffCNN	2854	$> 5K$	–	2812	$> 5K$	–
	LSTMNet	157	208	1135.7	146	248	972.7
	HierNet	2	2	2.3	2	2	2.0
ASCADr	PolyCNN	5	286	711.7	4	326	445.7
	EffCNN	–	–	–	–	–	–
	LSTMNet	4	7	8.3	5	6	9.0
	HierNet	2	2	2.3	2	3	2.7
CHES20	PolyCNN	689	$> 5K$	–	1946	$> 5K$	–
	EffCNN	–	–	–	–	–	–
	LSTMNet	2	4	3.7	2	2	4.0
	HierNet	1	2	2.0	2	2	2.0

the segment length has been set to $5K$, and the overlap length is set to $2K$.

We train HierNet for $4M$ steps utilizing the Adam optimizer. Following the previous literature [16], [18] of TN for SCA, it has been trained using a cosine decay with a linear warmup learning rate schedule. More precisely, the learning rate is linearly increased from 0 to the maximum value of 2.5e-4 over t_{warmup} training steps. Subsequently, it is gradually reduced to $0.004 \times$ 2.5e-4 over the remaining $4M - t_{warmup}$ steps, following a cosine learning rate schedule. The default value for t_{warmup} is set to $1M$; however, it was observed that, in some instances, the model did not train effectively for this default value. In such cases, t_{warmup} was increased to either of $2M$ or $4M$.

D. Comparison with Benchmark Models in the Presence of Random Delay on Long Traces

In this section, we compare HierNet with the benchmark models in the presence of random delay on long traces. To achieve this, we conduct experiments on full-length traces from the three datasets, which are $100K$, $250K$, and $62.5K$ features long, respectively, with added random delay. The comparisons are carried out in two parts. In the first part, we use all available training traces from the datasets to train the models. In the second part, we compare their performance when fewer training traces are used to train them.

Experimental Setup: to introduce random delay effect, we added random shifts in the traces. In the profiling traces, the random delay has been introduced using a profiling desync of 600. In other words, we shifted each profiling traces by a random displacement in the range $[0, 600)$. The data augmentation has been performed using a desync of 400. Evaluations of the trained models were conducted for two attack desyncs: 600 and 1000. We use the minimum number of attack traces required to reach the guessing entropy 1, denoted by T_{GE1}, as the performance metric for the DL models.

Results with All Training Traces: We begin by comparing HierNet with the benchmark models, utilising all profiling traces from the respective datasets to train the model. Therefore, we utilize $50K$ traces from the ASCADf dataset, and $200K$ traces each from the ASCADr and CHES20 datasets for training. The results are detailed in Table V. From the data presented in the table, it becomes apparent that HierNet exhibits notable performance, requiring approximately 2 to 3 traces to reach the guessing entropy 1 across all datasets. Conversely, the performance of the PolyCNN model is poor across all three datasets, with relatively better performance in the ASCADr dataset. Similarly, the EffCNN model exhibits poor performance across all datasets. Specifically, our investigation revealed that the training of the model on the ASCADr and CHES20 datasets progresses too slowly to reduce the training loss significantly below the level of random guessing, even after 60 hours of training in our experimental environment (as illustrated in Appendix D). In contrast, HierNet models require only 24 to 40 hours for the complete training. The LSTMNet model performs significantly worse on the ASCADf dataset, although its performance is comparable to HierNet on the ASCADr and CHES20 datasets.

Comparison with Reduced Number of Training Traces: We perform further evaluation using a reduced number of profiling traces to train the models. Specifically, we compare HierNet and LSTMNet on the ASCADr and CHES20 datasets, utilizing a random subset of $50K$ profiling traces from the datasets' total of $200K$ profiling traces to train the models. Note that, due to their poor performance on the full training data, PolyCNN and EffCNN models are not included in this comparison. Additionally, since LSTMNet performs poorly compared to HierNet with the full training data of the ASCADf dataset, we do not compare them again using a lesser number of training traces from that dataset. The results are presented in Table VI. From the table, it is evident that HierNet models can still reach the guessing entropy 1 using only three or

TABLE VI
THE T_{GE1} VALUES (LOWER IS BETTER) FOR THE LSTMNET, AND HIERNET MODELS WHEN THE MODELS ARE TRAINED USING $50K$ PROFILING TRACES. THE COLUMNS LABELED *Best*, *Med.*, AND *Avg.* RESPECTIVELY DISPLAY THE BEST, MEDIAN, AND AVERAGE T_{GE1} VALUES COMPUTED FROM THREE INDEPENDENT TRAINING TRIALS FOR EACH MODEL. ENTRIES MARKED WITH '-' INDICATE THAT THE AVERAGE VALUE IS UNAVAILABLE BECAUSE THE MODEL FAILED TO REACH THE GUESSING ENTROPY 1 USING $5K$ ATTACK TRACES IN SOME OF THE TRAINING RUNS.

Dataset	Model	Attack Desync 600			Attack Desync 1000		
		Best	Med.	Avg.	Best	Med.	Avg.
ASCADr	LSTMNet	2987	4893	–	2615	2812	–
	HierNet	2	2	2.3	2	3	2.7
CHES20	LSTMNet	2	$> 5K$	–	4	$> 5K$	–
	HierNet	1	2	1.7	2	2	2.0

fewer attack traces in both datasets. In contrast, the LSTMNet models necessitate over $2K$ attack traces on the ASCADr dataset to reach the guessing entropy 1. On the CHES20 dataset, though the LSTMNet model has reached the guessing entropy 1 using fewer than five attack traces in one training run, it fails to reach the same using less than $5K$ traces in the remaining two runs.

In summary, PolyCNN and EffCNN exhibit poor performance on longer traces. While LSTMNet shows more resilience to increasing trace length, achieving comparable performance to HierNet may require significantly more training data. These results clearly demonstrate HierNet's advantage over the three benchmark models in handling longer traces. The following section further compares HierNet with the benchmark models against the clock jitter countermeasure.

E. Comparison with Benchmark Models in the Presence of Clock Jitter Effect

An advantage in attacking countermeasures involving global trace desynchronizations, such as in many random delay countermeasures, lies in the ease with which these effects can be simulated during the training phase. Consequently, data augmentation can be readily applied, increasing the effective training data size and enhancing the robustness of the DL models to such misalignments. In contrast, the clock jitter countermeasure introduces small, local desynchronization throughout the full traces, leading to significant variation in distances between peaks within and across traces. While various data augmentation techniques are available for simulating the clock jitter effect [10], these approaches result in significant information loss due to the removal of informative features during the processing. Consequently, relying solely on data augmentation during training of the DL models to perform effectively against the clock jitter countermeasure may be infeasible.

In this section, we assess HierNet's performance against clock jitter by conducting attacks after introducing the clock jitter effect into the traces. It is crucial to note that prior studies, such as [20], have demonstrated successful attacks against the clock jitter effect under the assumption of a stronger adversary who has access to both the clean version and the noisy version of the traces. However, as in [10], [16], we consider a weaker adversary who lacks the access to the clean version of the noisy traces.

Generation of Clock Jitter Effect: We employed the methodology outlined in [10], [16], [20] to introduce the clock jitter effect into the traces. Specifically, we generated a noisy trace for each trace within the dataset using the following procedure. For every feature in the original trace, we performed one of the following three actions with equal probabilities: a) excluding the feature from the noisy trace, b) including the feature in the noisy trace, and c) including the feature along with an additional feature, where the additional feature is the average of the current and following features in the source trace. Notably, this method aligns with the application of the clock jitter effect using the algorithm proposed by Wu et al. [20], with the parameter `clock_jitters_level` set to 1. It is pertinent to mention that Hajra et al. [16] also employed the same algorithm for introducing the clock jitter effect. However, prior to applying the clock jitter effect, they doubled the trace length by duplicating each feature in the source trace twice to avoid the loss of any informative features. In contrast, we do not perform such preprocessing. Therefore, our attack setup is more challenging compared to that of [16].

TABLE VII
SELECTED ATTACK WINDOWS FOR THE EXPERIMENTS WITH ADDED CLOCK JITTER EFFECT.

	Dataset		
	ASCADf	ASCADr	CHES20
Attack Window	$[40K, 50K]$	$[78K, 88K]$	$[46K, 56K]$

Increasing the Number of Traces in the Training Data: In Section VI-D, it was observed that the performance of the DL models improves with the increase of the number of training traces, although HierNet achieves its optimal performance with a significantly smaller number of training traces. To investigate whether the DL models can also be robust to clock jitter by using more training traces, we conduct experiments with an increased number of training traces. More precisely, the number of training traces for a dataset was increased to x-fold as follows: for each trace in the clean dataset, x noisy traces were added to the noisy dataset. Each of the x noisy traces was independently created by introducing clock jitter effects to the corresponding clean trace. Since the process adds x noisy traces in the noisy dataset for each trace in the clean dataset, it increases the size of the noisy dataset x-fold. Note that increasing the number of training traces in this manner does not increase the information content of the dataset, as multiple traces are generated from the same clean

TABLE VIII
THE T_{GE1} VALUES (LESSER IS BETTER) FOR DIFFERENT DL MODELS ON THE THREE DATASETS WITH ADDED CLOCK JITTER EFFECT. WE HAVE PERFORMED THE ATTACKS ON A SELECTED ATTACK WINDOW OF SIZE $10K$ FOR THESE EXPERIMENTS. AS BEFORE, EACH MODEL HAS BEEN INDEPENDENTLY TRAINED THREE TIMES. WE HAVE SHOWN THE MEDIAN OF THE THREE RESULTS IN THE TABLE.

Dataset	Training Data Size	Model			
		PolyCNN	EffCNN	LSTM	HierNet
ASCDf	$100K$	$> 5K$	$> 5K$	$> 5K$	55
	$200K$	544	384	$> 5K$	23
	$400K$	442	60	197	22
	$800K$	262	34	160	17
ASCDr	$200K$	$> 5K$	$> 5K$	$> 5K$	20
	$400K$	$> 5K$	88	$> 5K$	16
	$800K$	25	40	23	13
CHS20	$200K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$
	$400K$	$> 5K$	$> 5K$	$> 5K$	10
	$800K$	$> 5K$	$> 5K$	$> 5K$	12

traces. However, creating multiple traces by introducing clock jitter effects independently to the same clean trace should help the DL models become robust to the variations introduced by the clock jitter effect.

Selection of Attack Window: Due to constraints in our computational resources, we could not conduct attacks on the full-length traces. Instead, the attacks were carried out on an attack window of size $10K$. It should be noted that, earlier, Hajra et al. [16] also conducted attacks on selected attack windows with a length of $10K$. For each dataset, they chose the attack windows so that the windows contain the maximum number of high SNR features. We adopt the same attack windows for the experiments in this section. The details of these attack windows are outlined in Table VII. It is important to note that the selection of the attack windows assumes the feasibility of SNR-based window selection, which may not hold in many practical attack scenarios. However, we emphasize that window selection was performed to minimize compute usage, and we hope that the conclusions drawn from these experiments will also apply to full-length traces, albeit with a different scaling of the number of training traces.

Training Details: All benchmark models have been trained as outlined in Appendix C. The HierNet models were trained according to the procedure detailed in Section VI-C. As in Section VI-D, data augmentation was applied to all models during training. Specifically, each training trace was shifted by a random displacement within the range $[0, 200)$ on-the-fly during the training.

Results: The attack results of the DL models under different numbers of training traces are presented in Table VIII. Similar to Section VI-D, the table uses T_{GE1} – the minimum number of attack traces required to reach the guessing entropy 1 – as the metric for assessing the effectiveness of the models. In these experiments also, each DL model was independently trained three times, and the table displays the median value of the three results.

In the case of the ASCADr dataset, the results of the DL models are presented for three distinct sizes of the training data: $200K$, $400K$, and $800K$. As evident from the table, with $200K$ training traces, HierNet achieves guessing entropy 1 with approximately 20 attack traces, whereas all other models fail to reach guessing entropy 1 even with as many as $5K$ attack traces. As the number of training traces increases, the other DL models can reach the guessing entropy 1. However, they require four times more training traces to achieve performance comparable to HierNet.

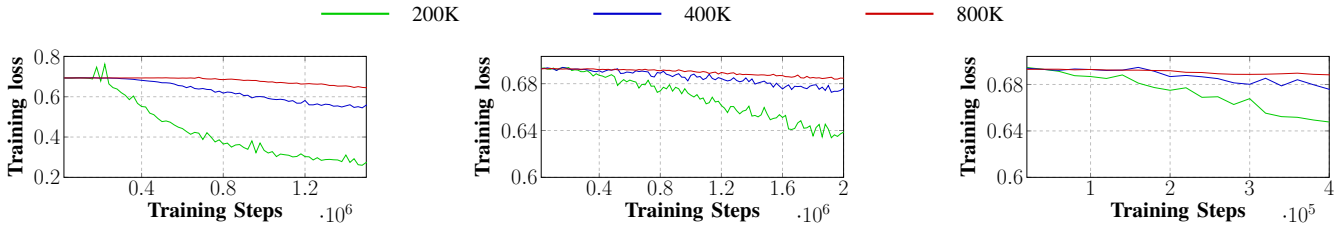
For the ASCADf dataset, the presented table displays attack results across four distinct training data sizes: $100K$, $200K$, $400K$, and $800K$. With $100K$ training traces, only HierNet can reach the guessing entropy 1 with fewer than $5K$ attack traces. As the number of training traces increases to $800K$, the performance of EffCNN approaches that of HierNet. Conversely, the PolyCNN and LSTMNet models fail to reach performance comparable to HierNet, even with eight times more training traces.

The CHES20 dataset poses a challenging scenario for the CNN and LSTM-based DL models, though HierNet perform well. Specifically, HierNet achieves its optimal performance (reaching the guessing entropy 1 with approximately 10 attack traces) with only $400K$ training traces. However, the CNN and LSTM-based models fail to reach the same even using $5K$ attack traces. Furthermore, even when the models are trained with more training traces, their performance does not improve. In fact, in our experiments, we observe that as the number of training traces increases, these models become challenging to train, as indicated in Figure 3. We speculate that these models struggle to learn meaningful information from the traces during the training. Therefore, when trained with a smaller number of training traces, the models tend to overfit the noise, leading to a relatively faster decrease in training loss. However, more training traces mitigate overfitting, leading to a reduced convergence rate in the training loss during training.

In conclusion, the results suggest that HierNet demonstrates robust performance against clock jitter countermeasures across all three datasets. In contrast, the benchmark models might fail to perform comparably to HierNet even when they are trained using eight times more training traces. The findings also reveal scenarios where HierNet excels while all the benchmark models fail to be trained effectively even with more training data, underscoring the need for further research to improve their performance in such contexts.

F. Training Time

In our experimental environment (Google Colab TPU), the training time of LSTMNet can be more than five times slower for traces with lengths around $10K$ features, while it is about two times slower for longer traces (around $100K$ features). The training time of the PolyCNN and EstraNet models varies between 3 to 12 times compared to HierNet on shorter traces (around $10K$ features), but their training becomes much slower on longer traces (around $100K$ features). In some cases, the training loss of these models did not decrease significantly below the level of random guessing, even after training for more than three times the total duration of HierNet. While an improved learning rate schedule might accelerate their training on longer traces, we still expect them to be slower than HierNet.



(a) Training loss of the LSTMNet model (b) Training loss of the PolyCNN model (c) Training loss of the EffCNN model

Fig. 3. Improvement in the training loss of the LSTMNet, PolyCNN, and EffCNN models during their training on the CHES20 dataset with the added clock jitter effect. The green, blue, and red lines, respectively, show the progress of the training loss for the models' training with $200K$, $400K$, and $800K$ training traces. It can be seen in the figures that the training loss of all three models does not reduce much while trained with $800K$ training traces, whereas it decreases faster while the models are trained with smaller $200K$ and $400K$ training traces. The behavior indicates that the models are overfitting to the noise while trained with fewer training traces. Though the overfitting is prevented by increasing the number of training traces to $800K$, the lack of enough improvement in the models's training loss indicates that the models are unable to extract useful information from the traces.

While comparing the training time of HierNet with EstraNet, it's worth noting that due to the overlapping segmentation in HierNet, its per-step training time is slightly higher than EstraNet's. Specifically, in our experiments, HierNet's per-step training time was between 1 and 1.5 times that of EstraNet across the three datasets. However, HierNet requires fewer training steps to converge, making the total training time comparable to EstraNet.

VII. Discussions and Future Works

a) Limitations of our Analysis: In the experiments involving random delay (Section VI-D), some DL models, including HierNet, exhibit continued improvement with their training's progress on the ASCADr and CHES20 datasets. However, the rate of improvement gradually diminishes over time. Due to computational constraints, we terminated the training when the rate of improvement became very slow. Therefore, there is a possibility of achieving slightly improved results for some DL models, including HierNet, through better training. However, we expect that the results would not deviate significantly from those reported in Table V. We also want to point out that the CNN and LSTM-based models have been trained for a duration more than two or three times longer than the TN-based models.

b) HierNet against Random Delay Interrupt Countermeasure: The Random Delay Interrupt (RDI) countermeasure inserts random delays into multiple intermediate positions of traces, causing drastic variation in the distances between the PoIs across the traces. Since HierNet segments the traces into multiple segments and it is shift-invariant in each segment, it assumes that the intermediate distances between the PoIs within each segment remain approximately constant. Such an assumption may not be true in the presence of RDI. Therefore, HierNet may not be directly applicable against RDI countermeasures. Notably, the study by Wu et al. [20] on synthetic RDI datasets demonstrated that DL models find RDI countermeasures challenging to overcome. Alternatively, various previously proposed pre-processing techniques [20]–[22] can be used to mitigate the RDI effects from the traces before training the DL models. It should also be noted that these pre-processing techniques may not completely eliminate the desynchronizations introduced by RDI. The pre-processed traces may still exhibit misalignment, albeit to a lesser extent. Therefore, HierNet, which has demonstrated its effectiveness against traces with bounded desynchronizations, could be a suitable choice for attacking the pre-processed traces. Investigating this aspect could be a future research direction.

c) HierNet against Shuffling Countermeasure: In the shuffling countermeasure [11], the order of execution of multiple parallel operations of a cryptographic algorithm (e.g., sub-byte operations of different bytes in an AES encryption) is randomly shuffled, resulting in different permutations in the positions of leakages of the respective operations across traces. This shuffling might cause the relative distances between the PoIs to vary significantly, posing a challenge to the usefulness of DL models like EstraNet on such traces. However, HierNet, which segments entire traces into multiple segments and processes each segment independently, can still be employed against shuffling countermeasures. To illustrate this approach, consider targeting the sub-byte operations of the first round in an encryption scheme. If the approximate length of each sub-byte operation is known, the traces can be segmented accordingly, and HierNet can be applied based on that segmentation. However, utilizing HierNet in this manner necessitates knowledge of the approximate length of the operations. Such knowledge might be deduced when the source code of the implementation is available, following the procedures proposed in [23].

VIII. Conclusions

DL-based SCA has shown significant success in overcoming various countermeasures. However, the proper selection of the attack window is critical for its effectiveness. Consequently, several studies have explored methods to enhance the performance of DL models on longer attack windows or long raw traces, thereby reducing the dependency on the attack window selection. Despite these efforts, the performance of most of the existing DL models deteriorates in the presence of jitter-based countermeasures or as trace length increases. Therefore, developing DL models that perform well on long raw traces while remaining robust to the jitter-based countermeasures remains an important yet challenging task.

EstraNet, a TN-based model, was recently introduced to perform SCA on long traces. The model has demonstrated state-of-the-art performance against various combinations of masking, random delay, and clock jitter countermeasures. In

this work, we closely analyze EstraNet and reveal that its performance gradually deteriorates as trace lengths increase. To address this limitation, we propose HierNet, a two-level hierarchical DL model for SCA. At the bottom level, HierNet employs EstraNet models to process smaller segments of the traces. At the top level, another TN model is used to integrate information from the outputs of the EstraNet models. This top-level TN model incorporates a novel self-attention layer with absolute positional encoding.

Extensive evaluations of HierNet have been conducted against various combinations of masking, random delay, and clock jitter countermeasures using several SCA datasets. The performance of HierNet has been compared with three existing benchmark DL models. The results indicate that HierNet outperforms the benchmark models on longer traces or against clock jitter countermeasure, showcasing its ability to reach the guessing entropy 1 using fewer than or close to 10 attack traces while the benchmark models are unable to reach the same using as many as $5K$ attack traces. Additionally, HierNet demonstrates significantly better performance on low training data scenarios. In conclusion, the success of HierNet establishes it as a promising method for SCA.

REFERENCES

[1] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *CRYPTO '96, California, USA,*, ser. LNCS, N. Koblitz, Ed., vol. 1109, 1996, pp. 104–113.

[2] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO, USA, 1999*, ser. LNCS, vol. 1666. Springer, 1999, pp. 388–397.

[3] J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): Measures and counter-measures for smart cards," in *E-smart 2001, France, Proceedings*, ser. LNCS, I. Attali and T. P. Jensen, Eds., vol. 2140. Springer, 2001, pp. 200–210.

[4] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *CHES, USA, 2002*, ser. LNCS, B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, 2002, pp. 13–28.

[5] W. Schindler, K. Lemke, and C. Paar, "A stochastic model for differential side channel cryptanalysis," in *CHES, UK, 2005*, ser. LNCS, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 30–46.

[6] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *TCHES*, vol. 2019, no. 1, pp. 209–237, 2019.

[7] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *SPACE, India, 2016*, ser. LNCS, vol. 10076. Springer, 2016, pp. 3–26.

[8] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *CRYPTO, USA, 1999*, ser. LNCS, vol. 1666. Springer, 1999, pp. 398–412.

[9] J. Coron and I. Kizhvatov, "An efficient method for random delay generation in embedded software," in *CHES, Switzerland, 2009*, ser. LNCS, vol. 5747. Springer, 2009, pp. 156–170.

[10] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing," in *CHES, Taiwan, 2017*, ser. LNCS, vol. 10529. Springer, 2017, pp. 45–68.

[11] C. Herbst, E. Oswald, and S. Mangard, "An AES smart card implementation resistant to power analysis attacks," in *ACNS, Singapore, 2006*, ser. LNCS, J. Zhou, M. Yung, and F. Bao, Eds., vol. 3989, 2006, pp. 239–252.

[12] L. Masure, N. Belleville, E. Cagli, M. Cornelie, D. Couroussé, C. Dumas, and L. Maingault, "Deep learning side-channel analysis on large-scale traces - A case study on a polymorphic AES," in *ESORICS, UK, 2020*, ser. LNCS, vol. 12308. Springer, 2020, pp. 440–460.

[13] X. Lu, C. Zhang, P. Cao, D. Gu, and H. Lu, "Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks," *TCHES*, vol. 2021, no. 3, pp. 235–274, 2021.

[14] G. Perin, L. Wu, and S. Picek, "Exploring feature selection scenarios for deep learning-based side-channel analysis," *Cryptology ePrint Archive*, 2021.

[15] E. Bursztein, L. Invernizzi, K. Král, D. Moghimi, J. M. Picod, and M. Zhang, "Generic attacks against cryptographic hardware through long-range deep learning," *CoRR*, vol. abs/2306.07249, 2023.

[16] S. Hajra, S. Chowdhury, and D. Mukhopadhyay, "EstraNet: An efficient shift-invariant transformer network for side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 1, p. 336–374, 2023.

[17] E. Bursztein, L. Invernizzi, K. Král, D. Moghimi, J. Picod, and M. Zhang, "Generalized power attacks against crypto hardware using long-range deep learning," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 3, pp. 472–499, 2024.

[18] S. Hajra, S. Saha, M. Alam, and D. Mukhopadhyay, "TransNet: Shift invariant transformer network for side channel analysis," in *AFRICACRYPT 2022, Fes, Morocco, 2022, Proceedings*, ser. LNCS, L. Batina and J. Daemen, Eds. Springer Nature Switzerland, 2022, pp. 371–396.

[19] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli, "Methodology for efficient CNN architectures in profiling attacks," *TCHES*, vol. 2020, no. 1, pp. 1–36, 2020.

[20] L. Wu and S. Picek, "Remove some noise: On pre-processing of side-channel measurements with autoencoders," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 389–415, 2020.

[21] F. Durvaux, M. Renauld, F. Standaert, L. van Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon, "Efficient removal of random delays from embedded software implementations using hidden markov models," in *CARDIS, Austria*, ser. Lecture Notes in Computer Science, S. Mangard, Ed., vol. 7771. Springer, 2012, pp. 123–140.

[22] G. Chiari, D. Galli, F. Lattari, M. Matteucci, and D. Zoni, "A deep-learning technique to locate cryptographic operations in side-channel traces," 2024.

[23] L. Masure and R. Strullu, "Side channel analysis against the ANSSI's protected AES implementation on ARM," *IACR Cryptol. ePrint Arch.*, p. 592, 2021.

[24] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ASCAD database," *J. Cryptogr. Eng.*, vol. 10, no. 2, pp. 163–188, 2020.

[25] D. Bellizia, F. Berti, O. Bronchain, G. Cassiers, S. Duval, C. Guo, G. Leander, G. Leurent, I. Levi, C. Momin, O. Pereira, T. Peters, F. Standaert, B. Udvarhelyi, and F. Wiemer, "Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher," *IACR Trans. Symmetric Cryptol.*, vol. 2020, no. S1, pp. 295–349, 2020.

APPENDIX A
RELATION BETWEEN $|\mathcal{H}|$ AND n

Let us formally define $\mathcal{H}$ (introduced in Section IV-A) to be the set $\{j : k_{GPA}(i-j) > \delta\}$ for some $\delta \in (0,1)$ and some $i \in \{0,1,\ldots,n-1\}$. Then

$$\begin{aligned}
|\mathcal{H}| &= |\{j : k_{GPA}(i-j) > \delta\}| \\
&= \left|\left\{j : \hat{\beta}_2 s_p (i/n - j/n - c_p)||\mathrm{W}_p\mathbf{p}||_2 < \sqrt{2\log\,(1/\delta)}\right\}\right| \\
&= \left|\left\{j : (i/n - j/n - c_p) < \frac{\sqrt{2\log\,(1/\delta)}}{\hat{\beta}_2 s_p||\mathrm{W}_p\mathbf{p}||_2}\right\}\right| \\
&= |\{j : (i/n - j/n - c_p) < C\}|
\end{aligned}$$

where $C = \frac{\sqrt{2\log\,(1/\delta)}}{\hat{\beta}_2 s_p||\mathrm{W}_p\mathbf{p}||_2}$. The parameters c_p and s_p are independent of n or $\hat{\beta}_2$ and $||\mathrm{W}_p\mathbf{p}||_2$ is constant. Therefore, when $\hat{\beta}_2$ remains the same, $|\mathcal{H}|$ increases almost linearly with n.

APPENDIX B
DETAILS OF DATASET

This section provides a detailed description of the datasets used.

ASCAD Fixed Key (ASCADf): The ASCAD fixed key dataset, referred to as the ASCADf dataset, is a collection of EM traces collected from a masked implementation of AES executed on the 8-bit microcontroller ATMega8515. This dataset comprises a total of $60K$ traces. Out of these, $50K$ traces were used for training, while two sets of $5K$ traces each were used for validation and testing. Each trace in the dataset is $100K$ features long. In line with earlier literature [13], [16], [19], [24], we aim to attack the third key byte of the first-round key and use the identity leakage model.

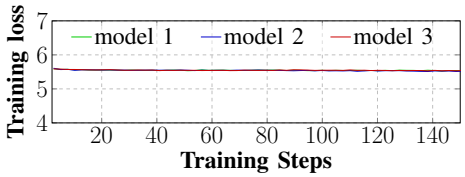
ASCAD Random Key (ASCADr): Similar to the ASCADf dataset, the ASCAD random key dataset, referred to as the ASCADr dataset, also corresponds to a masked implementation of AES executed on an 8-bit ATMega8515 microcontroller. However, for the ASCADf dataset, the secret key remains constant across all profiling traces, whereas for the ASCADr dataset, it varies randomly. The dataset consists of $200K$ profiling traces and $100K$ attack traces. Unless stated otherwise, all profiling traces have been utilized for training the DL models, while two disjoint random samples of $10K$ attack traces were used for validation and testing, respectively. Each trace in the dataset spans $250K$ features. To reduce memory usage, the traces were compressed to a length of $125K$ by replacing each consecutive pair of features of the original traces with their average. Therefore, the attacks were conducted on the compressed traces of length $125K$. Similar to the ASCADf dataset, we attack the third key byte of the first round key using the identity leakage model.

CHES 2020 CTF SW3 (CHES20): The CHES 2020 CTF presents a SCA challenge targeting the masked implementation of the Clyde-128 Tweakable Block Cipher (TBC) used in the Spook AEAD encryption scheme [25]. The challenge encompasses various datasets derived from distinct implementations of the cipher. In this study, we utilize the dataset corresponding to a second-order masked implementation executed on an ARM Cortex-M0 microcontroller. The dataset comprises $200K$ profiling traces and $500K$ attack traces. Unless stated otherwise, all profiling traces are employed for training the DL models, while for validation and testing, two separate and randomly selected sets of attack traces, each containing $10K$ traces, are utilized. It should be noted that Clyde-128 employs a (4×32)-bit state, with 4 representing the size of the non-linear S-box and 32 indicating the size of the linear L-box. Consistent with the approach in [16], we attack the four bits of the 17th column (from the right) of the state after the first round sbox operation.

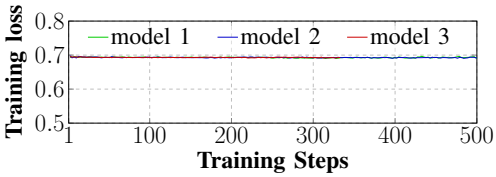
Appendix C
Details of the Benchmark Models

This section provides short descriptions of the benchmark DL models.

LSTMNet: LSTMNet [13] is a family of DL models that incorporate LSTM layers to aggregate information from multiple Points of Interest (PoIs). These models exhibit linear scalability with respect to input length, making those well-suited for processing very long traces. In their work [13], Lu et al. demonstrated LSTMNet's proficiency in attacking available



(a) The ASCADr Dataset.



(b) The CHES20 Dataset.

Fig. 4. Training loss of three independent training of EffCNN model on the ASCADr and CHES20 datasets.

full-length raw traces. For the ASCADf and ASCADr datasets, we trained the respective models available in their online repository[4] to conduct the attack. Since no model is available for the CHES20 dataset, following [16], we used the model designed for the ASCADr dataset to train on this dataset. The models were trained for a maximum of $4K$ epochs, employing the Adam optimizer with a learning rate of 1e-4.

PolyCNN: The PolyCNN model [12], introduced by Masure et al. [12], is designed for attacking implementations protected by code polymorphism. The model is comprised of multiple convolutional blocks succeeded by a global average-pooling layer. [12] has demonstrated that the model can perform very well on very long traces. Consequently, we employ PolyCNN as a benchmark model in our study. The model underwent training for a maximum of $5K$ epochs, utilizing the Adam optimizer with a learning rate of 1e-5.

EffCNN: In their work [19], Zaid et al. proposed a methodology for constructing CNN-based models for SCA. They successfully demonstrated the applicability of their methodology in creating compact CNN models suitable for both synchronized and desynchronized trace scenarios. However, it is worth noting that when applying their methodology to very long traces with large trace desynchronizations, the resultant models become large and exhibit slow training speeds. Despite these considerations, we utilize models constructed through their methodology as benchmark models in our study. It is important to emphasize that distinct models have been constructed based on their methodology for different datasets. The training process involved a maximum of $2K$ epochs, utilizing the Adam optimizer with a learning rate of 2.5e-5.

Appendix D
Plots of Training Loss of EffCNN Model on the ASCADr and CHES20 Datasets

In Figure 4, we illustrate the EffCNN model's training loss convergence in the experiments with the ASCADr and CHES20 datasets. Figure 4a displays the training loss for the ASCADr dataset, indicating negligible improvement even

[4]https://github.com/lxj-sjtu/TCHES2021_Pay_attention_to_the_raw_traces

after 150 epochs. Notably, training for 150 epochs on this dataset took approximately 50 hours, whereas HierNet models achieved full convergence within 24 to 40 hours. Similarly, Figure 4b shows that even after 500 epochs (equivalent to 80 hours of training), the training loss did not significantly decrease compared to random guessing.