EstraNet: An Efficient Shift-Invariant Transformer Network for Side-Channel Analysis*

Suvadeep Hajra[1], Siddhartha Chowdhury[1] and Debdeep Mukhopadhyay[1,2]

[1] Indian Institute of Technology Kharagpur, Kharagpur, India {suvadeep.hajra,siddhartha.chowdhury27}@kgpian.iitkgp.ac.in,debdeep@cse.iitkgp.ac.in
[2] New York University, Abu Dhabi, UAE debdeep@nyu.edu

Abstract. Deep Learning (DL) based Side-Channel Analysis (SCA) has been extremely popular recently. DL-based SCA can easily break implementations protected by masking countermeasures. DL-based SCA has also been highly successful against implementations protected by various trace desynchronization-based countermeasures like random delay, clock jitter and shuffling. Over the years, many DL models have been explored to perform SCA. Recently, Transformer Network (TN) based model has also been introduced for SCA. Though the previously introduced TN-based model is successful against implementations jointly protected by masking and random delay countermeasures, it is not scalable to long traces (having a length greater than a few thousand) due to its quadratic time and memory complexity. This work proposes a novel shift-invariant TN-based model with linear time and memory complexity. The contributions of the work are two-fold. First, we introduce a novel TN-based model called EstraNet for SCA. EstraNet has linear time and memory complexity in trace length, significantly improving over the previously proposed TN-based model's quadratic time and memory cost. EstraNet is also shift-invariant, making it highly effective against countermeasures like random delay and clock jitter. Secondly, we evaluated EstraNet on three SCA datasets of masked implementations with random delay and clock jitter effect. Our experimental results show that EstraNet significantly outperforms several benchmark models, demonstrating up to an order of magnitude reduction in the number of attack traces required to reach guessing entropy 1.

Keywords: SCA · Transformer Network · Shift-invariance

1 Introduction

The power consumption or electromagnetic emission of a CMOS device depends on the data being processed within the device. Side-Channel Analysis (SCA) exploits this dependency to recover the secret key of a cryptographic device. Non-profiling SCA such as differential power analysis (DPA) [KJJ99], correlation power analysis (CPA) [BCO04], and mutual information analysis [GBTP08] perform attacks on a target device without prior characterization of its leakage behavior. In contrast, profiling SCA assumes that the adversary possesses a clone of the target device under his control. The attacker utilizes the clone device to characterize the leakage behavior, learning an approximate leakage model for the target device. This learned model is then used to perform actual attacks. Examples of profiling attacks include template attack [CRR02] and stochastic attack [SLP05].

Profiling attacks have received significant attention in the SCA literature due to their ability to provide worst-case evaluations of cryptographic devices against SCA. Classical

*This work has been accepted in IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), volume 2024, number 1.

profiling attacks, such as the template attack, heavily rely on the selection of informative sample points, commonly referred to as Points-of-Interests (POIs), before conducting the attack. However, selecting POIs becomes challenging for implementations protected by countermeasures like masking [CG00], random delay, and clock jitter [WP20]. In recent years, Deep Learning (DL) has emerged as a highly successful approach for profiling SCA [MZ13, MPP16]. DL-based SCA (DLSCA) demonstrates remarkable performance even without precise POI selection. Instead, approximately selecting an attack window containing some POIs is sufficient for DLSCA to yield favorable results. Recent studies [LZC$^+$21, PWP21, KP22] have shown that the effectiveness of DL models can be significantly enhanced by conducting attacks on larger attack windows or full-length traces. Utilizing a larger attack window increases the likelihood of including more POIs, thus improving the performance by combining their leakages. Additionally, it is relatively easier to find a larger attack window containing some POIs. This work aims to improve the performance of DLSCA when applied to longer traces or larger attack windows.

Various deep learning (DL) models, including the Feed Forward Network (FFN) [MZ13, MHM13, MPP16], Convolutional Neural Network (CNN) [MPP16, CDP17, BPS$^+$20, ZS20, PSK$^+$18], and Recurrent Neural Network (RNN) [MPP16, Mag19, LZC$^+$21], have been extensively investigated for profiling SCA. Recently, a Transformer Network (TN)-based model called TransNet was introduced in [HSAM22] for profiling SCA, demonstrating successful attacks against implementations protected by masking and random delay countermeasures. Since TNs are better at capturing the dependency among distant POIs than other DL models like CNNs or RNNs [VSP$^+$17], they are a natural choice for those SCAs which requires combining leakages from distant POIs. Furthermore, it was shown in [HSAM22] that TN could be made shift-invariant, making TransNet highly effective against large trace desynchronizations. However, the quadratic time and memory complexity of TransNet with respect to the trace length makes it impractical for traces having lengths greater than a few thousand. This study introduces a novel TN-based model called EstraNet, which exhibits linear time and memory complexity. The linear complexity enables EstraNet to scale effectively to traces with lengths exceeding $10K$. Additionally, EstraNet is shift-invariant, making it robust against countermeasures like random delay and clock jitter. We conduct evaluations of EstraNet on three large-scale datasets, demonstrating comparable or significantly better performance compared to three CNN and LSTM-based benchmark models. More precisely, the contributions of the work are as follows:

1. We propose EstraNet[1], a novel TN-based model for SCA. EstraNet has a linear time and memory complexity in terms of trace length. EstraNet is also shift-invariant, making it effective against countermeasures like random delay and clock jitter. Our contributions in EstraNet architecture are two-fold:

    (a) The self-attention layer is the major component of a TN. We propose a novel self-attention layer, called GaussiP attention, with linear time and memory complexity. The attention layer incorporates relative positional encoding making it suitable for the shift-invariance of EstraNet.

    (b) Due to the incompatibility of conventional normalization techniques, such as batch normalization and layer normalization in TNs for SCA, we introduce a novel normalization approach called layer-centering in EstraNet.

2. We conducted experimental evaluations of EstraNet on three datasets of masked implementations. The main observations of our experiments are as follows:

    (a) We compared the performance of EstraNet with three CNN and LSTM-based benchmark models on the datasets. Additionally, we introduced random displacements in the traces to assess the models' robustness against random delay.

[1] The Tensorflow implementation can be available at https://github.com/suvadeep-iitb/EstraNet.git

The results indicate that EstraNet performs similarly or significantly better than the benchmark models. More precisely, it requires upto 90% less attack traces to reach the guessing entropy 1 compared to the benchmark models.

(b) We conducted additional comparisons between EstraNet and the benchmark models on the datasets after adding clock jitter effect. The experiments demonstrate that EstraNet can reach the guessing entropy 1 using fewer than 100 attack traces most of the time, while the benchmark models struggle to reach the same using as many as $5K$ traces. Even in cases where the benchmark models performs relatively well, they still required an order of magnitude more attack traces compared to EstraNet.

(c) We conducted several studies to assess the influence of several hyperparameters on the performance of EstraNet. Additionally, we performed an ablation study to analyze the impact of different design choices and training setup.

The organization of the paper is as follows. In Section 2, we introduce the necessary notations and SCA background. Section 3 briefly describes vanilla TN along with its prime component, self-attention operation. The section also briefly outlines an approach to make the self-attention and, thus, the TN models linear in time and memory complexity. In Section 4, we propose a novel self-attention operation with relative positional encoding and linear time and memory cost. Section 5 introduces the overall architecture of EstraNet. In Section 6, we provide the experimental results. Section 7 discusses the limitations of EstraNet and depicts some future work directions. Finally, in Section 8, we conclude the work.

2 Preliminaries

2.1 Notations

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

2.2 Side-Channel Analysis

The power consumption or electromagnetic (EM) emission of a semiconductor device depends on the values being manipulated within the device. SCA exploits this behavior of semiconductor devices to gain information about some intermediate sensitive variables of a cryptographic implementation and, hence, the device's secret key. More precisely, in an SCA, an adversary takes control of the target device, also known as Device Under Test (DUT), and collects power or EM measurements, referred to as traces, by executing the encryption (resp. decryption) algorithm multiple times with different plaintexts (resp. ciphertexts). Then the adversary performs a statistical test to infer the device's secret key.

SCA can be of two types: profiling SCA and non-profiling SCA. In a profiling SCA, the adversary is assumed to possess a clone of the DUT under his control. Using the clone device, he can build a profile of the DUT's power consumption or EM emission characteristic and use that profile for performing the actual attack. On the other hand, in a non-profiling SCA, the adversary does not possess any clone device and, thus, cannot build any power/EM profile of DUT. Instead, he tries to recover the secret key from the traces of DUT only. In this paper, we consider profiling SCA only.

2.3 Profiling SCA

A profiling SCA is performed in two phases. In the first phase, known as the profiling phase, the adversary sets some known key in the clone device and collects a large number of traces by executing the encryption (resp. decryption) operations on some known plaintexts (resp. ciphertexts) using the device. For each trace, the adversary computes the value of an intermediate secret variable $Z = F(X, K)$, where X represents a component of the random plaintext (or ciphertext), K represents a component of the (possibly random) key, $F(\cdot, \cdot)$ is a cryptographic primitive. Then the adversary uses the traces to build a model for

$$\mathbb{P}[\mathbf{L}|Z] = \mathbb{P}[\mathbf{L}|F(X, K)] \tag{1}$$

where $\mathbf{L}$ represents a random vector corresponding to the traces. The conditional probabilities $\mathbb{P}[\mathbf{L}|Z]$ serve as the leakage templates in the second phase.

In the second phase, also known as the attack phase, the adversary collects several trace-plaintext pairs $\{(\tilde{\mathbf{l}}^i, \tilde{p}^i)\}_{i=0}^{T_a-1}$, where $\tilde{\mathbf{l}}^i$, $\tilde{p}^i$ are the i-th trace and plaintext (or ciphertext) respectively, and T_a is the total number of attack traces, executing the DUT for varying plaintexts. For all the traces, the secret key k^* is unknown but fixed. Finally, the adversary computes the score for each possible key as

$$\hat{\delta}_k = \sum_{i=0}^{T_a-1} \log \mathbb{P}[Z = F(\tilde{p}^i, k)|\mathbf{L} = \tilde{\mathbf{l}}^i] \propto \sum_{i=0}^{T_a-1} \log \left(\mathbb{P}[\mathbf{L} = \tilde{\mathbf{l}}^i|Z = F(\tilde{p}^i, k)] \times \mathbb{P}[F(\tilde{p}^i, k)]\right) \tag{2}$$

The key $\hat{k} = \text{argmax}_k\, \hat{\delta}_k$ is chosen as the predicted key. If $\hat{k} = k^*$ holds, the prediction is said to be correct. The rank of the correct key in the list of all the possible keys sorted by their scores $\hat{\delta}_k$ is used as a metric for the degree of success of the attack. This attack is also called a Template attack as the estimated $\mathbb{P}[\mathbf{L}|Z]$ in Eq. 1 can be considered as the leakage template for different values of the sensitive variable Z.

2.4 Deep Learning based Profiling SCA

In Deep Learning (DL) based profiling SCA, the adversary trains a DL model which takes a trace L as input and generates a probability distribution over all possible values of the sensitive variable Z. More precisely, let $f(\cdot; \theta^*)$ be the trained DL model with θ^* be the model parameters learned during training. Thus, the output of the DL model for a trace $\mathbf{l}$ can be written as

$$\mathbf{p} = f(\mathbf{l}; \theta^*) \tag{3}$$

where $\mathbf{p} \in \mathbb{R}^{|\mathcal{Z}|}$ such that $\mathbf{p}[i]$, for $i = 0, \cdots, |\mathcal{Z}| - 1$, represents the predicted probability for the intermediate variable $Z = i$. During the attack phase, given the set of attack trace-plaintext pairs $\{(\tilde{\mathbf{l}}^i, \tilde{p}^i)\}_{i=0}^{T_a-1}$, the score of each key $k \in \mathcal{K}$ is computed as

$$\hat{\delta}_k = \sum_{i=0}^{T_a-1} \log \mathbf{p}^i[F(\tilde{p}^i, k)] \tag{4}$$

where $\mathbf{p}^i = f(\tilde{\mathbf{l}}^i; \theta^*)$ is the predicted probability vector for the i-th trace. Like template attack, $\hat{k} = \text{argmax}_k\, \hat{\delta}_k$ is chosen as the guessed key. Alternatively, the rank of the correct key in the list of all possible keys sorted by their scores $\hat{\delta}_k$ can be considered as a metric for the degree of the attack's success.

Various DL models (Feed Forward Network [MZ13, MHM13, MPP16], Convolutional Neural Network [MPP16, CDP17, BPS+20, ZS20, PSK+18], Recurrent Neural Network [MPP16, Mag19, LZC+21]) have been used for SCA. Recently, [HSAM22] has introduced a shift-invariant TN model, TransNet, for SCA. However, their proposed model only applies to short traces (having lengths less than a few thousand) as the time and memory complexity of the model is quadratic in trace length. This work introduces a shift-invariant TN with linear time and memory complexity. Thus, in the next section, we describe the architecture of TN. We also briefly outline the existing methods to make the TN models' self-attention operation; thus, the TN models themselves shift-invariant and have linear computational costs. Section 4 proposes a novel shift-invariant self-attention operation for SCA having a linear cost.

3 Transformer Network

Like all other DL models, TN also has a layered structure. It consists of multiple transformer layers stacked one after another. Each transformer layer takes an input 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 transforms it into another sequence $\mathrm{Y} = [\mathbf{y}_0, \ldots, \mathbf{y}_{n-1}]^T \in \mathbb{R}^{n \times d}$ where n corresponds to the trace length or sequence length and d is the dimension of each feature vector. The transformer layer consists of the self-attention layer and position-wise feed-forward layer. More precisely, if we denote the self-attention layer by the function $f_{SA} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$ and the position-wise feed-forward layer by $f_{PFF} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$, then the output Y of a transformer layer can be computed as

$$\begin{aligned}\hat{\mathrm{Y}} &= f_{SA}(\mathrm{X}) + \mathrm{X}\\ \mathrm{Y} &= f_{PFF}(\hat{\mathrm{Y}}) + \hat{\mathrm{Y}}\end{aligned}$$

The function f_{PFF} independently transforms each feature vector using a feed-forward network, thereby enhancing the non-linear characteristics of the model. In contrast, the self-attention layer captures the interdependencies between input features by transforming each feature based on its relation to other features. As a result, the self-attention layer plays a critical role in TN's ability to capture the dependency among the distant features.

In the context of SCA, the input X corresponds to the input of an intermediate layer, where n represents the input length (which is equal to the trace length for the first layer), and d represents the feature dimension of the preceding layer. Similarly, Y corresponds to the output of the layer. The self-attention layer possesses the ability to combine leakage information from multiple POIs, leading to higher SNR outputs. Specifically, it can combine the leakages of different shares from a masked implementation to reconstruct the unmasked secret in some output feature vectors.

In the following section, we present a brief description of the self-attention layer.

3.1 Self-attention Layer

Given the input sequence $\mathrm{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}]^T \in \mathbb{R}^{n \times d}$, the self-attention layer computes the output sequence $\hat{\mathrm{V}} = [\hat{\mathbf{v}}_0, \ldots, \hat{\mathbf{v}}_{n-1}]^T \in \mathbb{R}^{n \times d_v}$ as follows

$$\hat{\mathbf{v}}_i = \sum_{j=0}^{n-1} \mathrm{softmax}\left(\frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d}}\right) \mathbf{v}_j = \sum_{j=0}^{n-1} \frac{\exp\left(\mathbf{q}_i^T \mathbf{k}_j / \sqrt{d}\right)}{\sum_{l=0}^{n-1} \exp\left(\mathbf{q}_i^T \mathbf{k}_l / \sqrt{d}\right)} \mathbf{v}_j \tag{5}$$

where

$$\mathbf{q}_i = \mathrm{W}_q\mathbf{x}_i, \quad \mathbf{k}_i = \mathrm{W}_k\mathbf{x}_i, \quad \mathbf{v}_i = \mathrm{W}_v\mathbf{x}_i$$

for $i = 0, \ldots, n-1$, $\mathrm{W}_q, \mathrm{W}_k \in \mathbb{R}^{d_k \times d}$ and $\mathrm{W}_v \in \mathbb{R}^{d_v \times d}$. The final output of the self-attention layer is computed as $\hat{\mathrm{Y}} = \hat{\mathrm{V}}\mathrm{W}_o^T + \mathrm{X}$ where $\mathrm{W}_o \in \mathbb{R}^{d \times d_v}$ is the projection matrix which projects the d_v-dimensional $\hat{\mathbf{v}}_j$s back into d-dimensional vector space. The matrices $\mathrm{W}_q, \mathrm{W}_k, \mathrm{W}_v$ and W_o are the parameters of the DL model which are learned during the training and the d_k and d_v are two hyper-parameters known as the key and value dimension respectively. The scalar softmax $\left(\frac{\mathbf{q}_i^T\mathbf{k}_j}{\sqrt{d}}\right)$ can be thought of as the attention $\hat{\mathbf{v}}_i$ pays to the input feature $\mathbf{x}_j$. This attention mechanism plays the major role in the TN's ability to capture long-distance dependency. If there exists some dependency between two input features, say $\mathbf{x}_i$ and $\mathbf{x}_j$, the attention from $\hat{\mathbf{v}}_i$ to $\mathbf{x}_j$ can be large, making the i-th output feature $\hat{\mathbf{y}}_i = \mathrm{W}_o\hat{\mathbf{v}}_i + \mathbf{x}_i$ dependent on both $\mathbf{x}_i$ and $\mathbf{x}_j$, thus, capturing the interrelations between those. Moreover, unlike CNN and RNN, the self-attention layer can capture the dependency between $\mathbf{x}_i$ and $\mathbf{x}_j$ in a constant number of steps even when the distance between i and j is large. Indeed, in [VSP+17], it has been argued that TN is better than CNN and RNN in capturing long distant dependency. [HSAM22] has demonstrated that the TN's ability of capturing long distant dependency can be utilized to make it highly effective in attacking software implementation of masked countermeasure in which the leakages of multiple shares (that can be far apart in time dimension) need to be combined for a successful attack.

Multihead Self-Attention One self-attention operation with a set of the parameters $\mathrm{W}_q, \mathrm{W}_k$ and W_v is called one attention head. In practice, several attention heads are parallelly used in the self-attention layer. Thus, an H-head self-attention layer is computed as

$$\begin{aligned} \hat{\mathrm{V}}^{(i)} &= f_{SA}(\mathrm{X}; \mathrm{W}_q^{(i)}, \mathrm{W}_k^{(i)}, \mathrm{W}_v^{(i)}), \quad i = 0, \ldots, H-1 \\ \hat{\mathrm{V}} &= \mathrm{concat}\left(\hat{\mathrm{V}}^{(0)}, \ldots, \hat{\mathrm{V}}^{(H-1)}\right), \\ \hat{\mathrm{Y}} &= \hat{\mathrm{V}}\mathrm{W}_o^T + \mathrm{X} \end{aligned}$$

where, the operation concat $\left(\hat{\mathrm{V}}^{(0)}, \ldots, \hat{\mathrm{V}}^{(H-1)}\right)$ denotes the row-wise concatenation of the matrices $\hat{\mathrm{V}}^{(0)}, \ldots, \hat{\mathrm{V}}^{(H-1)}$. Thus, in this new setting, $\hat{\mathrm{V}}^{(i)} \in \mathbb{R}^{n \times d_v}$, $\hat{\mathrm{V}} \in \mathbb{R}^{n \times Hd_v}$ and $\mathrm{W}_o \in \mathbb{R}^{d \times Hd_v}$. For simplicity, by self-attention, we will imply the single-head self-attention layer only, though our observations can be easily extended to multihead self-attention.

One main drawback of the vanilla self-attention operation is that any parallel implementation of self-attention has quadratic memory and computation cost with respect to the input length. In SCA, the trace length can be very large (in the order of 10^5). Quadratic complexity of the self-attention layer prevents TN from being applied to very long traces [HSAM22]. Several variations of self-attention operation have been introduced, which operate in linear time and memory. In the next section, we describe one such approach.

3.2 Self-attention with Linear Complexity

Rewriting the last term of Eq. (5), the self-attention operation can be given by

$$\hat{\mathbf{v}}_i = \frac{\sum_{j=0}^{n-1} \exp\left(\mathbf{q}_i^T\mathbf{k}_j/\sqrt{d}\right)\mathbf{v}_j}{\sum_{j=0}^{n-1} \exp\left(\mathbf{q}_i^T\mathbf{k}_j/\sqrt{d}\right)}. \tag{6}$$

In [KVPF20], Katharopoulos et al. have replaced the exponential function of the form $\exp(\mathbf{q}^T\mathbf{k}/\sqrt{d})$ by a positive function $k(\mathbf{q},\mathbf{k})$ such that $k(\mathbf{q},\mathbf{k})$ is factorizable as $\phi(\mathbf{q})^T\phi(\mathbf{k})$ for some feature map $\phi : \mathbb{R}^{d_k} \to \mathbb{R}^{d'_k}$ and $k(\mathbf{q},\mathbf{k}) \geq 0$ for all $\mathbf{k},\mathbf{q} \in \mathbb{R}^{d_k}$. A function $k(\mathbf{q},\mathbf{k})$ which is factorizable as $\phi(\mathbf{q})^T\phi(\mathbf{k})$ is known as (positive semi-definite) kernel function [Wik22]. Thus, replacing the exponential function in Eq. (6) by a kernel $k(\cdot,\cdot)$ such that $k(\cdot,\cdot) \geq 0$, the vector $\hat{\mathbf{v}}_i$ can be computed as

$$\hat{\mathbf{v}}_i^T = \frac{\sum_{j=0}^{n-1} k\left(\mathbf{q}_i,\mathbf{k}_j\right)\mathbf{v}_j^T}{\sum_{j=0}^{n-1} k\left(\mathbf{q}_i,\mathbf{k}_j\right)} = \frac{\sum_{j=0}^{n-1} \phi(\mathbf{q}_i)^T\phi(\mathbf{k}_j)\mathbf{v}_j^T}{\sum_{j=0}^{n-1} \phi(\mathbf{q}_i)^T\phi(\mathbf{k}_j)} = \frac{\phi(\mathbf{q}_i)^T\sum_{j=0}^{n-1} \phi(\mathbf{k}_j)\mathbf{v}_j^T}{\phi(\mathbf{q}_i)^T\sum_{j=0}^{n-1} \phi(\mathbf{k}_j)}. \tag{7}$$

Since all $\hat{\mathbf{v}}_i$s share the terms $\sum_{j=0}^{n-1}\phi(\mathbf{k}_j)\mathbf{v}_j^T$ and $\sum_{j=0}^{n-1}\phi(\mathbf{k}_j)$ in Eq. (7), those can be computed using linear time and memory[2]. However, since they have used a different kernel $k(\mathbf{q},\mathbf{k})$ than the $\exp(\mathbf{q}^T\mathbf{k})$, the resultant self-attention operation differs from the vanilla softmax self-attention (i.e. Eq. (5)).

3.2.1 Feature Map for Softmax Self-attention

Recently, several works [PPY+21, CLD+21] have proposed self-attentions which approximate the softmax self-attention and works in linear time and memory. The theoretical foundation of the works lies in the approximation of the Gaussian kernel, i.e., the kernel of the form $\exp\left(-||\mathbf{q}-\mathbf{k}||_2^2/2\sigma^2\right)$ using random Fourier features [RR07, YSC+16, CRW17]. More precisely, the Fourier feature map defined as

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

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

$$exp(-||\mathbf{q}-\mathbf{k}||_2^2/2) \approx \phi_{\text{fr}}(\mathbf{q})^T\phi_{\text{fr}}(\mathbf{k}).$$

Thus, $\phi_{\text{fr}}(\mathbf{x})$ is a feature map for the Gaussian Kernel. Peng et al. [PPY+21] have used the above feature map for the Gaussian kernel to obtain a feature map for the kernel $\exp\left(\mathbf{q}^T\mathbf{k}\right)$. Concretely, $\exp(\mathbf{q}^T\mathbf{k})$ can be written as

$$\exp\left(\mathbf{q}^T\mathbf{k}\right) = \exp\left(\frac{||\mathbf{q}||_2^2}{2}\right)\exp\left(-\frac{||\mathbf{q}-\mathbf{k}||_2^2}{2}\right)\exp\left(\frac{||\mathbf{k}||_2^2}{2}\right)$$

Thus, using the feature map:

$$\phi_{\text{tri}}(\mathbf{x}) = \frac{\exp\left(||\mathbf{x}||_2^2/2\right)}{\sqrt{d_e}}\left[\sin(\mathbf{w}_0^T\mathbf{x}),\ldots,\sin(\mathbf{w}_{d_e-1}^T\mathbf{x}),\cos(\mathbf{w}_0^T\mathbf{x}),\ldots,\cos(\mathbf{w}_{d_e-1}^T\mathbf{x})\right]^T, \tag{9}$$

we can approximate the kernel $\exp\left(\mathbf{q}^T\mathbf{k}\right)$ as $\phi_{\text{tri}}(\mathbf{q})^T\phi_{\text{tri}}(\mathbf{k})$ where d_e is a hyper-parameter known as the dimension of the kernel feature map. However, in [CLD+21], Choromanski et al. pointed out that ϕ_{tri} might lead to unstable behavior of self-attention due to potentially negative components like $\sin(\mathbf{w}_j^T\mathbf{x})$s and $\cos(\mathbf{w}_j^T\mathbf{x})$s in ϕ_{tri}. They resolved the issue by proposing positive random feature map:

$$\phi_{\text{pos}}(\mathbf{x}) = \frac{\exp\left(-||\mathbf{x}||_2^2/2\right)}{\sqrt{d_e}}\left[\exp(\mathbf{w}_0^T\mathbf{x}),\ldots,\exp(\mathbf{w}_{d_e-1}^T\mathbf{x})\right]^T, \tag{10}$$

where $\mathbf{w}_0,\ldots,\mathbf{w}_{d_e-1}$ are as defined in Eq. (8). Since $\phi_{\text{pos}}(\mathbf{x})$ can have only positive components, it solves the unstable behavior of ϕ_{tri}.

[2] Note that, with this reformulation of self-attention operation, the network's ability to learn long-distance dependency is not compromised as every input feature is still connected to every output feature by a constant number of steps. However, such reformulation introduces some approximation errors [CLD+21].

3.3 Relative Positional Self-attention with Linear Complexity

In self-attention of the form of Eq. (5), the attention an output feature $\hat{\mathbf{v}}_i$ pays to an input feature $\mathbf{x}_j$ does not depend on their positions i.e. the indices i and j. More precisely, the attention paid by the output feature $\hat{\mathbf{v}}_i$ to the input feature $\mathbf{x}_j$ is proportional to $k(\mathbf{q}_i, \mathbf{k}_j)$ which is a function of the vectors $\mathbf{q}_i$ and $\mathbf{k}_j$ not their positions i and j. Thus, if the input sequence is permuted, the output sequence of the self-attention layer will also be permuted similarly. However, in SCA, we want the attention to be more on the POIs rather than all sample points having equal attention. Moreover, in the presence of countermeasures like random delay and clock jitters, the distances between the POIs remain approximately same though their absolute positions (the indices at which they appear) vary from trace to trace. Thus, we want the attention an output feature $\hat{\mathbf{v}}_i$ pays to an input feature $\mathbf{x}_j$ to depend on their relative positions or $i - j$. Such modeling of attention probabilities is referred to as relative positional encoding. In self-attention with relative positional encoding, the terms of the form $\exp\left(\mathbf{q}_i^T \mathbf{k}_j\right)$ in Eq. (5) is generalized by a positive function of the form $f(\mathbf{q}_i, \mathbf{k}_j, i - j)$ [SUV18, DYY+19].

In self-attention with relative positional encoding and linear complexity, the kernel $k\left(\mathbf{q}_i, \mathbf{k}_j\right)$ in Eq. (7) is replaced by another kernel of the form $k_r\left(\mathbf{q}_i, \mathbf{k}_j, i - j\right)$ such that the new kernel is (approximately) factorizable as $k_r\left(\mathbf{q}_i, \mathbf{k}_j, i - j\right) = \phi_q(\mathbf{q}_i, i)^T \phi_k(\mathbf{k}_j, j)$. Thus, by replacing the terms $k(\mathbf{q}_i, \mathbf{k}_j)$ in Eq. (7) by the relative position aware kernel, we get self-attention with relative positional encoding, which can be performed in linear time and memory as follows:

$$\hat{\mathbf{v}}_i^T = \frac{\sum_{j=0}^{n-1} k_r\left(\mathbf{q}_i, \mathbf{k}_j, i - j\right) \mathbf{v}_j^T}{\sum_{j=0}^{n-1} k_r\left(\mathbf{q}_i, \mathbf{k}_j, i - j\right)} = \frac{\phi_q(\mathbf{q}_i, i)^T \sum_{j=0}^{n-1} \phi_k(\mathbf{k}_j, j) \mathbf{v}_j^T}{\phi_q(\mathbf{q}_i, i)^T \sum_{j=0}^{n-1} \phi_k(\mathbf{k}_j, j)}. \tag{11}$$

In literature, several self-attention with relative positional encoding which work in linear time and memory have been introduced. For example, [LCW+21, LSL+21, SLP+21] have used feature maps of the form $\phi_q(\mathbf{q}_i, i) = \phi(\mathrm{M}_i \mathbf{q}_i)$ and $\phi_k(\mathbf{k}_j, j) = \phi(\mathrm{N}_j \mathbf{k}_j)$ where the matrix $\mathrm{M}_i^T \mathrm{N}_j$ is a function of $i - j$ and $\phi(\cdot)$ is given by Eq. (9) or (10). [LCW+21, LSL+21] have further generalized the matrices M_i and N_js by some small DL models. In [Che21], Chen et al. have used feature map of the form $\phi_q(\mathbf{q}_i, i) = \mathrm{M}_i \tilde{\phi}_q(\mathbf{q}_i)$ and $\phi_k(\mathbf{k}_j, j) = \mathrm{N}_j \tilde{\phi}_q(\mathbf{k}_j)$ such that $\mathrm{M}_i^T \mathrm{N}_j$ becomes a function of $i - j$. [LLC+21] has used the kernel of the form $k\left(\mathbf{q}_i, \mathbf{k}_j, i - j\right) = \exp\left(\mathbf{q}_i^T \mathbf{k}_j + b_{j-i}\right)$. Here $b_{-(n-1)}, \ldots, b_0, \ldots, b_{n-1}$ are some relative positional biases whose values are also learned during training. They have further shown that the self-attention operation with the above kernel can be performed in $\mathcal{O}(n \log\, n)$ time using Fast Fourier Transform (FFT). [HSGB21] has used a kernel of the form $k\left(\mathbf{q}_i, \mathbf{k}_j, i - j\right) = \exp\left(\mathbf{q}_i^T \mathbf{k}_j + \phi_r(i)^T \phi_r(j)\right)$ where $\phi_r(\cdot)$ is a feature map for the position indices i and j satisfying $\phi_r(i)^T \phi_r(j)$ to be a function of $i - j$.

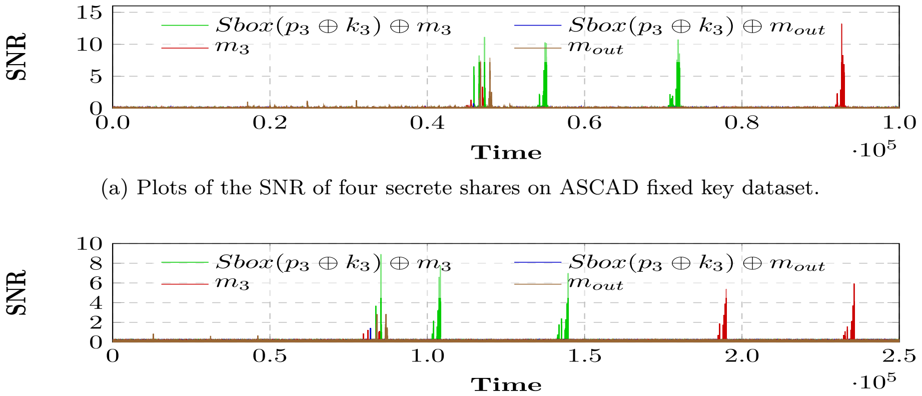
In this next section, we propose a novel attention for SCA.

4 Self-attention in SCA

The informative sample points in power or EM traces are sparse. In other words, only a few sample points in the traces are high SNR sample points, and the rest of those are noisy. To see this, we plot the SNR of four secret shares on two very widely used SCA datasets, namely ASCAD fixed key[3] and ASCAD random key[4] datasets in Figure 1. A peak or a high value in the plots indicates the informative sample points. It can be seen in the figures that the SNR at most of the sample points is close to zero, implying the informativeness of only a few sample points. Thus, the self-attention operation should be

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

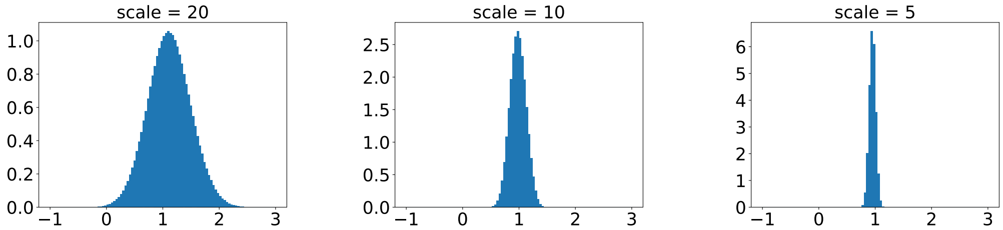
[4] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key



(a) Plots of the SNR of four secrete shares on ASCAD fixed key dataset.

(b) Plots of the SNR of four secrete shares on ASCAD random key dataset.

Figure 1: Plots of the informative sample points on two widely used SCA datasets.



(a) Histogram for scale 20.

(b) Histogram for scale 10.

(c) Histogram for scale 5.

Figure 2: Plots of the histogram of the attention values of the randomly initialized self-attention scheme of [HSGB21] for three different scales.

able to put high attention to only a few sample points while putting close to zero attention to the rest.

However, the existing self-attention with linear complexity like [LCW+21, LSL+21, HSGB21] puts significantly non-zero attention to most of the input feature vectors making the output feature vectors influenced by the noisy (low SNR) sample points. Figure 2 plots the histogram of the attention scores of the randomly initialized self-attention scheme of [HSGB21] for three different values of the scale hyper-parameter. As depicted in the plots, the attention scores are centered around 1, indicating that the self-attention scheme assigns attention close to 1 to the majority of input feature vectors. Similar observations have been made for other self-attention schemes, such as the one proposed by Li et al. [LSL+21]. Another disadvantage of those schemes is that their attention scores result from complicated interactions among many parameters. Consequently, it is difficult to train the network to generate sparse attention scores.

To address the aforementioned limitations of existing self-attention mechanisms, this section introduces a novel attention that employs a Gaussian kernel on the relative positions of input features to generate attention scores. The proposed self-attention produces sparse attention scores, as discussed in Section 4.2. By utilizing relative positional encoding, it achieves shift invariance. Additionally, the proposed method exhibits linear time and memory complexity with respect to the trace length, enabling scalability to longer traces.

4.1 GaussiP: Gaussian Positional Attention

This section introduces the proposed Gaussian Positional attention, also called GaussiP attention. The GaussiP attention exhibits linear time and memory complexity with respect to the input length, making it computationally efficient. The degree of sharpness and sparseness in the attention scores can be controlled by adjusting a suitable parameter. Furthermore, the attention mechanism enables high attention to be assigned to distant features, facilitating the flow of information over long distances.

The subsequent sections delve into the details of the GaussiP attention. Firstly, we describe the kernel function utilized in the GaussiP attention. Next, we introduce the feature map employed for factorizing the kernel function, enabling efficient computation. Lastly, we address a significant drawback of the proposed kernel function and present our solution, which involves utilizing multiple heads in the attention.

4.1.1 Deciding the Kernel Function

Unlike most of the existing self-attention, we use a Gaussian kernel for our proposed attention:

$$k_r\left(\mathbf{q}_i, \mathbf{k}_j, i-j\right) = \exp\left(-\frac{||\phi_q(\mathbf{q}_i, i) - \phi_k(\mathbf{k}_j, j)||_2^2}{2}\right) \tag{12}$$

with

$$\phi_q(\mathbf{q}, i) = \begin{bmatrix} \beta_1 \mathbf{q} \\ \beta_2 s_p \mathrm{W}_p(\mathbf{b} + i\mathbf{p}) \end{bmatrix} \text{ and } \phi_k(\mathbf{k}, j) = \begin{bmatrix} \beta_1 \mathbf{k} \\ \beta_2 s_p \mathrm{W}_p(\mathbf{b} + j\mathbf{p} + c_p n \mathbf{p}) \end{bmatrix} \tag{13}$$

for $i, j = 0, 1, \ldots, n-1$ where $\beta_1, \beta_2 \in [0, +\infty)$ are two hyperparameters, $\mathbf{p}, \mathbf{b} \in \mathbb{R}^{d_p}$ are two predefined constants, $\mathrm{W}_p \in \mathbb{R}^{d_p \times d}$ is a matrix with entries drawn from a uniform random distribution, and $s_p \in (0, +\infty)$, $c_p \in [-1, 1]$ are two trainable parameters. Thus, with the above defined ϕ_q and ϕ_k, the proposed kernel takes the form

$$k_r(\mathbf{q}_i, \mathbf{k}_j, i-j) = \exp\left(-\frac{\beta_1^2 ||\mathbf{q}_i - \mathbf{k}_j||_2^2 + \beta_2^2 s_p^2 (i - j - c_p n)^2 ||\mathrm{W}_p \mathbf{p}||_2^2}{2}\right) \tag{14}$$

The part $\beta_1^2 ||\mathbf{q}_i - \mathbf{k}_j||_2^2$ in the above equation influences the kernel scores based on the contents of the input feature vectors while the part $\beta_2^2 s_p^2 (i - j - c_p n)^2 ||\mathrm{W}_p \mathbf{p}||_2^2$ influences the scores based on the relative positions, i.e., $i - j$ of the feature vectors. In our initial set of experiments, we found that the first part does not positively effect the performance of EstraNet. Thus, we set $\beta_1 = 0$ in Eq. (13) which simplifies $\phi_q(\mathbf{q}, i)$ and $\phi_k(\mathbf{k}, i)$ as

$$\phi_q(i) = \beta_2 s_p \mathrm{W}_p(\mathbf{b} + i\mathbf{p}) \quad \text{and} \quad \phi_k(j) = \beta_2 s_p \mathrm{W}_p(\mathbf{b} + j\mathbf{p} + c_p n \mathbf{p}) \tag{15}$$

resulting into the following simplified kernel:

$$k_{GPA}(i-j) = \exp\left(-\frac{||\phi_q(i) - \phi_k(j)||_2^2}{2}\right) = \exp\left(-\frac{\beta_2^2 s_p^2 (i - j - c_p n)^2 ||\mathrm{W}_p \mathbf{p}||_2^2}{2}\right) \tag{16}$$

The above equation shows several important properties of the proposed kernel function. Firstly, it can be observed that the maximum value of the kernel output occurs when $i - c_p n = j$. In simpler terms, the i-th output feature vector assigns maximum attention to the $(i - c_p n)$-th input feature vector. Thus, the attention mechanism facilitates the flow of information from the $(i - c_p n)$-th index to the i-th index, enabling the learning of long distant dependencies. Secondly, the precision of the attention can be controlled

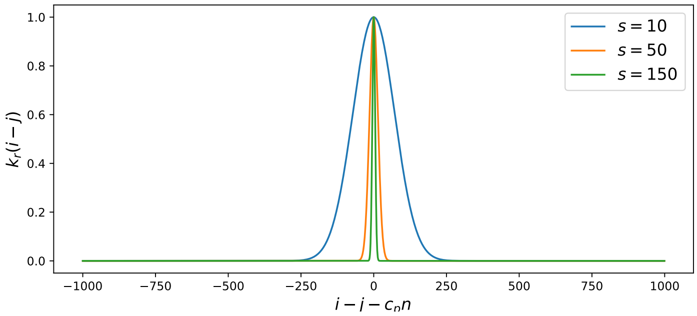


Figure 3: Plots of $k_{GPA}(i-j)$ vs. $(i-j-c_p n)$ for different values of $s(=\beta_2 s_p)$.

by appropriately setting the hyper-parameter β_2 or learning an appropriate value for the parameter s_p during the training process. Let us denote $s = \beta_2 s_p$. If the value of s is large, the attention will be concentrated within a small region of the input traces. Conversely, if the value of s is small, the attention will be spread over a larger region. To illustrate this, Figure 3 displays the kernel scores for various values of s. It can be observed that as the value of s increases, the kernel scores become more concentrated in smaller regions.

4.1.2 Deciding the Feature Map for the Kernel Function

Since, we are using a Gaussian kernel, following the work of [CLD+21], a feature map with only positive entries can be given for the kernel as follows:

$$\phi'(\mathbf{x}) = \frac{\exp\left(-||\mathbf{x}||_2^2\right)}{\sqrt{d_e}} \left[\exp(\mathbf{w}_0^T\mathbf{x}), \ldots, \exp(\mathbf{w}_{d_e-1}^T\mathbf{x})\right]^T$$

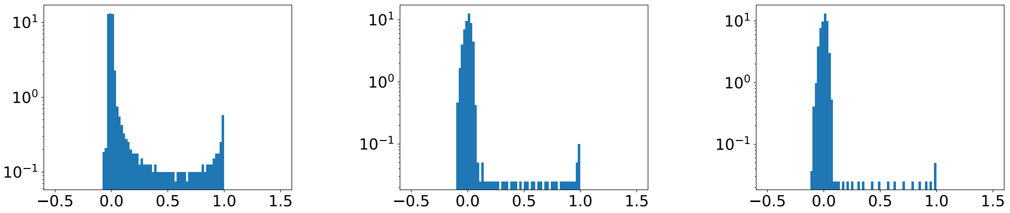
However, we have observed that the approximation error of ϕ' significantly increases as the norms of the input features $\mathbf{x}$ become larger. It should be noted that in order to concentrate the attention scores on a smaller segment of the input traces, the norms of the input features need to be sufficiently large. As an alternative, the Fourier features ϕ_{fr} defined in Eq.. (8) can be used as the feature map for the kernel. However, as pointed out in [CLD+21], the kernel scores $k_{GPA}(i-j)$ approximated by $\phi_{\text{fr}}(\phi_q(i))^T\phi_{\text{fr}}(\phi_k(j))$ can be potentially negative, causing the increase in the variance of the normalized kernel score $k_{GPA}(i-j)/\sum_{k=0}^{n-1} k_{GPA}(i-k)$, which, in turn, causes the unstable behavior of the self-attention. To address this issue, we propose to approximate the normalizing factor of the kernel given in Eq. (16) in a closed form as follows:

$$\sum_{k=0}^{n-1} k_{GPA}(i-k) \approx \int_{x=-\infty}^{\infty} k_{GPA}(x)dx \approx \frac{1}{\beta_2 s_p ||\mathrm{W}_p\mathbf{p}||_\mathbf{2}} \tag{17}$$

The justification for the approximation can be found in Appendix A. Using the above approximation in the proposed attention, the expression of the output feature vectors $\hat{\mathbf{v}}_i$ can be given as:

$$\hat{\mathbf{v}}_i^T = \frac{\sum_{j=0}^{n-1} k_{GPA}\left(i-j\right)\mathbf{v}_j^T}{\sum_{k=0}^{n-1} k_{GPA}\left(i-k\right)} \approx \beta_2 s_p ||\mathrm{W}_p\mathbf{p}||_\mathbf{2} \left[\phi_{\text{fr}}(\phi_\mathbf{q}(\mathbf{i}))^\mathbf{T} \sum_{\mathbf{j=0}}^{\mathbf{n-1}} \phi_{\text{fr}}(\phi_\mathbf{k}(\mathbf{j}))\mathbf{v}_\mathbf{j}^\mathbf{T}\right] \tag{18}$$

where $\phi_{\text{fr}}(\cdot)$ is given by Eq. 8, and $\phi_q(i) = \beta_2 s_p \mathrm{W}_p(\mathbf{b} + i\mathbf{p})$ and $\phi_k(j) = \beta_2 s_p \mathrm{W}_p(\mathbf{b} + j\mathbf{p} + c_p n\mathbf{p})$.



(a) Histogram for $s = 10$.

(b) Histogram for $s = 50$.

(c) Histogram for $s = 150$.

Figure 4: Plots of the histogram of the unnormalized kernel scores approximated by the Fourier feature map ϕ_{tri}. The histogram is plotted for three different values of s (i.e. $\beta_2 s_p$).

Table 1: The set of hyper-parameters of the proposed GaussiP attention.

Notation	Description	Notation	Description
d	model dimension	d_k	key dimension
d_v	value dimension	d_e	dimension of kernel feature map
n	input or trace length	H	number of heads
β_2	distance based scaling in attention kernel		

4.1.3 Making the Attention Distribution Multi-modal

The major problem with the Gaussian kernel is that its attention scores follow a uni-modal distribution. In other words, it puts high attention over a small contiguous segment of the traces and assigns small attention scores to the rest of the parts (as seen in Figure 3). The uni-modal attention distribution limits the flow of information from one region to a distant region. To make the attention distribution multi-modal, we use multi-head attention (kindly refer to Section 3.1 for a description of multi-head attention) with different heads having different s_p and c_p parameters. Since different heads have different c_p values, they put high attention to different parts of the input traces allowing the flow of information to an output feature vector from different regions of the input traces. Similarly, separate s_p for each head enables them to scale the attention independently. We found that the proper initialization of c_p and s_p for each head is crucial for the successful training of EstraNet. We initialize s_p for all heads to the same value 1. However, we initialize c_p for h-th head to $(1 + 2h)/2H$ for $h = 0, 1, \ldots, H - 1$. In other words, c_ps are initialized so that different attention heads focus on different parts of the input sequence. Appendix B plots the attention probabilities learned at the attention heads of a EstraNet model.

The set of hyper-parameters of the proposed GaussiP attention are shown in Table 1.

4.2 Difference with the Existing Alternatives

The functional distinctions between the proposed GaussiP attention and existing foremost alternatives are summarized in Table 2. Notably, the attention scores in the GaussiP attention exhibit sparsity. Figure 4 illustrates that, for significantly large values of $s = \beta_2 s_p$, the majority of attention scores are concentrated around zero, while only a few scores are substantially greater than zero. This characteristic aligns well with the nature of side-channel traces, where informative sample points are typically sparse. While methods such as Luo et al. [LLC+21] offer greater flexibility in learning arbitrary probability distributions, they require complex operations such as Fast Fourier Transform (FFT) for efficient

Table 2: Differences between the proposed GaussiP attention and the foremost self-attentions (with linear complexity and relative positional encoding) in TN literature.

	Self-attention	
	Conventional	Proposed
Attention form	$\hat{\mathbf{v}}_i^T = \frac{\sum_{j=0}^{n-1} k_r(\mathbf{q}_i,\mathbf{k}_j,i-j)\mathbf{v}_j^T}{\sum_{j=0}^{n-1} k_r(\mathbf{q}_i,\mathbf{k}_j,i-j)}$	$\hat{\mathbf{v}}_i^T = \beta_2 s_p \Vert \mathrm{W}_p \mathbf{p} \Vert \sum_{\mathbf{j=0}}^{\mathbf{n-1}} \mathbf{k_r}(\mathbf{i}-\mathbf{j})\mathbf{v}_\mathbf{j}^\mathbf{T}$
Kernel form	$k_r(\mathbf{q}_i, \mathbf{k}_j, i-j)$ $= \exp\left(\phi_q(\mathbf{q}_i, i)^T \phi_k(\mathbf{k}_j, j)\right)$	$k_r(i-j) = \exp\left(-\Vert\phi_q(i) - \phi_k(j)\Vert^2\right)$
Kernel feature map	ϕ_{tri}, ϕ_{pos} or ϕ_{fr}	ϕ_{fr}
Positional Encoding	linear or trigonometric	linear

implementation. The self-attention proposed by Guo et al. [GZL19] employs a Gaussian prior over input features to bias the attention scores based on relative distances. However, unlike our scheme, they utilize a softmax-based self-attention mechanism. Additionally, their scheme emphasizes high attention to nearby features, whereas our scheme can put high attention to distant features. Although the self-attention approach presented by Liutkus et al. [LCW+21] can be represented as a Gaussian kernel with a Fourier feature map for certain hyper-parameter configurations, their attention scores are maximized for $i - j = 0$. In other words, in their scheme, each position or sample point assigns maximum attention to itself, making the propagation of information from one region to a distant region challenging. Conversely, in our proposed scheme, for sufficiently large β_2, the attention scores are maximized when $i - j \approx nc_p$, where n denotes the sequence length and $c_p \in [0, 1]$ represents a trainable parameter. This allows each output feature to allocate high attention to a distant region, enabling the flow of information over long distances.

In the subsequent section, we present the architectural design of EstraNet.

5 EstraNet Architecture

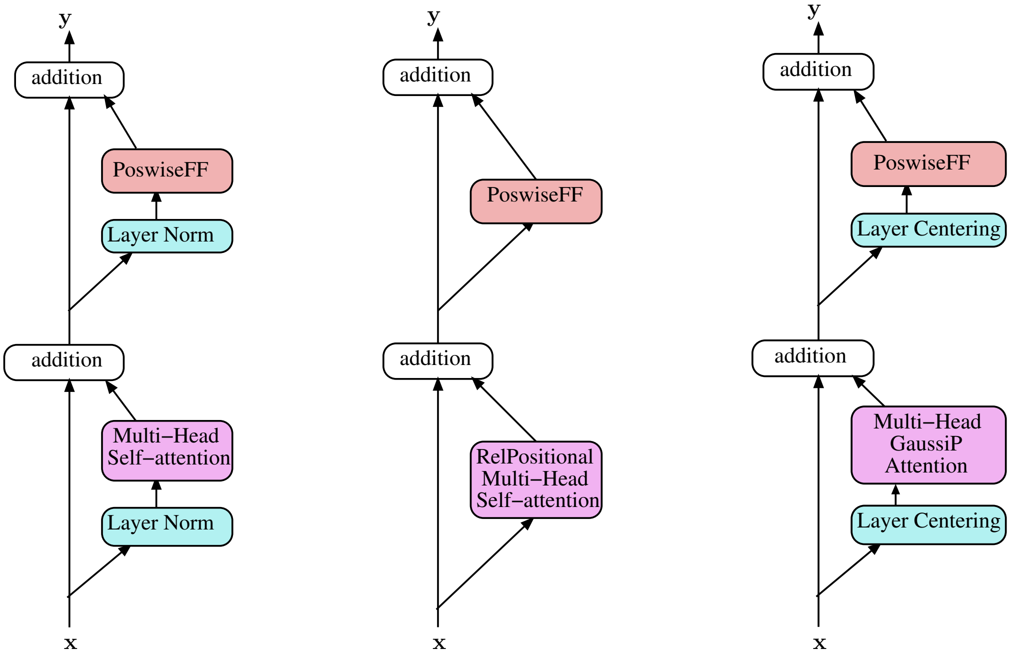
This section provides a detailed overview of the EstraNet architecture. In Section 5.1, we introduce a layering-centering normalization technique. Subsequently, in Section 5.2, we present the structure of a single layer of EstraNet. Finally, in Section 5.3, we describe the complete architecture of EstraNet.

5.1 Layer-Centering

In the conventional TN architecture, layer normalization is often employed to ensure the stability of training [XYH+20]. However, it has been observed in [HSAM22] that applying layer normalization to TN layers makes the network untrainable for the SCA datasets. Our experiments also found that incorporating either layer normalization or batch normalization in EstraNet layers makes its performance poor. Consequently, we introduce a novel "layer-centering" operation as an alternative approach.

To understand the layer-centering operation, let us assume that $[\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}]^T \in \mathbb{R}^{n \times d}$ represents the input to the layer. Given the input, we start by computing the mean of each vector in the input sequence:

$$\mu_i = \frac{1}{d} \sum_{j=0}^{d-1} \mathbf{x}_i[j], \text{ for } i = 0, \ldots, n-1$$



(a) Standard TN layer.

(b) TransNet layer.

(c) EstraNet layer.

Figure 5: Figure 5a shows a single layer of standard TN with pre-layer normalization [XYH+20]. Figure 5b shows a single layer of TransNet proposed in [HSAM22]. Figure 5c depicts a single layer of the proposed EstraNet.

where $\mathbf{x}[j]$ denotes the j-th element of $\mathbf{x}$ and d is the model/feature dimension. Finally, the input vectors are re-centered as

$$\bar{\mathbf{x}}_i = \mathbf{x}_i - \mu_i + \mathbf{c}_{lc}, \text{ for } i = 0, \ldots, n-1 \tag{19}$$

where $\mathbf{c}_{lc} \in \mathbb{R}^d$ is a trainable parameter. The resulting sequence $[\bar{\mathbf{x}}_0, \ldots, \bar{\mathbf{x}}_{n-1}]$ is the output of the layer in the layer-centering operation. Specifically, each element of the input sequence is first centered to $\mathbf{0}$ and then re-centered to $\mathbf{c}_{lc}$, which is a parameter learned during the training process. It is worth noting that while layer-normalization involves re-centering and re-scaling each vector in the input sequence, layer-centering involves only re-centering the elements without any re-scaling.

5.2 Single Layer of EstraNet

The EstraNet layer, as illustrated in Figure 5c, is similar to a standard TN layer depicted in Figure 5a. However, it incorporates the proposed multi-head GaussiP attention from Section 4.1 instead of the vanilla multi-head self-attention and the layer-centering instead of the layer-normalization. Compared to the TransNet layer shown in Figure 5b, EstraNet incorporates the novel multi-head GaussiP attention. The attention operation has a linear time and memory complexity with respect to the input length. Conversely, the multi-head self-attention employed in the TransNet layer exhibits quadratic memory and time complexity. Furthermore, the TransNet layer lacks any normalization layer, whereas the EstraNet layer utilizes layer-centering for input normalization.

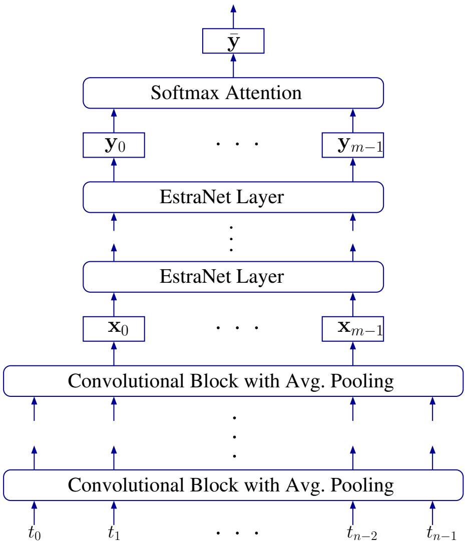


Figure 6: EstraNet Architecture.

5.3 EstraNet Architecture

EstraNet follows the general multilayer architecture of TN models with some notable exceptions. The architecture is shown in Figure 6. The input to the EstraNet model is a one-dimensional trace denoted as $\mathbf{t} = [t_0, \ldots, t_{n-1}] \in \mathbb{R}^n$. However, the EstraNet layer assumes the input to be two-dimensional. Thus, we pass each input trace through several, say L_{conv}, convolutional and average-pooling layers, which convert the input traces $\mathbf{t}$ into a sequence of vectors $\mathrm{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}]^T \in \mathbb{R}^{m \times d}$ where m is the (possibly reduced) length of the output sequence X and d is the dimension of its feature vectors. Note that by setting a large value for the pool size, denoted as ps, of the average-pooling layers, we can make $m \ll n$ enabling the efficient processing of the sequence by the following layers of the EstraNet. Then the output of the final convolutional layers X is passed through several EstraNet layers resulting in the output $\mathrm{Y} = [\mathbf{y}_0, \ldots, \mathbf{y}_{m-1}]^T \in \mathbb{R}^{m \times d}$. The output sequence Y of m d-dimensional feature vectors is then reduced into a single vector $\bar{\mathbf{y}} \in \mathbb{R}^d$ using a (multi-head) softmax-attention layer. The vector $\bar{\mathbf{y}}$ is then passed to the classification layer to generate prediction probabilities.

In the following section, we present the results of our experiments.

6 Experimental Results

This section presents the experimental evaluation of EstraNet. We provide an overview of the datasets used for the evaluation in Section 6.1. The methodology for selecting the attack window is described in Section 6.2. We provide the detailed information about the benchmark models used in this study in Section 6.3. The training process of EstraNet is explained in Section 6.4. Section 6.5 elaborates on the experiment setup and evaluation methods. A comparative study between EstraNet and the benchmark models in the presence of a combinations of masking, random delay, and clock jitter countermeasures is provided in Sections 6.6 to 6.8. In Section 6.9, we perform an ablation study to investigate the impact of various design choices in EstraNet. Section 6.10 discusses the impact of several hyperparameter choices in EstraNet's performance. The training time of EstraNet is compared with that of the benchmark models in Section 6.11. Finally, Section 6.12 investigates the impact of data augmentation in the shift-invariance of EstraNet.

6.1 Dataset Details

For evaluating EstraNet, we select three datasets of software implementation of ciphers protected by masking countermeasures. Since, in the software implementation of masking countermeasures, different shares of the intermediate secret variable leak at different regions of the power/EM traces, they are the ideal candidates for evaluating EstraNet's ability to capture long-distance dependency. Additionally, we have added random delay and the clock jitter effect [WP20] in the traces to evaluate the shift-invariance of EstraNet. This section provides the details of the datasets.

ASCAD Fixed Key (ASCADf) ASCAD fixed key dataset[5] is a collection of $60K$ traces of a first-order masked implementation of AES running on an 8-bit ATMega8515 microcontroller. We divided the entire dataset into three splits: profiling, validation, and test containing $50K$, $5K$, and $5K$ traces, respectively. Following the common practice in the literature [BPS+20], we attack the third S-box operation of the first round of the cipher. We use the identity leakage model to generate the labels for the profiling traces as it is found to perform better than Hamming weight leakage model in previous studies [BPS+20].

ASCAD Random Key (ASCADr) As the ASCADf dataset, ASCAD random key dataset[6] is a collection of traces of a first-order masked implementation of AES running on an 8-bit ATMega8515 microcontroller. However, in the ASCADf dataset, the secret key is fixed for all profiling traces, whereas in the ASCADr dataset, the secret key randomly varies for each profiling trace. The profiling split of the dataset contains $200K$ traces, while the attack split contains $100K$ traces. We created validation and test splits by selecting $10K$ traces for each split from the $100K$ attack traces. Like the ASCADf dataset, we attack the third S-box operation of the first round of the cipher. We use the identity leakage model to generate the labels for the profiling traces on this dataset also.

CHES 2020 CTF SW3 (CHES20) Clyde-128 is a tweakable block cipher that supports side-channel resilient and efficient bit-slicing implementation on 32-bit microprocessors [BBB+20]. Spook SCA CTF[7] is an SCA challenge for masked implementations of the Clyde-128. The challenge consists of multiple datasets for different implementations of the cipher. We select the dataset collected from a second-order masked implementation of the cipher running on an ARM Cortex-M0 microcontroller. The dataset contains $200K$ and $500K$ profiling and attack traces. We select all $200K$ profiling traces as the train set and $10K$ traces for validation and test set each from the attack traces. Since Clyde-128 is an LS design, it works on (4×32)-bit state, with 4 being the size of the non-linear S-box and 32 being the size of the linear L-box. We target the four bits of the 17^{th} column from the left of the first round S-box operation. We chose the 17^{th} column as we found the SNR for these bits to be high. Since each bit of an S-box is processed separately in the bit-slicing implementation of the cipher, we use a multilabel loss having a sigmoid function for each output bit of the S-box (as introduced in [ZXF+19, ZXF+21]) to attack the four target bits.

The statistics of all three datasets are summarized in Table 3.

6.2 Attack Window Selection

The trace length of the above three datasets varies from 62500 (for the CHES20 dataset) to $250K$ (for the ASCADr dataset). Instead of performing the evaluation on the full-length

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

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

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

Table 3: Dataset statistics.

	ASCADf	ASCADr	CHES20
Profiling dataset size	50000	200000	200000
Validation dataset size	5000	10000	10000
Test dataset size	5000	10000	10000
Trace length	20000	10000	10000

traces, we perform it on a selected attack window from the full-length traces. We consider two sizes for the attack window: $10K$ and $40K$. Thus, for the first set of experiments, we select an attack window of size $10K$ for each of the three datasets and perform attacks on the window. And for the second set of experiments, we use an attack window of size $40K$.

Table 4: The selected attack windows on the three datasets.

Dataset	Attack Window Size	
	$10K$	$40K$
ASCADf	$[40000, 50000]$	$[30000, 70000]$
ASCADr	$[78000, 88000]$	$[70000, 110000]$
CHES20	$[46000, 56000]$	$[20000, 60000]$

The selected attack windows for the three datasets are given in Table 4. The attack windows are selected using SNR-based methods. Thus, we calculated the SNR at each sample point and selected a window of respective size such that the window contains the most number of high SNR sample points. In many scenarios (e.g., in the presence of countermeasures like masking, desynchronization, and clock jitter), the SNR-based method may not work. However, note that one can still be able to identify the attack window (with size in the order of $10K$, which is significantly large) based on a combination of the knowledge of the implementation and intelligent guess (such as in scheme-aware threat model [MCLS23]). In the worst case, an adversary can iteratively repeat the attack by selecting different attack window at different segments of the traces. For example, if the first round of the cipher spans over $100K$ sample points, one can repeat the attack on multiple windows of size $40K$, such as $[0, 40K]$, $[20K, 60K]$, $[40K, 80K]$, and $[60K, 100K]$. Note that, recent research [LZC+21] has demonstrated the possibility of directly performing attacks on traces with a length in the order of $100K$ and achieving good results. However, the results presented in Section 6.8 reveal that the performance of these models significantly deteriorates in challenging scenarios, such as in the presence of clock jitter. Therefore, EstraNet can be a better alternative for the worst-case security evaluation in such situations. From now on, we will use the terms “window size” and “trace length” interchangeably to refer to the size of the attack window.

6.3 Benchmark Models

This section briefly describes the (existing) DL models used as the benchmark models in our experiments. Many DL models have been introduced in SCA literature over the last few years. Among all those models, we have selected three models which have been introduced to deal with long traces and/or large trace desynchronizations. The models are briefly described below. The detailed architectures of the models are given in Appendix C.

6.3.1 PolyCNN [MBC+20]

In [MBC+20], Masure et al. have proposed a CNN model to attack AES implementation protected by code polymorphism. Using their proposed model, they successfully recovered the secret key using less than 20 attack traces. Since their model is suitable for long traces, we use it as a benchmark model. We trained the model on the three datasets for $10K$ epochs using Adam optimizer and a constant learning rate of 1e-5.

6.3.2 EffCNN [ZBHV20]

In [ZBHV20], Zaid et al. have proposed a methodology for creating CNN models to be effective against desynchronized traces. They have further demonstrated that their methodology can be used to construct CNN models to perform successful attacks on several datasets. Thus, their models are good candidates for being benchmark models. We constructed three models for the three datasets and used those as the benchmark models. The models have been trained for $2K$ epochs using Adam optimizer and a constant learning rate of 2.5e-5.

6.3.3 LSTMNet [LZC+21]

In [LZC+21], Lu et al. proposed to use LSTM-based models to perform attacks on full-length traces. Their experimental study demonstrated that the LSTM-based models could be used to conduct successful attacks on both synchronized and desynchronized datasets. Thus, we use their models as benchmark models. On the ASCADf and ASCADr datasets, we took the respective models from their online repository[8] and trained the models. Since no LSTM-based model is available for the CHES20 dataset, we use their model for the ASCADr dataset to train on the CHES20 dataset. The models were trained for $4K$ epochs using Adam optimizer and a constant learning rate of 1e-4.

6.4 Training Details and Hyper-parameter settings of EstraNet

6.4.1 Training details

We use the cross-entropy loss and Adam optimizer to train EstraNet. For the learning rate schedule, cosine-decay with linear warmup schedule [ZLLS20] has been used. More precisely, we increase the learning rate linearly from 0 to 2.5e-4 for t_{warmup} steps of gradient update and then gradually decreases to $0.004 \times$ 2.5e-4 following a cosine curve for the remaining $t_{max} - t_{warmup}$ steps where t_{max} is the maximum training steps and t_{warmup} (satisfying $t_{warmup} < t_{max}$) is the number of warmup steps. In Appendix D, we describe the learning rate schedule in more detail.

6.4.2 Hyper-parameter settings

We adopted the common hyper-parameters like the number of EstraNet layers L, and model dimension d from [HSAM22]. The value dimension d_v and the number of heads H in GaussiP attention have been set to 32 and 8, respectively. Following [BPS+20] and [HSAM22], the kernel width of the first convolutional layer has been set to 11. The kernel width of the subsequent convolutional layers has been set to 3 (as in [LZC+21]). Unless stated otherwise, we set the number of convolutional layers and the pool size of each average-pooling layer to 2 and 10, respectively. However, we found that, on the ASCADf dataset, a pool size of 10 leads to poor performance as the sampling rate of the dataset is comparatively low. Thus, we set it to a slightly smaller value, 8. The dimension of the feature map of the GaussiP attention kernel (denoted as d_e) has been set to 512. We set t_{max}, the maximum training steps, and t_{warmup}, the warmup steps of EstraNet training to $4M$ and $1M$ for ACSADf and ASCADr datasets. However, using $1M$ as the warmup steps (t_{warmup}) leads to unstable training on the CHES20 dataset. Thus, we set it to $2M$. We found that the scaling hyper-parameter β_2 influences the performance of EstraNet highly.

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

Table 5: The minimum number of traces (lesser is better) required to reach guessing entropy 1 by EstraNet and the benchmark models for trace length $10K$. The models have been evaluated on attack traces with attack desync 0 (no desync), 200 and 400. The columns titled *Best*, *Med.*, and *Avg.* respectively show the best, median and average results of three independently trained models.

Dataset	Model	Attack Desync 0			Attack Desync 200			Attack Desync 400		
		Best	Med.	Avg.	Best	Med.	Avg.	Best	Med.	Avg.
ASCADf	PolyCNN	65	68	83.0	44	54	54.7	36	44	42.3
	EffCNN	40	42	44.7	24	26	26.0	18	28	24.7
	LSTM	16	26	30.7	21	37	32.3	21	31	35.7
	EstraNet	13	15	14.3	12	13	12.7	9	13	12.3
ASCADr	PolyCNN	21	28	28.0	9	10	10.7	9	9	10.3
	EffCNN	32	35	36.0	15	22	22.0	16	20	20.7
	LSTM	5	6	7.7	6	9	9.0	6	7	7.7
	EstraNet	5	5	5.0	5	6	5.7	4	5	5.0
CHES20	PolyCNN	15	22	19.7	15	19	20.3	16	18	19.0
	EffCNN	34	58	58.0	36	74	62.3	30	90	74.0
	LSTM	4	46	47.3	5	67	47.0	5	46	45.7
	EstraNet	5	6	6.7	5	6	6.7	4	7	7.3

Therefore, we tuned it over three values: 10, 50, and 150 for each dataset. The rest of the hyper-parameter values have been found based on some initial experiments. Section 6.10 discusses the influence of several important hyper-parameters in EstraNet's performance.

6.5 Experimental Setup

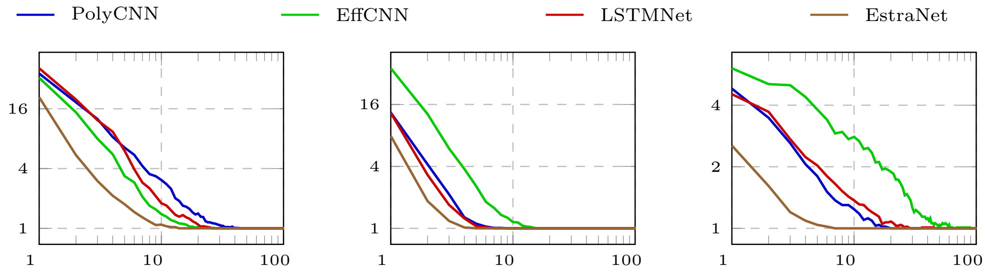
Since we aim to evaluate the robustness of the models to misalignments in the traces, we trained each model on desynchronized profiling traces. Data augmentation techniques were also applied by introducing random displacements to each profiling trace on the fly during training. Thus, in different epochs, the same profiling trace is shifted by different values. It should be noted that this data augmentation was also implemented when training the benchmark models. To reduce training time, early stopping of training was employed. In other words, we evaluated the trained model at regular intervals during the progress of the training and stopped the training when no significant improvement was observed on the validation dataset. The intermediate model that exhibited the best performance on the validation dataset was selected for the final evaluation on the test dataset.

The guessing entropy of each model on a dataset is computed by repeating the attack 100 times on randomly permuted traces of the dataset. For each experiment, we repeat each model's training (starting from the random initialization) three times. We report the best, median, and average results of the repeated experiments.

6.6 Experimental Results for Trace Length $10K$

This section presents a comparative analysis of EstraNet with the benchmark models for the attack window size of $10K$ (refer to Section 6.2 for details on attack window selection). To assess the robustness of the deep learning (DL) models against random delay countermeasures, we applied a profiling desynchronization of 200. In other words, we independently desynchronized each profiling trace by a maximum displacement of 200. During training, we have used data augmentation. More precisely, we randomly desynchronized each profiling trace by a maximum displacement of 200 on the fly (refer to Section 6.5 for detailed description). The trained models were then evaluated on the attack set with attack desynchronizations of 0, 200, and 400. Each DL model was trained independently three times. Table 5 provides the best, median, and average values of T_{GE1} (the minimum number of traces required to achieve guessing entropy 1) obtained from the three trained models for the three attack desynchronization scenarios.

Based on the observations from the table, it can be noted that on the ASCADf dataset, EstraNet consistently outperforms the other three methods. It demonstrates an improvement of more than 50% in the majority of cases in terms of all three metrics and across all three attack desynchronization scenarios. On the ASCADr dataset, EstraNet performs significantly better than EffCNN and similar to LSTMNet with respect to all three metrics. For the dataset, while PolyCNN shows similar performance to EstraNet for attack desynchronizations of 200 and 400, it performs poorly compared to EstraNet for attack desynchronization of 0. Finally, on the CHES20 dataset, EstraNet showcases more than 60% improvement over PolyCNN and EffCNN in terms of all three metrics. Although LSTMNet has comparable results to EstraNet in terms of the best T_{GE1} on that dataset, it requires 6 to 12 times more traces to achieve guessing entropy 1 according to the median or average values. In summary, it can be concluded that EstraNet consistently performs significantly better, with improvements of over 50%, compared to the benchmark models in the majority of cases, though gains are marginal sometimes. Figure 8 plots the median guessing entropy of the DL models with respect to the number of attack traces on the three datasets for attack desync 400. The plots also support the above observations.



(a) ASCADf dataset.

(b) ASCADr dataset.

(c) CHES20 dataset.

Figure 8: Plots of the guessing entropy vs. the number of attack traces for the experiments with trace length $10K$. The attack traces are desynchronized by a maximum displacement of 400.

6.7 Experimental Results for Trace Length $40K$

This section presents a comparative analysis with the benchmark models for the attack window size of $40K$. In contrast to Section 6.6, where the trained models were evaluated against smaller attack desynchronizations (0, 200, and 400), this section evaluates the models in the presence of larger attack desynchronization (600 and 1000). For the experiments, a profiling desynchronization of 600 was used, resulting in the random shifting of each profiling trace by a maximum displacement of 600. Furthermore, data augmentation was incorporated during training by additionally desynchronizing each profiling trace on-the-fly with a maximum displacement of 400. For each dataset, we independently trained each DL model three times. Table 6 provides the best, median, and average values of T_{GE1} (the minimum number of attack traces required to achieve guessing entropy 1) obtained from the three trained models.

In this scenario, some trained models failed to reach guessing entropy 1 using $5K$ attack traces. Entries with a dash symbol ('-') in Table 6 indicate that the average T_{GE1} is not

Table 6: The minimum number of traces (lesser is better) required to reach guessing entropy 1 by EstraNet and the benchmark models for trace length $40K$. The models have been evaluated on attack traces with attack desyncs 600 and 1000. The columns titled *Best*, *Med.*, and *Avg.* respectively show the best, median and average results of three independently trained models. The '-' entries in the table indicate that the average value is not available as some of the independently trained models failed to reach guessing entropy 1 using $5K$ attack traces.

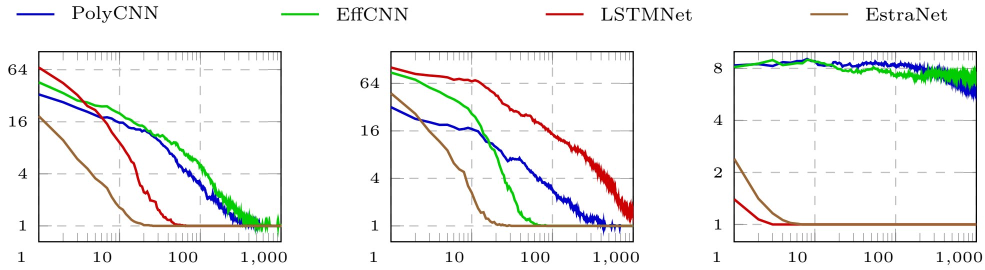
Dataset	Model	Attack Desync 600			Attack Desync 1000		
		Best	Med.	Avg.	Best	Med.	Avg.
ASCADf	PolyCNN	343	534	–	548	553	–
	EffCNN	353	631	–	198	747	–
	LSTM	42	72	242.3	30	84	164.7
	EstraNet	21	22	23.0	22	26	25.3
ASCADr	PolyCNN	185	271	482.7	236	246	541.7
	EffCNN	41	72	63.7	49	85	76.3
	LSTM	823	1158	1292	1260	1722	1794
	EstraNet	23	36	55.7	28	31	62.0
CHES20	PolyCNN	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–
	EffCNN	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–
	LSTM	2	4	–	3	3	–
	EstraNet	4	11	22.3	5	7	21.2

available as some of the independently trained models failed to reach guessing entropy 1 using $5K$ attack traces. While comparing the results of EstraNet with the benchmark models on the ASCADf dataset, EstraNet demonstrates a significant improvement of at least 90% compared to PolyCNN and EffCNN. It also showcases improvements ranging from 26 to 90% compared to LSTMNet. Moreover, in terms of training stability, EstraNet performs better than the other models as PolyCNN and EffCNN fail to reach the guessing entropy once out of the three training trials, and LSTMNet performs poorly in terms of the average T_{GE1}. On the ASCADr dataset, EstraNet exhibits substantial improvements of 85 to 90% compared to PolyCNN and LSTMNet for all three metrics. It also demonstrates an improvement of 10 to 60% compared to EffCNN. In the CHES20 dataset, none of the PolyCNN and EffCNN models could bring down T_{GE1} below $5K$. Although EstraNet performs slightly worse than LSTMNet in terms of the best and median T_{GE1} values on this dataset, it demonstrates more stability with an average T_{GE1} of around 20. In contrast, in one of the three training trials, LSTMNet failed to bring down the T_{GE1} below $5K$. In summary, it can be concluded that EstraNet offers significant improvements (up to 90%) over the benchmark models on the ASCADf and ASCADr datasets. Though LSTMSNet performs slightly better than EstraNet in terms of the best and median T_{GE1} on the CHES20 dataset, its performance is significantly unstable compared to EstraNet. Figure 10 visually depicts the median guessing entropy of different methods with respect to the number of attack traces for attack desync 1000, further confirming the observations of Table 6.

6.8 Experimental Results in the Presence of Clock Jitter Effect

This section presents a comparative analysis of EstraNet with the benchmark models on three datasets with added clock jitter effect[9]. We pre-processed the traces using the

[9] It is worth noting that prior works [CDP17, WP20] have examined the effectiveness of DLSCA against the clock jitter effect. However, the work of [WP20] assumes a slightly different attack setup where the



(a) ASCADf dataset.
(b) ASCADr dataset.
(c) CHES20 dataset.

Figure 10: Plots of the guessing entropy vs. the number of attack traces for the experiments with trace length $40K$. The attack traces are desynchronized by a maximum displacement of 1000.

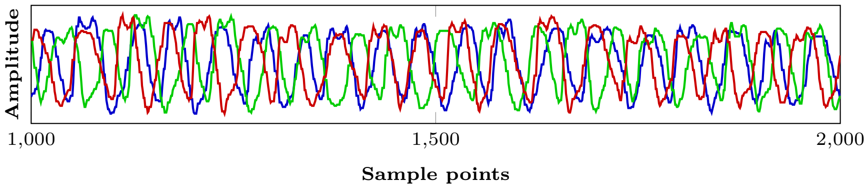


Figure 11: Sample traces after adding clock jitter effect.

approach proposed in [CDP17, WP20] to introduce the clock jitter effect into the datasets. Appendix E provides the details of the algorithm employed to add the clock jitter effect. Figure 11 illustrates several sample traces after adding the clock jitter effect. As in the previous experiments, the DL models in this section were trained with data augmentation, where each profiling trace was independently desynchronized by a maximum displacement of 200 during training. Also, as in the previous experiments, we independently trained each DL model three times. Table 7 presents the best, median, and average of T_{GE1} (the minimum number of attack traces required to achieve guessing entropy 1) values obtained in the three training trials. It should be noted that the elastic alignment technique [vWWB11] is a well-known method used to align traces affected by the clock jitter. Therefore, for each dataset, we trained the benchmark models on the misaligned traces (i.e., traces obtained after adding the clock jitter effect) and the traces obtained after aligning the misaligned traces using elastic alignment. Both sets of results are presented in Table 7.

Analysis of the table reveals that on the ASCADf dataset, both PolyCNN and EffCNN perform poorly on the misaligned traces, although their performance improves significantly after elastic alignment. However, they still require 5 to 12 times more traces to reach guessing entropy 1 compared to EstraNet. In contrast, none of the LSTMNet models are able to reach guessing entropy 1 using $5K$ traces even after elastic alignment. On the ASCADr dataset, both PolyCNN and LSTMNet exhibit poor performance. With one exception, none of the models can reach guessing entropy 1 using $5K$ traces. Though EstraNet fails to reach guessing entropy 1 in one of three training trials on the dataset, it shows an improvement of almost 70% over EffCNN in terms of the best T_{GE1}. For the CHES20 dataset, both PolyCNN and LSTMNet perform poorly, as all of their models require either more than or close to $5K$ traces to reach the guessing entropy 1. Although the best EffCNN model requires 576 traces to reach guessing entropy 1, its other models require either close to or greater than $5K$ traces for the same. In contrast, all EstraNet

adversary possesses a clean trace alongside each noisy trace in the dataset. In contrast, we consider a weaker adversary with no clean traces as in [CDP17].

Table 7: The minimum number of attack traces (lesser is better) required to reach guessing entropy 1 by EstraNet and the benchmark models on the datasets with the added clock jitter effect. The column titled *Elastic Alignment* indicates whether the traces have been aligned using elastic alignment [vWWB11] prior to perform the attack. The columns titled *Best*, *Med.*, and *Avg.* respectively show the best, median and average results of three independently trained models. The '-' entries in the table indicate that the average value is not available as some of the independently trained models failed to reach guessing entropy 1 using $5K$ attack traces.

Dataset	Model	Elastic	T_{GE1}		
		Alignment	Best	Med.	Avg.
ASCADf	PolyCNN	No	610	$> 5K$	–
		Yes	219	412	350.3
	EffCNN	No	1814	2384	2327.7
		Yes	345	394	562.3
	LSTM	No	$> 5K$	$> 5K$	–
		Yes	$> 5K$	$> 5K$	–
	EstraNet	No	42	42	48.3
ASCADr	PolyCNN	No	$> 5K$	$> 5K$	–
		Yes	363	$> 5K$	–
	EffCNN	No	1422	1540	2148.3
		Yes	405	458	449.0
	LSTM	No	$> 5K$	$> 5K$	–
		Yes	$> 5K$	$> 5K$	–
	EstraNet	No	117	566	–
CHES20	PolyCNN	No	$> 5K$	$> 5K$	–
		Yes	$> 5K$	$> 5K$	$> 5K$
	EffCNN	No	576	4651	–
		Yes	$> 5K$	$> 5K$	$> 5K$
	LSTM	No	$> 5K$	$> 5K$	–
		Yes	3708	$> 5K$	–
	EstraNet	No	26	28	32.0

models require only 25 to 40 traces to reach the guessing entropy 1. In summary, the benchmark models fail to reach guessing entropy 1 using $5K$ attack traces in the majority of cases, and even in those cases they perform well, their performance is an order of magnitude worse than that of EstraNet.

Finally, we would like to highlight that in the presence of clock jitter, the relative distances between the POIs fluctuates significantly across different traces. Although the design of GaussiP attention assumes constant relative distances between the POIs in all traces, EstraNet demonstrates better robustness to such fluctuations compared to the benchmark models. This robustness can be attributed to two key factors. Firstly, GaussiP attention uses Gaussian attention which is resilient to minor variations in relative distances. For instance, if an attention head assigns significant attention to a trace segment $[s, t]$, the attention output can remain almost same even if some POIs shift their positions within the segment. Secondly, in EstraNet, the input traces propagate though multiple convolutional and average-pooling layers before reaching the first GaussiP attention layer. As a result, some of the fluctuations in relative distances are absorbed during the propagation through the average-pooling layers. Indeed, the above experimental results indicate that EstraNet achieves superior robustness to fluctuations in the relative distances introduced by the

Table 8: Comparison of the use of softmax-attention and global average-pooling in EstraNet. Both models have been trained thrice on the ASCADf dataset using the setup of Section 6.7. We report the number of attack traces required to reach the guessing entropy 1 in the three training runs of the two models.

Model	Attack Desync 600			Attack Desync 1000		
	1	2	3	1	2	3
EstraNet with Softmax-attention	22	26	21	26	28	22
EstraNet with global average-pooling	36	101	1734	39	110	1327

clock jitter effect compared to the benchmark models.

6.9 Ablation Study

The EstraNet model incorporates two novel layers: the GaussiP attention layer and the layer-centering normalization layer. Additionally, EstraNet utilizes a softmax-attention layer. This section aims to explore the impact of integrating these layers into EstraNet and determine their contribution to its improved performance.

6.9.1 Ablation study of softmax-attention

In this section, we assess the performance of EstraNet without using any softmax-attention altogether. Thus, we replace the softmax-attention with a global average-pooling layer, keeping all other hyper-parameters same. Table 8 presents a comparison between EstraNet with global average-pooling and the vanilla EstraNet (i.e., EstraNet with softmax-attention) on the ASCADf dataset using an attack window size of $40K$. Note that, as before, both models were trained three times. The table displays the minimum number of attack traces required to reach the guessing entropy of 1 (denoted as T_{GE1}) obtained from the three training runs for each model.

From the table, we observe that the T_{GE1} for EstraNet with softmax-attention ranges from 21 to 28. In contrast, for EstraNet with global average-pooling, it reaches as high as 1734 in some training runs. This indicates that employing softmax-attention instead of global average-pooling in EstraNet leads to a significant improvement in its performance.

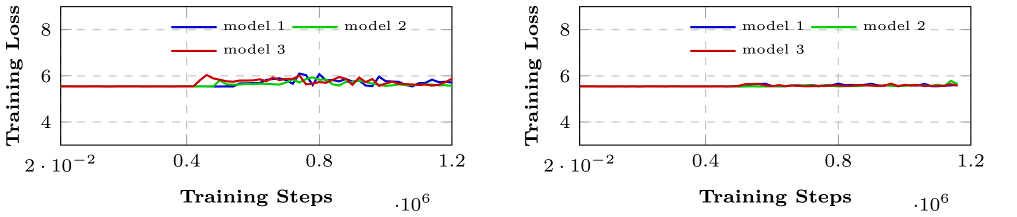
6.9.2 Ablation study of layer-centering layer

This section focuses on assessing the impact of the newly introduced layer-centering layer in EstraNet. Toward that goal, we conducted experiments using two other normalization methods, namely layer normalization and batch normalization. We also included the results of EstraNet without any normalization, as [HSAM22] reported its effectiveness in TransNet. The pool size hyper-parameter was varied for each normalization method, and the attack performance was evaluated. Each experiment was repeated three times, and the results are presented in Table 9.

From the table, it can be observed that EstraNet with layer-centering maintains consistent performance across different pool sizes. Conversely, for other normalization methods or without any normalization, the performance of EstraNet deteriorates significantly as the pool size decreases from 8 to 4. Moreover, in the case of batch normalization, EstraNet fails to achieve the guessing entropy of 1 using considerably fewer than $5K$ attack traces for all pool sizes. These results highlight the robustness of EstraNet when using layer-centering compared to other alternatives.

Table 9: Attack results of EstraNet using various normalization methods. All models have been independently trained thrice on the ASCADf dataset using the setup of Section 6.7. We report the number of attack traces required to reach the guessing entropy 1 in the three training runs of each model.

Normalization Method	Pool size	Attack Desync 600			Attack Desync 1000		
		1	2	3	1	2	3
Layer Centering	8	22	26	21	26	28	22
	6	23	28	17	23	27	17
	4	25	23	35	24	22	35
No Normalization	8	33	34	29	42	28	21
	6	18	51	313	37	33	381
	4	117	899	$> 5K$	87	1045	$> 5K$
Layer Normalization	8	483	2380	53	274	1735	57
	6	4635	67	3854	3612	80	3596
	4	$> 5K$	140	$> 5K$	$> 5K$	133	$> 5K$
Batch Normalization	8	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$
	6	$> 5K$	4500	$> 5K$	$> 5K$	4138	$> 5K$
	4	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$



(a) EstraNet models with self-attention of [HSGB21]. (b) EstraNet models with self-attention of [LCW+21].

Figure 12: Training loss vs. training steps for EstraNet with its self-attention replaced by the self-attentions proposed in [HSGB21] (Figure 12a) and [LCW+21] (Figure 12b).

6.9.3 Ablation study of the proposed GaussiP attention layer

This section assesses the performance of EstraNet by substituting the proposed GaussiP attention layer in EstraNet with alternative self-attention layers that incorporate relative positional encoding and have linear complexity. In the TN literature, several such self-attention layers have been introduced (as discussed in Section 3.3). Due to computational constraints, it is not feasible to verify all of these options. Hence, we select the self-attentions proposed in [LCW+21] and [HSGB21] as alternatives to GaussiP attention. We employ the same training setup to train the modified EstraNet models with the GaussiP attentions replaced by the ones proposed in [LCW+21] and [HSGB21]. Figure 12 displays the training loss of the modified EstraNet models over the course of the initial 1.2 million training steps. The figures demonstrate that the training loss of the EstraNet models with alternative self-attention layers does not even start to decrease even after 1.2 million training steps, indicating the challenge of training these models effectively. It is worth noting that it might be possible to adopt the existing self-attention layers in the context of SCA through non-trivial modifications. Nonetheless, the results presented in Figure 12 suggest that such adaptations is subject to intense research.

6.10 Choice of the Hyper-parameters

In our experiments, we found that setting most of the hyper-parameters of EstraNet to their default values provides good results. However, selecting the appropriate value for a few hyper-parameters is crucial for its good performance. This section illustrates the choice of those hyper-parameters.

6.10.1 Choice of number of heads in GaussiP attention layer

In all our previous experiments, we found that setting the number of heads, denoted as H, in the GaussiP attention layer to the default value of 8 resulted in good performance. In this section, we aim to evaluate the sensitivity of EstraNet's performance to the choice of H. To accomplish this, we trained EstraNet models using different values of H on the ASCADf datasets while keeping the remaining hyperparameters at their default values. Similar to earlier experiments, we trained three independent EstraNet models for each value of H and report the T_{GE1} (the number of attack traces required to reach guessing entropy 1) in Table 10.

The table shows that the performance of EstraNet is significantly unstable for smaller values of H, such as 4 and 6. More precisely, for $H = 4$ and $H = 6$, EstraNet requires up to 367 and over $5K$ attack traces, respectively, to reach guessing entropy 1, whereas, for the default $H = 8$, it requires fewer than 30 traces in the all the three training trials. This unstable behavior of EstraNet can be attributed to the limited flow of information to distant sample points when using smaller values of H. Indeed, as explained in Section 4.1.3, increasing the number of heads in the GaussiP attention layer allows for more information flow from one sample point to distant sample points. Therefore, EstraNet performs very well with significantly larger values of H, such as 8 and 10. However, the performance slightly deteriorates for $H = 12$. We attribute this deterioration in performance to the increased number of parameters in the GaussiP attention layer when using a very large value of H. In conclusion, these experiments reveal that there exists an optimal range for the choice of H in EstraNet. In our experiments, we have found that an H value within the range of $8 - 10$ consistently performs well across the datasets.

Table 10: Attack results of EstraNet for the different values of H (the number of heads in GaussiP attention) hyper-parameters. As before, for each value of H, the model has been independently trained thrice on the ASCADf dataset using the setup of Section 6.7. We report the number of attack traces required to reach the guessing entropy 1 in the three training runs.

H	Attack Desync 600			Attack Desync 1000		
	1	2	3	1	2	3
4	367	23	11	226	29	13
6	12	$> 5K$	29	14	$> 5K$	25
8	23	28	17	23	27	17
10	17	17	22	14	18	20
12	22	65	28	27	53	28

6.10.2 Choice of distance based scaling in GaussiP attention

The selection of the distance-based scaling hyper-parameter, denoted as β_2 in Table 1, plays a critical role in achieving satisfactory performance with EstraNet. Table 11 presents the experimental results of attacking the ASCADf dataset using an attack window size of $10K$ for four distinct values of β_2 in EstraNet. The findings demonstrate the sensitivity of EstraNet's performance to the choice of β_2. Specifically, for $\beta_2 = 10$ and 300, EstraNet

Table 11: The minimum number of attack traces (lower is better) required to reach the guessing entropy 1 on the ASCADf dataset with attack window size $10K$ by EstraNet for different values of β_2. For each experiment, the average result of three independently trained models is reported.

β_2	Attack Desync		
	0	200	400
300	$> 5K$	$> 5K$	$> 5K$
150	19.3	15.0	15.7
50	70.0	67.3	64.0
10	$> 5K$	$> 5K$	$> 5K$

Table 12: The minimum number of attack traces (lower is better) required to reach the guessing entropy 1 on the ASCADf dataset by EstraNet for different values of d_e. The columns titled *Best*, *Med.*, and *Avg.* respectively show the best, median and average results of three independently trained models.

d_e	Attack Desync 0			Attack Desync 200			Attack Desync 400		
	Best	Med.	Avg.	Best	Med.	Avg.	Best	Med.	Avg.
512	14	22	19.3	8	18	15.0	9	17	15.6
256	13	15	16.3	9	11	13.0	12	14	14.3
128	12	14	14.0	11	12	11.7	12	14	13.7

fails to attain a guessing entropy of 1 even with as many as $5K$ traces, whereas it achieves its optimal performance with $\beta_2 = 150$. Notably, with a trace length (attack window size) of $10K$, EstraNet performs most effectively with $\beta_2 = 150$ on the ASCADf and CHES20 datasets. However, for the ASCADr dataset, the optimal performance is observed with $\beta_2 = 50$. In general, for the trace length $10K$, we observed that EstraNet exhibits favorable performance within the range of β_2 values between 50 and 150. Furthermore, our investigations reveal that as the trace length increases from $10K$ to $40K$, the range of β_2 values associated with EstraNet's optimal performance increases. For instance, for the attack window size of $40K$, the preferred β_2 values on the ASCADr and CHES20 datasets are 200 and 450, respectively. Notably, these values are respectively 4 and 3 times that of the optimal values obtained for the attack window size of $10K$. In conclusion, we recommend tuning the β_2 hyper-parameter based on some validation data. We also want to mention that the performance of EstraNet generally improves as the value of β_2 gets larger. However, setting β_2 to a too-large value makes the EstraNet model untrainable. In other words, while training an EstraNet model with a very large β_2, the training loss does not fall below the level of random loss. Therefore, we suggest gradually increasing the value of β_2 during the tuning process until the model becomes untrainable.

6.10.3 Choice of the feature map dimension of GaussiP attention kernel

In all previous experiments, we have utilized a feature map dimension of 512 for the GaussiP attention kernel (referred to as d_e in Table 1), which is significantly large. However, employing a large value of d_e significantly increases the memory and compute requirements for training the EstraNet model. Therefore, this section aims to assess the performance of EstraNet when using smaller values of d_e. Table 12 presents a performance comparison between EstraNet with $d_e = 512$ and EstraNet with reduced feature map dimensions: 256 and 128. The experimental results indicate that the performance of EstraNet with reduced feature map dimensions is comparable to that achieved with $d_e = 512$. These findings suggest that the performance of EstraNet remains stable for a wide range of d_e, indicating the possibility of making EstraNet more memory and compute efficient using a smaller d_e.

Table 13: Training time of the benchmark models compared to EstraNet.

Dataset	EstraNet	PolyCNN	EffCNN	LSTMNet
ASCADf	1.0x	5.9x	7.0x	6.3x
ASCADr	1.0x	4.5x	12.8x	4.8x
CHES20	1.0x	3.0x	5.7x	0.9x

6.11 Training Time: Comparison with the Benchmarks

This section presents a comparative analysis of the training times between EstraNet and the benchmark models. Table 13 provides the comparison for the trace length $10K$. Upon examining the table, it is evident that the training time of PolyCNN is three to six times longer than that of EstraNet. Similarly, the training time of EffCNN is six to thirteen times greater compared to EstraNet. On the other hand, the training time of LSTMNet is approximately five to six times larger on the ASCADf and ASCADr datasets, while it is almost the same on the CHES20 dataset. Hence, with few exceptions, the training of EstraNet is approximately three to thirteen times faster than the benchmark models. Note that training EstraNet may involve tuning certain hyperparameters (β_2 in particular). However, given that the training time of a single EstraNet model is significantly smaller than that of the benchmark models, the hyperparameter tuning process can be completed in a comparable time to the training time of the benchmark models.

6.12 Shift-invariance: Dependence on Data Augmentation

The shift-invariance of a DL model is defined for infinite-length input [HSGB21]. However, when dealing with inputs of finite length, the model's shift-invariance tends to break down near the input boundaries. Additionally, incorporating subsampling layers (e.g., average-pooling layer) into a DL model further diminishes its shift-invariance [Zha19, HSAM22]. The reduced shift invariance could lead to overfitting during the model's training. Moreover, a model that exhibits greater shift-invariance is inherently less reliant on desynchronization in profiling traces and data augmentation [HSAM22]. This section evaluates the impact of the employed data augmentation on the shift-invariance of EstraNet. Toward that goal, we repeated the experiments of Section 6.6. However, this time, the models are trained without any data augmentation. We compare the performance of EstraNet while trained with and without data augmentation in Table 14.

The findings of Table 14 reveal that, on the ASCADr and CHES20 datasets, the performance of EstraNet remains almost identical whether or not data augmentation is used during training. Conversely, on the ASCADf dataset, the model's performance significantly declines when trained without data augmentation. This observed drop in performance on the ASCADf dataset can be attributed to the relatively lesser number of the profiling traces it contains, amounting to only $50K$ as opposed to the $200K$ profiling traces present in both the ASCADr and CHES20 datasets. The limited training samples cause EstraNet to overfit while trained without data augmentation, resulting in significantly poorer performance than the model trained with data augmentation.

Table 17 of Appendix F compares the shift-invariance of EstraNet with that of the benchmark models. Like EstraNet, the performance of the benchmark models also vastly deteriorates on the ASCADf dataset while the deterioration is marginal on the ASCADr and CHES20 datasets. While comparing the performance of EstraNet to the benchmark models in the absence of data augmentation, EstraNet performs significantly better on the ASCADr and CHES20 datasets. On the ASCADf dataset, PolyCNN and LSTMNet fail to reach guessing entropy 1 using $5K$ attack traces while EstraNet and EffCNN shows similar performance.

Table 14: EstraNet's performance with and without data augmentation. The rest of the experimental setup is the same as in Section 6.6.

Dataset	Attack	w Data Augmentation			w/o Data Augmentation		
	Desync	Best	Med.	Avg.	Best	Med.	Avg.
ASCADf	0	13	15	14.3	79	390	286.7
	200	12	13	12.7	139	350	319.3
	400	9	13	12.3	124	311	308.0
ASCADr	0	5	5	5.0	6	7	7.0
	200	5	6	5.7	6	6	6.0
	400	4	5	5.0	5	6	6.3
CHES20	0	5	6	6.7	4	4	7.3
	200	5	6	6.7	3	6	7.0
	400	4	7	7.3	4	4	6.3

7 Limitations and Future Works

Comparing the results for the trace length of $40K$ (Table 6) with those for the trace length of $10K$ (Table 5), it is evident that the performance of EstraNet slightly deteriorates with the increase in the trace length beyond $10K$ though the deterioration is significantly less compared to the other DL models. To investigate the decline in EstraNet's performance on longer traces further, we conducted additional experiments utilizing the ASCADf dataset with a trace length of $60K$ sample points. For the experiments, we employed the attack setup of Section 6.7. The results of these experiments are provided in Table 15. Upon examination of Table 15, it becomes evident that the performance of EstraNet has deteriorated significantly as compared to the performance for the trace lengths of $10K$ (Table 5) and $40K$ (Table 6) though the performance is significantly better than the benchmark DL models. Therefore, improving EstraNet's performance further on longer traces (e.g., with a trace length $> 40K$) can be an important research direction to explore.

Table 15: The minimum number of traces (lesser is better) required to reach guessing entropy 1 by the DL models on the ASCADf dataset for trace length $60K$. The columns titled *Best*, *Med.*, and *Avg.* respectively show the best, median, and average results of the three independently trained models. The '-' entries in the table indicate that the average value is not available as some of the independently trained models failed to reach guessing entropy 1 using $5K$ attack traces.

Model	Attack Desync 600			Attack Desync 1000		
	Best	Med.	Avg.	Best	Med.	Avg.
PolyCNN	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–
EffCNN	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–
LSTM	4809	$> 5K$	–	4391	$> 5K$	–
EstraNet	111	3342	–	131	3771	–

The resilience of a DL model to low SNR traces constitutes a pivotal requirement for ensuring the model's efficacy across a wide spectrum of datasets. To assess the robustness of EstraNet on low SNR datasets, we conducted further experiments on the ASCADf dataset by adding Gaussian noise with standard deviations (std.) of 2 and 4. The addition of Gaussian noise reduces the peak SNR by approximately 30% and 60%, respectively. These experiments were conducted on traces of length $10K$, employing the experimental setup of Section 6.6. The resulting findings are presented in Table 16. By comparing the results of EstraNet in Table 16 with those in Table 5, we observe that the performance of EstraNet has deteriorated marginally for noise std. 2. However, the performance is significantly deteriorated for noise std. 4, though the performance is remarkably better than the benchmark models. Nonetheless, whether the performance of EstraNet can be improved further on low SNR scenarios by using a better training method or improving the model architecture can be an interesting future work.

Table 16: The minimum number of traces (lesser is better) required to reach guessing entropy 1 by the DL models on the ASCADf dataset with added Gaussian noise. The columns titled *Best*, *Med.*, and *Avg.* respectively show the best, median, and average results of the three independently trained models. The '-' entries in the table indicate that the average value is not available as some of the independently trained models failed to reach guessing entropy 1 using $5K$ attack traces.

Noise Std.	Model	Attack Desync 0			Attack Desync 400		
		Best	Med.	Avg.	Best	Med.	Avg.
2	PolyCNN	736	881	947.3	382	407	459.0
	EffCNN	307	394	528.3	139	210	198.7
	LSTM	342	935	1267.7	381	849	973.7
	EstraNet	26	45	39.0	27	39	39.7
4	PolyCNN	3322	4080	3915.7	1847	3180	–
	EffCNN	3509	3950	3862.7	2098	2360	2405.3
	LSTM	3835	4878	–	2539	2716	–
	EstraNet	341	624	731.0	482	559	630.0

The experimental results of Sections 6.6 and 6.7 reveal that EstraNet exhibits significant performance improvements over the benchmark models when faced with random delay or clock jitter countermeasure in combination with masking countermeasure. In the presence of random delay or masking countermeasures, the relative distances between the POIs remain approximately the same. However, in the presence of countermeasures such as random delay interrupt [CK09] and shuffling [HOM06], the relative distances between the POIs vary drastically. It would also be interesting to investigate the adaptation of the TN-based model against these countermeasures. It is worth mentioning that existing research [DRS$^+$12, WP20] has demonstrated that these countermeasures can significantly deteriorate the performance of DL models. However, they have also introduced the Hidden Markov Model and autoencoder-based methods to denoise traces protected by such countermeasures, albeit in weaker settings. Similar approaches could be explored in conjunction with EstraNet to make it robust to these countermeasures.

When comparing EstraNet with the benchmark models in terms of their attack performance on desynchronized traces, EstraNet demonstrates a remarkable improvement with up to 90% reduction in the number of attack traces required to reach the guessing entropy 1. Additionally, EstraNet showcases significant improvements over the benchmark models on the datasets with clock jitter effects. On such scenarios, while the benchmark models often struggle to reach the guessing entropy 1 even with $5K$ attack traces, EstraNet models can reach it using less than 100 traces most of the time. However, it is important to note that this does not imply that CNN-based and RNN-based models cannot be developed to perform well on such scenarios. Thus, developing CNN and RNN-based DL models to perform well against datasets with highly desynchronized traces and clock jitter effects can be explored in the future.

8 Conclusions

Deep learning (DL) models have shown great success in SCA; however, selecting an appropriate model architecture remains crucial in achieving good performance. This work introduces a novel DL model called EstraNet for SCA. EstraNet exhibits linear time and memory complexity, significantly improving quadratic time and memory complexity of the previously proposed TN-based model, TransNet. This linear complexity makes EstraNet applicable to traces with lengths exceeding $10K$. Moreover, EstraNet is shift-invariant, making it resilient to misalignments in the traces.

The EstraNet architecture incorporates two major contributions. First, we propose a novel attention operation called GaussiP attention, which has linear time and memory costs. By incorporating relative positional encoding within the attention operation, EstraNet achieves shift-invariance. Additionally, the sparsity of attention probabilities in GaussiP attention makes it particularly suitable for SCA. Second, due to the limitations of standard normalization techniques, such as batch normalization and layer normalization for EstraNet, we introduce a novel normalization method called layer-centering. The proposed normalization method improves the stability of EstraNet training significantly.

We conducted extensive experimental evaluations of EstraNet on three datasets consisting of masked implementations. We introduced random displacements to the datasets to assess EstraNet's robustness against random delay countermeasure. The results demonstrate that EstraNet achieves up to 90% decrease in the number of attack traces required to reduce the guessing entropy to 1 compared to three benchmark models. When comparing the attack performance of EstraNet with the benchmark models on datasets incorporating clock jitter effects, EstraNet provides up to an order of magnitude reduction in the number of attack traces required to reach guessing entropy 1. Additionally, we investigated the impact of various hyperparameters on EstraNet's performance. Overall, the experimental findings highlight EstraNet's potential as a promising avenue for advancing SCA research.

Acknowledgments

The authors express their gratitude to the anonymous reviewers for their valuable insights and suggestions that greatly contributed to the enhancement of this paper. Additionally, they would like to acknowledge the Prime Minister's Research Fellowship, India for funding the first author and the computing infrastructure. Furthermore, the authors are pleased to thank the Department of Science and Technology (DST), Government of India, the IHUB NTIHAC Foundation at C3i Building, IIT Kanpur, and the Centre on Hardware-Security Entrepreneurship Research & Development (HERD), Meity, India, for partially supporting additional lab infrastructure.

References

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES, USA, 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[BPS$^+$20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.*, 10(2):163–188, 2020.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *CHES, Taiwan, 2017*, volume 10529 of *LNCS*, pages 45–68. Springer, 2017.

[CG00] Jean-Sébastien Coron and Louis Goubin. On boolean and arithmetic masking against differential power analysis. In *CHES, USA, 2000*, volume 1965 of *LNCS*, pages 231–237. Springer, 2000.

[Che21] Peng Chen. PermuteFormer: Efficient relative position encoding for long sequences. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *EMNLP 2021, Dominican Republic, 2021*, pages 10606–10618. ACL, 2021.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *CHES, Switzerland, 2009*, volume 5747 of *LNCS*, pages 156–170. Springer, 2009.

[CLD$^+$21] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *ICLR 2021, Virtual Event, Austria, May 3-7, 2021.* OpenReview.net, 2021.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES, USA, 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.

[CRW17] Krzysztof Marcin Choromanski, Mark Rowland, and Adrian Weller. The unreasonable effectiveness of structured random orthogonal embeddings. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *NIPS 2017, December 4-9, 2017, USA*, pages 219–228, 2017.

[DRS$^+$12] François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon. Efficient removal of random delays from embedded software implementations using hidden markov models. In Stefan Mangard, editor, *CARDIS, Austria*, volume 7771 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2012.

[DYY$^+$19] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL, Italy, 2019*, pages 2978–2988. Association for Computational Linguistics, 2019.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES, USA, 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.

[GZL19] Maosheng Guo, Yu Zhang, and Ting Liu. Gaussian transformer: A lightweight approach for natural language inference. In *AAAI 2019, Hawaii*, pages 6489–6496. AAAI Press, 2019.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS, Singapore, 2006*, volume 3989 of *LNCS*, pages 239–252, 2006.

[HSAM22] Suvadeep Hajra, Sayandeep Saha, Manaar Alam, and Debdeep Mukhopadhyay. TransNet: Shift invariant transformer network for side channel analysis. In Lejla Batina and Joan Daemen, editors, *AFRICACRYPT 2022, Fes, Morocco, 2022, Proceedings*, LNCS, pages 371–396. Springer Nature Switzerland, 2022.

[HSGB21] Max Horn, Kumar Shridhar, Elrich Groenewald, and Philipp F. M. Baumann. Translational equivariance in kernelizable attention. *CoRR*, abs/2102.07680, 2021.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO, USA, 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[KP22] Sengim Karayalcin and Stjepan Picek. Resolving the doubts: On the construction and use of resnets for side-channel analysis. *IACR Cryptol. ePrint Arch.*, page 963, 2022.

[KVPF20] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 2020.

[LCW$^+$21] Antoine Liutkus, Ondrej Cífka, Shih-Lun Wu, Umut Simsekli, Yi-Hsuan Yang, and Gaël Richard. Relative positional encoding for transformers with linear complexity. In Marina Meila and Tong Zhang, editors, *ICML 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7067–7079. PMLR, 2021.

[LLC$^+$21] Shengjie Luo, Shanda Li, Tianle Cai, Di He, Dinglan Peng, Shuxin Zheng, Guolin Ke, Liwei Wang, and Tie-Yan Liu. Stable, fast and accurate: Kernelized attention with relative positional encoding. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *NeurIPS 2021, 2021, virtual*, pages 22795–22807, 2021.

[LSL$^+$21] Yang Li, Si Si, Gang Li, Cho-Jui Hsieh, and Samy Bengio. Learnable fourier features for multi-dimensional spatial positional encoding. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *NeurIPS 2021, 2021, virtual*, pages 15816–15829, 2021.

[LZC$^+$21] 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. *TCHES*, 2021(3):235–274, 2021.

[Mag19] Houssem Maghrebi. Deep learning based side channel attacks in practice. *IACR Cryptol. ePrint Arch.*, 2019:578, 2019.

[MBC$^+$20] Loïc Masure, Nicolas Belleville, Eleonora Cagli, Marie-Angela Cornelie, Damien Couroussé, Cécile Dumas, and Laurent Maingault. Deep learning side-channel analysis on large-scale traces - A case study on a polymorphic AES. In *ESORICS, UK, 2020*, volume 12308 of *LNCS*, pages 440–460. Springer, 2020.

[MCLS23] Loïc Masure, Valence Cristiani, Maxime Lecomte, and François-Xavier Standaert. Don't learn what you already know: Scheme-aware modeling for profiling side-channel analysis against masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):32–59, 2023.

[MHM13] Zdenek Martinasek, Jan Hajny, and Lukas Malina. Optimization of power analysis using neural network. In *CARDIS, Germany, 2013*, volume 8419 of *LNCS*, pages 94–107. Springer, 2013.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *SPACE, India, 2016*, volume 10076 of *LNCS*, pages 3–26. Springer, 2016.

[MZ13] Zdenek Martinasek and Vaclav Zeman. Innovative method of the power analysis. *Radioengineering*, 22(2):586–594, 2013.

[PPY$^+$21] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. Random feature attention. In *ICLR 2021, Virtual Event, Austria, 2021.* OpenReview.net, 2021.

[PSK$^+$18] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *SPACE, India, 2018*, volume 11348 of *LNCS*, pages 157–176. Springer, 2018.

[PWP21] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *Cryptology ePrint Archive*, 2021.

[RR07] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS, British Columbia, Canada, 2007*, pages 1177–1184. Curran Associates, Inc., 2007.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *CHES, UK, 2005*, volume 3659 of *LNCS*, pages 30–46. Springer, 2005.

[SLP$^+$21] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021.

[SUV18] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *NAACL-HLT, USA, 2018*, pages 464–468. Association for Computational Linguistics, 2018.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS, USA, 2017*, pages 5998–6008, 2017.

[vWWB11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In Aggelos Kiayias, editor, *CT-RSA, USA, 2011*, volume 6558 of *LNCS*, pages 104–119. Springer, 2011.

[Wik22] Kernel method — Wikipedia, the free encyclopedia, 2022. [Online; accessed 29-August-2022].

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

[XYH+20] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *ICML, 2020*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020.

[YSC+16] Felix X. Yu, Ananda Theertha Suresh, Krzysztof Marcin Choromanski, Daniel N. Holtmann-Rice, and Sanjiv Kumar. Orthogonal random features. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *NIPS 2016, December 5-10, 2016, Spain*, pages 1975–1983, 2016.

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

[Zha19] Richard Zhang. Making convolutional networks shift-invariant again. In *ICML, 2019, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7324–7334. PMLR, 2019.

[ZLLS20] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020. https://d2l.ai.

[ZS20] Yuanyuan Zhou and François-Xavier Standaert. Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks. *J. Cryptogr. Eng.*, 10(1):85–95, 2020.

[ZXF+19] Libang Zhang, Xinpeng Xing, Junfeng Fan, Zongyue Wang, and Suying Wang. Multi-label deep learning based side channel attack. In *AsianHOST, China, 2019*, pages 1–6. IEEE, 2019.

[ZXF+21] Libang Zhang, Xinpeng Xing, Junfeng Fan, Zongyue Wang, and Suying Wang. Multilabel deep learning-based side-channel attack. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(6):1207–1216, 2021.

Appendices

A Approximation of GaussiP Attention's Normalization Factor

In Eq (17), we have used the following approximation:

$$\sum_{j=0}^{n-1} k_{GPA}(i-j) = \sum_{j=0}^{n-1} \exp\left(-\frac{\beta_2^2 s_p^2 (i-j-c_p n)^2 ||\mathrm{W}_p \mathbf{p}||_2^2}{2}\right) \approx \frac{1}{\beta_2 s_p ||\mathrm{W}_p \mathbf{p}||}$$

In this section, we provide a justification for the same. Toward that goal, let us define the shifted kernel function as follows:

$$\bar{k}(j) = k_{GPA}(j + c_p n) = \exp\left(-\frac{\beta_2^2 s_p^2 ||\mathrm{W}_p \mathbf{p}||_2^2 j^2}{2}\right)$$

Thus, the shifted kernel $\bar{k}(j)$ attains its maximum at $j = 0$ and decreases monotonically as j goes to either $+\infty$ or $-\infty$. Thus, we have

$$\begin{aligned}
\sum_{j=0}^{n-1} k_{GPA}(i-j) \quad &= \quad \sum_{j=0}^{n-1} \bar{k}(i-j-c_p n) \quad = \sum_{j=i-c_p n-n+1}^{i-c_p n} \bar{k}(j) \\
\approx \sum_{j=-\infty}^{\infty} \bar{k}(j) \quad &= \quad \bar{k}(0) + \sum_{j=-\infty}^{-1} \bar{k}(j) + \sum_{j=1}^{\infty} \bar{k}(j) \\
\approx \bar{k}(0) + \int_{j=-\infty}^{0} \bar{k}(j) + \sum_{j=0}^{\infty} \bar{k}(j) \ &= \ \bar{k}(0) + \int_{j=-\infty}^{\infty} \bar{k}(j) \ = \ 1 + \int_{j=-\infty}^{\infty} \bar{k}(j) \\
\approx \int_{j=-\infty}^{\infty} \bar{k}(j) \quad &= \quad \frac{\sqrt{2\pi}}{\beta_2 s_p ||\mathrm{W}_p \mathbf{p}||} \\
\approx \frac{1}{\beta_2 s_p ||\mathrm{W}_p \mathbf{p}||_2}
\end{aligned}$$

In the above derivation, there are four steps of approximations. Note that, in the last approximation step, we have ignored the constant factor $\sqrt{2\pi}$ of $\sqrt{2\pi}/\beta_2 s_p ||\mathrm{W}_p \mathbf{p}||_2$ as a small constant factor like 2 and 3 can be consumed in the hyper-parameter β_2. In the third approximation step, we have ignored the additive term 1 of $1 + \int_{j=-\infty}^{\infty} \bar{k}(j)$. The approximation can be justified as, for longer traces, $\int_{j=-\infty}^{\infty} \bar{k}(j) \gg 1$ holds. Similarly, the approximations in the third approximation step are also quite tight for longer traces. However, the first approximation i.e. the approximation of $\sum_{j=i-c_p n-n+1}^{i-c_p n} \bar{k}(j)$ by $\sum_{j=-\infty}^{\infty} \bar{k}(j)$ can be significantly loose in some scenarios. To justify the approximation, we will consider two scenarios. In the first scenario, let us assume that $i - c_p n - n + 1 \ll 0 \ll i - c_p n$. In that case

$$\sum_{j=i-c_p n-n+1}^{i-c_p n} \bar{k}(j) = \sum_{j=-\infty}^{\infty} \bar{k}(j) - \left(\sum_{j=-\infty}^{i-c_p n-n} \bar{k}(j) + \sum_{j=i-c_p n+1}^{\infty} \bar{k}(j) \right)$$

Since $\bar{k}(j)$ decays exponentially to 0 when j goes further away from 0, $\sum_{j=-\infty}^{i-c_p n-n} \bar{k}(j) + \sum_{j=i-c_p n+1}^{\infty} \bar{k}(j)$ remains close to 0 in this scenarios. In the second scenario, let us assume that $i - c_p n \ll 0$. Thus, in this scenario

$$\sum_{j=i-c_p n-n+1}^{i-c_p n} \bar{k}(j) \approx 0$$

as each $\bar{k}(j)$ for $j \in [-c_p n - n + 1, -c_p n]$ is close to 0. However, note that normalizing the attention scores by a value close zero can lead to the explosion of the attention scores which may lead to the unstable behaviour of the model. On the other hand, by approximating $\sum_{j=i-c_p n-n+1}^{i-c_p n} \bar{k}(j)$ by the upper limit $\sum_{j=-\infty}^{\infty} \bar{k}(j) \approx 1/\beta_2 s_p ||\mathrm{W}_p \mathbf{p}||_2$ avoids such problem. Overall, we find that approximating the normalizing constant $\sum_{j=0}^{n-1} k_{GPA}(i-j)$ by $1/\beta_2 s_p ||\mathrm{W}_p \mathbf{p}||_2$ leads to better stability of EstraNet.

B Plots of the Attention Probabilities in the GaussiP Attention Heads

Figure 13 plots the attention scores learned at different heads of the first layer's GaussiP attention operation of EstraNet on the ASCADr datasets. The value at i-th row and j-th column of the images indicates the attention scores from $\mathbf{q}_i$ to $\mathbf{k}_j$. From the plots, we observe that the attention scores are the same for (i, j)-pairs having the same $i - j$,

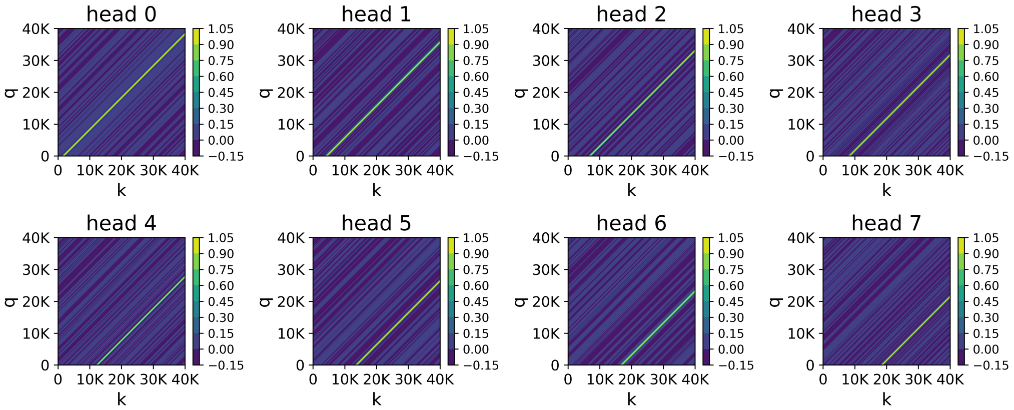


Figure 13: The contour plots of the attention scores learned at the eight different heads of the first layer's GaussiP attention operations of EstraNet on the ASCADf dataset. The attention probabilities are represented as a $4K \times 4K$ image A where $A[i, j]$ represents the attention from $\mathbf{q}_i$ to $\mathbf{k}_j$.

implying that the attention scores are the function of the relative distance of the i and j. Additionally, the attention scores at each attention head are high for a small range of relative distances and close to zero for the rest showing the sparsity of the attention scores.

C Detailed Architecture of the Benchmark Models

PolyCNN The PolyCNN model is a CNN model proposed in [MBC+20]. It has six convolutional blocks followed by a global average-pooling layer. Each convolutional block is composed of a convolutional, a batch-normalization, a ReLU, and an average-pooling layer. The number of features of the convolutional layers are respectively set to $10, 20, 40, 40, 80$ and 100. The kernel width of the first convolutional layer is set to 10 and to 11 for the rest. The pool size and stride of the average-pooling layer are set to 25 in the first convolutional block and to 5 for the rest. We have used this model for trace length $40K$. However, this model was not applicable to the smaller trace length $10K$ as the trace length becomes 1 after the first 5 convolutional blocks. Thus, we removed the one (third) convolutional block from the original PolyCNN model to use it for trace length $10K$.

EffCNN In [ZBHV20], Zaid et al. proposed a methodology for constructing CNN models robust to desynchronizations in the traces. Their models have three convolutional blocks followed by a flattening layer and three fully-connected layers. Each convolutional block is composed of a convolutional, a SELU activation, a batch-normalization, and an average-pooling layer. The number of features in the convolutional layers are respectively set to 32, 64 and 128. The kernel widths of the layers are respectively set to $1, T/2, n/I$ where n is the trace length, T is the maximum assumed amount of desynchronization, and I is the assumed number of POIs in the traces. Similarly, the pool sizes and the strides in the average-pooling layers are respectively set to $2, T/2, n/I$. For all experiments, we set I to 10. For the experiments of Sections 6.6 and 6.8, we set $T = 400$ and for the experiments of Section 6.7, we set $T = 1000$.

LSTMNet In [LZC+21], Lu et al. have proposed the LSTM-based models. The models for the desynchronized ASCADf and ASCADr datasets have six convolutional blocks, followed by bidirectional LSTM layers, followed by softmax-attention. Each convolutional block is

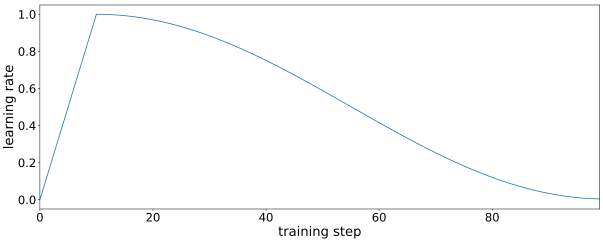


Figure 14: Plots of learning rate vs. training step for cosine decay with linear warmup learning rate schedule.

composed of a convolutional layer, a batch-normalization layer, an ELU activation, and an average-pooling layer. The number of filters of the convolutional layers are respectively set to $4, 8, 16, 32, 64$ and 128. The pool sizes of the average-pooling layers are all set to 2. The kernel width of all convolutional layers except the first one is set to 3. The kernel width of the first convolutional layer is respectively set to 26 and 63 for the ASCADf and ASCADr datasets. [LZC+21] has not provided any model for the CHES20 dataset. Thus, we use the model of the ASCADr dataset to perform the attacks on the CHES20 dataset.

D Learning Rate Schedule for EstraNet

In cosine decay with linear warmup learning rate scheduling, the learning rate is first linearly increased from 0 to some maximum value, say l_{max} over the first t_{warmup} steps. Then the learning rate is gradually decreased to some minimum value l_{min} over the remaining $t_{max} - t_{warmup}$ steps where l_{max} and l_{min} are respectively the maximum and minimum learning rate, t_{max} is the total number of training steps, and t_{warmup} is the number of warmup steps. Thus, the learning rate at t-th training step is given by

$$l_t = \begin{cases} \frac{t}{t_{warmup}} \times l_{max} & \text{for } t \leq t_{warmup} \\ l_{min} + \frac{l_{max} - l_{min}}{2} \times \left(1 + \cos\left(\frac{t - t_{warmup}}{t_{max} - t_{warmup}} \times \pi\right)\right) & \text{for } t > t_{warmup} \end{cases}$$

Figure 14 plots the learning rate vs training steps in cosine decay with linear learning rate schedule for $t_{max} = 100$, $t_{warmup} = 10$, $l_{max} = 1$ and $l_{min} = 0.004$.

E Adding Clock Jitter Effect

For adding clock jitter effect, we perform the pre-processing of each trace of the datasets as follows. For each sample points in the original trace we perform one of the following three actions each with one-third probability: a) we do not add it in the new trace, b) simply add it or c) add it along with another additional sample points with magnitude equal to the average of the sample point and the next sample point. Note that the above pre-processing is equivalent to adding clock jitter effect using the algorithm of [WP20] with the *clock_jitters_level* set to 1. However, after doing the above pre-processing, none of the DL models performed well[10] as many informative sample points were getting removed from the traces during the pre-processing making the informative sample points more sparser in the pre-processed traces. We circumvented the loss of informative sample points during the pre-processing as follows. Prior to adding the clock jitter effect in the traces, we double the number of sample points in the traces by repeating each sample points twice.

[10] In fact, on one dataset, only EstraNet worked significantly well. The rest of the methods were not working well.

Table 17: The minimum number of traces (lesser is better) required to reach guessing entropy 1 by EstraNet and the benchmark models for trace length $10K$. The models have been evaluated on attack traces with attack desync 0 (no desync), 200, and 400. The columns titled *Best*, *Med.*, and *Avg.* respectively show the best, median, and average results of three independently trained models. During the training of the models, no data augmentation is used.

Dataset	Model	Attack Desync 0			Attack Desync 200			Attack Desync 400		
		Best	Med.	Avg.	Best	Med.	Avg.	Best	Med.	Avg.
ASCADf	PolyCNN	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–
	EffCNN	277	440	384.0	257	288	298.0	730	839	1064.3
	LSTM	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–
	EstraNet	79	390	286.7	139	350	319.3	124	311	308.0
ASCADr	PolyCNN	14	19	21.3	11	17	17.0	48	84	93.0
	EffCNN	13	16	17.7	13	14	15.0	42	75	64.3
	LSTM	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–	$> 5K$	$> 5K$	–
	EstraNet	6	7	7.0	6	6	6.0	5	6	6.3
CHES20	PolyCNN	14	19	19.3	13	14	17.7	16	24	22.3
	EffCNN	12	22	23.0	13	27	23.0	22	26	28.3
	LSTM	6	17	14.0	5	14	11.7	7	16	13.3
	EstraNet	4	4	7.3	3	6	7.0	4	4	6.3

Note that such pre-processing is equivalent to doubling the sampling rate of the trace acquisition setup. Since, during the addition of clock jitter effect, at most one consecutive sample point of the original traces gets removed, by repeating each sample point of the original trace twice prior to adding clock jitter effect, we make sure that each sample point of the original traces appears at least once in the pre-processed traces while having the clock jitter affect at the same time.

F Ablation Study of Data Augmentation

In this Section, we investigate the impact of the data augmentation on the shift-invariance of the DL models. Toward that goal, we performed the experiments of Section 6.6 without any data augmentations. The resultant outcomes are detailed in Table 17. The findings show a significant decline in the performance of all models on the ASCADf dataset while the models are trained without data augmentations. Specifically, PolyCNN and LSTMNet were unable to reach the guessing entropy 1 even using $5K$ attack traces. The performance of EstraNet is similar to or significantly better than EffCNN on the dataset. The LSTMNet model fails to reach the guessing entropy 1 on the ASCADr dataset, although it performed closely to EstraNet on the CHES20 dataset. On the ASCADr and CHES20 datasets, EstraNet demonstrated slightly superior performance compared to PolyCNN and EffCNN. Overall, the better performance of EstraNet than the benchmark models suggests better shift-invariance of EstraNet architecture.