IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 3, pp. 235–274. DOI:10.46586/tches.v2021.i3.235-274

Pay Attention to Raw Traces: A Deep Learning Architecture for End-to-End Profiling Attacks

Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, Haining Lu

Shanghai Jiao Tong University, Shanghai, China.
{luxiangjun,zcsjtu,loccs_cp,dwgu,hnlu}@sjtu.edu.cn

Abstract. With the renaissance of deep learning, the side-channel community also notices the potential of this technology, which is highly related to the profiling attacks in the side-channel context. Many papers have recently investigated the abilities of deep learning in profiling traces. Some of them also aim at the countermeasures (e.g., masking) simultaneously. Nevertheless, so far, all of these papers work with an (implicit) assumption that the number of time samples in raw traces can be reduced before the profiling, i.e., the position of points of interest (PoIs) can be manually located. This is arguably the most challenging part of a practical black-box analysis targeting an implementation protected by masking. Therefore, we argue that to fully utilize the potential of deep learning and get rid of any manual intervention, the end-to-end profiling directly mapping raw traces to target intermediate values is demanded.

In this paper, we propose a neural network architecture that consists of encoders, attention mechanisms and a classifier, to conduct the end-to-end profiling. The networks built by our architecture could directly classify the traces that contain a large number of time samples (i.e., raw traces without manual feature extraction) while whose underlying implementation is protected by masking. We validate our networks on several public datasets, i.e., DPA contest v4 and ASCAD, where over 100,000 time samples are directly used in profiling. To our best knowledge, we are the first that successfully carry out end-to-end profiling attacks. The results on the datasets indicate that our networks could get rid of the tricky manual feature extraction. Moreover, our networks perform even systematically better (w.r.t. the number of traces in attacks) than those trained on the reduced traces. These validations imply our approach is not only a first but also a concrete step towards end-to-end profiling attacks in the side-channel context.

Keywords: Side-channel analysis · Deep learning · End-to-end profiling · Attention mechanism

1 Introduction

Side-channel analysis (SCA), introduced in [Koc96] for the first time, takes advantage of the fact that cryptographic algorithms are not entirely black-box when implemented in the real world. It exploits the possible leakages of the sensitive internals via some side-channels, e.g., execution time, power consumption, electromagnetic radiation, etc., to recover the secret of a cryptographic device.

As one branch of the SCA, the profiled SCA has been continually explored and studied since the first method template attacks (TA) [CRR02] was proposed. Following the seminal TA, many machine learning techniques [SA08, HGM$^+$11, BL12, HZ12, LBM14] are gradually introduced and adopted into the side-channel field. Essentially, the differential analysis in the attack phase of profiled SCA is based on the distinguishment among different classes.

Received: 2020-10-15 Accepted: 2020-12-15 Published: 2021-07-09

Hence, in theory, any machine learning techniques that help to differ (no matter it is directly performed as classification or regression) samples could be used in profiling.

To distort the distinguishment among classes, the countermeasures reducing the relations between the internals and leakages [KJJ99, AG01] are proposed almost at the same time with the Differential Power Analysis (DPA). The presence of countermeasures, particularly masking, is also an important reason that some machine learning techniques are preferred, since the Support Vector Machine (SVM) [HGM+11, HZ12, BL12], Random Forest (RF) [LBM14, LPB+15], and deep learning could naturally handle the non-linear problems and deal with the high dimensional input better than classical methods like TA. Moreover, many results indicate that deep learning could directly cater to masking countermeasures [MPP16], even with some additional noise (i.e., misalignment) [CDP17, PSB+18].

We remark that one of the most critical advantages of deep learning is that they could learn to extract and combine features automatically and thus find a better representation of data. However, when looking into the recent literature applying neural networks to SCA, there is always an (implicit) assumption that the adversary could carry out a manual feature extraction (i.e., selecting PoIs or locating a short region containing PoIs) to significantly reduce the length of raw traces before the profiling. This assumption is usually too strong or even invalid in a practical analysis targeting a masked implementation. So far, no literature in this direction has discussed this issue in-depth, which leads to a limitation of the universality of the current research. As a consequence, the power of these methods is weakened due to the impractical manual feature extraction.

The primary motivation of our paper is to conduct successful end-to-end[1] profiling. Particularly, we want to omit the tricky manual feature extraction on long[2] raw traces when the prior knowledge of masking countermeasures is lacking. Yet, there is no such approach in current literature handling the requirements straightforward, to the best of our knowledge. Therefore, in this paper, we propose a new neural network architecture consisting of three components, i.e., encoder, attention mechanism, and classifier, which conduct the operations of encoding, attending, and classifying, respectively. Our architecture is designed to automatically extract and combine the leakages of sensitive intermediate values on raw traces that are too long to be handled by the mainly focused convolutional neural networks (CNNs) or multilayer perceptrons (MLPs). In our architecture, the junior encoder first encodes the raw data and yields the fine-grained features. The senior encoder then non-linearly combines these features in time sequences by the recurrent layer. Following the encoders, an attention block that focuses on the important time steps is merged to reduce the hardness of learning a recurrent layer on long time sequences. Finally, a classifier generates the output probability.

1.1 Related Works

There are many methods aiming to attack the masked implementations. This kind of attack is usually called high-order attacks, including both the profiled [Sch08, LP07, MDM16] and non-profiled [Mes00] phenomenons. In the profiled high-order attacks, the method like Gaussian mixture models [LP07] and neural networks [MPP16, CDP17, KPH+19, ZBHV20] could carry out successful attacks without knowing the mask, while we argue that they still face the feature extraction issue (which is also discussed at the end of [WAGP20]) in more practical scenarios. All of these methods can not be regarded as end-to-end profiling unless more evidence on raw traces is given. We notice there is also another roadmap to conduct high-order attacks that the adversary detects the PoIs of all the sensitive intermediate

[1]We call the profiling that maps raw data (e.g., raw traces) to final targets (e.g., unmasked intermediate values) without manual feature representations (e.g., feature extraction or combination) end-to-end profiling in this paper.

[2]Since the traces are essentially the signals changing over time, we use the word *long* to describe the traces containing a lot of time samples.

values jointly based on statistical tools and then combine them. The positive results have been reported in [DSV+15, DS16]. While in this paper, we focus on conducting end-to-end profiling in practical scenarios (i.e., raw traces with masking), which naturally includes the selection task. In other words, our architecture could be regarded as including a learnable built-in PoIs selector.

Our work is also related to a series of papers focusing on the attention mechanism and its variations [BCB15, HKG+15, RLL+17, VSP+17, LJD19, KKL20]. The attention mechanism has been widely used in Natural Language Processing (NLP) and Automatic Speech Recognition (ASR). Usually, it is used as a built-in component of the decoder agents in the sequence-to-sequence (seq2seq) problems. We use it in our paper to further combine the leakage features and accelerate the convergence of networks.

1.2 Our Contributions

The contributions of our work can be summarised as follows:

1. **An architecture for practical end-to-end profiling.** We propose a new neural network architecture in the context of SCA, which gets rid of the manual feature extraction when handling extremely long traces (e.g., raw traces) whose underlying implementation is protected by masking. With this architecture, we can mount end-to-end profiling without any prior knowledge about implementations.

2. **Introduce new structures into the SCA.** We point out that a locally-connected (LC) layer could be used as a junior encoder to extract useful time samples in a narrow region and overcome the drawbacks of convolutional layers. Surprisingly, we do not find any discussion about this kind of simplest layer in the side-channel community. Moreover, while the attention mechanism is prevalent in NLP and ASR in the deep learning community today, we are the first to introduce this mechanism upon a recurrent layer into SCA[3], to the best of our knowledge.

3. **Satisfying results on public datasets.** To our best knowledge, we are the first that achieve end-to-end profiling in the SCA. In the profiling phase, we could handle the raw traces over 100,000 time samples in each. Concretely, we directly train networks on the raw traces of DPA contest v4.1 and two ASCAD datasets. For DPA contest v4.2, we use the first 300,000 time samples in profiling to save time. All of these networks above converge and carry out successful attacks. Moreover, our networks perform even systematically better (w.r.t. the number of traces in attacks) than those trained on the length reduced traces.

4. **Explore the attention mechanism in the SCA context.** In addition to the public datasets, we also apply the architecture to our own dataset acquired from the power consumption of ATmega128A running our simulated masking demo. We utilize this dataset as an additional baseline to further explore how the fundamental structures work in our network.

The rest of this paper is organized as follows. In Section 2, we provide the basics of profiled SCA, high-order attacks and some necessary backgrounds about the neural networks. In Section 3, we propose the new architecture and describe each component of it in detail. We also explain the motivations of such a design and why our architecture works. In order to validate our statements of end-to-end profiling, in Section 4, we test the network instances of our architecture on different datasets. We also dive into the trained networks in Section 5, showing the behaviors and the interactions of components in

[3]Jin et al. also apply the attention mechanism (upon a convolutional layer) to SCA in [JZHY20], while their mechanism is different from ours in terms of purpose and means.

practice. Finally, in Section 6, we conclude and discuss some potential research directions in the future.

2 Background

In this section, we give the notations of variables and functions used in the context of SCA. For completeness, we describe the route of carrying out a profiled SCA. We also introduce the basics of neural networks which are used in our architecture.

2.1 Notations

We use the upper-case letter X to denote the random variables, the bold format $\boldsymbol{X}$ to denote the corresponding random vectors. The realizations are denoted as lower-case letters x and $\boldsymbol{x}$, respectively. Throughout this paper, we use $\boldsymbol{L}$ to denote the trace of a device, X to denote the plaintext, K to the secret key and Z to the sensitive target. The variable Z is related to X and K (e.g., $Z = Sbox(X \oplus K)$). For an acquired trace set, it is a set of realizations of variable $\boldsymbol{L}$ and could be denoted as $\boldsymbol{\mathcal{L}} = \{\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_N\}$, where N is the number of traces. Naturally, the corresponding sets of plaintexts, keys and sensitive targets are $\mathcal{X}, \mathcal{K}$, and $\mathcal{Z}$, which is consisted of the entries $x_{i,j}, k_{i,j}$ and $z_{i,j}$, respectively, where j is the byte index. We also use sans-serif upper-case (e.g., H) and lower-case (e.g., h) letters to represent matrices and vectors, respectively, when describing the architecture of neural networks. We will omit the subscripts for conciseness whenever there is no ambiguity.

2.2 Profiled Side-Channel Analysis

A profiled attack consists of two phases: an offline profiling (training) phase and an online attack (testing) phase. In the profiling phase, the adversary collects the traces with known labels (i.e. targets) as $\{(\boldsymbol{l}_i, z_i)\}_{i=1,\ldots,N}$. The goal of profiling is usually to estimate a probability density function (e.g., $\mathrm{P}[\boldsymbol{L}|Z = z]$ in TA) that could be used to classify the traces which do not occur in the profiling set. In the attack phase, the adversary acquires a new set of traces $\{(\boldsymbol{l}_i)\}_{i=1,\ldots,N_a}$ with a fixed unknown key k^*, where N_a is the number of traces for attack. The probability $\mathrm{P}[\boldsymbol{L}|K = k]$ for each key hypothesis could be set through $\mathrm{P}[\boldsymbol{L}|Z = z]$ since the x is known. The classification is then carried out by posterior transformed by Bayes' Theorem:

$$\mathrm{P}[K|\boldsymbol{L} = \boldsymbol{l}] = \frac{\mathrm{P}[\boldsymbol{L}|K = k] \cdot \mathrm{P}[K = k]}{\mathrm{P}[\boldsymbol{L} = \boldsymbol{l}]} \ . \tag{1}$$

As we want to use all of the information of the attack set but a single trace, the eventual posterior p_k observing the whole trace set is calculated as the score to reveal the underlying key (with the hypothesis that the $\boldsymbol{l}$s are independent):

$$p_k = \mathrm{P}[k|\boldsymbol{\mathcal{L}}] = \frac{\left(\prod_{i=1}^{N_a} \mathrm{P}[\boldsymbol{L} = \boldsymbol{l}_i|K = k]\right) \cdot \mathrm{P}[K = k]}{\prod_{i=1}^{N_a} \mathrm{P}[\boldsymbol{L} = \boldsymbol{l}_i]} \ . \tag{2}$$

To prevent the numeric issue of underflow, the p_k could also be equivalently calculated by the sum of the log-posterior. Finally, the key candidate with the maximum score is chosen as the key guess.

2.3 High-Order Attacks

One of the most common countermeasures to protect cryptographic implementations from SCA is masking, which hides the sensitive intermediate values by randomness. The (Boolean) masking applied on the Sbox can be represented as a series of XOR operations among $d + 1$ shares, namely, $z_m = Sbox(x \oplus k) = z'_m \oplus m_1 \oplus \cdots \oplus m_d$, where $m_1, \ldots, m_d$ are the *masks* and z'_m is the *masked value.*

The fundamental idea of high-order attacks is combining the leakages of shares. Then the first-order DPA with different distinguishers can just be carried out on these combinations. However, with raw traces, the $d + 1$ shares make this combination very complicated. In the worst case, the adversary has to pre-compute all of the possible combinations of $d + 1$ time samples, and thus yield the new traces of length $\binom{|\boldsymbol{l}|}{d+1}$, where $|\boldsymbol{l}|$ is the length of original traces. In recent research, the MLPs and CNNs are used to automatically combine leakages without knowing the implementation details. We note that besides the combination complexity, the architectures of MLPs and CNNs themselves also restrict the profilings on extremely long traces. We will discuss this in detail later in Section 3.1.

2.4 Neural networks

Neural networks are usually composed of multiple layers, and each layer is composed of multiple neurons. The different behavior of networks is mainly affected by how these neurons are connected within and between layers. For the convenience of describing our architecture, we briefly introduce some basics of neural networks, specifically, the useful layers and the attention mechanism.

2.4.1 Convolutional layer

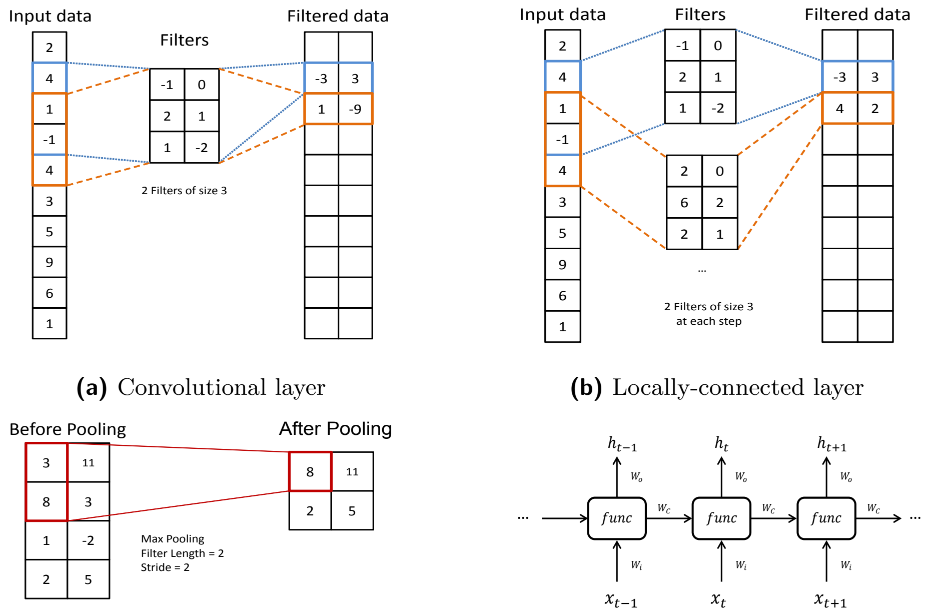
The convolutional layer is one of the most famous neural network layers because of the great success in the computer vision field with Pooling layers. It conducts convolution operations to the input by the filters as a group of sliding windows. The weights of filters are shared among the spatial steps, which detect the same patterns at different positions. The left-hand side of Figure 1 shows a convolution operation with 2 filters of size 3 and a maximum pooling with a window of size 2. As mentioned in many other papers, the output of the stacked convolutional layer are usually called feature maps or abstract features which discriminate from the raw data. Meanwhile, these abstract features also have some natural properties like shift-invariance, and thus the CNNs are good candidates to handle the desynchronization in the side-channel context [MPP16, CDP17].

2.4.2 Locally-connected layer

We use the term locally-connected (LC) in this paper to specifically refer to the layer in Figure 1b. The only difference between it and the convolutional layer is that the LC layer does not share the weights among steps when sliding along the inputs. This means the filters do not learn to extract common features and each element of the output is only related to a limited small region of the input. That is, the LC layer gets more freedom to focus on details in a particular region. This property is pretty useful when we desire exhaustive features that are independent of each other in different positions. This kind of layer has been used in face recognitions [TYRW14] and speaker recognitions [CLS⁺15] to extract local features.

2.4.3 Recurrent layer

The recurrent layer is dedicated to exhibiting temporal dynamic behavior, and thus it could learn the dependence of time samples along the time sequence. To generate the output,



(a) Convolutional layer

(b) Locally-connected layer

(c) Pooling layer

(d) Recurrent layer

Figure 1: Examples of neural network layers.

the recurrent layer infers not only the input of the current time step but also the output of the last. While the recurrent architecture theoretically handles the time-dependency, it results in a significantly long path when the gradients are backpropagated, causing the vanishing or the exploding of gradients. For this reason, the vanilla recurrent layer can not really deal with a very long sequence. To mitigate this drawback, gated recurrent layers and the variants are proposed [HS97, GS00, CvMG+14]. Among all of these improvements, we focus on the Long-Short Term Memory (LSTM) in this paper. The formula of LSTM goes as follows,

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ \widetilde{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \widetilde{C}_t \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{3}$$

where i, f and o are the input, forget and output gate, respectively, C is the memory state, h is the hidden state.

2.4.4 Attention mechanism

Attention mechanism has recently demonstrated success in a wide range of deep learning tasks, including machine translation [BCB15, VSP+17, KKL20], imagine captioning [XBK+15, AHB+18], speech recognition [CBS+15] and sentence summarization [SLM17]. In these seq2seq models, the basic idea of attention is to allow the decoders to refer back to any of the encoders' output, thereby finding out the related parts that highly contribute to generating the expected output at the specific step of the decoders. Therefore, the attention mechanism can be interpreted as a function which maps an encoded input sequence to an output with a variable query.

In the context of seq2seq problems, the attention mechanism creates an implicit soft alignment between entries in the output sequence and entries in the input sequence, which can give useful insight into the model's behavior. In the context of side-channel, this insight could be interpreted as the regions of interest that the sensitive values leak. We give more details about the attention mechanism we used in Section 3.

3 Our architecture

In this section, we evaluate the shortcomings of MLPs and CNNs by explaining the causes of the high dimension issue in the side-channel context. To overcome these shortcomings, we propose concrete design principles for our new architecture. We then introduce three components (i.e., encoder, attention mechanism, and classifier) guided by these principles and explain why the overall architecture works in the side-channel context. Moreover, we discuss some different choices of connections to build up variants of our architecture.

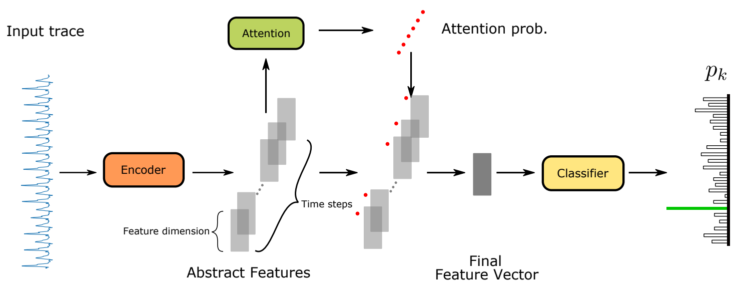


Figure 2: Abstract architecture from the side-channel point of view.

3.1 Motivation and design principles

Although the full control of a profiling device is a common assumption of profiled SCA, the situations could differ in practice. From the security analyzers' point of view, the implementations in the device under test are usually open-source and could even be modified. Therefore, the operations at every time clock are known to the analyzers. From the adversaries' point of view (which is more practical), the most common phenomenon is that they can only control the input (i.e., the plaintext) and reset the key of a profiling device. Intuitively, under the latter assumption, the complexity of attacking a masked implementation is expected to increase rapidly. In the worst case, to ensure the sensitive leakages being acquired, the adversary has to measure a quite long operation sequence and use all of the time samples to conduct an attack.

The MLPs and CNNs seem to be good choices at first glance since several references have shown their ability to profile traces without knowing the mask (on the dimension reduced datasets). However, a more practical scenario could easily let the attack fall into the worst case breaking the implicit assumptions that the trace dimension could be easily reduced in advance. Meanwhile, a trace containing hundreds of thousands of time samples is quite usual (especially for software implementations), e.g., the DPA contest v4.2 (over 1,600,000 samples) and ASCAD (250,000 samples) dataset. This is the actual magnitude of time samples on which the adversary has to deal with in more practical scenarios. To this end, considering the end-to-end profiling is distinctly meaningful.

Shortcomings of typical MLPs and CNNs. The shortcomings of MLPs are apparent when the dimension of input is increasing (i.e., the length of raw traces is increasing). As all of the layers in MLPs are fully-connected (FC) layers, each neuron of the current layer is contributed by all of the neurons from the last. That means with the increasing input dimension, each neuron of the hidden layer is related to more weights corresponding to the noise in input and thus makes the training harder. From another perspective, the property of fully-connection also restricts the number of neurons in the first hidden layer since a weight matrix between a high dimensional input and a large number of neurons could quickly run out the memory of GPU cards off-the-shelf. The problem of CNNs is that they are not good at reducing the overall dimension if we admire a good performance. This could be inspired by the VGG, ResNet and GoogLeNet-liked CNN architectures that all of them increase (usually double) the channels (i.e., the number of convolutional filters) when reducing the spatial dimension. As a result, the dimension of the final feature map is not reduced significantly compared to the input or could even be increased (like the 50-layer ResNet). Therefore, CNNs meet a similar dimension issue like MLPs when connecting the final feature map to the FC layer (additional FC layer for further feature combination or the final classifier). Moreover, from the perspective of memory utilizing, it is even more tricky to build a proper CNN instance when the input dimension is high, since in general, the feature maps generated by the convolution in the first several layers are already quite memory consuming.

We also notice the popularity of utilizing the global average pooling at the end of CNNs, since this kind of pooling avoids the massive connection to FC layers. Nevertheless, a high dimensional input indicates that even at the end of a network, the global average pooling still has to average the features on too many spatial positions. Intuitively, this introduces too much noise into the final feature vector. Some results in Section 4.3 imply that this kind of architecture does not really fit the end-to-end context. It is also important to point out that the state-of-the-art (w.r.t. specific dataset) architectures in SCA do not entirely follow the rules of thumb in the image classification field. Yet, as mentioned in [WAGP20], the results that these architectures against long traces are still lacking.

Design principles. According to the analysis above, we present some principles that we use in designing:

1. avoid heavily (fully-) connection between high dimensional layers
2. avoid expanding feature dimension when the spatial (temporal) dimension is still high
3. fine-grained feature should be extracted and compressed with lightweight network structures at the first several layers
4. layers before the final classifier should have the ability to select and combine these fine-grained features

Of course, with the richness of today's neural networks, there could be many other potential architectures solving the problems in practical SCA. In this paper, we propose one possibility and demonstrate that it is an actual step towards the end-to-end profiling.

3.2 Encoder component

The first component of our architecture is the encoder component. It extracts and combines the leakages to get the abstract features, and hence the upper component could recover the underlying intermediate values. With the principles in mind, we use the LC layer and recurrent layer (i.e., LSTM) as the junior and senior encoder, respectively, for the synchronized traces. For the desynchronized traces, to extract the shift-invariant feature map, we use the stacked convolutional layers as junior encoder instead. We emphasize that by designing them with the upper components in our architecture, lightweight stacked convolutional layers do not raise the problems once in typical CNNs when handling the long raw traces.

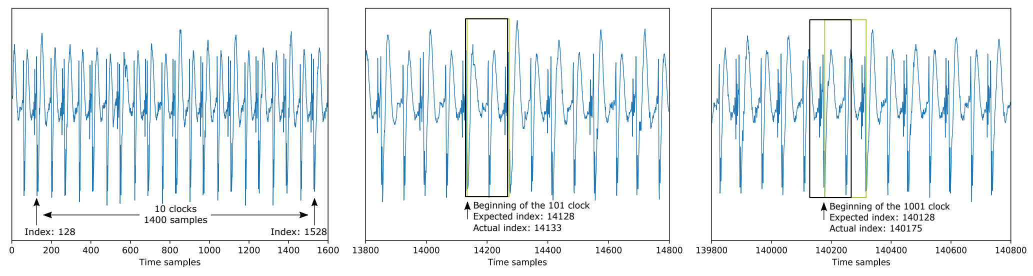
3.2.1 Junior encoder

By default (synchronized situation), we use the LC layer as the junior encoder. In this paper, we always use a single LC layer with one filter per step. The filter size and stride are chosen by trial and error, while the guideline is that the filter size should cover an integer number of clock cycles and be divided by stride. Empirically, the filter size is chosen as the length of one or two clock cycles while the stride is half of the size for overlapping. The basic methodology underlying is compressing the time samples in several clock cycles to a single feature sample. The overlapping is conducted to smooth this compression. As a result, one or two clock cycles are represented by several feature samples. The non-linear activation function is also omitted to keep the junior encoder a simple sequence of affine transformations.

One reason for using the LC layer is that it significantly mitigates the effect of high dimension since each neuron only focuses on a narrow region from the input. Compared to the convolutional layer, the filter weights associated with the specific position could be fine tuned independently and the output of each neuron is not affected by the noise far from it, which leaves more freedom to extract independent and variable features. Similar conclusions can be derived when compared to the FC layer. In a word, the LC layer decouples each neuron from the whole high dimensional input, which is crucial to generate fine-grained features with high quality. Benefiting from its own property, the LC layer also naturally reduces the dimension of the input data (if we set the stride with a proper value), and thus does not yield a high dimensional internal output. From another aspect, these locally-connections could be regarded as a sequence of transformations that are analogues of the ones in the linear discriminant analysis (LDA). That is, ideally, each small region is transformed to be more distinguishable. Note that although the number of weights in the LC layer increases linearly with the increase of the input dimension, the increase of computation time of this layer is negligible since it could be well parallelized.

Why the convolutional layer can not be used in the same way? So far, it seems that compared to the convolutional layer, the advantage of LC layer that significantly reduces the dimension is not based on the layer itself but the way we use it (i.e., one filter per step, the size of the filter and the value of stride). In Figure 3, some evidence is given to show that the convolutional layer can not be used in the same way due to the different connection structures. The fundamental problem is that, in the convolutional layer, the beginning of filters and one clock cycle can not always be aligned in the sliding process. This problem is caused by the desynchronization between the clock frequency of the device and the sampling rate of the oscilloscope (i.e., the clock frequency can not divide the sampling rate), so that the length of each clock cycle is not accurately an integer (while the size of the filter and stride must be). It is shown in Figure 3b that the actual start of clock cycle 101 is 14133 while the sliding filters expect it as 14128. Furthermore, this deviation is accumulated with sliding (Figure 3c). Note that one purpose of sharing weights between steps in the convolutional layer is to make an agreement that each filter extracts one particular pattern. The changing deviation means the pattern of fragment slided by the only filter is continuously shifting (this shifting is nonperiodic, different from when the stride is 1), which significantly harms the agreement among steps. In contrast, steps in LC layer do not try to reach an agreement with each other and thus do not care about the deviation. As a result, the convolutional layer can hardly be used in the same way like our LC layer in practical scenarios.

The convolutional layer is still preferred in desynchronized cases. Despite the shortcomings of typical CNNs, the convolutional layer itself is a powerful tool to extract shift-invariant features when the stride is set to 1 (which is the most common case). Therefore, we still use the stacked convolutional layers to profile the desynchronized traces. Thanks to the upper components of our architecture, we only need some lightweight (w.r.t. the number of filters) convolutional layers and do not really care about the final spatial dimension[4] of the feature maps.



(a) First 10 clocks **(b)** 100 clocks **(c)** 1000 clocks

Figure 3: Increasing deviation between the beginnings of filters and clock cycles.

3.2.2 Senior encoder

The recurrent layer, specifically, the LSTM, is used as a senior encoder in our architecture to further combine the fine-grained features extracted by the junior encoder. In our use case, the LSTM works under a seq2seq mode that, at every time step, it gets a feature vector in and yields a feature vector out. Another basic setting of LSTM is the number of units which is also known as the dimension of the internal memory (also the dimension of output vector at each step by default). In the practical phenomenon we concerned, the number of units in LSTM is set to 128 or 256 to balance the memory needed for complex inner operations and the yielded feature dimension[5]. The activation functions of gates and internal memory are set to *sigmoid* and *tanh*, respectively.

One important reason for choosing LSTM is to obey the design principles. Although the compressing applied by the LC layer reduces the input dimension, the actual internal dimension of a network still depends on the specific dataset. From the perspective of designing a stable architecture, the risk of using the FC layer to combine features should be avoided. Moreover, once we want to change the junior encoder to stacked convolutional layers to deal with the desynchronized traces, LSTM is much less sensitive to the overall dimension by avoiding fully-connection, and thus makes our architecture more pluggable to different junior encoders. In a word, the senior encoder should be able to combine the fine-grained features at any positions in a sequence of any length, while the consuming of memory should be as little as possible (which is important for profiling raw traces). In our opinion, the recurrent layer is the best choice among the layers nowadays.

Another reason is that the LSTM could learn to control the data flow in itself, and thus it could potentially learn very complex operations among the time steps. Specifically,

[4]This is also the temporal dimension. We use both terms interchangeably according to which component we discuss, as they are the same thing from different aspects.

[5]We remark the temporal dimension has been reduced by the junior encoder. There is no doubt that a larger dimension consumes more resources, but the trade-off of resources and representing capacity always exist in neural network designing.

it learns the weight matrices related to the input, forget and output gates that directly control the behaviors at each time step. That is, the LSTM could learn to remember useful information and forget (or just never remember) the useless after going through enough sequences. For the output at each time step, it is essentially a mapping of the sub-sequence until the current step, representing a potential useful abstract feature vector. This property helps to further extract and combine the fine-grained features which are (potentially) long-time dependent on each other (i.e., leakages spread among many clock cycles and/or the shares leak at different positions).

The purpose of using seq2seq mode is that encoding a too long sequence in our supposed scenario into a single feature vector will definitely make the training very hard (LSTM still faces the gradient issue in practice). As a result, we need some mechanism to reduce the hardness of training. Ideally, if we can choose some critical time steps or at least shorten the time sequences automatically, the training will be much easier. Therefore, we select the seq2seq mode which exposes the hidden state at each time step to satisfy the necessary precondition.

Bi-directional LSTM. To better profile the time-dependency among the leakages, we utilize two LSTMs that go through the input sequences in both forward and backward directions as depicted in Figure 4. The outputs of two LSTMs could be concatenated along the feature axis, as in Figure 4a (corresponding to variant 1 of our architecture). Consequently, the output at each time step could get the information from not only the past but also the future. This concatenation intuitively makes the time steps between two leakages more informative. Besides, since one of the LSTM reads the input backward, the order of accessing the sensitive leakages is reversed from the forward one. Two LSTMs may learn different kinds of combinations of features. Sometimes, we do not want the outputs of LSTMs to interact with each other through the merging operation nor limit the representing flexibility of higher layers. Therefore, we hold the outputs of two LSTMs independently, as shown in Figure 4b (corresponding to variant 2 of our architecture). In this case, we also use independent attention components to focuse on the specific time steps of each direction. Concretely, we use $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$ to represent the forward output and the backward output at each time step. For the former case, the output matrix $\mathbf{H}$ goes like

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_S \end{bmatrix} = \begin{bmatrix} \overrightarrow{\mathbf{h}}_1 & \overrightarrow{\mathbf{h}}_2 & \cdots & \overrightarrow{\mathbf{h}}_S \\ \overleftarrow{\mathbf{h}}_1 & \overleftarrow{\mathbf{h}}_2 & \cdots & \overleftarrow{\mathbf{h}}_S \end{bmatrix} , \tag{4}$$

for the latter case, the two matrices go like

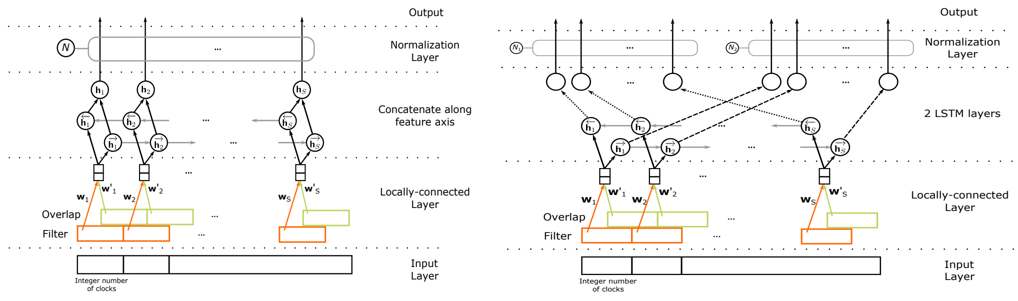
$$\begin{aligned} \overrightarrow{\mathbf{H}} &= \begin{bmatrix} \overrightarrow{\mathbf{h}}_1 & \overrightarrow{\mathbf{h}}_2 & \cdots & \overrightarrow{\mathbf{h}}_S \end{bmatrix} , \\ \overleftarrow{\mathbf{H}} &= \begin{bmatrix} \overleftarrow{\mathbf{h}}_1 & \overleftarrow{\mathbf{h}}_2 & \cdots & \overleftarrow{\mathbf{h}}_S \end{bmatrix} , \end{aligned} \tag{5}$$

respectively, where S is the length of the input sequence.

3.3 Attention mechanism

In this section, we introduce the attention mechanism in our architecture, explaining why we need such a mechanism and how it works.

We recall the LSTM with sequential output is chosen to build our senior encoder, in order to avoid encoding the whole sequence to a single feature vector. According to the properties of LSTM, the underlying data flow at the specific time step indicates that from the beginning to the current step, what information is remembered, combined and yielded. There is no doubt that unless the LSTM has visited the leakages of all shares, it can not generate an informative abstract feature that is helpful to recover the key. In contrast,



(a) Concatenated output **(b)** Two independet outputs

Figure 4: Two encoder instances using different wrapping strategy.

once all clock cycles that leak sensitive information are passed through, the rest of the sequence is just noise disturbing the learning. This inference indicates that there are some time steps (at least one step) being more informative than others. Once the most informative steps are found out, both the time dimension and the training hardness could be reduced simultaneously. The attention mechanism does exactly what we need.

In this paper, we create an analogue of the attention mechanism proposed by Bahdanau et al. [BCB15]. Our attention mechanism first evaluates the importance of each time step by a time-distributed FC layer that contains only one neural perceptron without bias. This layer shares the weights among the time steps (like a convolutional one), and thus it scores each time step with the same standard. Based on the output of this time-distributed FC layer, a batch normalization layer is appended to give each step additional flexibility adjusting the attention scores independently. Finally, the scores are normalized (by a softmax function) to generate probabilities used in the weighted sum. Concretely, our attention mechanism works as below:

$$\begin{aligned} \mathbf{a'} &= batchnorm(\mathbf{v}^T\mathbf{H}) \\ \mathbf{a} &= softmax(\mathbf{a'}) \\ \mathbf{r} &= \mathbf{H}\mathbf{a}^T \end{aligned} \tag{6}$$

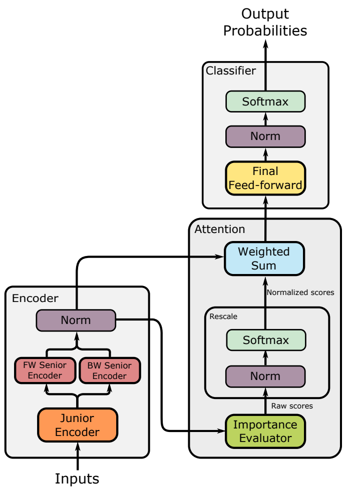
where $\mathbf{v}$ is the trainable parameter vector of the single perceptron whose dimension is the same as $\mathbf{h}$, $\mathbf{v}^T$ denotes its transpose and $\mathbf{a'}$ is the vector of attention scores.

Here we give two detailed interpretations of why our attention mechanism helps from the training viewpoint: 1) The attention mechanism is essentially weighted summing up all time steps and generating a single vector as representation. In other words, it controls the proportion of the gradients that backpropagated through each step of LSTM by the attention probabilities. Therefore the network is mainly updated by the gradients associated with the high attention probabilities, getting rid of not knowing which time steps are most informative automatically. 2) The attention mechanism could also be regarded as a kind of soft time truncating method. The time truncating method [Jae02,Sut13] uses two parameters s_1 and s_2 to control the number of forward-pass timesteps between updates and the number of timesteps to which to apply backpropagation through time, respectively. From this point of view, the parameter s_1 in our method is set by the attention probabilities but with a variable value associated with the interval between where the probability is large enough. Meanwhile, the s_2 is also a variable value that always equals the current step count. Furthermore, being different from an actual time truncating method, our attention mechanism could also be regarded as truncating the rest of the steps after the last significantly attended step, and thus reduces the influence of noise after all of the necessary leakages have been accessed.

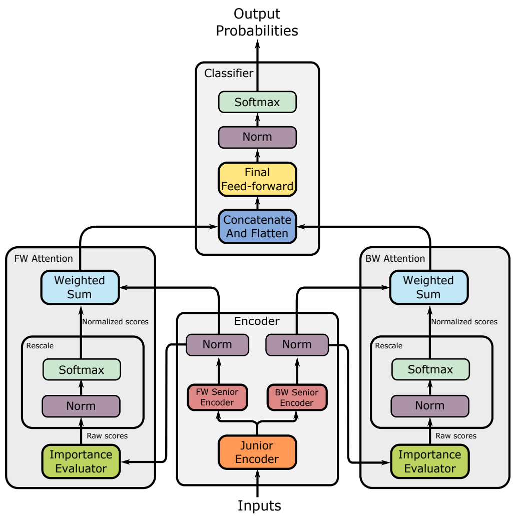
Why not utilize the attention mechanism after the junior encoder? To answer this question, we recall that for a high-order DPA, the combination of leakages of shares is necessary. According to [DZFL14], the centered product combination function is the best possible combination method in noisy situations. Therefore, if the attention mechanism is directly used upon the fine-grained feature, the combination of these features will be a weighted addition. As a result, a sub-optimal operation is chosen to combine the leakages. From our point of view, the gated recurrent layer is a better candidate to learn the best possible combination. More evidence is presented in Section 4.3 that the network with attention mechanism on the junior encoder (i.e., stacked convolutional layers) performs much worse than those with attention mechanism on the senior encoder (i.e., LSTM), even the junior encoder is modified to be more complex.

3.4 Classifier component

To recover the targeted intermediate value, we need a component at the end of our architecture to classify the abstract features we get so far. In general, a FC layer with softmax as an activation function is the most used output layer format in the classification task. The softmax normalizes the logits and makes them more interpretable (i.e., probabilities of classes). In the context of SCA, these probabilities are necessary to mount the subsequential attack, and thus we apply this format to our output layer. That is, our recover component is as simple as a FC layer with a softmax activation function.



(a) Variant 1: Concatenating the outputs of two LSTMs along the feature axis



(b) Variant 2: Utilizing two independent LSTMs with attention mechanisms

Figure 5: Two variants of our architecture.

4 Experimental results of profiling and attack

In this section, we provide the results of experiments to explore more interests of our architecture. We show the profiling and attack results on several datasets to evaluate the feasibility of the architecture when it actually faces the raw traces without any feature extraction. We carry out these experiments on the public datasets (e.g., DPA contest, ASCAD) and traces collected from a microcontroller ATmega128A (we refer to this dataset as AT128 hereafter). Before the detailed results of each dataset, we also briefly introduce the basics of the implementation. Nevertheless, we again emphasize that none of our experiments utilize these basics as necessary prior knowledge. According to the 'no-free-lunch' theory, we build the networks and configure the training differently among these datasets. On each dataset, we present the results of the best network instance we get. For conciseness, the detailed topologies of networks are given in Appendixes A to C, while a brief index of experiments is given in Table 1. All of the networks are built up by Keras 2.2.4 [C+15] with TensorFlow 1.13.1 [AAB+15] as backend, and the code to reproduce our experiments is available in a Github repository[6]. These networks are trained on our workstation equipped with Intel I9-9900X CPU, 2 Nvidia RTX 2080Ti GPU cards, and 128GB RAM. More details about the training cost are given in Appendix D. Additionally, although the software implementations are mainly focused, the results on hardware are also illustrated in Appendix E.

Table 1: Summary of experiments.

Section	Dataset	Additional RD	# of traces
4.1.1	DPA contest v4.1	None	7/1(known mask)
	DPA contest v4.2	None	30/120(7th subset)
4.1.2	ASCAD v1	None	8
	ASCAD v2	None	10
4.1.3	AT128-N	None	3
	AT128-F	None	4
4.2.1	ASCAD v1	52/100	22/20
	ASCAD v2	126/200	4/3
4.2.2	AT128-N	44/100	3/3
	AT128-F	44/100	3/4

RD: Random delay.

# of traces: The number of taces that reduces the guessing entropy to 0.

4.1 Results on synchronized traces

In this section, we present the results of profiling and attack on synchronized datasets. We use the unmasked value of Sbox in the first round as label (i.e., $Sbox(x \oplus k)$). The guessing entropy is calculated from 100 random attacks. It is depicted as the function of the number of traces used in attacks to evaluate efficiency. No pre-processing is conducted to the traces but standardizing (zero-mean unit-variance).

4.1.1 DPA contest v4

The DPA contest v4 provides measurements of 2 protected implementations of AES. The first version v4.1 implements a masked AES-256 with Rotating Sbox Masking (RSM), while the second version v4.2 implements a masked AES-128 with the same masking scheme by assembly code. Both of them are implemented on ATmega163 smart card and acquired by the electromagnetic radiation. We refer to [NSGD12] and [BBD+14] for more details about these two implementations.

Training setups for v4.1 set. The available amount of traces of v4.1 set is 100,000. We choose the first 39,000 traces for profiling and the following 1000 traces for validations and

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

attacks. We remark that the holders of DPA contest use a fixed key to acquire the traces of v4.1. This could be a problem under the end-to-end scenario. Since the key is fixed, the bijection from the plaintext to the output of Sbox is also fixed. In this case, from the neural network viewpoint, classifying the traces according to $Sbox(x \oplus k)$ is equivalent to according to x. The network might learn to classify the traces by the leakages of plaintext, which is not expected. Consequently, the first 10,000 time samples are removed to exclude the disturbance of the plaintext[7]. Since plaintext is always known, this could be easily carried out. The number of time samples is then reduced to about 420,000. In this dataset, the first byte is chosen as the target. The network is built by our architecture variant 1 in Figure 5a. Finally, it is trained with Adam optimizer, batch size 32 and learning rate 0.0001.

Training setups for v4.2 set. We use all of the 16 subsets associated with different keys to train networks on this dataset. For each subset, we choose 4500 traces for profiling and 500 traces for validations and attacks. The number of time samples in the raw traces is over 1,600,000 which is too large for practical profiling with limited time, and thus we use the first 300,000 time samples (almost including the entire first round of AES). We also focus on the first byte in this dataset and build the network according to our architecture variant 1. Finally, the network is trained with Adam optimizer, batch size 64 and learning rate 0.0001.

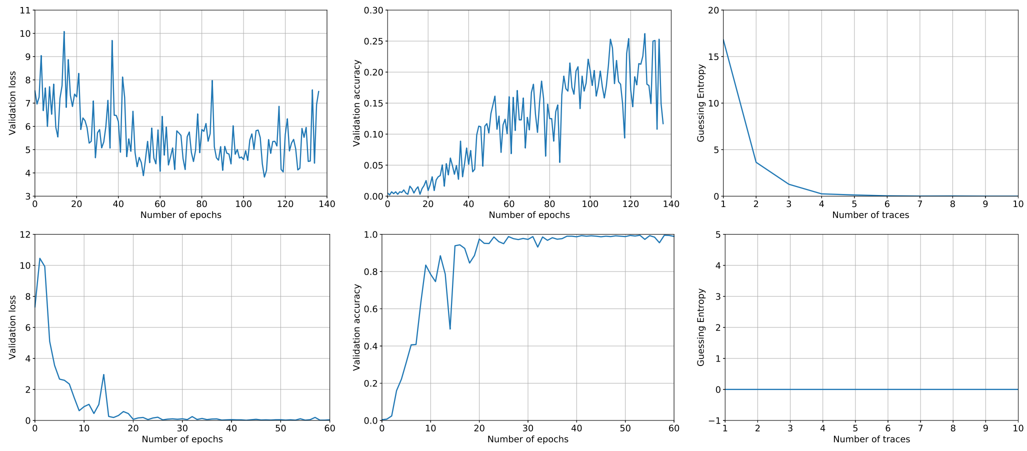


Figure 6: Results of DPA contest v4.1, with (bottom) and without (top) the knowledge of mask value.

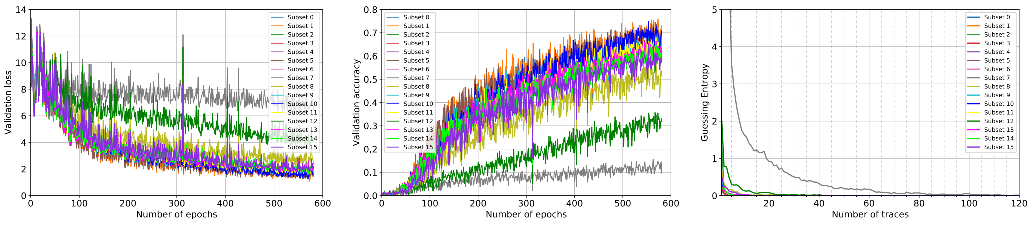


Figure 7: Results of DPA contest v4.2

The results of the dataset v4.1 are given in Figure 6. As the figure shows, the validation

[7]This is not a weakness of our architecture, but an issue caused by the natural property of the bijection function in AES. Acquiring traces with a variable key could completely avoid this.

loss metric saturates after about 50 epochs, while the corresponding validation accuracy keeps increasing to about 20%. The network which gives the lowest validation loss is used to conduct 100 random attacks. The result with respect to the number of traces in attack is depicted on the right-hand side of the figure. This result is very similar to the best performances shown in the recent papers (see Appendix F)[8], while we do not utilize any prior knowledge of the mask value nor select the PoIs. For further comparison, the result knowing the mask value (which is the case in the referred papers) is shown in the second row of Figure 6. In this case, the key could be recovered with only a single trace. The results of this dataset verify that even the raw traces are extremely long, our network could extract and classify the leakages very efficiently without the manual selection of PoIs.

The results of the dataset v4.2 are given in Figure 7 where we show the loss, accuracy, and guessing entropy for the first byte in each subset. The curves of loss and accuracy imply the 7th subset (with partial key 0x2F) is harder to attack, which is verified by the guessing entropy in the attack phase. One intuitive explanation of this phenomenon is that the specific mask candidates obstacle the profiling corresponding to some key values. However, since the holders do not provide more traces, further investigation of this observation is left in a future work. Besides the unbalanced performance among different subsets, all of the attacks could reduce the guessing entropies to 0 in a reasonable number of traces.

4.1.2 ASCAD

Our architecture applies a successful attack on the trace sets of DPA contest v4. However, both implementations manipulate the lightweight RSM scheme. Therefore, the ASCAD dataset [PSB+18] is selected as a validation to attack the Boolean masking scheme where the mask value is completely random. Moreover, the newest released version of this dataset is a perfect baseline to test our architecture, as the raw traces are long enough and the key in the training set is variable. We refer to the ASCAD dataset with fixed (resp., variable) key as ASCAD v1 (resp., v2) hereafter. In ASCAD, a masked AES-128 algorithm is implemented on an 8-bit AVR microcontroller (ATmega8515). The measurements are collected through electromagnetic radiation. The masking scheme makes use of the classical *table re-computation* method introduced in [AG01,PR07].

Training setups for v1 set. For this trace set, we use 50,000 traces for profiling and 10,000 traces for validations and attacks. In each raw trace, there are 100,000 time samples, which are directly used in experiments. Since the masks of the first two bytes are set to 0 by the authors for elementary attack tests, the third byte is selected as the target. The network for this dataset is built by our variant 2 in Figure 5b. Finally, it is trained with Adam optimizer, batch size 8 and learning rate 0.0001.

Training setups for v2 set. For the v2 trace set, we use all of the 200,000 traces with a variable key to train the network and 10,000 traces with a fixed key to validate and attack. The raw traces of ASCAD v2 contains 250,000 time samples which are directly used in our profiling. Due to the same reason as the v1 set, the third byte is selected as the target. The network for this dataset is also built by our architecture variant 2. Finally, it is trained with Adam optimizer, batch size 200 and learning rate 0.0001.

We give the results of the ASCAD v1 dataset in Figure 8, observing that our trained network could reduce the guessing entropy to 0 very efficiently (in 10 traces), though the raw traces are used. To our best knowledge, this result is state-of-the-art, even compared to the results based on the selected PoIs (see Appendix F). In addition, we note there is a step at about 380 epochs in both curves of validation loss and accuracy. We infer the

[8]We refer to Figure 5 and Figure 3 in [PHJ+19] and [KPH+19], respectively, for more details.

cause of this observation is that the learning rate is too large when training reaches epoch 380, and thus the network declines. In other words, we could further optimize the network if we use a lower learning rate after 380 epochs. We do not dive into this learning rate tuning game as our main target is end-to-end profiling, not finding an optimal network instance.

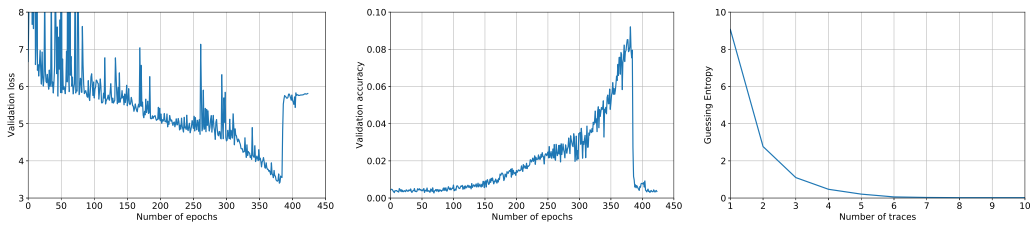


Figure 8: Results of synchronized ASCAD v1.

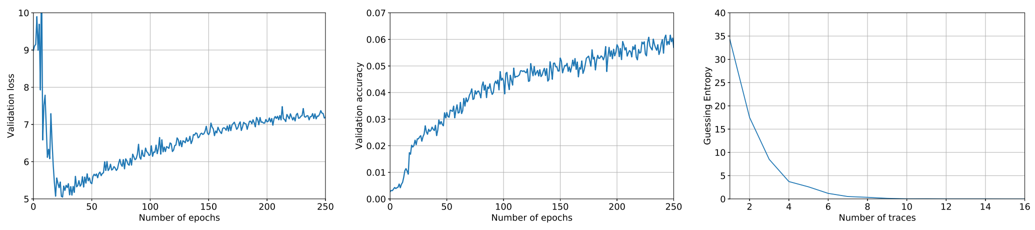


Figure 9: Results of synchronized ASCAD v2.

The results of the ASCAD v2 dataset are presented in Figure 9. On this dataset, again, the partial key is correctly recovered within 10 traces. To our best knowledge, this is a state-of-the-art attacking result on this dataset. Meanwhile, there is no report of directly profiling and attacking the raw traces of this dataset before ours. We can observe that the validation loss is increasing from the epoch 25, which usually indicates overfitting in profiling. However, the validation accuracy is also increasing while we verify the network is still being optimized (w.r.t. guessing entropy). We infer the reason for this strange observation is that the network capacity is not tuned to be optimal and/or the amount of data is still not quite enough for end-to-end profiling. As a result, the network is becoming less confident about the overall classification result while classifying more test traces around margins correctly. Despite this, our network instance is already good enough for a practical end-to-end attack.

4.1.3 AT128

For a more thorough study of our network architecture, we acquire the power consumption from a device we fully controlled (an 8-bit AVR microcontroller, ATmega128A). To control the leakages better, we do not actually implement a protected AES. We simulate the leakages of sensitive internals by an input array (with 9 bytes) through the assembly instructions like LD and ST, which is very similar to what Masure et al. presented in [MDP20]. We conduct two simulations, where the only difference is that we insert much more redundant NOP instructions in the second one to make the leakages of each byte further apart. Thence, we simulate a more extreme leaking situation which is more difficult to conduct end-to-end profiling. We refer to the datasets acquired from these two implementations as AT128-N (N for near) and AT128-F (F for far). Both of them are collected by the Pico 3203D oscilloscope at a sampling rate of 125MS/s while the chip runs at 11MHz. With our acquirement setups, each dataset contains 200,000 traces, while each trace contains 47,000 time samples.

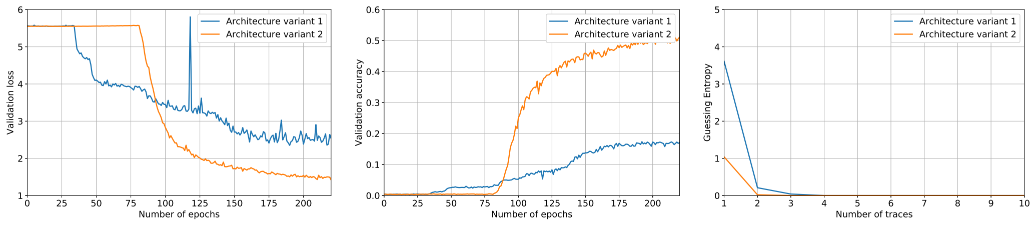


Figure 10: Results of synchronized AT128-N.

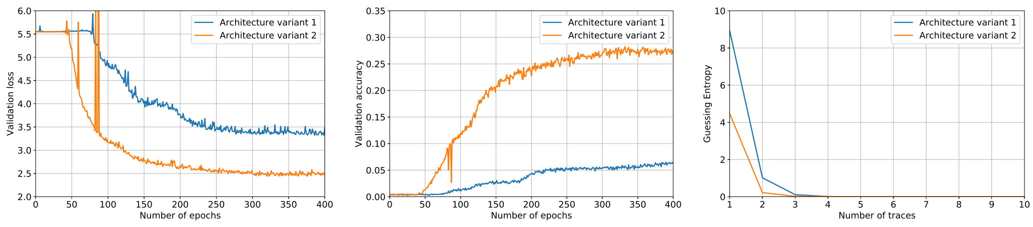


Figure 11: Results of synchronized AT128-F.

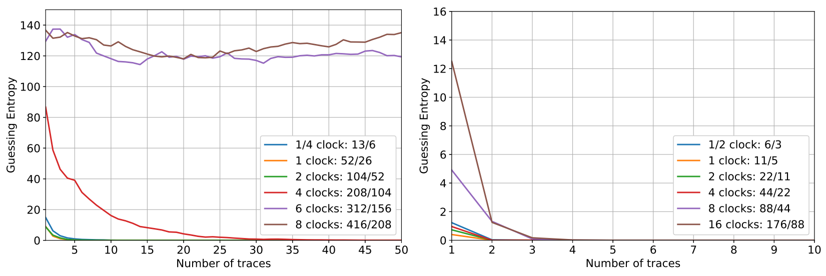
Training setups. We carry out experiments on both of the datasets using 190,000 traces for profiling and 10,000 traces for validations and attacks. On these two datasets, we use the simulated output of Sbox as the label (i.e., the XORed value of two input bytes which simulate the m and $Sbox(x \oplus k) \oplus m$). The network for AT128-N is built by our architecture variant 1. It is trained with Adam optimizer, batch size 128 and learning rate 0.001 with decay rate 0.8 per 30 epochs. The target of AT128-N is $x_4 \oplus x_6$. The network for AT128-F is built by our architecture variant 2 whose target is $x_2 \oplus x_7$. The training setups are the same as AT128-N except for the batch size is 16. To further compare the universality of these two variants, the best networks for each dataset are exchanged and retrained (with their original training setups) from scratch.

We first show the results of the dataset AT128-N in Figure 10. We see that the networks built by two variants perform similarly in the aspect of guessing entropy. Both of them need only several traces to recover the key. Differently, the network built by the variant 2 takes about 200 epochs to reach the accuracy of 50%, which is much better than the network built by the variant 1. Then the results of AT128-F are given in Figure 11. In this dataset, the tend of differences between the variants of architecture are similar to the AT128-N. The network built by the variant 2 still performs better.

According to the results, the network based on the variant 2 handles both leakage conditions better. Recalling that the batch size of the variant 2 is 16 compared to the 128 of the variant 1, the advantage of the variant 1 is that it could converge with a larger batch size when the number of training samples is the same, and thus get a sound network for attack faster. For completeness, we point out it can not reach the same accuracy as the variant 2 (about 3% lower on AT128) even the batch size is set to 16. Furthermore, we can not find an instance of variant 1 to attack ASCAD datasets, which indicates the variant 2 is potentially more suitable to complicated leakages. We remark that although these are empirical conclusions, they help to simulate a more knowledgable adversary. In contrast, we still choose a better architecture variant by trial and error on the public datasets.

4.1.4 Choices of the filter size and stride in the LC layer

Finally, we also present more results about the influence of the filter size and stride in the LC layer. Although we have stated in Section 3.2.1 that the filter size choosing as the length of one or two clock cycles while the stride being half of it is a good empirical choice for general cases, the values of these two hyper-parameters are definitely not limited. Choosing these two parameters are making trade-offs between the local resolution (related to the quality of fine-grained features) and the number of time steps (more related to the training time). There is usually a wide range of filter size and stride where the networks converge. To demonstrate this, we show several searching tests on AT128-N and ASCAD v1 (both are tested with architecture variant 2) in Figure 12.



(a) ASCAD v1 **(b)** AT128-N

Figure 12: Results of different choices on the filter size and stride in LC layer.

4.2 Results on desynchronized traces

In this section, the traces are randomly (but fixed during profiling and attack) shifted within an interval from the original raw dataset, to simulate the random delay. The networks are similar to the ones trained on synchronized traces but replacing the junior encoder from a LC layer to stacked convolutional layers. Since all of the datasets we used are roughly centered, we approximately rescale the traces to the interval -1 to 1 by dividing an integer. In order to make the networks converge stably under the desynchronized scenario, we also conduct the data augmentation, i.e., additional shifts, to expand the profiling set. We remark that the value of additional shifts for augmentation is variable during the profiling. We also point out that the guessing entropies shown in this section are not directly comparable to the ones given in Section 4.1 because of the data augmentation.

4.2.1 Desynchronized ASCAD

Settings on v1 set. On this dataset, the length of the delay interval is set to 52 (the length of one clock cycle) and 100 to evaluate our network under different degrees of random delay. The length of the augmentation interval is set to 80 for both cases. The time samples are divided by 64. The training setups are the same as the synchronized dataset.

Settings on v2 set. On this dataset, the length of the delay interval is set to 126 (the length of one clock cycle) and 200. The length of the augmentation interval is set to 20 for both cases. The time samples are divided by 128. The batch size is reduced to 40, as even the lightweight stacked convolutional layers require more memory of GPU card compared to the LC layer.

The results of the ASCAD v1 and v2 are presented in Figure 13. As the trace in the ASCAD v1 dataset are too few to support end-to-end profiling in desynchronized cases, a

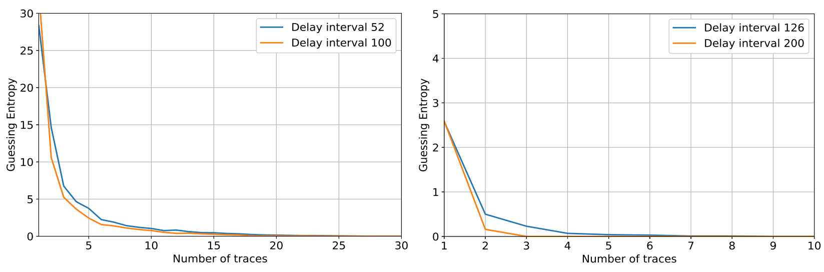


Figure 13: Guessing entropy of desynchronized datasets ASCAD v1 (left) and ASCAD v2 (right).

large augmentation interval is chosen to expand the profiling set 80 times. With such an expansion, our network needs about 20 traces to recover the key. Although considering the data augmentation, our result is not strictly better compared to the state-of-the-arts [ZBHV20, WAGP20] carried out on the PoIs, it is still the first end-to-end result in the desynchronized context for ASCAD. Moreover, we show in Section 4.3 that the networks designed for PoIs do not necessarily fit the end-to-end scenario (even with the same level of augmentation). For the v2 dataset, thanks to the better measurement quality and more traces for profiling, our network reduces the guessing entropy to 0 in 10 traces with less augmentation.

4.2.2 Desynchronized AT128

Settings on AT128-N. We set the length of delay interval to 44 (the length of 4 clock cycles) and 100 on this dataset. The length of the augmentation interval is set to 6 and the batch size is reduced to 100. The value of time samples is divided by 16384 (2^{14}).

Settings on AT128-F. We also set the length of the delay interval to 44 and 100 on this dataset, since the length of one clock cycle is the same between these two datasets. The length of the augmentation interval is set to 6 and 20 according to the length of delay interval 44 and 100, respectively. The batch size is also reduced to 100.

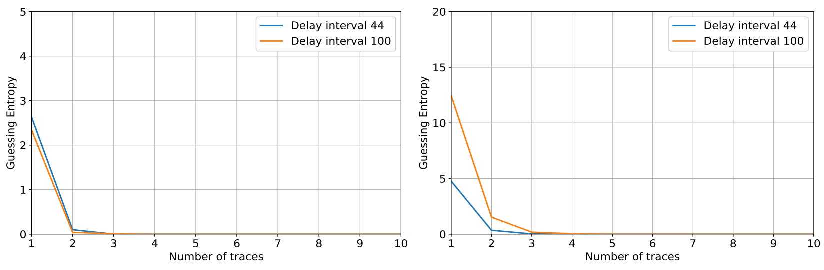


Figure 14: Guessing entropy of desynchronized datasets AT128-N (left) and AT128-F (right).

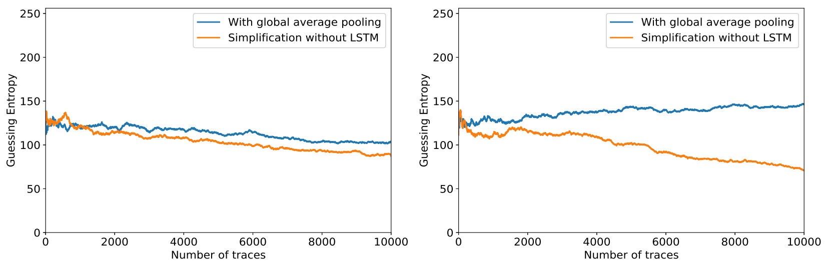
The results of AT128-N and AT128-F are presented in Figure 14. As it can be observed, in all of these four cases, we could reduce the guessing entropy to 0 within 4 traces. We carry out more augmentation on the AT128-F when the delay interval is set to 100. Otherwise, the guessing entropy needs over 100 traces to reduce to 0, which indicates the AT128-F is a more difficult dataset for end-to-end profiling, especially when desynchronization occurs.

4.3 More experiments for comparison

In this section, we conduct more experiments to compare our architecture with others that are potentially effective in end-to-end profiling.

4.3.1 Global average pooling and simplifications of our architecture

First, we explore the CNNs with global average pooling and a simplification of our architecture (i.e., utilize attention on junior encoder directly without LSTM as mentioned in Section 3.3). Concretely, we build the CNNs with global average pooling by modifying the networks proposed for desynchronized traces. We simply increase the filters of each convolutional layer to make sure the final feature dimension is the same as the output of LSTM. For the simplification of our architecture, we also increase the filters of each convolutional layer and remove the LSTM layers. To provide a baseline, we consider the dataset AT128-N and ASCAD v1 with the same setups (i.e., delay and augmentation interval) in desynchronized scenarios. We also keep the training setups the same unless we have to reduce the batch size (as the filters of convolutional layers are increased). More details of the networks are given in Appendix C.



(a) AT128-N **(b)** ASCAD v1

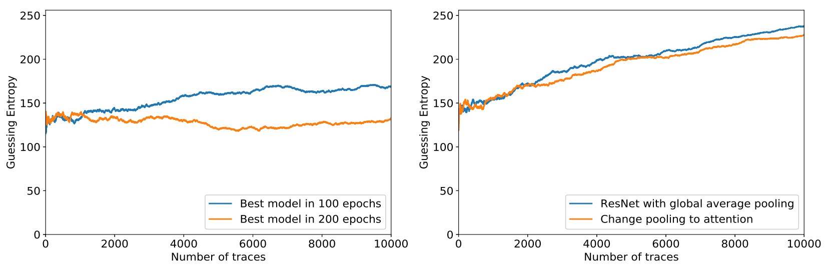
Figure 15: Guessing entropy of CNNs with global average pooling and the simplification of our architecture.

As depicted in Figure 15, both of the CNNs with global average pooling and the simplification of our architecture fail to carry out a successful attack on these two datasets. The results of the global average pooling indicate that it could mix the features up when the final spatial dimension is still high. Meanwhile, the results of the simplification of our architecture imply the recurrent layer is critical in combining features. A simple attention structure (weighted sum) is far from enough.

4.3.2 Applying the state-of-the-arts to raw traces

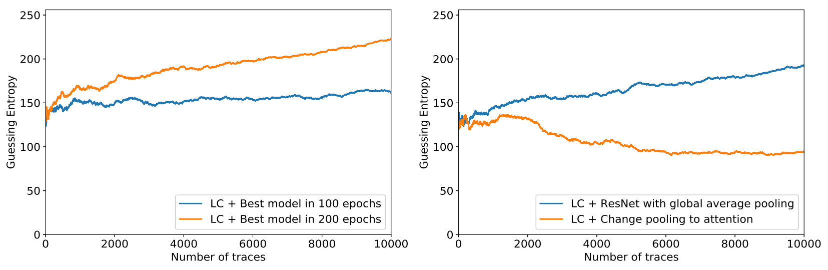
In addition, we also apply the networks proposed in [ZBHV20] and [ZS20] to the raw traces of the ASCAD v1 dataset. For the network proposed by Zaid et al. [ZBHV20], we use the one cycle policy with more epochs (100/200) since the raw traces are more difficult to profile. As the training is conducted on raw traces, we have to reduce the batch size to 50 instead of 100 in the original paper. For the ResNet proposed by Zhou et al. [ZS20], besides testing the original network, we also replace the global average pooling to attention mechanism. We reduce the batch size to 8 (as the ResNet is quite memory consuming) and train them in 200 epochs. We illustrate the results in Figure 16.

The above experiments are mounted on the desynchronized traces where our architecture does not utilize the LC layer to reduce the dimension either. From the viewpoint of just comparing different architectures, the results are clear enough. Nevertheless, the networks proposed in [ZBHV20] and [ZS20] are not designed for raw traces. To make the comparison as fair as possible, we also insert an additional LC layer into these two networks as the first layer to reduce the dimension prior. To this end, we use the synchronized traces as the LC layer cannot handle the desynchronized scenario. The further results are given in Figure 17 that the modified networks still do not carry out a successful attack. This negative observation indicates that the LC layer does not essentially help CNN architectures to profile raw traces. With all these interesting investigations, we find that transforming even the state-of-the-art architectures on PoIs to raw traces is not trivial.



(a) Network proposed by Zaid et al. **(b)** Network proposed by Zhou et al.

Figure 16: Guessing entropy of two state-of-the-art CNNs applied to raw traces.



(a) Network proposed by Zaid et al. **(b)** Network proposed by Zhou et al.

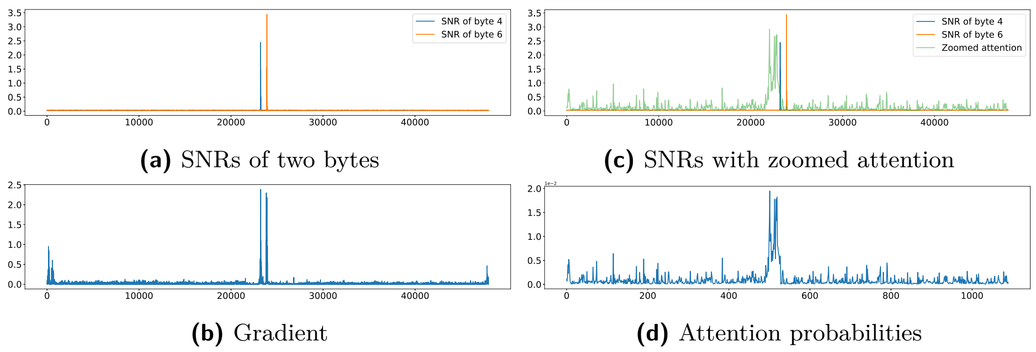
Figure 17: Guessing entropy of two state-of-the-art CNNs (with additional LC layer) applied to raw traces.

5 Investigations of attention mechanism

Although the networks built by our architecture could converge on raw traces to carry out a successful attack, we are more interested in how the networks actually achieve this, specifically, whether the attention mechanism works as we designed. To this end, we investigate the attention probabilities in the trained networks with the results of leakage detections, gradients and the states of LSTMs. Through these investigations, we reveal how our architecture accomplishes the feature extraction and combination. In this part of paper, we focus on the networks trained by the datasets AT128-N and ASCAD v1.

5.1 Leakages, gradients and attention probabilities

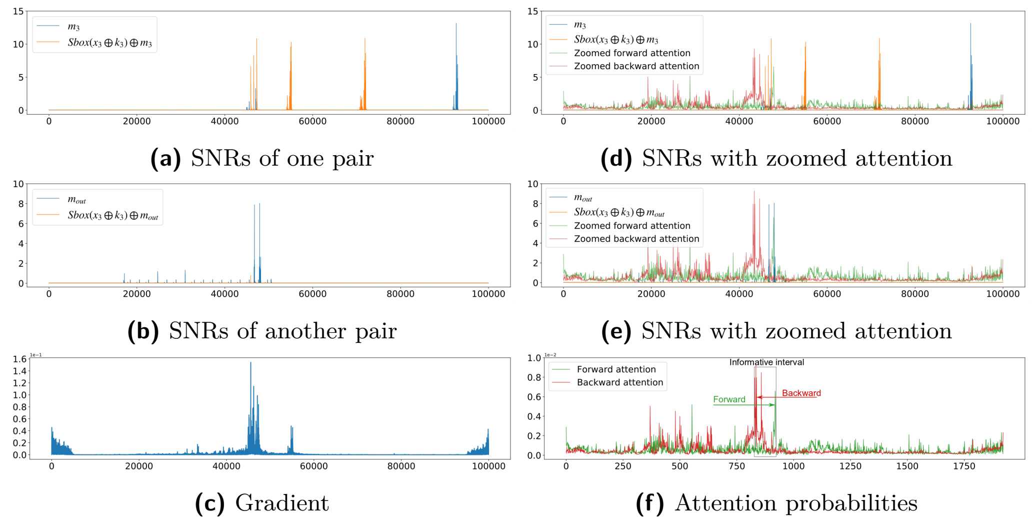
For a deep learning process conducted on raw traces, one of the most important issues is whether the network could extract the PoIs automatically. In this section, we first give the results of leakage detections which imply the positions of leakages. Then we show the absolute gradient of the network (w.r.t. the inputs) which corresponds to the positions where the network is sensitive to. If the positions indicated by the leakage detections and the gradient are highly consistent, we could reasonably infer that the networks extract the PoIs successfully. In addition, by showing the attention probabilities, we could disclose whether the attention mechanism helps to locate the PoIs.



(a) SNRs of two bytes **(c)** SNRs with zoomed attention

(b) Gradient **(d)** Attention probabilities

Figure 18: Results of SNRs, gradient and attention for a network trained on AT128-N.



(a) SNRs of one pair **(d)** SNRs with zoomed attention

(b) SNRs of another pair **(e)** SNRs with zoomed attention

(c) Gradient **(f)** Attention probabilities

Figure 19: Results of SNRs, gradient and attention for a network trained on ASCAD v1.

We illustrate the results of a network (based on architecture variant 1) trained on AT128-N in Figure 18. We first depicted the SNRs as the results of the leakage detections of x_4 and x_6[9] in subfigure (a). Compared to the absolute gradient in subfigure (b), we see the positions of distinct peaks in gradient and SNRs are highly consistent with each other, and thus our network could extract the PoIs accurately from raw traces. Besides, when compare to the attention probabilities, we find the peaks of attention are on the left of those in gradient and SNRs, but at a very close distance. This observation could be explained by that the network is dominated by the backward LSTM. As we use the attention mechanism with LSTM, a distinct peak in attention does not only indicate that the current time step of LSTM is important but also imply that the corresponding interval on the raw trace before (according to the direction) this time step contains information related to the classification target. Consequently, it is reasonable that the attention mechanism pays

[9]We use the manipulations of the input array to simulate the leakages in AT128.

particular attention to the time steps after the backward LSTM has accessed the leakages of both x_4 and x_6.

Similarly, in Figure 19, we show the results of a network (based on architecture variant 2) trained on ASCAD v1. To thoroughly identify the leakages, we calculate SNRs on two pairs of intermediate values, namely, $(Sbox(x_3 \oplus k_3) \oplus m_3,\ m_3)$ and $(Sbox(x_3 \oplus k_3) \oplus m_{out},\ m_{out})$, which are presented in subfigure (a) and (b), respectively. We then give the absolute gradient (w.r.t. the inputs) of this network, showing the network is sensitive to where the mask and the masked value leak. It seems that the leakages of $Sbox(x_3 \oplus k_3) \oplus m_3$ and m_3 are preferred by our network. Noting that we use the architecture variant 2 this time, there are two attention instances in the trained network. The attention probabilities of the forward and backward attention instances are simultaneously shown in subfigure (f), and compared to the SNRs in subfigure (d) and (e) respectively. According to Figure 19(f), both attention instances pay special attention just after LSTMs go through the time steps between 834 and 919, and these time steps correspond to an interval [43368, 47788] on raw traces. In other words, the interval [43368, 47788] is suggested by attention instances to be the most informative interval on raw traces. Not surprisingly, this interval includes the 700 time samples (in the interval [45400, 46100]) which are manually selected by the authors of [PSB+18].

5.2 Interactions between the gate state and attention

Since the data flow in LSTM is self-controlled by the inside gates, quantitatively evaluating how the attention affect it is quite tricky. Nevertheless, we can gain some insight into the internal mechanisms of the LSTM by studying the gate activations when the networks process test data.

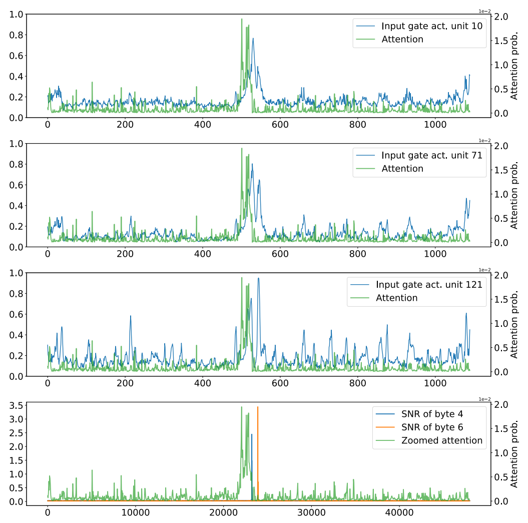


Figure 20: Activations of the input gates in a backward LSTM. The attention probabilities and SNRs are also given to investigate the interactions.

We first select some active units in LSTM from a network trained on AT128-N. We judge the units' activeness based on the gradient of the network(w.r.t. the output of LSTM), where the most active unit generates the most accumulated absolute gradient through the whole time steps. In Figure 20, we plot the activations of the input gate in unit 10, 71 and 121. We observe that the input gates are highly activated at a close distance before the time steps are attended, which is concrete evidence that the LSTM does extract information before attention asks it to yield. The activations of the input gates are also highly related to the SNRs where the two peaks correspond to the leakages of x_4 and x_6, which indicates the input gates let more information in when it is needed. These observations are consistent with the results in the last subsection that the attention mechanism is highly related to the feature extraction and combination. Although not all of the units (about 20 units in the network we explored) in LSTM could be observed with such a clear behavior on input gate, this result verifies that the attention mechanism does help the LSTM to shorten the time steps during which the memory should be kept.

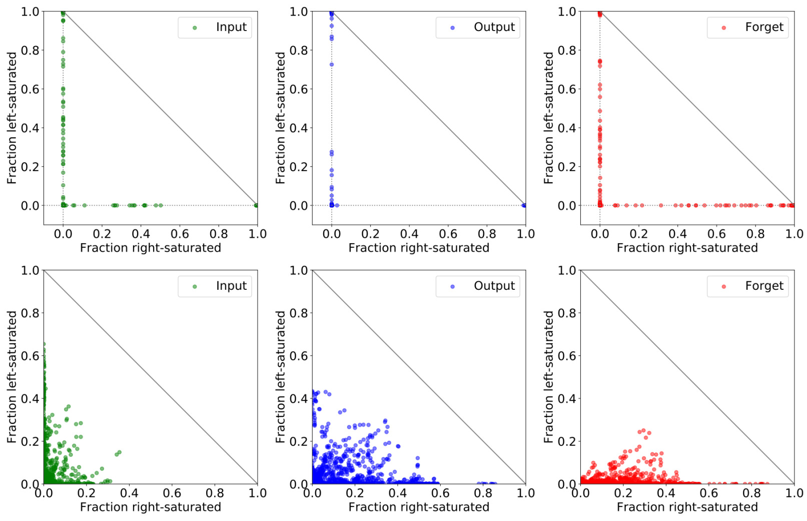


Figure 21: Fractions of saturation for units. The first row shows the units from an unconverged network. All the units are on the axes. The second row shows the units from a converged network. The units perform more differently among time steps. Each subfigure contains 1280 dots.

Then we explore the statistics of the input, forget and output gates to show the behavior of LSTM in a more macroscopical view. We are particularly interested in looking at the distributions of saturation regimes of the gates, where we define a gate to be left or right saturated if its activation is less than 0.1 or more than 0.9, respectively, or unsaturated otherwise. To illustrate these distributions, we calculate the fraction of times for each unit that the gates get into left or right saturated. The results are depicted in Figure 21 (with an unconverged network for comparison), where the units are collected from 10 test samples, and thus there are 1280 dots in each subfigure.

In the context of side-channel analysis, PoIs are usually in a limited number, and hence ideally, the input gate of a unit should be right saturated (let the information in) in several time steps and left saturated in the rest. That is, units occurring at the upper left of the plot (not on the y axis since the fraction of right saturated should not be 0) is the optimal scenario. The analysis of the output gate follows the same logic. As for the forget gate, it really depends on the implementation that determines how long the memory should be kept. Generally speaking, in the side-channel field, the LSTM need to be trained to find the differences (informative or not) among time samples, so that the gates with both saturations (behave more differently among steps) are supposed performing better than those who never be saturated nor saturated on one direction (i.e., units on the axes).

6 Conclusion

In this paper, we propose an end-to-end profiling approach by introducing a new neural network architecture that consists of encoders, attention mechanisms and a classifier. Compared to the current popular architectures in side-channel like CNNs and MLPs, our architecture can directly profile traces that are significantly longer (e.g., raw traces). This property makes our architecture more suitable for end-to-end profiling in which the implementation is protected by masking. Since in this condition, selecting PoIs is quite challenging.

We build the networks guided by our architecture and conduct the attacks on several datasets. To our best knowledge, we are the first that successfully carry out end-to-end profiling directly on the raw traces that contain over 100,000 time samples in each. With these trained networks, we can break the implementations of the datasets, like DPA contest v4 and ASCAD, with very few traces which could be even fewer than the networks trained on length reduced dataset. By replacing the junior encoder from a LC layer to stacked convolutional layers, we could also handle the desynchronized cases. To further explore how our architecture works, we investigate the attention mechanism finding it is highly related to the gradient and the behaviors of LSTM. These investigations indicate how the attention mechanism helps to accomplish feature extraction and combination. Finally, we believe our approach is a first step towards the end-to-end profiling in the context of side-channel.

Besides the successful attacks, there is still some space to improve our architecture. One possible direction is replacing the recurrent layers since the recurrent structures that can not be parallelized will slow down the training process. The self-attention proposed in [VSP+17], which could be parallelized, is a promising candidate as it has set off a revolution of abandoning LSTM in the NLP field. We leave this to be a future work. Considering the richness of nowadays' neural network, it should be interesting to explore more new architectures and analyze their effectiveness to the SCA.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No.62072307) and the Shanghai Science and Technology Committee scientific & research project (No. 19511103903). We thank the anonymous reviewers for their valuable comments on how to improve this paper.

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AG01] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and aes, secure against some attacks. In *Cryptographic Hardware and Embedded*

Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings, number Generators, pages 309–318, 2001.

[AHB+18] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 6077–6086, 2018.

[BBD+14] Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and improvements of the DPA contest v4 implementation. In *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings*, pages 201–218, 2014.

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[BL12] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, pages 263–276, 2012.

[C+15] François Chollet et al. Keras. https://keras.io, 2015.

[CBS+15] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 577–585, 2015.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 45–68, 2017.

[CLS+15] Yu-hsin Chen, Ignacio Lopez-Moreno, Tara N. Sainath, Mirkó Visontai, Raziel Alvarez, and Carolina Parada. Locally-connected and convolutional neural networks for small footprint speaker recognition. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 1136–1140, 2015.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 13–28, 2002.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural*

Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1724–1734, 2014.

[DS16] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 240–262, 2016.

[DSV$^+$15] François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Jean-Baptiste Mairy, and Yves Deville. Efficient selection of time samples for higher-order DPA with projection pursuits. In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, pages 34–50, 2015.

[DZFL14] A. Adam Ding, Liwei Zhang, Yunsi Fei, and Pei Luo. A statistical model for higher order DPA on masked devices. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 147–169, 2014.

[GS00] Felix A. Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000, Neural Computing: New Challenges and Perspectives for the New Millennium, Como, Italy, July 24-27, 2000, Volume 3*, pages 189–194, 2000.

[HGM$^+$11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptographic Engineering*, 1(4):293–302, 2011.

[HKG$^+$15] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701, 2015.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, pages 249–264, 2012.

[Jae02] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[JZHY20] Minhui Jin, Mengce Zheng, Honggang Hu, and Nenghai Yu. An enhanced convolutional neural network in side-channel attacks and its visualization. *CoRR*, abs/2009.08898, 2020.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[KKL20] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.

[KPH$^+$19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):148–179, 2019.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: an approach based on machine learning. *IJACT*, 3(2):97–115, 2014.

[LJD19] Shikun Liu, Edward Johns, and Andrew J. Davison. End-to-end multi-task learning with attention. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 1871–1880, 2019.

[LP07] Kerstin Lemke-Rust and Christof Paar. Gaussian mixture models for higher-order side channel analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, pages 14–27, 2007.

[LPB$^+$15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, pages 20–33, 2015.

[MDM16] Zdenek Martinasek, Petr Dzurenda, and Lukas Malina. Profiling power analysis attack based on MLP in DPA contest V4.2. In *39th International Conference on Telecommunications and Signal Processing, TSP 2016, Vienna, Austria, June 27-29, 2016*, pages 223–226, 2016.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, pages 238–251, 2000.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, pages 3–26, 2016.

[NSGD12] Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. RSM: A small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas. In *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1173–1178, 2012.

[PHJ$^+$19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):209–237, 2019.

[PR07] Emmanuel Prouff and Matthieu Rivain. A generic method for secure sbox implementation. In *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, pages 227–244, 2007.

[PSB$^+$18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[RLL$^+$17] Colin Raffel, Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. Online and linear-time attention by enforcing monotonic alignments. *CoRR*, abs/1704.00784, 2017.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, pages 411–425, 2008.

[Sch08] Werner Schindler. Advanced stochastic methods in side channel analysis on block ciphers in the presence of masking. *J. Mathematical Cryptology*, 2(3):291–310, 2008.

[SLM17] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083, 2017.

[Sut13] Ilya Sutskever. *Training recurrent neural networks.* University of Toronto Toronto, Canada, 2013.

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1701–1708, 2014.

[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 *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010, 2017.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):147–168, 2020.

[XBK$^+$15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015.

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

[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.

A Networks for synchronized traces

A.1 DPA contest v4.1

More details about training and network:

Number of traces for profiling: 39000
Number of traces for validations and attacks: 1000
batch size: 32
learning rate: 0.0001
Number of units in each LSTM: 256
Filter size: 70
Stride: 35
Regularization in LC layer: l2 (1e-5)
Regularization in LSTM: None
Dropout rate: 0.5

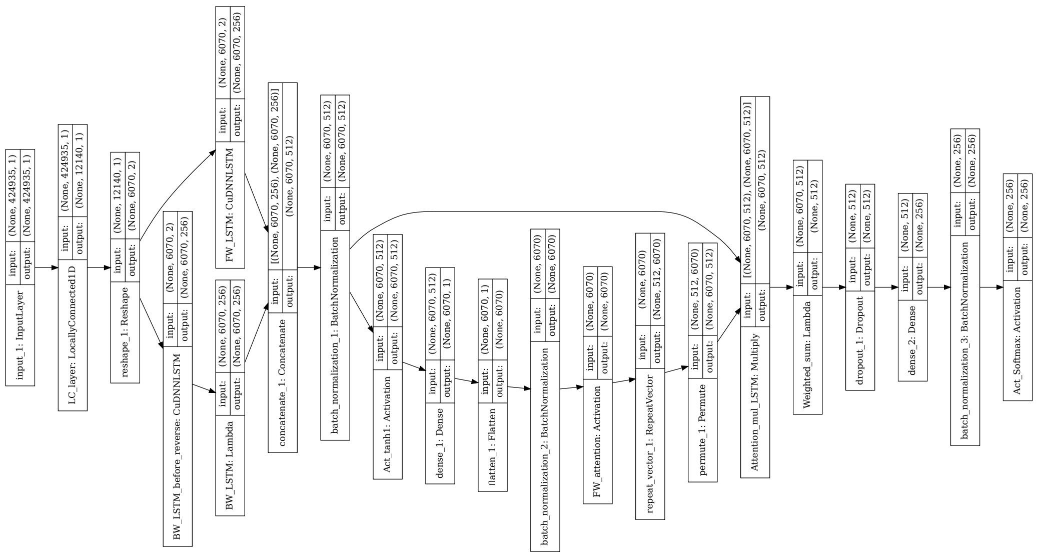

input_1: InputLayer | input: (None, 424935, 1) | output: (None, 424935, 1)
LC_layer: LocallyConnected1D | input: (None, 424935, 1) | output: (None, 12140, 1)
reshape_1: Reshape | input: (None, 12140, 1) | output: (None, 6070, 2)
BW_LSTM_before_reverse: CuDNNLSTM | input: (None, 6070, 2) | output: (None, 6070, 256)
FW_LSTM: CuDNNLSTM | input: (None, 6070, 2) | output: (None, 6070, 256)
BW_LSTM: Lambda | input: (None, 6070, 256) | output: (None, 6070, 256)
concatenate_1: Concatenate | input: [(None, 6070, 256), (None, 6070, 256)] | output: (None, 6070, 512)
batch_normalization_1: BatchNormalization | input: (None, 6070, 512) | output: (None, 6070, 512)
Act_tanh1: Activation | input: (None, 6070, 512) | output: (None, 6070, 512)
dense_1: Dense | input: (None, 6070, 512) | output: (None, 6070, 1)
flatten_1: Flatten | input: (None, 6070, 1) | output: (None, 6070)
batch_normalization_2: BatchNormalization | input: (None, 6070) | output: (None, 6070)
FW_attention: Activation | input: (None, 6070) | output: (None, 6070)
repeat_vector_1: RepeatVector | input: (None, 6070) | output: (None, 512, 6070)
permute_1: Permute | input: (None, 512, 6070) | output: (None, 6070, 512)
Attention_mul_LSTM: Multiply | input: [(None, 6070, 512), (None, 6070, 512)] | output: (None, 6070, 512)
Weighted_sum: Lambda | input: (None, 6070, 512) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 256)
batch_normalization_3: BatchNormalization | input: (None, 256) | output: (None, 256)
Act_Softmax: Activation | input: (None, 256) | output: (None, 256)

Figure 22: Network for DPA contest v4.1.

A.2 DPA contest v4.2

More details about training and network:

Number of traces for profiling: 4500 in each subset
Number of traces for validations and attacks: 500 in each subset
batch size: 64
learning rate: 0.0001
Number of units in each LSTM: 256
Filter size: 140
Stride: 35
Regularization in LC layer: l2 (1e-5)
Regularization in LSTM: None
Dropout rate: 0.5

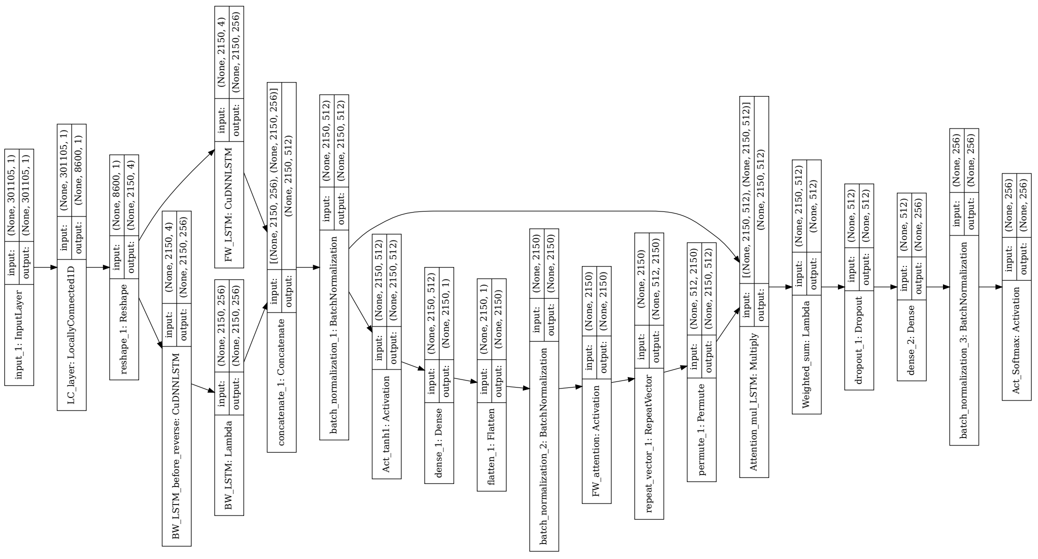

input_1: InputLayer input: (None, 301105, 1) output: (None, 301105, 1)
LC_layer: LocallyConnected1D input: (None, 301105, 1) output: (None, 8600, 1)
reshape_1: Reshape input: (None, 8600, 1) output: (None, 2150, 4)
BW_LSTM_before_reverse: CuDNNLSTM input: (None, 2150, 4) output: (None, 2150, 256)
FW_LSTM: CuDNNLSTM input: (None, 2150, 4) output: (None, 2150, 256)
BW_LSTM: Lambda input: (None, 2150, 256) output: (None, 2150, 256)
concatenate_1: Concatenate input: [(None, 2150, 256), (None, 2150, 256)] output: (None, 2150, 512)
batch_normalization_1: BatchNormalization input: (None, 2150, 512) output: (None, 2150, 512)
Act_tanh1: Activation input: (None, 2150, 512) output: (None, 2150, 512)
dense_1: Dense input: (None, 2150, 512) output: (None, 2150, 1)
flatten_1: Flatten input: (None, 2150, 1) output: (None, 2150)
batch_normalization_2: BatchNormalization input: (None, 2150) output: (None, 2150)
FW_attention: Activation input: (None, 2150) output: (None, 2150)
repeat_vector_1: RepeatVector input: (None, 2150) output: (None, 512, 2150)
permute_1: Permute input: (None, 512, 2150) output: (None, 2150, 512)
Attention_mul_LSTM: Multiply input: [(None, 2150, 512), (None, 2150, 512)] output: (None, 2150, 512)
Weighted_sum: Lambda input: (None, 2150, 512) output: (None, 512)
dropout_1: Dropout input: (None, 512) output: (None, 512)
dense_2: Dense input: (None, 512) output: (None, 256)
batch_normalization_3: BatchNormalization input: (None, 256) output: (None, 256)
Act_Softmax: Activation input: (None, 256) output: (None, 256)

Figure 23: Network for DPA contest v4.2.

A.3 ASCAD v1

More details about training and network:

Number of traces for profiling: 50000
Number of traces for validations and attacks: 10000
batch size: 8
learning rate: 0.0001
Number of units in each LSTM: 128
Filter size: 52
Stride: 26
Regularization in LC layer: l2 (1e-3)
Regularization in LSTM: None
Dropout rate: 0.5

A.4 ASCAD v2

More details about training and network:

Number of traces for profiling: 200000
Number of traces for validations and attacks: 10000
batch size: 200
learning rate: 0.0001
Number of units in each LSTM: 128
Filter size: 125
Stride: 62
Regularization in LC layer: l2 (1e-3)
Regularization in LSTM: UnitNorm constraint on recurrent weight matrix

A.5 AT128-N

More details about training and network:

Number of traces for profiling: 190000
Number of traces for validations and attacks: 10000

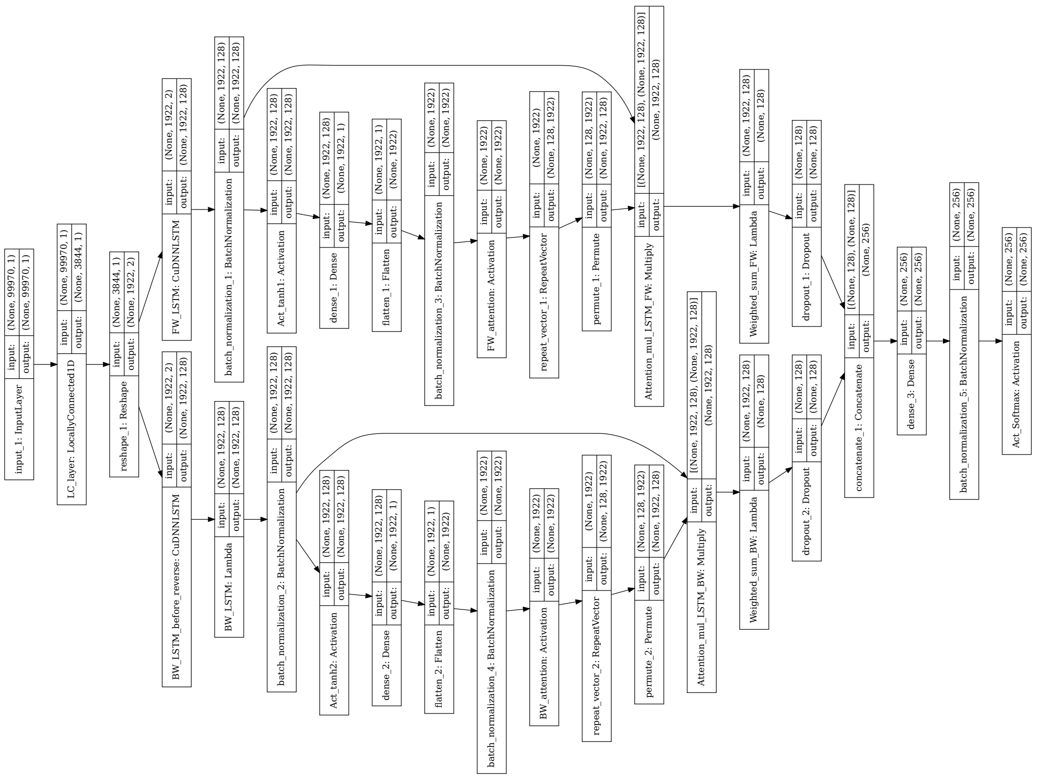

input_1: InputLayer
LC_layer: LocallyConnected1D
reshape_1: Reshape
FW_LSTM: CuDNNLSTM
BW_LSTM_before_reverse: CuDNNLSTM
batch_normalization_1: BatchNormalization
BW_LSTM: Lambda
Act_tanh1: Activation
batch_normalization_2: BatchNormalization
dense_1: Dense
Act_tanh2: Activation
flatten_1: Flatten
dense_2: Dense
batch_normalization_3: BatchNormalization
flatten_2: Flatten
FW_attention: Activation
batch_normalization_4: BatchNormalization
repeat_vector_1: RepeatVector
BW_attention: Activation
permute_1: Permute
repeat_vector_2: RepeatVector
Attention_mul_LSTM_FW: Multiply
permute_2: Permute
Attention_mul_LSTM_BW: Multiply
Weighted_sum_FW: Lambda
Weighted_sum_BW: Lambda
dropout_1: Dropout
dropout_2: Dropout
concatenate_1: Concatenate
dense_3: Dense
batch_normalization_5: BatchNormalization
Act_Softmax: Activation

Figure 24: Network for synchronized ASCAD v1.

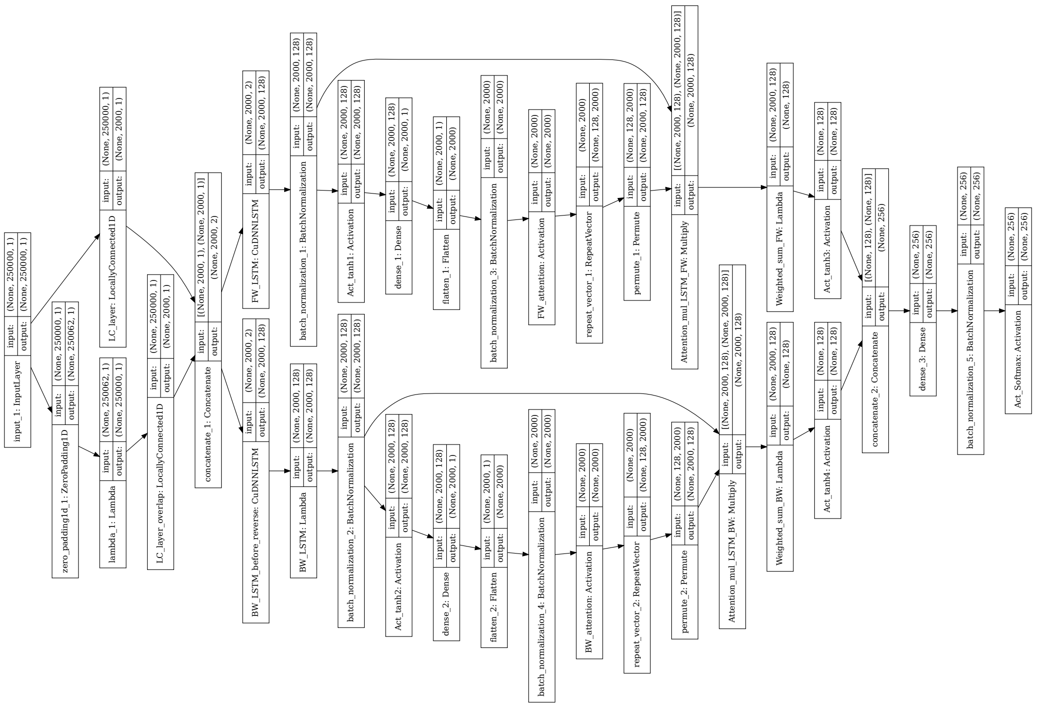

input_1: InputLayer
zero_padding1d_1: ZeroPadding1D
LC_layer: LocallyConnected1D
lambda_1: Lambda
LC_layer_overlap: LocallyConnected1D
concatenate_1: Concatenate
FW_LSTM: CuDNNLSTM
BW_LSTM_before_reverse: CuDNNLSTM
batch_normalization_1: BatchNormalization
BW_LSTM: Lambda
Act_tanh1: Activation
batch_normalization_2: BatchNormalization
dense_1: Dense
Act_tanh2: Activation
flatten_1: Flatten
dense_2: Dense
batch_normalization_3: BatchNormalization
flatten_2: Flatten
FW_attention: Activation
batch_normalization_4: BatchNormalization
repeat_vector_1: RepeatVector
BW_attention: Activation
permute_1: Permute
repeat_vector_2: RepeatVector
Attention_mul_LSTM_FW: Multiply
permute_2: Permute
Attention_mul_LSTM_BW: Multiply
Weighted_sum_FW: Lambda
Weighted_sum_BW: Lambda
Act_tanh3: Activation
Act_tanh4: Activation
concatenate_2: Concatenate
dense_3: Dense
batch_normalization_5: BatchNormalization
Act_Softmax: Activation

Figure 25: Network for synchronized ASCAD v2.

batch size: 128
learning rate: 0.001 with decay rate 0.8 per 30 epochs
Filter size: 44
Stride: 22
Number of units in each LSTM: 128
Regularization in LC layer: None
Regularization in LSTM: UnitNorm constraint on recurrent weight matrix

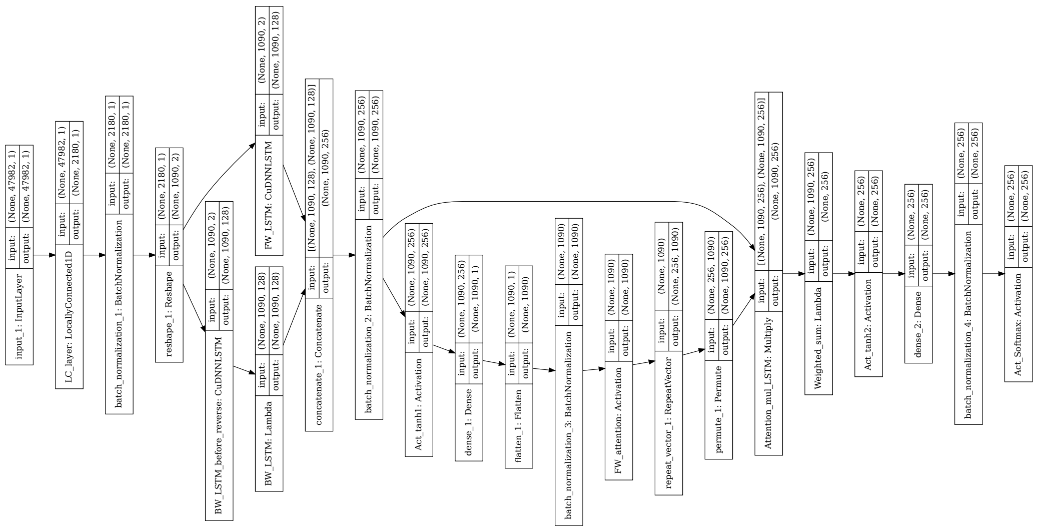

input_1: InputLayer
LC_layer: LocallyConnected1D
batch_normalization_1: BatchNormalization
reshape_1: Reshape
FW_LSTM: CuDNNLSTM
BW_LSTM_before_reverse: CuDNNLSTM
BW_LSTM: Lambda
concatenate_1: Concatenate
batch_normalization_2: BatchNormalization
Act_tanh1: Activation
dense_1: Dense
flatten_1: Flatten
batch_normalization_3: BatchNormalization
FW_attention: Activation
repeat_vector_1: RepeatVector
permute_1: Permute
Attention_mul_LSTM: Multiply
Weighted_sum: Lambda
Act_tanh2: Activation
dense_2: Dense
batch_normalization_4: BatchNormalization
Act_Softmax: Activation

Figure 26: Network for synchronized AT128-N.

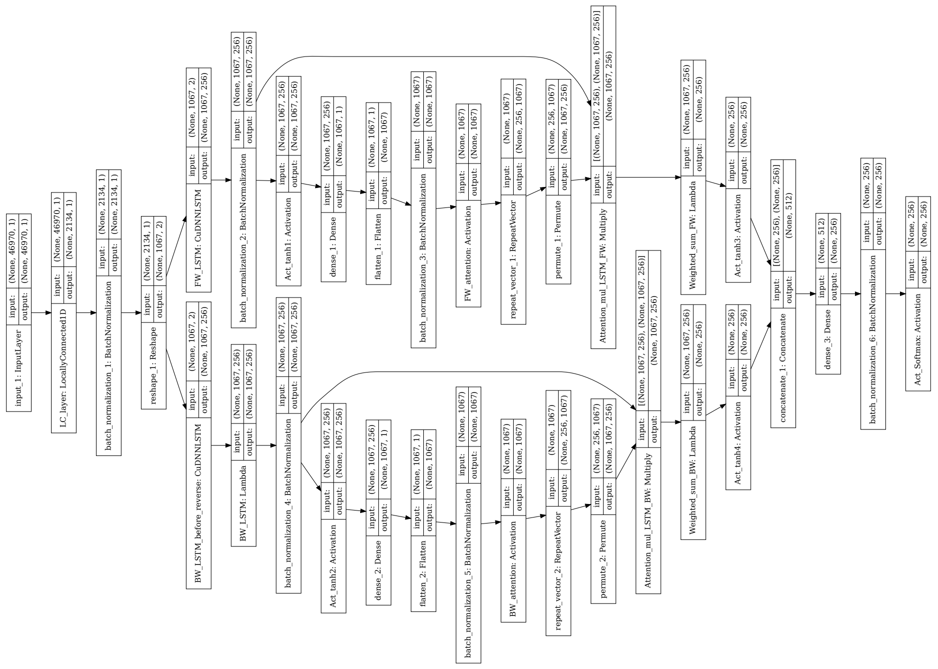

input_1: InputLayer
LC_layer: LocallyConnected1D
batch_normalization_1: BatchNormalization
reshape_1: Reshape
FW_LSTM: CuDNNLSTM
BW_LSTM_before_reverse: CuDNNLSTM
BW_LSTM: Lambda
batch_normalization_2: BatchNormalization
batch_normalization_4: BatchNormalization
Act_tanh1: Activation
Act_tanh2: Activation
dense_1: Dense
dense_2: Dense
flatten_1: Flatten
flatten_2: Flatten
batch_normalization_3: BatchNormalization
batch_normalization_5: BatchNormalization
FW_attention: Activation
BW_attention: Activation
repeat_vector_1: RepeatVector
repeat_vector_2: RepeatVector
permute_1: Permute
permute_2: Permute
Attention_mul_LSTM_FW: Multiply
Attention_mul_LSTM_BW: Multiply
Weighted_sum_FW: Lambda
Weighted_sum_BW: Lambda
Act_tanh3: Activation
Act_tanh4: Activation
concatenate_1: Concatenate
dense_3: Dense
batch_normalization_6: BatchNormalization
Act_Softmax: Activation

Figure 27: Network for synchronized AT128-F.

A.6 AT128-F

More details about training and network:
Number of traces for profiling: 190000
Number of traces for validations and attacks: 10000
batch size: 16
learning rate: 0.001 with decay rate 0.8 per 30 epochs
Number of units in each LSTM: 256
Filter size: 44
Stride: 22
Regularization in LC layer: None
Regularization in LSTM: UnitNorm constraint on recurrent weight matrix

B Networks for desynchronized traces

B.1 Desynchronized ASCAD v1

More details about training and network:
Number of traces for profiling: 50000
Number of traces for validations and attacks: 10000
batch size: 8
learning rate: 0.0001
Number of units in each LSTM: 256
Filter size: 28 in first layer, 3 in the rest
Stride: 1
Regularization in Conv. layer: None
Regularization in LSTM: UnitNorm constraint on recurrent and kernel weight matrix

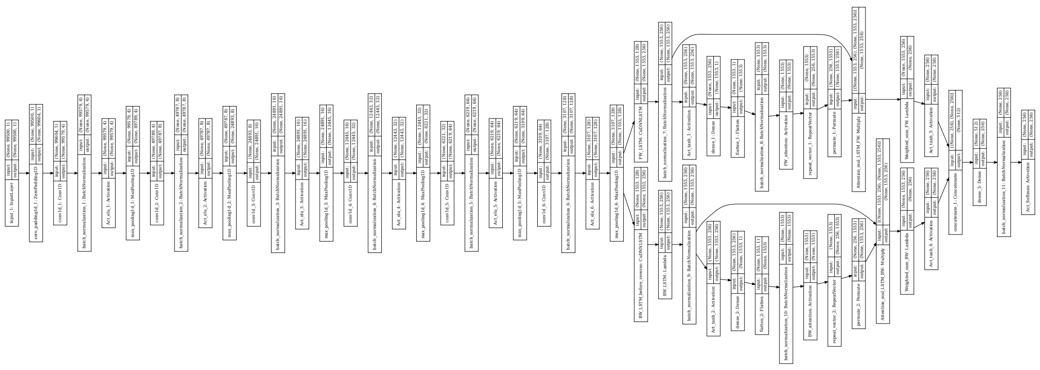


Figure 28: Network for desynchronized ASCAD v1.

B.2 Desynchronized ASCAD v2

More details about training and network:
Number of traces for profiling: 200000
Number of traces for validations and attacks: 10000
batch size: 40
learning rate: 0.0001
Number of units in each LSTM: 256
Filter size: 63 in first layer, 3 in the rest
Stride: 1

Regularization in Conv. layer: None
Regularization in LSTM: UnitNorm constraint on recurrent and kernel weight matrix

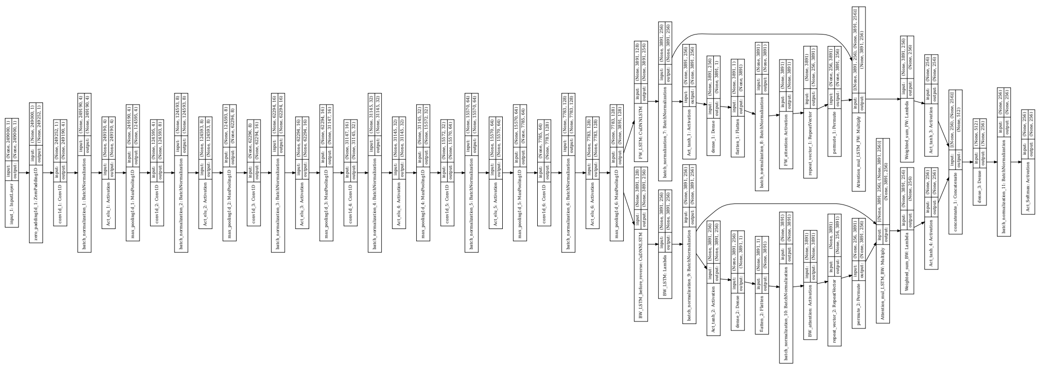


Figure 29: Network for desynchronized ASCAD v2.

B.3 Desynchronized AT128-N

More details about training and network:
Number of traces for profiling: 190000
Number of traces for validations and attacks: 10000
batch size: 100
learning rate: 0.0001
Number of units in each LSTM: 256
Filter size: 11 in first layer, 3 in the rest
Stride: 1
Regularization in Conv. layer: None
Regularization in LSTM: None

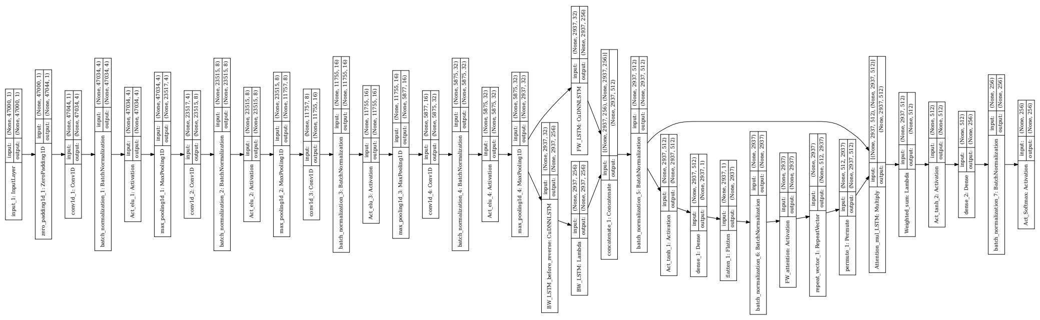


Figure 30: Network for desynchronized AT128-N.

B.4 Desynchronized AT128-F

More details about training and network:
Number of traces for profiling: 190000
Number of traces for validations and attacks: 10000
batch size: 100
learning rate: 0.0001
Number of units in each LSTM: 256

Filter size: 11 in first layer, 3 in the rest
Stride: 1
Regularization in Conv. layer: None
Regularization in LSTM: None

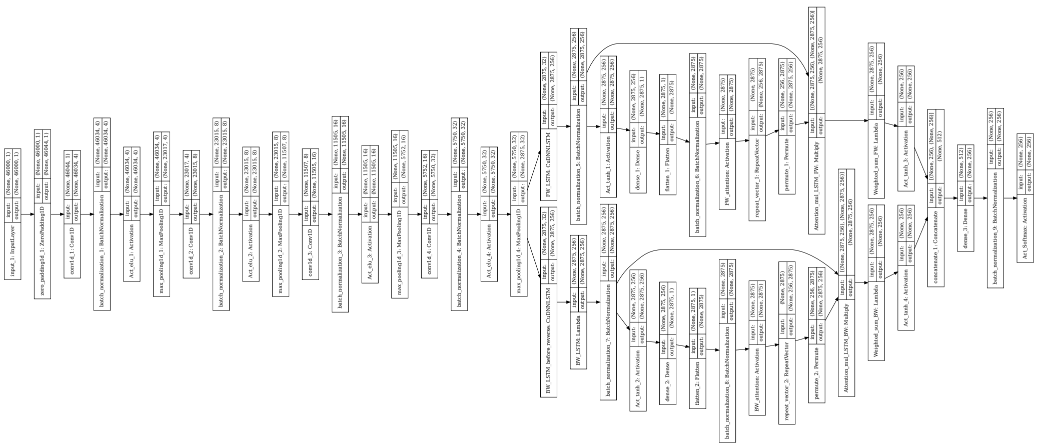


Figure 31: Network for desynchronized AT128-F.

C Networks for additional comparisons

C.1 Modified stacked convolutional layers with global average pooling

We show the stacked convolutional layers with global average pooling which are used in Section 4.3 in Figure 32 and Figure 33

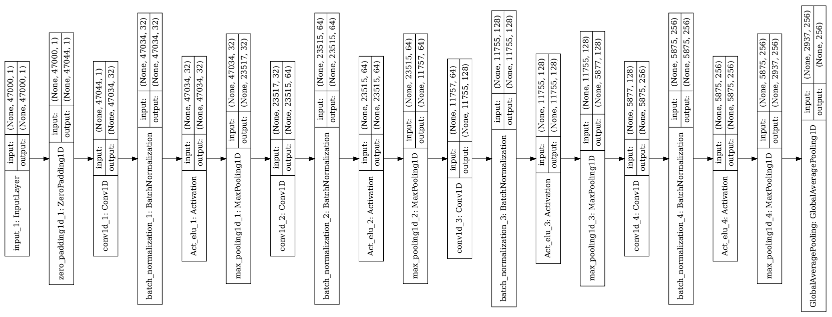


Figure 32: Stacked convolutional layers for AT128-N.

C.2 Network proposed by Zhou et al.

Since Zhou et al. do not provide the network as open source, we implement a ResNet according to their descriptions. More details about the network topology are shown in Figure 34.

D Computation cost

The computation cost of training the networks in Section 4 is shown below in table 2. The training time of each epoch is mainly determined by the length of raw traces, the

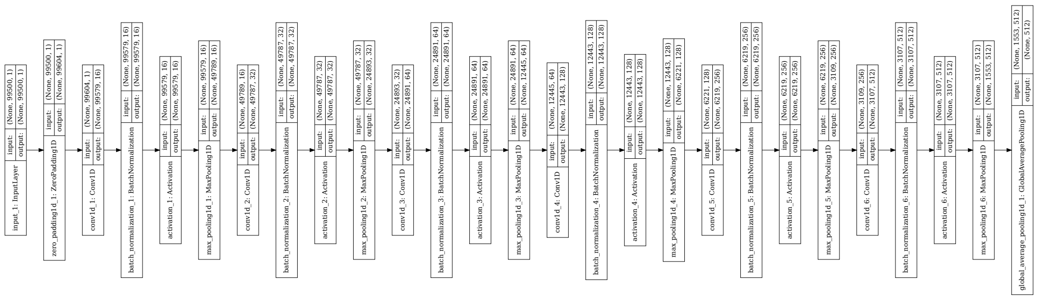


Figure 33: Stacked convolutional layers for ASCAD v1.

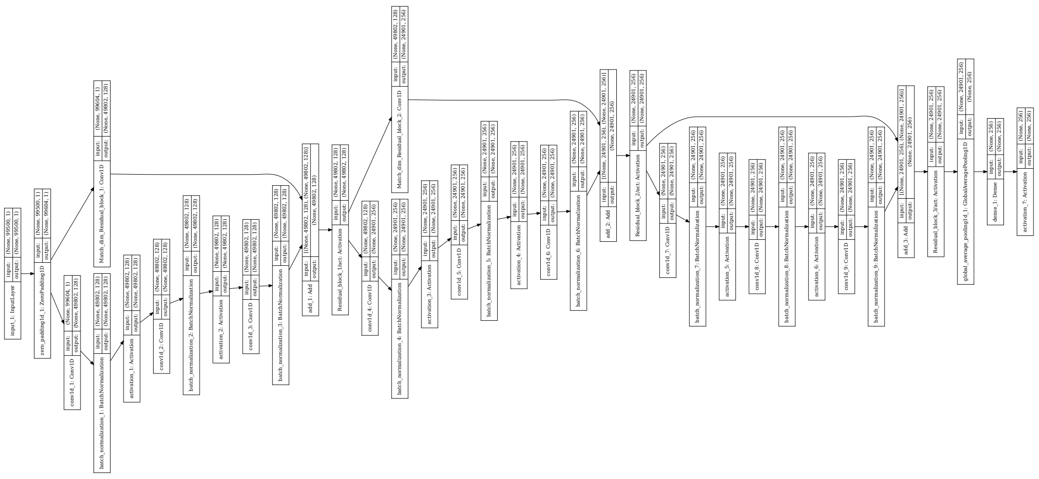


Figure 34: Zhou's ResNet. We implement this according to their descriptions.

size of training sets, and the corresponding batch size. The number of epochs for the first successful attack is also affected by the additional random delay.

E Experiment on hardware implementation

Our architecture focusing on feature extracting and combining on long raw traces is mainly designed for the software implementations where the masking scheme costs more clocks and the leakages of shares spread in a long interval. However, we are also interested in how our approach performs in hardware scenarios. In hardware implementations, the raw traces are usually much shorter, and thus we do not expect a superior result compared to the CNNs and MLPs. We test our architecture on the dataset AES_HD used in [PHJ+19] and illustrate the results in Figure 35. Our approach takes about 800 traces to recover the key, which is close to the state-of-the-art (about 700 traces) presented in [ZBHV20]. We refer to the source code in our Github repository for more details of this attack.

F Comparisons with networks trained on reduced traces

To keep the plots of our results easy to read, we do not directly show the results of previous works in Figures 6 and 8 but illustrate the comparisons in Figure 36.

Table 2: Summary of training cost.

Sec.	Dataset	RD	Variant	Batch size	Epoch time (s)	# of epochs
4.1.1	DPA v4.1	None	1	32	945	10
	DPA v4.2	None	1	64	615	80
4.1.2	ASCAD v1	None	2	8	1040	85
	ASCAD v2	None	2	200	660	5
4.1.3	AT128-N	None	1	128	250	35
	AT128-F	None	2	16	1425	45
4.2.1	ASCAD v1	52/100	2	8	910	90/100
	ASCAD v2	126/200	2	40	3220	40/60
4.2.2	AT128-N	44/100	1	100	1050	70/80
	AT128-F	44/100	2	100	1000	190/245

RD: Random delay.

# of epochs: The number of epochs that for the first time the network could reduce the GE to 0.

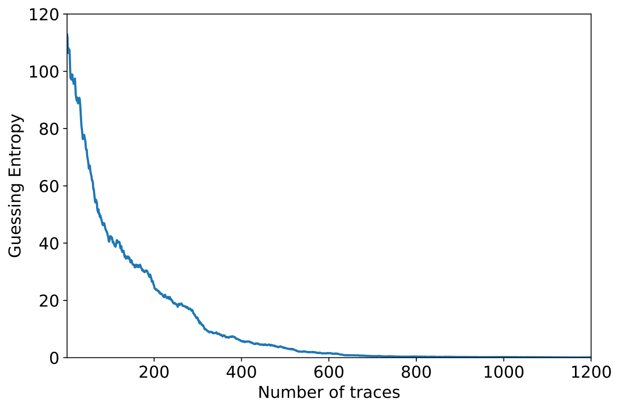
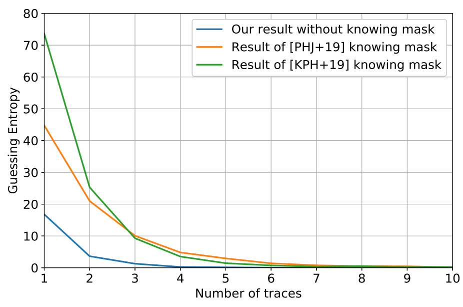
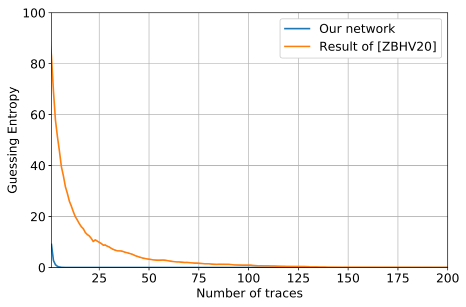


Figure 35: Guessing entropy of AES_HD



(a) DPA contest v4.1



(b) ASCAD v1

Figure 36: Comparisons of guessing entropy.