A Comparison of Multi-task learning and Single-task learning Approaches

Thomas Marquet[1] and Elisabeth Oswald[1,2] $^{[0000-0001-7502-3184]}$

[1] Digital Age Research Center (D!ARC), University of Klagenfurt, Austria
[2] University of Birmingham, UK

Abstract. In this paper, we provide experimental evidence for the benefits of multi-task learning in the context of masked AES implementations (via the ASCADv1-r and ASCADv2 databases). We develop an approach for comparing single-task and multi-task approaches rather than comparing specific resulting models: we do this by training many models with random hyperparameters (instead of comparing a few highly tuned models). We find that multi-task learning has significant practical advantages that make it an attractive option in the context of device evaluations: the multi-task approach leads to performant networks quickly in particular in situations where knowledge of internal randomness is not available during training.

Keywords: Side Channel Attacks · Masking · Deep Learning · Multi-Task Learning

1 Introduction

The rapid adoption of deep learning methods as an alternative to classical statistics in profiled side-channel attacks is due to their superior capability to efficiently utilize information from multiple tracepoints. However, many deep learning architectures still rely on the conventional thinking of traditional statistics-based attacks, where a single intermediate target is learned at a time, resulting in a single learning task being executed.

Multi-task learning is a technique where multiple optimisation functions and their respective ground truths are combined in order to improve the learning of a deep network model. As is so often the case, the idea for multi-task learning is inspired by how humans often learn: in order to perform a new task, humans often try to combine the knowledge gained from (many) related tasks. And maybe to learn a new task that is difficult, humans might first try to learn a related easier task and then move on to the harder problem. These ideas are the key to multi-task learning.

Most fields connected to pattern recognition are investing heavily in the multi-task paradigm with a clear trend to move towards multi-task learning. The learning goals in side channel analysis can be linked to goals within the field of pattern recognition. However,so far multi-task learning has not shown significant promise in the side channel setting.

We believe that this is mostly because the architectures introduced in the initial works are suboptimal. In the first two works by Maghrebi [5] and Masure and Strullu [8], a classical multi-task architecture as summarised in [14] has been used. However, our previous work [6], showed that this kind of architecture is typically inferior to single-task training. However, better ways of defining multi-task architectures are available, and the goal of this paper is to convince readeres that such approaches have strong benefits over single-task learning.

1.1 Summary of Contributions and Outline

Using the public databases ASCADv1-r and ASCADv2, we study the application of multi-task learning in the context of masked AES-128 implementations. Using different masking schemes allows us to explore very different scenarios and really highlight the benefits of multi-task learning approaches. The work of our paper is done assuming no knowledge about the Boolean mask during profiling and attack. With masking countermeasures, the trace complexity of the problem increases exponentially with the number of masks/shares. Therefore even one unknown mask decreases greatly the chances of finding a successful model. After providing some notation and background in Sect. 2, we provide empirical evidence for the improvement of the single-task approach [7] in Sect. 3; we introduce the methodology and present our first experiment on the ASCADv1-r dataset in Sect. 4. We continue with experiments on the ASCADv2 dataset in Sect. 5 and summarise our findings in Sect. 6.

- We propose a methodology for comparing deep learning approaches rather than resulting trained models.
- We demonstrate that the adoption of an "expert" based approach is not only beneficial for multi-task learning but also for single-task architectures.
- We provide evidence that multi-task architectures benefit from shared variables, like masks, even if they are not provided as labels during training.
- We show that multi-task learning enables to find more working models than single-task learning at a fraction of the training time.

We provide the code and our extracted datasets used in this paper at:

- https://github.com/sca-research/multi-vs-single-experiments/
- https://dataverse.harvard.edu/privateurl.xhtml?token=ca37179e-5219-4d49-b0df-719f27924b5e

1.2 Relevant related works

Mahgrebi [5] introduces the multi-task learning paradigm to the side channel community. The model defined in this paper, is a model that learns bit per bit the masked intermediate values, and the related mask. It provides attacks on the ASCADv1 database but with extracted samples.

The paper [10] continues and improves the work done by the community on the ASCADv1-r dataset. Their main discussion evolves around the point of interest selection, which is a critical step in profiled attacks. They set a new state of the art in the scenario where no knowledge about the randomness is assuming during training, and where the training is based on the raw traces (i.e. no points of interest must be extracted).

The introduction and characterisation of the ASCADv2 dataset are done in Masure and Strullu[8]. They provide an extracted version of the dataset and then propose a first utilisation of multi-task learning in a scenario where the knowledge of randomness is assumed during profiling, but also in two cases when a part of the randomness isn't assumed (permutations, and then the multiplicative mask). They set the first state-of-the-art for this dataset, with 60 traces with a single multi-task learning model. However, they use an architecture that doesn't fully take advantage of the benefits of multi-task learning, as the model possesses a branch for each task and then does not connect those branches again.

The paper by Masure et al. [7] explores the idea of training two models simultaneously. In their architecture, they apply a loss function on the predicted combined probabilities of their respective softmax outputs. This approach relaxes the difficulty of the problem, as the network doesn't have to learn how to combine the leakages. The authors solely focus on the PI metric and don't present any attack outcomes.

A further study of multi-task learning is provided in [6]. They propose a new best attack on the ASCADv2 dataset in the scenario where randomness is fully assumed. The authors introduce several novel multi-task architectures and they also utilise custom layers to combine knowledge from multiple branches.

2 Preliminaries

To perform side-channel attacks with deep learning, the attacker typically operates over two stages. The first stage consists of exploring the device leakage, building a dataset, and then training a model. In the second stage, the attacker performs the attack. Such side-channel attacks are often called profiled attacks.

Deep nets aim to learn the leakage distribution of intermediate values. A specific intermediate x corresponds to the chosen points of interest from a leakage trace l and we refer to these points by l_x.

An approach to deep learning consists of the choice of an architecture and a training regime; we focus on single-task vs multi-task architectures specifically. An architecture is a set of layers (of different types) and their connections. Training an architecture results in a deep net model which we call m_θ, whereby θ is the set of hyperparameters corresponding to the model. The output layers of those models are by default unactivated. We call $\sigma(x)$ the activation of the layer x by a softmax function.

2.1 Profiling based on Deep Learning

To perform attacks on newly observed traces, an attacker must first train one or multiple models to build a distinguisher. To build a training dataset, one must obtain a clone of the device to capture training traces. Once the dataset is built, it is possible to train and then utilise the models. Since it is unlikely that the model will recover the target key in only one trace, one must combine the predictions from multiple traces. We note $S_{i,j}$ the scores related to the target byte i of the key, obtained on the trace j picked randomly from the set of attack traces N_a. The recovery of the key will be done using the sum of the logarithm of $S_{i,j}$ in the following way $d[k_i] = \sum_{j=1}^{N_a} log(S_{i,j})$

2.2 Training Methodology

To enable meaningful comparisons, we use the same learning rate, optimizer, and the number of epochs across all training regimes. The only difference is that we train all bytes together in the multi-task models instead of byte by byte training in the single-task models (the multi-task models consist of multiple identical copies of the respective single-task models; we explain them in more detail later in this paper).

Our datasets are split into three different sets: training (size N_T), validation (size N_V), and attack (size N_A) dataset. All results reported in this paper are made on the attack dataset. Training and validation sets are labeled with intermediates derived from random keys. In the ASCADv1-r dataset, fixed keys are provided and therefore our attack dataset is labeled with a fixed key. However, in the ASCADv2 dataset, only random keys are provided. A simple re-labeling of the keys and plaintexts allow us to artificially fix the key. We use a callback to save the best model from the training phase. This callback will either monitor the validation accuracy in the case of single-task models, or the minimum validation accuracy over all bytes, in the case of multi-task learning. Even though accuracy is a problematic metric in side-channel analysis as demonstrated first in Cagli et al. [1] and then in Picek et al. [11], it is difficult to find a better way to judge the quality of the model during the training phase.

2.3 Comparison Methodology

We wish to compare different approaches to deep learning in the context of masked AES implementations. We consider an approach as “better” if desireable qualities (accuracy, time to find a model, time for training) are favorable compared to other approaches. Comparing approaches cannot be done by picking just one or two models because individual models are not necessarily representative of an approach. To compare approaches we need to compare a lot of models (of each approach), and then check how many models of one approach outperform the models from another approach.

This type of comparison also ties in with the reality of side channel evaluations: a model that works well on one device, might not perform well on another device. However, if an approach performs well, then this means that it is likely that we can find a good model for any device. Thus a good approach is beneficial in the long run for an evaluator.

Summarising, we consider the quality of an approach as the reliability with which the approach is going to yield performing models. It might be very hard to estimate the best model possible with an approach, however, the average performance on many hyperparameter sets is a reliable metric. Therefore, to compare approaches, we instantiate a lot of models with random hyperparameters.

2.4 Computing resources

We are using two GPUs: one Nvidia A30 with 24GB of dedicated memory, and one Nvidia A4000 with 16GB of dedicated memory. In addition to the GPUs, we're using 4 cores of an AMD EPYC at 2.6GHz with 128 GB of RAM. As OS we use an Ubuntu 22.04.1 kernel, with TensorFlow 2.10.1.

2.5 ASCADv1-r

The paper of Prouff et al. [13] introduces the dataset in 2018, along with a characterization of the leakage characteristics. the ASCADv1-r dataset possesses a total of 300k traces of a Boolean masking implementation of an AES. The device on which the dataset has been acquired through electromagnetic emissions is a simple 8-bit microcontroller and therefore is very leaky. About two third of the traces are based on using random keys, intended for training and validation purposes, and the rest is based on a fixed key to perform attacks. Because 250k traces are more than enough for training, we decided to only include 60k traces from the random key split, and 10k traces from the fixed key traces.

The dataset contains the encryption up to the beginning of the second round of the AES. The masking scheme is a simple Boolean masking with two shares. Before encryption, a masked SubBytes table $SubBytes^*$ is precomputed using the randomness r_{in} and r_{out}, and during the computation of a masked encryption round, all state bytes are masked by a state mask r_i. The input to the masked SubBytes step is protected by r_{in}, and the corresponding output is protected by r_{out}, as defined by $SubBytes^*$. The output is then remasked with r_i.

The usual target on this dataset is the Subbytes outputs s_i along with the state byte share $(s_i \oplus r_i, r_i)$. However, to leverage the power of multi-task learning, we need a shared mask for all learning tasks, and consequently, we target the less leaky intermediate $(t_i \oplus r_{in}, r_{in})$.

2.6 ASCADv2

Introduced in 2020, the ASCADv2 dataset is less researched in the literature perhaps because it is based on a better protected implementation of AES. The dataset has been generated using the EM waves from an STM32 microcontroller with an ARM Cortex M4. It has in total of 800k traces with a million points per trace. This is because it has been purposely over-sampled and should be resampled for better use. All traces are created based on random keys and therefore it is necessary to artificially fix the key in the attack time. Also, it is very important to shuffle the files, as during acquisition, physical perturbations impacted on the leakage from the device. Therefore, one might observe different attack results when attacking different files if, initially, the training dataset doesn't contain traces from all files. After shuffling the traces, we pick 300k traces at random to build our dataset.

The masking scheme is a shuffled affine masking as defined in [3]. It possesses a non-zero multiplicative mask (note: a few zeros are present in the dataset), and an additive mask. Those masks are respectively noted α and β. Those masks are common for all bytes of the state, implying that the representation of the state bytes around the Subbytes operation is $\alpha \otimes x \oplus \beta$. On the other operations, the masking is done with α and a state mask. We do not use the state mask in this paper. Permutations over all 16 state bytes are present over the whole encryption except MixColumns, in which only the column elements are permuted.

We kept the original notation from the ASCAD database. The subkey of the byte i, is denoted by k_i, along with the Subbytes input and outputs, respectively t_i, s_i. The multiplicative mask is noted r_m, the additive mask for the Subbytes inputs is r_{in}, and the additive mask from the Subbytes outputs is r_{out}. The permuted state bytes are noted respectively k_j, t_j, s_j.

2.7 Custom layers

To fit our design needs, we use several custom layers that were defined in [6], and we include the basic principle to combine outputs of layers for ease of reading. The custom layers compute the joint probability distribution of two variables x, y given the probability distributions of two layers (that we understand to be statistically independent) that depend on x and a function of x, y:

$$\begin{cases} f_{\oplus}(x,y)[i] = \sum_{j=0}^{255} x[j] \times y[i \oplus j] \;\; \forall \; i \;\in\; [0,255] & (1) \\ f_{\otimes}(x,y)[i] = x[0] + \sum_{j=1}^{255} x[j] \times y[i \otimes j] \;\; \forall \; i \;\in\; [0,255] & (2) \end{cases}$$

The function $f_{\otimes}$ has to discriminate the first case where $j = 0$, being a null element. We decided that in this case, the probabilities of x should be unchanged. The use of these functions will become clear from the architecture.

2.8 Multi-Task Learning

Multi-task learning is a very natural concept introduced in Caruana [2]. It can be defined as a network architecture that has multiple outputs and is trained with multiple labels. During training, multi-task models try to optimise multiple objective functions. Across all domains where deep learning is the state-of-the-art, multi-task learning approaches are among the most used, see [14].

The main potential benefits according to Caruana [2], are the following: data amplification, attribute selection, eavesdropping, and representation bias. **Data**

amplification can also be understood as data augmentation. The idea is that the samples from different tasks sharing features might share the signal but not the noise as the noise is independent of the signals. By training both tasks together, the noise is effectively reduced. The concept of **attribute selection**, is subsequent to the concept of data amplification. As the noise is reduced and the signal clearer for a shared feature, the relevant inputs will be easier to find. **Eavesdropping** is a very interesting case where two tasks share a feature. But this feature is hard to understand for one task and easy to learn for the other. Then, the first task is going to benefit from collaborating with the second task since the knowledge of the feature will be shared. And finally, **representation bias** is the idea that a model trained with multiple labels is going to yield more consistent results. Since the training of deep networks is a stochastic process, the "path" taken by the gradient depends heavily on the initialisation. However, since there are multiple objectives to learn, the gradient will take the path that is best for all objectives instead of just one. In addition to a regularisation effect on the network the model weights will prefer a reduced set of representations.

However, those strengths might also be the drawbacks of this technique. Some potential benefits listed above assume that tasks share features. If there is no feature shared between the different objectives, this technique can yield sub-optimal models as the gradient will struggle to find a common "path". Even though, some results [9], [15] have shown that including unrelated tasks in the training procedure can lead to improvements, it is very hard to infer from previous literature if one set of task will perform better trained jointly. One reason is the lack of a cross-domain formalism that would explain when tasks collaborate or compete.

3 Utilising Experts to Improve Single-Task Architectures

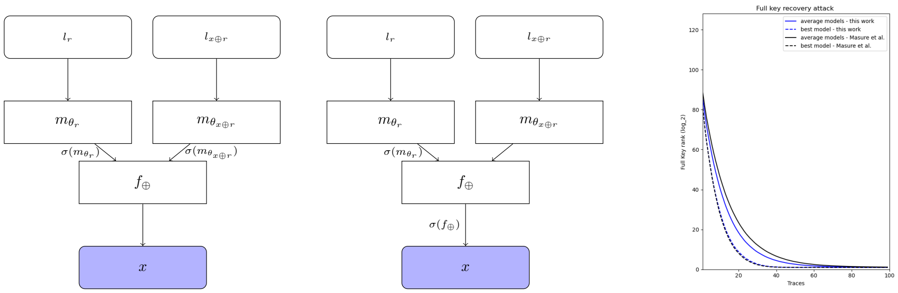
Only in very recent work [6], multi-task learning appears to show any real benefit over single-task learning. This begs the question why the existing multi-task architectures that were proposed in the side channel community do not work as well as they should. In this section we show that a seemingly small "tweak" in combining learned distributions within a network can significantly benefit its performance. We do this by referring back to a recent single-task architecture that combines information that is learned about a mask and the corresponding masked value, and modify it by borrowing inspiration from [6], and then show that our tweak improves the network's performance in a single task setting. We will use this tweak then in our multi-task architectures as well.

The initial single-task architecture proposed in [7] produces probabilities from the activated outputs of two "models", trained with one loss function. One model that we denote m_{θ_r} is expected to learn the mask r, and the other model denoted $m_{\theta_{x \oplus r}}$ is supposed to learn the masked intermediate $x \oplus r$. This technique feels natural in the sense that the probability distribution of x can be derived by the learned distributions of the two branches. We provide a visual representation of this architecture in Fig. 1a.

In [6] the idea of "weighing" up learning results was introduced in the context of multi-task learning. Their way of combining the learning from related branches can be understood as a "multi-gate mixture of experts model", see [4]. In a multigate mixture of experts model the idea is that one branch acts as an "expert" for one or more other branches, by contributing its learning to update the learning of the other branch(es). We find that this idea is appealing when different parts of a network learn intermediate values that are related to the same internal randomness (aka mask) and thus use this to further improve the recent results of [7].

We suggest that the idea of using experts can also be applied to single learning tasks. For instance, we can view the related learning tasks for $x \oplus r$, and r as training two experts who can influence each other. Their outputs are combined via a custom layer and only thereafter a softmax layer is applied to produce a distribution for x. This is in line with the design of [6], but simplified, and adapted to the single-task approach from [7]. We provide a visual representation of this architecture in Fig. 1b.

To evaluate if there is merit in this idea, we run a set of experiments on the ASCADv2 dataset. The setup of this experiment is the same as the extracted scenario in Figure. 5 and therefore the final architectures used in the experiment are based on Figure. 6. We are targeting the unmasked values of the subbytes inputs t_j to compare the better positioning of the softmax activation σ. Knowledge of the multiplicative mask r_m and the permutations j_i is being assumed. Therefore we have in this experiment $x_i = r_m \otimes t_j$ and $r = r_{in}$.



(a) Masure proposition

(b) Our proposition

(c) Comparison

Fig. 1: Single-task architecture difference from Masure et al.[7], with a full key recovery comparison

We can see in Figure. 1c that the architecture where we use the learning of one share as an expert for the other share increases the average performance of the models and has the same amount of successful model as the approach from Masure et al. [7]. These results are obtained with the same randomly chosen set of hyperparameters.

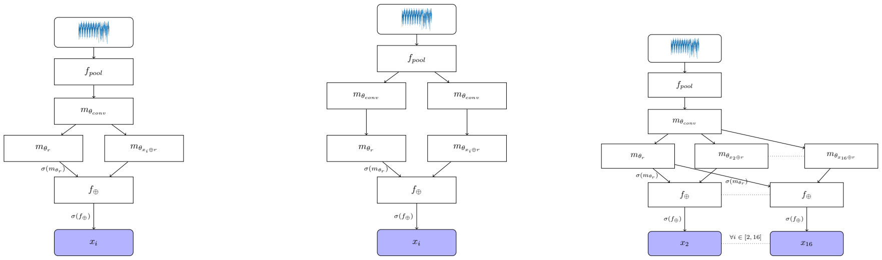
4 ASCADv1-r: Comparing Multi-Task and Single-Task Architectures

In this section we will use the idea of experts for all architectures alike.

4.1 Assumptions, Contributions and State of the Art

ASCADv1-r has been extensively researched by the community in all sorts of scenarios and it has come to light that this dataset leaks many intermediate values, giving many opportunities for attacks. In this paper, we use only the raw traces, without access to any randomness. The state of the art in this scenario is an attack using **1** trace, in [10], while targeting the SubBytes outputs. We choose another target and therefore a straightforward comparison is difficult. We choose to attack the SubBytes inputs t_i because they all share the same mask across all state bytes, and thereby enable this multi-task learning approach. Recall that the shared mask is r_{in}. Therefore we note in this section $x_i = t_i$ and $r = r_{in}$

4.2 Architectures



(a) Single-task-xor model for one byte

(b) Single-task-twin-xor model for one byte

(c) Multi-task model for all bytes

Fig. 2: Architectures used in the Ascadv1 experiment

We provide a visual representation of all network architectures that we use in the context of the ASCADv1-r dataset in Figure. 2. All architecturs take as input the raw traces from the ASCADv1-r dataset. Those raw traces are 250k samples long and include the full execution of the masking scheme up to the end of the first round. Inspired by the work in [10] in relation to the NOPOI scenario, we give to each model a weighted average pooling of the traces. This pooling takes place in the f_{pool} layer. After this layer, which in the multi-task architecture is provided to all tasks, the model is going to split into multiple branches supposed to learn each one a piece of the puzzle t_i.

We have two models in the single-task learning approach. This is because we were interested in comparing the impact of shared convolutions in a single-task scenario. Even though it is not given multiple labels, each branch is supposed to learn a different part of the problem. This might cause an increase in the difficulty of learning.

4.3 Training many models

We train 50 CNNs for each approach. We do not train all bytes for the single-task scenarios because of the significant overhead, instead we only compare byte 6 (7-th byte because the labels start from 0). The attacks are therefore also only on byte 6. All bytes are trained for the multi-task model and reach similar accuracies.

Training time. On our A30 GPU, each epoch for both single-task models takes about 17-25s while each epoch from the multi-task model takes 20-27s depending on the hyperparameters. This is relatively close since the input is huge and the "processing" layers are the bottleneck of the model. Therefore the difference in training time between the two approaches is almost irrelevant, except that in the case of the multi-task model, all bytes are trained.

Hyperparameter choices. The choice of hyperparameter is done fully randomly. Both models start with a layer that reduces the size of the input using weighted average pooling, followed by convolutions. Those convolutions are shared for both models, however, the architecture splits with different fully connected layers.

Set	Hyperparameter	Interval
θ_{conv}	convolution block	$[1, 3]$
	kernel of block i	$[16, 64]\ \forall\ i$
	filters	$[3, 16]$
	strides	$[2, 30]$
	pooling size	$[2, 5]$
$\theta_r, \theta_{x \oplus r}$	dense blocks	$[1, 5]$
	dense units	$[64, 512]$

Table 1: Table of the architectural hyperparameters

4.4 Results of the experiments

For every model trained, we perform 1000 key recovery attacks using 1000 randomly picked traces from the attack dataset. In this section, a model is deemed successful if the average guessing entropy on those 1000 experiments, is less than the full number of traces.

The main metric used to compare the approaches is the number of successful models n_{win}. Additionally, we compare the best models and the average performance of successful models. Those key metrics are given in Figure. 3; and Table. 2. Note that **unsuccessful approaches do not appear** in Figure. 3.

Model type	n_{win}
single-task-xor	0
single-task-twin-xor	2
multi-task	8

Table 2: Successful models

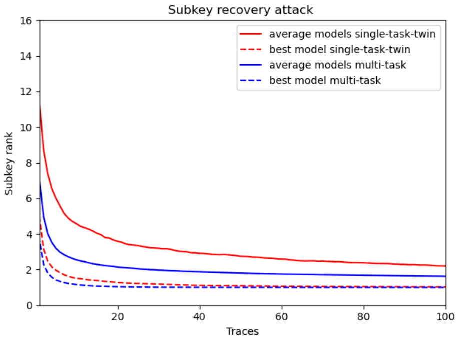


Fig. 3: 7-th subkey recovery attack results for the successful models

Single-task-xor vs single-task-xor-twin. In the case of single-task models, we see that the model that does not share the convolutions outperforms the one with shared convolutions. The extra hyperparameters help the model converge twice, while the single-task-xor architecture struggles to even find one model.

Single-task vs multi-task models. In this experiment, which is based on a traces that are very long, we hypothesise that the the main difficulty for any network is to find the leaking trace points; the amount of leakage in "the right trace points" is known to be strong. Our results suggest that multi-task learning seems to be better in finding working models. The average model performance for both successful approaches is low because only very few models managed to converge according to the results in Tab. . In the case of the single-task-twin-xor model, the best model reached rank one on average at trace 5, while the second model converged, after 444 traces. For the multi-task approach, while 6/8 models managed to reach on average rank 1 using fewer than 20 traces, the two other models were significantly less performant. This make-or-break success is highlighting the fact that the difficulty for the networks is to make sense of the samples. Because when they understand the leakage, they perform as expected. Another reason of this disparity, is the problematic relationship between accuracy and key ranking [1]. On this dataset, the networks are often likely to overfit and be over confident. Because they do not learn the leakage distributions, their first ”bad” predictions are hard to fix. In this case, multi-task learning brings a much better approach than single-task learning. While it requires almost as much time to train all bytes with this approach than training a single byte with single-task learning, it yields more often succesfull models, with a better overall performance.

5 ASCADv2

5.1 Assumptions and state of the art

The first best results were achieved by Masure and Strullu [8]. They required only **60** traces for successful key recovery in a setting where access to all randomness is assumed during training, but no knowledge about the randomness is assumed during the attack. Then an improvement of the state of the art has been achieved in [6], reducing the number of traces to recover the full key at **24** traces. Their results show that to perform the best attacks on ASCADv2, one has to target the input of the AES Sboxes t instead of the more commonly used output s because the signal in the inputs is stronger than the signal in the SubBytes output. We also select the SubBytes input in this work.

We do not work with raw traces, but select points of interest to contain only the leakages from all bytes of the SubBytes inputs of the first round, and their mask r_{in} (we thus assume knowledge about these locations for this purpose). However, we do not include knowledge about additive randomness, and the choice to extract “windows of interest” is to speed up experiments (we already considered a scenario with long traces in the previous section). Extracting points does not impact on the comparison between single-task and multi-task models, because the selection is helping both approaches (and perhaps slightly more the single task models).

In addition to the assumption needed to extract the samples from the raw traces, we give to both approaches the knowledge of the multiplicative mask r_m and the permutations. Therefore the only randomness not assumed during profiling and attack is the additive mask r_{in}. Since the multiplicative mask and the permutations are assumed, $x_i = r_m \otimes t_j$ and $r = r_{in}$ are in the following section for clarity purposes.

5.2 Input scenarios

To reduce the size of the traces given to the networks and ease the problem of finding relevant samples, we extracted points of interest using a standard SNR analysis. Then, we recombine the points of interest of different intermediate values into three different “extraction levels”: fully-extracted, separated, and concatenated.

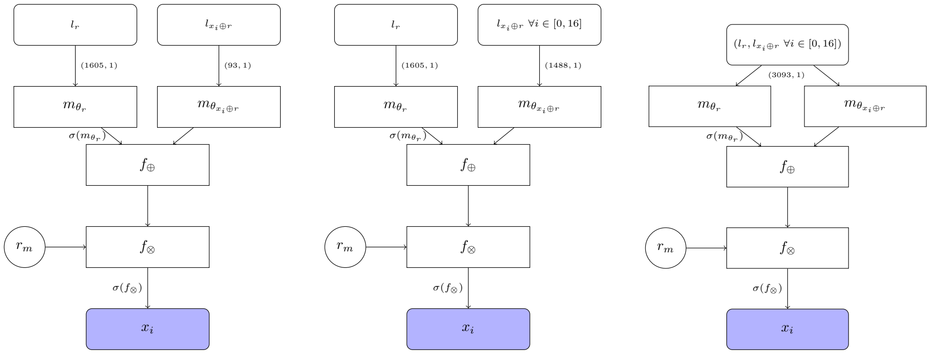
- “Fully-extracted” is a scenario where the samples from the mask but also the individual bytes of the targeted intermediate, are given as inputs independently.
- “Separated” is a scenario where the samples related to the mask are given independently from the samples related to all bytes of the intermediate. However, the latter are fed to the network in a concatenated manner.
- “Concatenated” is a scenario where all extracted samples are concatenated in a single input.

In both fully-extracted and separated scenarios, the leakage related to the mask r_{in} is given to the “mas” branch of the network. This will make it easier for the network as this branch will only see helpful samples. In the concatenated strategy, all extracted samples are concatenated to recreate a large trace. This will allow us to further investigates the attribute selection benefit of the multi-task approach.

In the ”fully-extracted” scenario, each byte samples are kept away from the others. This means that each model branch will have only the samples from what it should learn the distribution. This represents the best case scenario for the single-task models.

5.3 Architectures

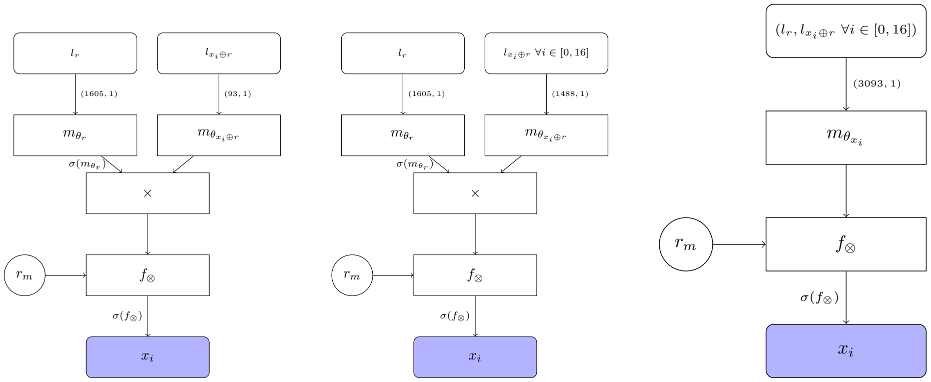
To compare against the existing architectures in the literature, which are the approach introduced in Masure et al. [7], and multi-task learning, we define three architecture types in this : single-task, single-task-xor and multi-task. We give a visualisation of the considered single-task-xor architectures in Figure. 4.



(a) fully-extracted (b) separated (c) concatenated

Fig. 4: Architectures for the single-task-xor approach

In the case of the single-task type, the architectures are provided in Figure. 4a, 4b. They are the same as before with the exception that the custom layer $f_\oplus$ is the multiplication layer. However, in the concatenated scenario, the architecture differs as can be seen in Figure. 5.



(a) fully-extracted (b) separated (c) concatenated

Fig. 5: Architectures for the single-task approach

The multi-task models are extensions of the architectures in Figure. 4 in the same manner as Figure. 2c. The branching is the same but the model is extended to all x_i.

5.4 Training many models

To better compare the approaches, we train many models with random sets of hyperparameters and observe metrics such as the number of successful models, the average number of traces to recover the full key for the successful models, and the best model in terms of number of traces for all approaches. All models in this section are CNNs.

The model that will represent the learning of the mask, m_{θ_r} has been pre-tuned. This means that we selected the hyperparameters for this training task on beforehand and tested them with the labels. We supply this model to all networks, and the only reason for pre-tuning is performance because we wish to obtain a good amount of working models and therefore need to reduce the variance in the hyperparameter choices.

A special case is the case of the architecture in Figure. 5c. This architecure has only one branch all the way to the end, and the branch supposed to act as the mask m_{θ_r} doesn't exist. So the hyperparameters chosen for $m_{\theta_{x_i \oplus r}}$ are given to the model named $m_{\theta_{x_i}}$ with $\theta_{x_i} = \theta_{x_i \oplus r}$. We choose the $\theta_{x_i \oplus r}$ randomly from the intervals defined in the table 3.

Fixed training hyperparameter. The number of epochs and the batch size have been chosen to have the best single-task-xor in the full-extracted scenario,

Hyperparameter	m_{θ_r}	$m_{\theta_{x_i \oplus r}}$
convolution block	1	$[1, 3]$
kernel of block i	32	$[4, 32]\ \forall i \in [1, cb]$
filters	16	$[3, 16]$
strides	10	1
pooling size	2	$[2, 5]$
dense blocks	2	$[1, 5]$
dense units	256	$[64, 512]$
batch norm	yes	yes

Table 3: Table of the architectural hyperparameters

with a slight number of extra epochs. We train all our models with 25 epochs and a batch size of 500. The chosen optimizer is Adam, with a learning rate of 0.0001. No search of any kind has been done for those hyperparameters.

Training time. The attacks proposed in the following subsection are done on the full key. This means that all bytes have been trained for all approaches and scenarios. In this case, models were trained across the GPUs, and therefore it is hard to compare the training time. But since the samples are extracted from the traces, the difference between training all bytes of the single-task model and the corresponding multi-task model is smaller. Training all bytes of the single-task approaches took around 2 times more than the multi-task approach.

5.5 Results of the experiments

To test the attack performances of our models in a full key recovery attack. We perform an attack with all models over 1000 experiments. In each experiment, we pick at random 1000 traces from the attack set and we try to recover the full key. A model is deemed successful if it recovers the key in all experiments. We note the number of successful models for each approach and each scenario in Table. 9. The figures 6, 7, 8 presents the average performance over all successful models, along with the best model found for each input scenario.

Single-task vs single-task-xor First, it appears clear that the single-task-xor is a better approach than the classical idea (i.e. single-task [13] , [12], [10]) that a deep net can learn to find the leakage from the sample, but also how to combine it. Doing a conditional probability between the output of a ”mask” branch and the output of an ”intermediate” branch, give significantly better results than the naive approach. This result confirms the results from [7], on actual attacks. On the 25 sets of hyperparameters chosen, the single-task didn’t get a single one successful in all scenarios.

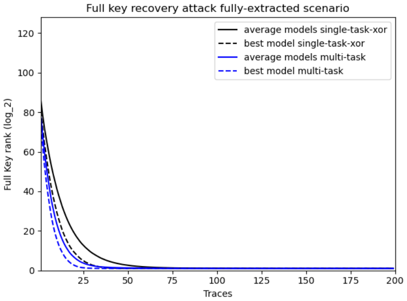


Fig. 6: fully-extracted

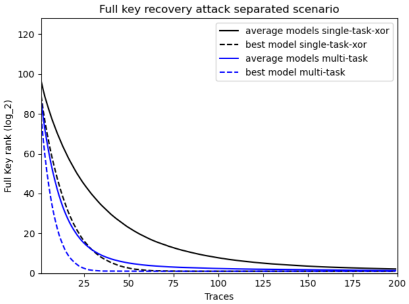


Fig. 7: separated

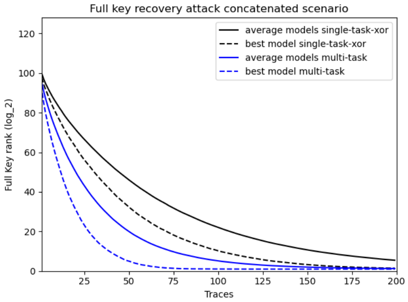


Fig. 8: concatenated

Strategy	Approach	n_{win}
fully-extracted	single-task	0
	single-task-xor	24
	multi-task	25
separated	single-task	0
	single-task-xor	16
	multi-task	22
concatenated	single-task	0
	single-task-xor	17
	multi-task	24

Fig. 9: Successful models

ASCADv2: Results of the experiments

Single-task-xor vs multi-task Secondly, we see that the "mask expert" (i.e. m_{θ_r}) has a much better understanding of the mask distribution and therefore helps the networks trained with this approach to outperform the lone single-task-xor models. In this experiment, the difficulty isn't in finding the tracepoints, as in the first experiment, but in how to not overfit. The regularisation effect of multi-task learning allows more hyperparameter sets to provide successful attacks, and the good models are overall better.

Fully extracted. In the fully extracted scenario, the average performance is close to the best model which itself is catching up with the best attack in a white box scenario reported in [6]. Even though there are obvious differences because here r_m and the permutations are assumed, it is still quite close. This means that most of the leakage has been captured by the model. The difference between the single-task-xor isn't huge, as in this very relaxed assumption, the problem isn't so hard. Still, one model didn't succeed to learn anything and the best model from the single-task-xor approach is inferior to the average performance of the multi-task approach.

Separated. The most interesting difference happens in the separated scenario. Where the samples related to each byte aren't extracted. The multi-task learning scenario is doing much better at identifying the good samples since it knows a little bit more about the problem. The best model is also very close to the fully extracted scenario. In a single-task-xor approach, only 64% of the hyperparameter sets yield a successful model against 88% from its multi-task counterpart.

Concatenated. Finally, in the concatenated, where the inputs are given altogether. We see a very significant difference in pure performance between the two approaches. Even the best model from the single-task-xor models is a lot less performant than the average multi-task model. Interestingly, this input scenario yields more successful models with both modeling strategies. But the difference between the two is still significant.

6 Conclusion

Throughout our two examples, we showcase the strengths of multi-task learning over multiple input sizes when the access of the randomness is not possible, or limited. From our point of view, the key takeaway points are the following:

- Single-task-xor learning, as introduced in [7] and [6], is improving a lot over the previous single-task learning paradigm.
- Training in a multi-task learning scenario will yield more models that can perform an attack.
- Training in a multi-task learning scenario improves the performance of successful models thanks to its regularisation effect.
- In addition to performances gains, the training time can be radically improved.

The results presented in this paper contribute to the community by showcasing the strength of multi-task learning against single-task approaches. We propose a methodology to compare approaches through the training of many models with random hyperparameters. Using this methodology, we highlight the difficulty in the previous single-task learning paradigm, to find models able to perform attacks without the knowledge of randomness thereby confirm [7]. We show evidence that multi-task learning has huge advantages over single-task learning methods at a minimal cost, especially in the case of large traces.

Acknowledgements. Thomas Marquet has been support by the KWF under grant number KWF-3520-31870-45842. Elisabeth Oswald has been supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725042).

References

1. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 45–68. Springer International Publishing, Cham (2017)
2. Caruana, R.: Multitask learning. In: Thrun, S., Pratt, L.Y. (eds.) Learning to Learn, pp. 95–133. Springer (1998). https://doi.org/10.1007/978-1-4615-5529-2_5, https://doi.org/10.1007/978-1-4615-5529-2_5
3. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. pp. 262–280. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
4. Ma, J., Zhao, Z., Yi, X., Chen, J., Hong, L., Chi, E.H.: Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1930–1939. KDD '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3219819.3220007, https://doi.org/10.1145/3219819.3220007
5. Maghrebi, H.: Deep learning based side-channel attack: a new profiling methodology based on multi-label classification. Cryptology ePrint Archive, Report 2020/436 (2020), https://eprint.iacr.org/2020/436
6. Marquet, T., Oswald, E.: Exploring multi-task learning in the context of two masked AES implementations. IACR Cryptol. ePrint Arch. p. 6 (2023), https://eprint.iacr.org/2023/006
7. Masure, L., Cristiani, V., Lecomte, M., Standaert, F.X.: Don't learn what you already know: Scheme-aware modeling for profiling side-channel analysis against masking. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(1), 32–59 (Nov 2022). https://doi.org/10.46586/tches.v2023.i1.32-59, https://tches.iacr.org/index.php/TCHES/article/view/9946
8. Masure, L., Strullu, R.: Side channel analysis against the ANSSI's protected AES implementation on ARM. Cryptology ePrint Archive, Report 2021/592 (2021), https://eprint.iacr.org/2021/592
9. Paredes, B.R., Argyriou, A., Berthouze, N., Pontil, M.: Exploiting unrelated tasks in multi-task learning. In: Lawrence, N.D., Girolami, M. (eds.) Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 22, pp. 951–959. PMLR, La Palma, Canary Islands (21–23 Apr 2012), https://proceedings.mlr.press/v22/romera12.html
10. Perin, G., Wu, L., Picek, S.: Exploring feature selection scenarios for deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 828–861 (Aug 2022). https://doi.org/10.46586/tches.v2022.i4.828-861, https://tches.iacr.org/index.php/TCHES/article/view/9842
11. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (Nov 2018). https://doi.org/10.13154/tches.v2019.i1.209-237, https://tches.iacr.org/index.php/TCHES/article/view/7339

12. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: Sok: Deep learning-based physical side-channel analysis. ACM Comput. Surv. **55**(11) (feb 2023). https://doi.org/10.1145/3569577, https://doi.org/10.1145/3569577
13. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. Cryptology ePrint Archive, Paper 2018/053 (2018). https://doi.org/10.1007/s13389-019-00220-8, https://eprint.iacr.org/2018/053, https://eprint.iacr.org/2018/053
14. Ruder, S.: An overview of multi-task learning in deep neural networks. CoRR **abs/1706.05098** (2017)
15. Zheng, Y., Fan, J., Zhang, J., Gao, X.: Exploiting related and unrelated tasks for hierarchical metric learning and image classification. IEEE Transactions on Image Processing **29**, 883–896 (2020). https://doi.org/10.1109/TIP.2019.2938321