Exploring Multi-Task Learning in the Context of Two Masked AES Implementations

Thomas Marquet, Elisabeth Oswald

Universität Klagenfurt, author.name@aau.at

Abstract. This paper investigates different ways of applying multi-task learning in the context of two masked AES implementations (via the ASCADv1 and ASCADv2 databases). We propose novel ideas: jointly using multiple single-task models (aka multi-target learning), custom layers (enabling the use of multi-task learning without the need for information about randomness), and hierarchical multi-task models (owing to the idea of encoding the hierarchy flow directly into a multi-task learning model). Our work provides comparisons with existing approaches to deep learning and delivers a first attack using multi-task models without randomness during training, and a new best attack for the ASCADv2 dataset.

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

1 Introduction

Deep learning techniques have fast become an alternative to the use of classical statistics in the context of profiled side channel attacks, because of their unrivalled ability to efficiently utilise information across many trace points. The approach taken by many deep learning architectures still somewhat depends on the thinking found in traditional statistics based attacks: as single intermediate target is learned at a time (thus a single learning task is performed).

Recent publications have begun to move beyond this single-task learning paradigm towards a multi-task learning approach: Mahgrebi [Mag20] explores a deep learning architecture to learn two intermediate values (bit-wise) on an AES implementation simultaneously; Masure and Strullu [MS21] revisit Mahgrebi's idea and learn many intermediate values simultaneously. They set a new record for a "non-dissecting" approach for the ASCADv2 dataset and successfully recover the key bytes with 60 traces when assuming knowledge of the masks during profiling. Their paper concludes by reflecting on the potential power of multi-task learning: "A further study of the advantages and drawbacks of such paradigm is yet to be done. Still, this could lead to help the SCA practitioner towards new milestones against protected implementations." (p. 21, [MS21]).

1.1 Summary of Contributions and Outline

We focus on the application of multi-task learning in the context of the masked AES-128 implementations that are the basis of the ASCADv1 and ASCADv2 databases. After providing some notation and background in Sect. 2, we focus on novel ideas for multi-target and multi-task learning in Sect. 3, we explain new multi-task learning architectures for ASCADv1 in Sect. **??** and ASCADv2 in Sect. 4.3, whereby our innovations are as follows:

1.1.1 Contributions in a white box scenario

- we propose the combination of multiple single-task networks via the multi-target attack strategy,
- we suggest hierarchical multi-task learning as means to encode relationships between the multiple learning tasks,
- we provide a comparison of hierarchical multi-task, multi-task and single-task architectures that suggests the importance of working with "related" learning tasks (rather than unrelated learning tasks), as well as the benefit of including hierarchies,
- we give a new best attack for the ASCADv2 dataset and propose a better target than previously considered.

1.1.2 Contributions in a grey box scenario

- we propose to leverage multi-task learning to tackle the plateau effect
- we suggest multiple architectures that leverages assumptions on the masking scheme in order to ease the learning
- we provide a comparison of those architectures against each other and the previously trained white box architectures
- we provide the first real one trace attack in a fully uninformed scenario (i.e. no PoI, no labels) on the ascadv1.

1.2 Relevant related works

[MS21] introduce the ASCADv2 database, and provide a first characterisation of the included traces. We base some of our network architecture for the model design for the ASCADv2 database on their work.

[HWW21] explain that it can be beneficial to use the data from the processing of the AES state bytes to train a single model representing an intermediate value. This is possible in the case of many software implementations, because each state byte undergoes the same operations (the same sequence of Assembly instructions) which means that their leakage is very similar.

[PWP22] investigate the impact that the selection of points of interest has when training deep networks. They observe that working with raw traces is sometimes possible (i.e. no points of interest are selected), which leads to a black box attack scenario, where an adversary needs no information about randomness during the training. They provide the best results for the ASCADv1 database: they achieve key recovery with just a single trace in many scenarios.

[VGS14] and [MOW14] consider how to combine the leakage from multiple intermediate values efficiently. The former introduce the idea of using belief propagation, and the latter combine probabilities using Bayes theorem.

[MCLS22] This paper consider the possibility of using a scheme aware training by training two models and propagating a loss on the predicted combined probabilities. Such training, relieves the network by giving him a better understanding of what he should learn. However the authors do not provide any attack results and only interest themselves in the P.I. metric.

2 Preliminaries

We consider side channel attacks that operate in two stages: a leakage identification stage where (if necessary) points of interest are selected and deep nets are trained, and a leakage exploitation stage, where the trained nets are used as classifiers in the context of differential side channel attacks.

We stick to as simple notation as possible and stay with the variable naming conventions of the ASCAD databases: upper case letters denote sets (which we overload and simultaneously use as random variables), and lower case letters denote realisations of the random variables (and equivalently elements of a set). All variable/set names taken (with no renaming) from the original papers (implementations/data sets), such that "matchign up" of our work with these original implementations is straightforward. The index i refers to the ith state byte, and we generally drop any indexing referring to points within a trace from our notation. Occasionally we require to refer to the j-th trace, which we put as an index (alongside the index indicating the state byte) to a variable.

We also use a consistent colour code for figures that describe custom layers/specific deep learning architecturs: importantly orange indicates output layers (they also correspond to the labels that are required for training).

2.1 Profiling based on Deep Learning

For the purpose of building a classifier for newly observed traces during the exploitation phase, a deep learning approach uses one (or more) trained models, which output values that can be understood as likelihood scores. In the context of our work, we are intested in recovering information about key values. Thus, our networks are configured to return per-trace log-likelihood scores S_i for 8-bit chunks of an AES secret key. To derive the log-likelihood score for the ith key chunk given an attack set of N_a traces, we just compute the sum $d[k_i] = \sum_{j=1}^{N_a} S_{i,j}$.

To measure the effectiveness of a deep learning architecture we consider two quantities of interest: the accuracy on the whole attack dataset and the ability of a model to reduce the average key rank.

2.1.1 Training Methodology

We use the same methodology across all datasets. To enable meaningful comparisons, we use the same overall architecture for single models and multi task models, with the same learning rate and optimizer. The only difference between the models is the number of fully connected branches and how the branches are connected. We design one branch per intermediate value, whereby an intermediate value may also refer to a mask value.

As per good practice, we divide the available data into training data, validiation data and attack (=test) data. All training happens on the training data set. We validate a learned model on a validation set of size N_v. During this validation phase we monitor the validation accuracy. Our best training model is selected based on the best validiation accuracy, and we use a tensorflow callback to retrieve this model. We then test the best training model by using it in an attacks with N_a attack traces: all accuracies that we report later on are these final attack (=test) accuracies.

2.2 Computing resources

We're using a single GPU Nvidia PNY A30 with 24GB of dedicated memory. In addition to the GPU, we're using 4 cores of an AMD EPYC at 2.6GHz with 128 GB of RAM. All that is running on an Ubuntu 22.04.1 kernel, with tensorflow 2.10.1.

Table 1: Summary of training, validation and test/attack dataset for ASCADv1-r

	N_{traces}	*start*	*stop*	key
Training	$50k$	0	75000	random
Validation	$10k$	180000	195000	random
Attack	$10k$	0	30000	fixed

2.3 Data Sets and Corresponding Notation

Our work is based on the ASCAD datasets, which are both based on masked AES implementations. We assume familiarity with low order masking, as well as typical software implementations of low order masking on standard micro-controllers, as we keep the following text as short as possible.

2.3.1 ASCADv1-r

The original ASCAD database (v1) features one data set of a masked AES implementation (on a simple 8-bit microcontroller) with varying keys, which we utilise in our work. The database is generous, each side channel trace offers many data points for inclusion in training. To fit with the constraints of our computing resources we extract trace information as summarised in Table 1.

The extracted datasets contain the information that relates to the masked computation of the AES SubBytes operation. The masking scheme is a simple two-share scheme, which precomputes a masked AES SubBytes Table $SubBytes^*$ prior to encryption. The masked SubBytes table is defined as $SubBytes^*[x] = SubBytes[x \oplus r_{in}] \oplus r_{out}$. During the computation of a masked encryption round, all state bytes t_i (i refers to the state byte index) are masked by a state mask r_i. Prior to the masked SubBytes step, the state bytes are are remasked, so that the input to $SubBytes^*$ is masked by r_{in}, and because of the definition of $SubBytes^*$, the corresponding output is masked by r_{out}. The SubBytes output is then again remasked so that it is protected by the state mask r_i. The accompanying write up for the data base already performs an analysis to highlight the most leaky intermediate variables, which are the masked input and output of the SubBytes operation ($t_i \oplus r_{in}$, $s_i \oplus r_i$) as well as the two involved masks r_i and r_in. Whilst the output mask r_{out} and the masked intermediate $s_i \oplus r_{out}$ also leak, their leakage is weak and hence typically ignored.

There have been a number of papers that reported, for a variety of network architectures and approaches, results for this database. Our approach is to work with the raw traces (thus no points of interest selection takes place). With this setting in mind, the best previous works are [BCS21] and [PWP21], who reach single trace success — although the latter does not achieve this for all key bytes.

2.3.2 ASCADv2

The ASCADv2 dataset contains traces from a masked and shuffled AES implementation (on a more complex 32-bit architecture). The full dataset contains 800k traces with random keys and inputs. Each trace has 1 million sample points: therefore we extract only a subset of the available points for training/attack purposes. The split of the available data into training, validation and attack (=test) data sets is summarised in Tab. 2.

The masking scheme is slightly more complex. It uses both a non-zero multiplicative mask β, as well as a Boolean mask α, i.e. each intermediate value x is represented by three shares: $(x \cdot \beta \oplus \alpha, \beta, \alpha)$ (the multiplication must be understood over the appropriate finite field). The SubBytes operation is based again on a pre-computed table. Shuffling happens throughout the encryption rounds: a permutation over 16 elements is used for all

Table 2: Summary of training, validation and test/attack dataset from ASCADv2

	N_{traces}	*start*	*stop*	key
Training	250k	0	250000	random
Validation	50k	250000	300000	random
Attack	200k	300000	500000	random

round operations bar MixColumns, in which only the column elements are permuted. The permutation affects the index of the state bytes.

The specific notation for the intermediate values is akin to the notation in ASCADv1-r and works as follows. The variable t_i denotes the i-th state byte prior to the SubBytes operation, s_i is the result of SubBytes. Key bytes are denoted by k_i. The multiplicative mask is called r_m (it is the same for all state bytes), and the additive masks are called r_{in} (before SubBytes), r_{out} (after SubBytes), and r_i (everywhere else).

3 Multi-Target and (Hierarchical) Multi-Task Deep Learning

Multi-target attack strategies predate the use of deep learning in the side channel community. Works such as [MOW14] and [VGS14] explored the possibilities of utilising the leakage from multiple intermediate values. These attack vectors do not always require learned models, but can be combined with deep learning: the idea then would be to learn multiple independent models representing the different intermediate values, and combining them using different techniques. In particular it is possible to plug machine learning models into belief propagation, see [GBO19].

Multi-task learning refers to the concept of jointly learning multiple tasks simultaneously [Car98] [Rud17]. The intuition is that this enables better use of available training data and therefore it acts as a type of data augmentation, which aids generalisation. The second intuition is that tasks that share features can benefit from each other: perhaps some features are easier to identify in one task than in others and thus this task produces "hints" for the other tasks. However it is also possible that hints are not helpful, and therefore "poor" hints might prevent a network from learning.

3.1 Multi-Target Learning

The intuition behind utilising the leakage from multiple intermediate values is that they enable to gather information more efficiently, thereby reducing the number of traces during an attack. In the context of deep learning, one would aim to train single task models for, e.g. the SubBytes input and SubBytes output. Then rather than taking the "better" model, the idea of multi-target attacks would be to combine the information from both learning tasks. The combination is straightforward, if we assume that the learning tasks are sufficiently independent: suppose we gather information from task l_1 and task l_2, and they both relate to the same key k: then $\Pr[K = k|l_1, l_2] = \Pr[K = k|l_1] \times \Pr[K = k|l_2]$.

Thus in multi-target learning, multiple single tasks are completed individually, and then eventually combined. This is in contrast to the next technique, multi-task learning.

3.2 Task and branches

Multi task learning leverages shared layers when training multiple tasks at the same time. As their name indicate it, shared layers are common to all the tasks and therefore the network must take an independant path for each task after those layers. That's where the idea of branches appears. Each branch is setup through combination of losses and labels,

to learn a specific part of the underlying leakage based on known information about the masking scheme.

We define a task T_x as the output of the network corresponding to the variable x. During the training process of a network, labels of x are provided to the corresponding output layer which is in our case a softmax layer, but could be any kind of layer that fits the output representation chosen. We note the activated task T_x output as $\alpha_x = softmax(U_x)$.

A multi task model has n_b branches, n_t tasks. In a white box scenario, we have $n_b = n_t$. However, in a scheme-aware scenario, we cannot label intermediates that possess a part of randomness. Therefore branches are combined with another to represent together $(x \oplus m, m)$, with the label representing x, meaning that there should be less tasks than branches.

We define a branch B the following way :

- The input of a branch is the output of the convolution block, which is shared with all branches.
- A branch possess only fully connected layers, regularization layers and activations functions.
- The output of a branch is an unactivated fully connected layer with units fitting the chosen output representation. As we chose the identity leakage model, we have 256 units. We note it U_x.

3.3 Custom layers used to regroup branches

In order to fit our designs needs, we define a number of custom layers that we define in the following section.

3.3.1 Xor and inverse multGF256

"GroupRecombine" as been defined in [MCLS22], as a custom layer performing conditionnal probabilities between the softmax layers of two models trained during the same process. Our iteration of this layer performs the following computation given two vectors x and y of size 256 :

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

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.

3.3.2 Inverse Sbox layer

The inverse sbox layer on itself assume the knowledge of the sboxes used in the implementation of the AES. However this is something that is commonly assumed in profiling attacks. The layer is defined according to the following principle with S_x and S_y respectively the input and the output of the layer:

$$f_{inv}(x)[i] = x[Sbox(i)] \quad \forall\ i\ \in\ [0,255]$$

3.4 Multi-Task Learning

In the deep learning community, Mahgrebi [Mag20] was the first to pick up on the idea of multi-task learning. An improved design by Masur and Strullu [MS21] achieves impressive results for the ASCADv2 database. The core idea behind the existing architectures in these two previous works is that each intermediate value is learned by an independent branch of the deep net, and that all branches are connected to several shared layers deeling with the higher level features. This is the canonical design of multi-task networks, as summarised in [Rud17].

We show this principle visualised for one intermediate value that is represented via two shares in Fig. 1b. There are two branches. The upper branch corresponds to learning the mask value, and the lower branch corresponds to learning the masked intermediate value. Both branches are connected to several shares layers. The network then outputs two classification outputs, which can be understood to represent the probabilities $\Pr[x \oplus m]$ and $\Pr[m]$, from which we can easily recover the probability $\Pr[x] = \sum_m \Pr[x \oplus m] \times \Pr[m]$ (because we may assume that masks are chosen independent from intermediate values).

3.5 Hierarchical Multi-Task Learning

In the classical approach to multi-task learning (and also multi-target learning 1a), all branches are weighted evenly when computing the distribution of the desired, unmasked intermediate value. Recall that the hope is that both branches share "something" such that learning some features jointly acts as data augmentation technique as well that the "easier to learn" branch provides hints for the other branch. But equally, it could be that one branch remains a much better classifier than the other branch. We define a set of tasks D_{ht}, that are encoded in the model from the results of tasks that are earlier in the model graph.

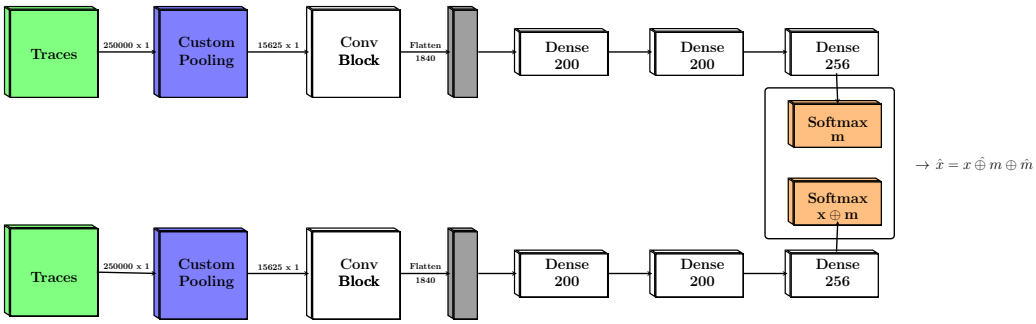
Therefore we propose to *encode* the hierarchical relationship(s) between branches as a learning task into the network. We illustrate this directly on the example of the simple two branch multi-task network that we showed in Fig. 1b; leading to the network architecture depicted in Fig. 1c. The network architecure contains a further layer, which aims at learning how to weigh the contributions of the branches and has $D_{ht} = [x]$. The design is slightly sophisticated because it is not possible to directly utilise the network outputs from the Softmax layer. The Softmax function is an exponential and therefore either amplifies or dampens characteristics of the distribution that is the result of the dense layers in each branch. Directly using the outputs of the dense layer is also not advisable. Consequently, we design a "Xor" layer which produces the distribution of x (given the distributions of $x+m$ and m) (twice, once for each combination of dense layer output and softmax output). The two distributions representing x are then summed, whereby this is a weighted sum, and the network learns the weights. This, after a further Softmax layer then leads to the final distribution for x.

Whilst we illustrated this approach for a network with two branches, it naturally generalises to networks with an arbitrary number of branches.

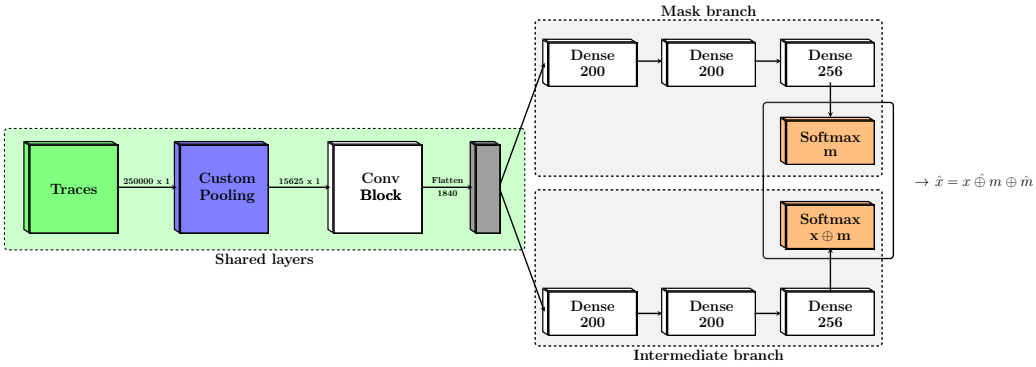
3.6 Scheme-aware Multi-Task Learning

The most challenging setting for learning is when during training only key knowledge and access is assumed. In the context of masked implementations, we would then assume that —because of a lack of access to internal randomness— the training data cannot be labelled with masks or masked values, but only the (unmasked) intermediate values. Also because of that, point of interest selection isn't possible.

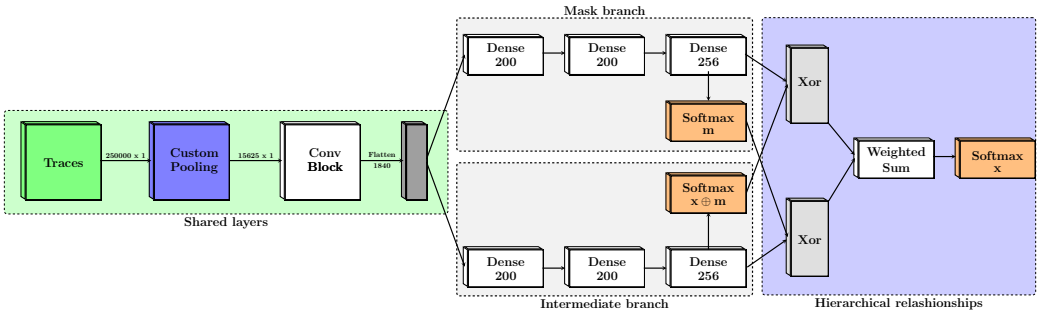
However, in the case where some information about the scheme is known, or simply because most schemes possess similarities in the relationships between masks and inter-



(a) Single task models



(b) Classical multi-task learning model



(c) Hierarchical multi-task learning model

Figure 1: Two approaches for multi-task learning in a white box scenario

mediate values, it is possible to design "scheme-aware" networks that because of their structure relaxes the learning process. This idea has been introduced in [MCLS22].

Given that the application of multi-task learning to masked implementations is based on designing branches that learn masks and masked values, it is non-trivial to come up with a way to apply multi-task learning when masks are unknown.

4 Multi task learning in a white box scenario

4.1 Assumptions, contributions and state of the art

"White box" scenario assumes that for each trace during profiling, we know the value of the mask. Even though the knowledge of the mask isn't assumed during the attack, it does decrease the difficulty of the profiling and therefore, the attack.

Ascadv1-r. This dataset is used in our case, to explore the possibilities of multi task learning in a relatively simple scenario. We go through multiple combinations of intermediates in order to understand in which case multi task learning is useful. It is also useful to understand the best performances possible with the model architecture chosen, which will be then used to attack in a scheme-aware setting. The state of the art in this setting, is one trace.

Ascadv2. In a more complex scenario, we're proposing to apply multi task learning to ascadv2. Exploration is a lot less possible on this dataset, there is only a few leakages usable. Instead of exploring we're offering an example of what can multi task learning can do to make it easier for an attacker, or evaluator. We're setting a new state of the art, recovering the key in around 24 traces, with a single model for all key bytes. The previous state of the art was set by [MS21] with 60 traces in a similar setting.

4.2 Ascadv1-r

The colored numbers in the tables of this section are the difference between the model in question and the related combination of individually trained models.

4.2.1 Core architecture

We build different sets of models for the ASCADv1-r database. One set of models represents the idea of building individual models for intermediate values. In order to ensure a like-for-like comparison with multi-task learning, these individual models are simply the separate branches of the multi-task models. All models share the design of the pooling, convolution and dense layers, which is as follows:

- **Weighted Pooling** Inspired by [PWP21] we're using custom layers to perform a weighted average pooling on the raw traces in order to reduce the size of the following network. We are pooling a total of 4 times to reduce the size from 250k samples to 15625 points. After each average pooling we perform a batch normalization and an alpha dropout.

- **Convolution block.** We use the convolutions from a CNN proposed by [PWP21]. It consist in only one convolution layer (kernel 34, strides 17 and filters 4) followed by an average pooling (pool size 2) and a batch normalization.

- **Dense Units.** Each prediction branch possess 2 dense layers of 200 units and one output layer followed by a softmax. The units in the dense layers are regularized using L2 norm and activated using a SeLu function.

4.2.2 Considering multiple intermediate values before SubBytes

As showed in [BCS21], the subbytes inputs can be rather useful in this dataset. In addition to the leakage with the input mask r_{in}, we noticed leakages with other shares. Therefore we explored the idea that perhaps combining information from several (shared) intermedative values that all related to the SubBytes input my help multi-task learning to succeed. In this scenario, we're again assuming knowledge of the masks. The (shared) intermediate values targeted are the following :

- $t_i \oplus r_{in}$
- $t_i \oplus r_i$
- $t_i \oplus r_i \oplus r_{in}$

It should be clear that the exlusive-or of these three intermediate values gives the unmasked SubBytes input t_i. Consequently, using the same core architecture as in the previous section, but designing five branches that learn the three intermediate values plus the masks, we now aim for a model that learns t_i.

Type	n_b	n_t	D_{ht}
m_{mt}	5	5	$\emptyset$
m_{hmt}	5	6	t_i

The same branches are used in the flat model and in the hierarchical model, however, we encode the relationships between tasks, linking them together to predict t_i. The design of the encoding is the following :

$$U_{t_i} = b + \sum_{\forall m \in [r_i, r_{in}, r_i \oplus r_{in}]} w_m * f_{\oplus}(U_{t_i \oplus m, \alpha_m}) + \sum_{\forall m \in [r_i, r_{in}]} w_{t_i \oplus m} * f_{\oplus}(\alpha_{t_i \oplus m}, U_m)$$

The tables Tab. 3a and 3b show the accuracies of the multi-task model and the hierarchical multi-task model relative to the accuracies obtained by training individual models in a multi-target setting to recover t_i. There are a number of similarities observable for both approaches: they both struggle to learn individual masks. However there is a striking difference in overall performance between the hierarchical multi-task model and both other models: encoding the hierarchical relationship between the intermediate values into the network, and enabling the network to learn the best weighted sum to recover t_i dramatically increases the accuracy for t_i.

As an aside it is interesting to observe that for bytes 8, 10, 11 and 14, the individual models do not learn anything about $t_i \oplus r_i$. Also for byte 4, the individual model is not learning anything about $t_i \oplus r_i \oplus r_{in}$.

Finally, we used the models in a key recovery attack. The attack is repeated over 1000 experiments, in which up to 10 traces are picked at random from the attack set. Using the hierarchical multi-task model enables to uniquely identify the *full* key using (on average) no more than two traces.

4.2.3 Considering multiple intermediate values before and after SubBytes

Encouraged by the improved results from combining three (shared) intermediate values, we now investigate the combination of intermediate values occuring before and after the SubBytes operation. Figure 3 shows the hierarchichal relationships levered. It learns the leakage of the input shares and output shares, whereby we encode the inverse SubBytes operation so that we effectively learn the unmasked input t_i from both the input and the output. We combine both "beliefs" and then add plaintext information to directly infer the key bytes k_i.

Table 3: Accuracies when recovering t_i, compared against the single task models

(a) Multi-task model

Multi task model for the recovery of t_i										
Byte	$t_i \oplus r_{in}$		$t_i \oplus r_i$		$t_i \oplus r_i \oplus r_{in}$		r_i		r_{in}		t_i	
$i = 3$	35.29	-6.21	22.91	16.61	15.80	-7.50	92.80	-4.20	72.61	-5.98	39.90	-5.75
$i = 4$	37.71	-4.59	14.27	1.38	11.33	11.00	70.66	-5.09	72.07	-6.52	35.32	-3.68
$i = 5$	36.20	-6.03	32.44	0.33	21.74	15.35	96.33	-1.81	72.52	-6.07	46.10	-5.11
$i = 6$	39.42	-3.62	23.59	7.34	21.03	15.55	97.87	-0.17	73.41	-5.18	43.32	0.03
$i = 7$	38.57	-3.37	18.24	11.87	21.31	12.43	96.27	-3.02	74.15	-4.44	40.58	0.77
$i = 8$	35.51	-4.82	15.77	15.38	15.28	10.29	82.20	-3.07	76.87	-1.72	39.51	6.50
$i = 9$	37.44	-13.4	17.83	2.36	16.01	10.18	70.83	-8.12	73.46	-5.13	36.85	-10.7
$i = 10$	41.02	-2.90	17.27	16.94	17.92	10.98	98.56	-0.12	75.89	-2.70	41.88	5.92
$i = 11$	45.51	-3.98	14.49	14.01	18.01	11.74	88.56	-2.77	74.29	-4.30	42.03	1.72
$i = 12$	44.24	-2.20	31.23	4.77	24.68	17.11	98.29	-0.99	79.61	1.02	52.69	1.23
$i = 13$	41.81	-1.50	19.57	11.07	16.66	10.00	69.40	-11.7	75.25	-3.34	40.08	-0.50
$i = 14$	35.52	-2.60	21.35	20.91	16.37	9.75	77.59	-11.8	74.91	-3.68	36.30	4.43
$i = 15$	48.73	6.36	44.54	-11.3	27.90	24.13	94.73	-3.63	77.65	-0.94	59.92	-8.82
$i = 16$	56.16	-3.47	24.69	4.24	26.97	19.46	77.91	-6.96	76.76	-1.83	51.25	-3.09

(b) Hierarchical multi-task model

Byte	$t_i \oplus r_{in}$		$t_i \oplus r_i$		$t_i \oplus r_i \oplus r_{in}$		r_i		r_{in}		t_i	
$i = 3$	43.49	1.99	29.47	23.17	23.32	0.02	94.89	-2.11	76.83	-1.76	80.25	34.6
$i = 4$	34.82	-7.48	13.44	0.55	13.08	12.75	74.24	-1.51	76.58	-2.01	67.37	28.37
$i = 5$	40.41	-1.82	32.66	0.55	25.17	18.78	98.54	0.40	77.89	-0.70	85.86	34.65
$i = 6$	38.87	-4.17	19.91	3.66	19.52	14.04	97.02	-1.02	75.56	-3.03	85.93	42.64
$i = 7$	40.62	-1.32	15.21	8.84	25.27	16.39	97.70	-1.59	77.89	-0.70	86.07	46.26
$i = 8$	38.66	-1.67	15.46	15.07	18.96	13.97	84.46	-0.81	80.31	1.72	81.37	48.36
$i = 9$	43.67	-7.13	17.17	1.70	20.50	14.67	73.27	-5.68	77.16	-1.43	77.76	30.17
$i = 10$	39.02	-4.90	15.11	14.78	21.03	14.09	96.41	-2.27	77.41	-1.18	83.00	47.04
$i = 11$	39.13	-10.4	13.06	12.58	16.05	9.78	81.94	-9.39	73.82	-4.77	77.72	37.41
$i = 12$	39.26	-7.18	26.05	-0.41	24.71	17.14	95.20	-4.08	76.62	-1.97	82.36	30.9
$i = 13$	37.23	-6.08	16.33	7.83	14.35	7.69	72.06	-9.05	75.98	-2.61	74.62	34.04
$i = 14$	41.53	3.41	16.13	15.69	17.62	11.00	77.75	-11.6	77.15	-1.44	80.73	48.86
$i = 15$	48.60	6.23	47.28	-8.58	27.18	23.41	95.39	-2.97	76.95	-1.64	88.67	19.93
$i = 16$	59.70	0.07	22.27	1.82	27.78	20.27	78.07	-6.80	77.08	-1.51	82.25	27.91

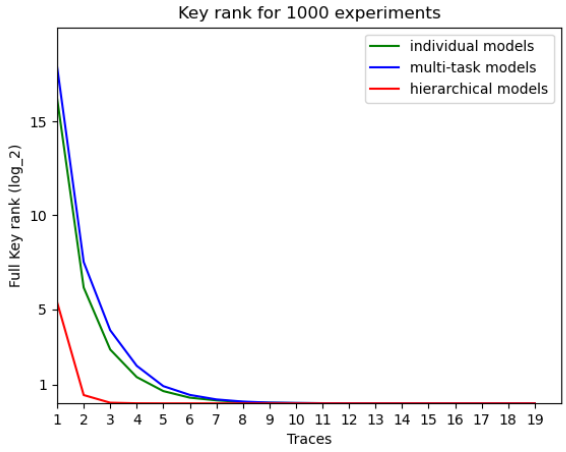


Figure 2: Full key recovery attacks for ASCADv1-r using multiple intermediates before SubBytes

Table 4: Accuracies when recovering k_i, compared against the single task models

(a) Multi-task model

Byte	$s_i \oplus r_i$	$t_i \oplus r_{in}$	r_i	r_{in}	s_i	t_i	k_i
$i = 3$	99.35 -0.27	44.92 3.42	91.43 -1.7	76.89 -5.57	90.84 -5.81	35.00 2.13	92.64 -4.86
$i = 4$	97.75 1.86	38.07 4.79	74.25 -2.82	75.77 -1.50	72.56 -0.28	28.71 -4.57	78.63 -0.74
$i = 5$	98.69 -0.45	43.99 1.76	97.73 -2.87	75.72 -0.41	96.45 -0.87	33.11 0.08	97.20 -0.82
$i = 6$	96.72 -2.64	34.05 -8.99	96.48 -1.54	77.05 -1.56	93.35 -4.07	26.67 -7.35	94.54 -3.48
$i = 7$	99.51 0.07	40.05 -1.89	95.49 -1.78	76.81 -3.80	95.06 -3.68	30.67 -1.15	95.89 -3.11
$i = 8$	96.69 -2.59	33.23 -7.1	84.49 -3.47	75.12 -0.78	81.95 -2.73	25.21 -6.60	84.59 -3.16
$i = 9$	97.03 -1.89	44.38 -6.42	83.78 -0.77	77.82 4.83	81.32 3.17	34.57 -5.22	86.64 2.53
$i = 10$	98.57 -1.02	40.12 -3.80	98.83 -0.73	77.86 0.15	97.45 -0.83	31.71 -2.76	98.13 -0.59
$i = 11$	98.51 0.10	39.19 -10.3	91.69 -1.41	77.18 0.36	90.36 0.41	30.63 -8.53	91.87 -0.77
$i = 12$	97.78 -2.00	43.97 -2.47	97.75 0.24	78.83 -1.53	95.66 -3.41	34.79 -2.00	96.75 -2.61
$i = 13$	98.91 -0.70	35.83 -7.48	68.01 -1.38	77.21 -13.1	67.39 -13.4	27.86 -6.62	73.24 -12.9
$i = 14$	98.16 -1.02	48.67 10.55	79.66 -2.79	75.8 -9.70	78.31 -10.4	37.07 6.65	83.75 -7.38
$i = 15$	97.35 -1.40	53.25 10.88	96.88 0.35	78.94 -1.48	94.33 -2.81	42.3 -0.07	95.76 -2.16
$i = 16$	98.60 -1.03	58.37 -1.26	77.39 1.01	79.6 -7.48	76.27 -8.32	46.23 -0.24	85.21 -4.90

(b) Hierarchical multi-task model

Byte	$s_i \oplus r_i$	$t_i \oplus r_{in}$	r_i	r_{in}	s_i	t_i	k_i
$i = 3$	98.23 -1.39	43.73 2.23	93.10 0.10	78.69 -3.9	86.54 -10.1	26.65 -6.22	97.75 0.25
$i = 4$	96.89 1.00	36.92 3.64	75.41 -0.34	74.05 -4.54	66.30 -6.54	21.59 -11.7	92.11 12.74
$i = 5$	96.40 -2.74	25.86 -16.4	90.11 -8.03	61.81 -16.8	79.61 -17.7	12.09 -20.9	94.09 -3.93
$i = 6$	98.55 -0.81	42.79 -0.25	96.93 -1.11	76.06 -2.53	93.15 -4.27	26.12 -7.9	99.09 1.07
$i = 7$	99.46 0.02	31.99 -9.95	97.82 -1.47	80.13 1.54	94.07 -4.67	19.97 -11.9	98.82 -0.18
$i = 8$	97.83 -1.45	35.01 -5.32	90.7 5.43	75.46 -3.13	84.79 0.11	20.89 -10.9	95.8 8.05
$i = 9$	97.02 -1.9	36.56 -14.2	77.7 -1.25	80.03 1.44	69.57 -8.58	22.99 -16.8	94.37 10.26
$i = 10$	98.60 -0.99	38.67 -5.25	98.60 -0.08	75.72 -2.87	95.24 -3.04	24.41 -10.1	99.08 0.36
$i = 11$	98.55 0.14	46.53 -2.96	89.58 -1.75	78.09 -0.50	83.49 -6.46	28.99 -10.2	97.1 4.46
$i = 12$	98.17 -1.61	42.60 -3.84	98.18 -1.10	72.55 -6.04	94.61 -4.46	23.93 -12.9	98.79 -0.57
$i = 13$	99.15 -0.46	39.00 -4.31	76.30 -4.81	78.35 -0.24	70.71 -10.1	23.59 -10.9	94.64 8.47
$i = 14$	97.52 -1.66	42.02 3.9	84.10 -5.26	76.34 -2.25	75.05 -13.7	25.04 -5.38	95.62 4.49
$i = 15$	97.12 -1.63	51.97 9.60	95.05 -3.31	75.43 -3.16	87.39 -9.75	31.3 -11.1	98.03 0.11
$i = 16$	98.92 -0.71	57.81 -1.82	80.14 -4.73	76.93 -1.66	74.56 -10.0	38.97 -7.50	97.88 7.77

Model	n_b	n_t	D_{ht}
m_{mt}	4	4	$\emptyset$
m_{hmt}	4	7	$t_i,\ s_i,\ k_i$

Again, the only difference between the two models, is the tasks encoded at the end. The encoding is the following :

$$\begin{cases} U_{t_i} = b + w_{r_{in}} * f_{\oplus}(U_{t_i \oplus r_{in}}, \alpha_{r_{in}}) w_{t_i \oplus r_{in}} * f_{\oplus}(\alpha_{t_i \oplus r_{in}}, U_{r_{in}}) & (3) \\ U_{s_i} = b + w_{r_i} * f_{\oplus}(U_{s_i \oplus r_i}, \alpha_{r_i}) + w_{s_i \oplus r_i} * f_{\oplus}(\alpha_{s_i \oplus r_i}, \alpha_{r_i}) & (4) \\ U_{k_i} = b + w_{t_i} * U_{t_i} + w_{s_i} * f_{inv}(U_{s_i}) & (5) \end{cases}$$

From Tables 4a and 4b we can see that both multi-task approach loses significantly in terms of accuracy, but hierarchical multi-task learning hugely benefits from the wealth of information which leads to an accuracy of over 95% on all key bytes in the attack data set. While impressive since we're using a single model to go from raw traces to key value, it has to be noted that the knowledge of the masks is assumed during profiling.

Like in the previous scenario, training all together does not appear to give any benefit on the intermediate values when combined using a multi-target strategy themselves. The real benefit appears to occur in the end leaf: the last loss function is more important than the other loss functions during the training. Looking at the accuracies on s_i and t_i, we can hypothesize that the network is trying to learn the intermediates in a complementary way, **each branch mitigating the errors of the other**. On the byte 9, the accuracies on s_i and t_i are much worse than the individual models. Yet the final accuracy on the key byte is 8% better.

As expected, when using these models in a full key recover attack, the outcomes are

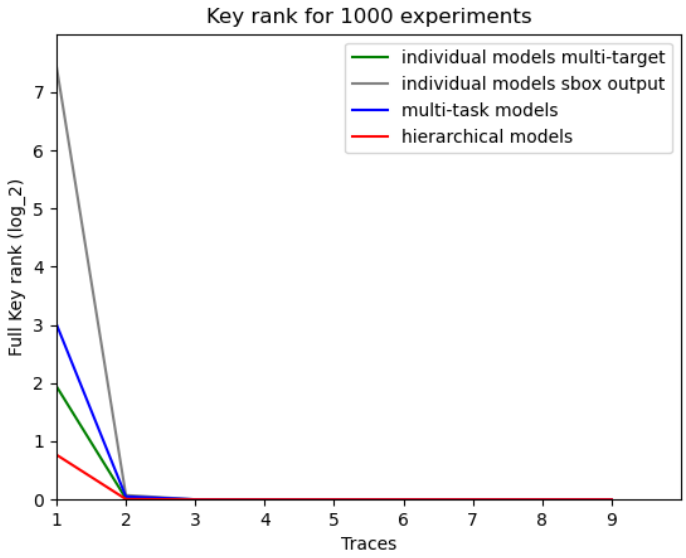


Figure 3: Full key recovery using SubBytes inputs and outputs (ASCADv1-r)

excellent, see Fig. 3. The hierarchical multi-task learning recovers the full key with a single raw trace robustly.

4.3 ASCADv2

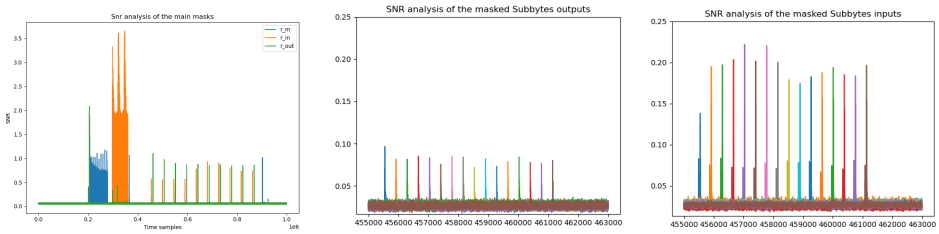
The ASCADv2 database is considerably newer and therefore much less analysed. In Masure and Strullu [MS21] the authors provide an excellent characterisation of the traces, and we took full advantage of this information in our work.

We follow the line of thought by Masure and Strullu, and first consider which trace points to include in our analysis—this is in stark contrast to what is possible for ASCADv1-r, where it is possible to work in a black box manner. Most of the work done on ascadv1 isn't replicable on this dataset simply because leakages that can be used on ascadv1 are not present or simply too weak on ascadv2. The best example of this is the state mask r_i which leaks up to 100% accuracy in the ascadv1. Here this mask doesn't leak better than 5%. Leveraging this mask and the potential combinaisons with $r_i \oplus r_{in}$ is therefore impossible here.

4.3.1 Points of interest selection

The point selection strategy (based on computing the SNR of intermediate values) detailed in Masure and Strullu [MS21], even though sufficient to perform successful attacks, omits inclusion, of what we found to be the most leaky part of the implementation: much leakage about the input mask r_{in} is not utilised and the subbytes inputs $r_m \otimes t_j \oplus r_{in}$. We therefore adapt our points of interest selection (also SNR based) as follows.

- r_m: We consider the entire interval where it leaks strongly (trace points 200000, 280000) but compress this down using a moving average with a window of 20 samples.
- r_{in}: We take the points where the snr is above a certain threshold, in total 1085 trace points
- r_{out}: We detect twelve peaks, and take 50 points around them for a total of 600 trace points



(a) r_m, r_{in}, r_{out} (b) $r_m \otimes s_j \oplus r_{out}$ (c) $r_m \otimes t_j \oplus r_{in}$

Figure 4: SNR analysis of the targets in our attack

- $r_m \otimes s_j \oplus r_{out}$: The SubBytes output is not leaking very strongly, and we take 100 traces points from within each state byte.
- $r_m \otimes t_j \oplus r_{in}$: The SubBytes input is leaking strongly, and we take 100 trace points from each state byte.
- $j = Perm(i)$: The permutation indices are leaking strongly, we take 1000 samples from across the entire encryption operation for each state byte.
- If some trace points are present twice because some interval overlaps, we remove them.

We note that the most leaky intermediates values are $s_j \oplus r_j$ and $t_j \oplus r_j$. However the mask r_j associated with these two values has a very low SNR and therefore we decided to exclude these two intermediates.

4.3.2 Data augmentation by training across state bytes

Whilst the shuffling implementation permutes the order of working through the state bytes, there is no whatsoever randomisation taking place within the processing of each state byte. This implies, that within each state byte, the order of operations is perfectly aligned across the 16 state bytes, and we take full advantage of this fact: we train a single model with a dataset constructed from samples corresponding to every bytes. This technique is taken from [HWW21] and allows a model to generalize to all bytes while decreasing the training time (or increasing the potential training set).

4.3.3 Core architecture)

In the case of ASCADv1-r we were able to use the individual branches of the multi-task architecture as a "reference" model for single task models. This strategy does not work with the ASCADv2 data: individual models based on individual branches do not learn any of the assigned tasks. Consequently, a like-for-like comparison as in the ASCADv1-r database is not possible, and architectures for the individual learning tasks were necessary. Each model is trained using data from all 16 key bytes.

4.3.4 Individual models

Inspired by the CNNs used by Masure in his paper, we decided to use a VGG16 like architecture. The structure is the following :

- **Convolution Block** : Convolution + Average pooling + BatchNormalization.

Table 5: Hyperparameters for Single Task Models

Parameter	Tested Values	Chosen Value
Number of Epochs	100	100
Batch Size	200, 500, 1000	500
Learning Rate	10^{-4}, 10^{-3}	10^{-3}
Optimiser	Adam	Adam

Table 6: Hyperparameters for Multi Task Models

Parameter	Tested Values	Chosen Value
Resnet Blocks	1,2,3,4,5	3
Number of Epochs	100	100
Batch Size	250, 500	250
Learning Rate	10^{-4}, 10^{-3}	10^{-3}
Optimiser	Adam	Adam

- **Numbers of blocks** : We experimented with 3,4 and 5 blocks. We settled for 4 as it showed the best performance.
- **Pooling** : Average Pooling of size 2.
- **Kernel size** : In VGG16 like architectures, the kernel size usually goes from 32 to 256 (doubling the size at every block). We decided after some tuning to start with a kernel size of 16.
- **Filters** : The convention is to use 11 filters, we investigated other choices, but found no better value, thus settled with 11.
- **Activation** : ReLu, as it is commonly used for VGG16.
- **Dense Layer block** : BatchNormalization + L1 and L2 regularization to reduce overfitting.
- **Number of units and dense block** : 2048 and 1, like Masure and Strullu, we did not experiment with other sizes.

The training hyperparameters are specified in the table 6. Note that the number of epochs do not matter as we overfit much before that but our strategy is to keep the best model on the validation set.

4.3.5 Multi task models

Again inspired by the work of Masure, we use ResNet based branches. However we are using our custom weighted pooling layer instead of the average pooling.

The hyperparameter tuning hasn't been done on the architecture of the network but rather on the regularizers present in most layers. All convolutions and dense layers are regularized used L1L2 Norm. Also various amount of dropout were used in between layers. This is where most of the tuning has been done.

4.3.6 Considering multiple intermediate values before and after SubBytes

Akin to how we approached the comparison for the ASCADv1-r dataset, we also performed experiments with ASCADv2. We built single-task models, and combined them using the multi-target strategy. We also built multi-task models and multi-target models. For all efforts we assumed full knowledge about the randomness during training.

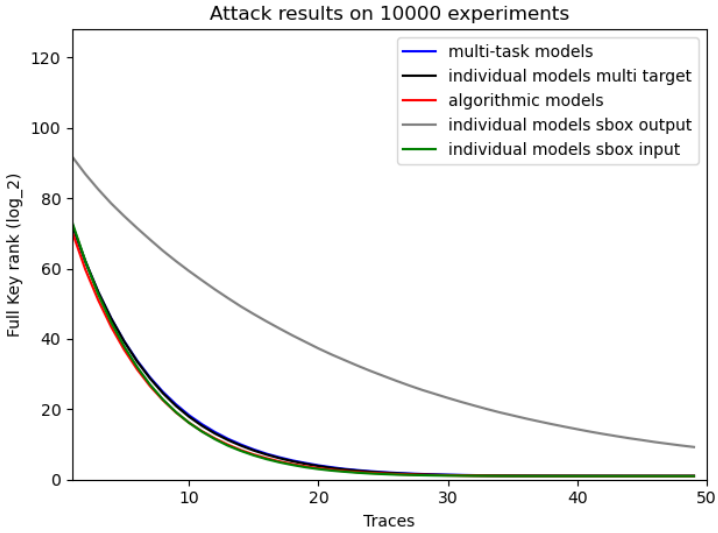


Figure 5: Full key recovery results for different models

Model	n_b	n_t	D_{ht}
m_{mt}	6	6	$\emptyset$
m_{hmt}	6	7	t_j

The 6 main branches correspond to the targets of the attack, developped in the section 4.3.1. The encoding of the hierarchical model m_{hmt} is the following :

$$\begin{cases} U_1 = f_{inv}(f_{\otimes}(f_{\oplus}(U_{s_j \oplus r_{out}}, \alpha_{r_{out}}), \alpha_{r_{\otimes}})) & (6) \\ U_2 = f_{inv}(f_{\otimes}(f_{\oplus}(\alpha_{s_j \oplus r_{out}}, U_{r_{out}}), \alpha_{r_{\otimes}})) & (7) \\ U_3 = f_{\otimes}(f_{\oplus}(U_{t_j \oplus r_{in}}, \alpha_{r_{in}}), \alpha_{r_{\otimes}}) & (8) \\ U_4 = f_{\otimes}(f_{\oplus}(\alpha_{t_j \oplus r_{in}}, U_{r_{in}}), \alpha_{r_{\otimes}}) & (9) \\ U_{t_j} = b + \sum_{k=1}^{4} w_k * U_k & (10) \end{cases}$$

The hierarchical multi task model uniquely reveals the correct secret key with 25 traces; this is as good as the best single-task model (using the SubBytes input) and the multi-task and multi-target models. These results are the best known results for the ASCADv2 dataset.

Even if the performances of multi task learning are worst than individual networks in 5, it has to be noted that the individual networks are fed with only the points that are related to the task to be learned, because the model couldn't learn anything otherwise.

4.3.7 Conclusion on whitebox multi-task learning

Throughout our two examples we show case a wide range of applications of multi task learning. The key takaways are the following :

- Training related tasks together, improve the training for all tasks.
- Training unrelated task together has negative to neutral impact. The increased difficulty by the propagation of multiple losses is to blame for that.

- Hierarchical multi task learning helps in both cases, as it improves when tasks are related and unrelated over the multi task learning scenario.
- Multi task learning increase the ability of a network to learn from large traces.
- Multi task learning provides a huge boost in learning and ease of hyperparameter tuning

5 Multi task learning in a scheme-aware scenario

The most challenging setting for learning is when during training only scheme knowledge is assumed. In the context of masked implementations, we would then assume that —because of a lack of access to internal randomness— the training data cannot be labelled with masks or masked values, but only the (unmasked) intermediate values. Again, due to the absence of randomness information, a point of interest selection might not be feasible.

Given that the application of multi-task learning to masked implementations is based on designing branches that learn masks and masked values, it is non-trivial to come up with a way to apply multi-task learning when masks are unknown. For this reason, we're looking for targets that we know share a mask in a scheme aware threat model.

Ascadv1-r. This dataset is very often used in a scenario where the masks are not known and many attacks have been showed successful. The state of the art succesfully recover the full key in 3 traces [PWP22]. The architecture of the branches of the network used in the whitebox scenario are the same as the one used here.

Ascadv2. This dataset has no state of the art in such scenario with no successful attacks so far.

5.1 Leveraging the subbytes inputs to unmask themselves

The idea with this model is to leverage the leakage of r_{in} and $t_i \oplus r_{in}$. Since all bytes of this intermediates are masked with the same randomness, we design an architecture with one branch that is connected to all the other with a xor-like layer. The idea is that while learning $(t_i \oplus r_{in}) \oplus r_{in}$, it might beneficial to learn at the same time $(t_{i+1} \oplus r_{in}) \oplus r_{in}$. This repeated 14 times for all the attackable bytes.

We train two unlabelled models, named m_0 and m_{cross}. The latter being an improved version of the first one.

Type	n_b	n_t	D_{ht}
m_0	15	14	$\emptyset$
m_{cross}	15	14	$\emptyset$

Both model have one branch for each possible masked intermediate, but only one for the mask. As said before, both architectures are going to combine the learning of the mask, from the masked intermediates. However, in the m_{cross} model, where adding an extra loss, that propagates the cross entropy between the predictions at the end of the mask branch, and the average mask prediction from the other branches. It is possible to recover this average mask prediction from the masked intermediate branches, simply by doing a xor operation between the labels Y_i and the predictions of the expected masked intermediates α_i. The losses $\mathcal{L}_m$ calculated for each models are the following:

$$\mathcal{L}_{m_0} = \sum_{i=3}^{16} crossentropy(\alpha_i, Y_i)$$

$$\mathcal{L}_{m_{cross}} = \mathcal{L}_{m_0} + crossentropy(\frac{1}{14}\sum_{i=3}^{16} f_{\oplus}(\alpha_{x_i \oplus m}, Y_i), \alpha_m)$$

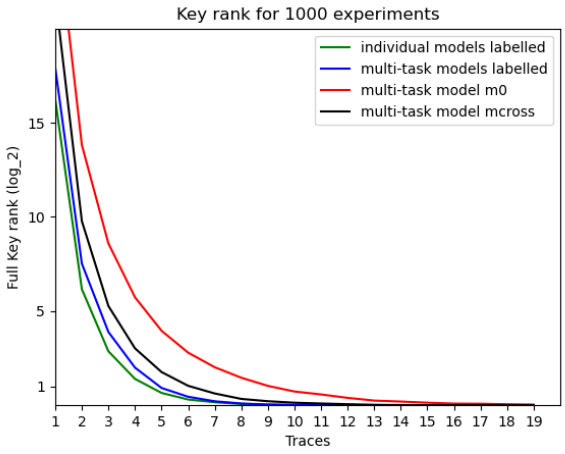


Figure 6: Comparison of unlabelled models against their labelled counterparts

This will perform a reindexing of the scores gathered on the $x_i \oplus m$ branch since Y_i correspond to the one hot encoded labels of t_i. Then with the summed scores, we add a cross entropy between the output of the mask branch and the information gathered on the rest. This technique will give a hint to the network.

We see clearly that the bottleneck of information reached by the labelled version isn't too far away from the best unlabelled models. We also see that linking the knowledge learned on the different branches, further improve the training. However, this is a special case where all bytes share a mask, which isn't always possible.

5.1.1 Is multi task learning helpful ?

To verify this, we trained 14 models in a two branches architecture. This scenario would be similar to the one in [MCLS22]. However, the models struggle to succesfully gather information on all bytes with our set of hyperparameters. This means that the hyperparameters would need to be tuned for each byte instead of together, increasing the tuning overhead. We can further hypothesize, that the amount of good set of hyperparameters with multi task learning should be greater than in single byte scenario, while the single byte scenario should be overall better (i.e. no free lunch theory). However, as the performances seen in 6, we're already reaching labelled performances, with our unlabelled networks.

6 Conclusion

Our results contribute to the research into using multi-task deep learning models in the context of side channel key recovery attacks. We make the suggestion of not just using multi-task models, but to exend them encoding hierarchical information about the relationships between the branches into the model. We observe that this can, sometimes significantly, improve the accuracy and with it the success rate in full key recovery. We also observe that using multiple related (shared) intermediate values is preferential over using just a single share and shared intermediate value. We propose new designs, which enable us also, for the first time, to design a multi-task learning model that does not require the knowledge of masks during training.

We believe more research is warranted for our new methods of multi-target and hierarchical multi-task training: the core of our idea is to encode relationships , and to propagate learned distributions from the branches of the multi-task model into a further

shared layer. This bears some resemblence to previous works on belief propagation. We also believe that our approach of defining custom layers is largely unexplored. The high customizability of libraries like TensorFlow makes this approach very practical, which enables more more tuned networks than what perhaps have been used by the side channel community so far.

References

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ASCAD with a single side-channel trace. Cryptology ePrint Archive, Report 2021/817, 2021. https://eprint.iacr.org/2021/817.

[Car98] Rich Caruana. Multitask learning. In Sebastian Thrun and Lorien Y. Pratt, editors, *Learning to Learn*, pages 95–133. Springer, 1998.

[GBO19] Joey Green, Tilo Burghardt, and Elisabeth Oswald. Not a free lunch but a cheap lunch: Experimental results for training many neural nets. *IACR Cryptol. ePrint Arch.*, page 1068, 2019.

[HWW21] Fanliang Hu, Huanyu Wang, and Junnian Wang. Cross-subkey deep-learning side-channel analysis. Cryptology ePrint Archive, Report 2021/1328, 2021. https://eprint.iacr.org/2021/1328.

[Mag20] Houssem Maghrebi. 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.

[MCLS22] Loïc Masure, Valence Cristiani, Maxime Lecomte, and François-Xavier Standaert. Don't learn what you already know: Grey-box modeling for profiling side-channel analysis against masking. Cryptology ePrint Archive, Report 2022/493, 2022. https://eprint.iacr.org/2022/493.

[MOW14] Luke Mather, Elisabeth Oswald, and Carolyn Whitnall. Multi-target DPA attacks: Pushing DPA beyond the limits of a desktop computer. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 243–261. Springer, 2014.

[MS21] Loïc Masure and Rémi Strullu. 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.

[PWP21] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2021/1414, 2021. https://eprint.iacr.org/2021/1414.

[PWP22] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022, Issue 4, 2022.

[Rud17] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.