Embedding belief propagation within a multi-task learning model : An example on Kyber's NTT

Thomas Marquet[1] and Elisabeth Oswald[1,2][0000−0001−7502−3184]

[1] Digital Age Research Center (D!ARC), Universität Klagenfurt, Austria
[2] University of Birmingham, UK
thomarquet@edu.aau.at
m.e.oswald@bham.ac.uk

Abstract. Domain-informed deep learning integrates specialized knowledge into deep neural networks through modifications to inputs, architectures, loss functions, or training processes. Such approach is usually employed to improve model performance in low/noisy data regime via domain-specific priors. Side-channel analysis is such a domain and even though straight deep learning has become a powerful tool for profiled side-channel, it still struggles on more complex datasets because of weak signals distributed across many datapoints. Initial attempts at integrating specific knowledge into models have been extremely successful to pushing boundaries of what can be achieved with deep learning. For example, integrating knowledge about the masking scheme in Masure et al [23] provides significant improvements in decreasing the profiling complexity. Marquet et Oswald [21,22] went further by leveraging redundant features across multiple tasks, such as shared randomness or common masking flow. In this work, we continue the discussion by using belief propagation on a larger graph to guide the learning. We introduce a multi-task learning model that explicitly integrates a factor graph reflecting the algebraic dependencies among intermediates in the computations of Kyber's inverse Number Theoretic Transform (iNTT). Such framework allow the model to learn a joint representation of the related tasks that is mutually beneficial. For the first time, we show that one can perform a belief propagation during training even when one does not have access to the internal randomness, on the masked shares, potentially improving greatly the performances of the attack.

Keywords: Side-channel analysis · Kyber · NTT · Masking· Multi-task learning · Deep learning · Belief propagation

1 Introduction

Side-channel attacks have been consistently a serious threat to the security of cryptographic implementations present in our embedded devices [14,13,25,20]. This class of attacks takes advantage of unintended information leakage from the physical implementations of the cryptographic primitives, such as power consumption or electromagnetic emissions, among many others. Even with the

demanding assumptions such attacks require, the newfound proximity between the attacker and the targets, along with the mass production of identical or near-identical hardware platforms, increased the relevance of attacks that leverage a clone of the target device, called profiled side-channel attacks. Naturally, Deep learning techniques became particularly relevant in this context and have transformed the landscape of side-channel analysis. Their main advantage over more traditional methods is the ability of neural networks to overcome a wide variety of countermeasures [19,3,27,35], but also to handle larger amounts of data, as deep learning methods can learn how to extract the signal from the traces without extensive manual pre-processing.

In a profiled attack setup, attackers correlate traces obtained from a profiling device, over which they have control, with sensitive intermediate values. At a minimum, this requires the attacker to have control over or the ability to observe the device's inputs. In many cases, attackers possess more information about the profiling device or target than these basic requirements. While this additional knowledge is often leveraged during the attack phase, it is rarely utilized during the profiling phase. One of the key advantages of deep learning models over classical machine learning techniques is their ability to incorporate such knowledge in very sophisticated ways. For example, one can integrate masking schemes directly into the network, as shown by Masure et al. [23] and Marquet et Oswald [21,22].

Post quantum cryptography is not different from previous standard algorithms with regard to the threat side-channel attacks pose, and the secure implementation of certain necessary computation has been proven to be particularly difficult. It is the case of the implementation of the Number Theoretic Transform (NTT) and its inverse (iNTT). One of the most powerful attack vectors is based on using belief propagation [30,31,33] on a graph representation of the NTT butterfly, to combine the information leakage from the many intermediate values that occur during the computation of the transforms.

Multi-task learning has been shown in Marquet et Oswald [21,22] to be a successful strategy to improve performances of deep learning templates, by encoding relationships between intermediates. The idea being that optimizing multiple objectives that share common sub-objectives helps to guide the learning towards a more successful region of the loss landscape. Another key result comes from Masure et al. [23], where the authors highlight that split the model in d-branches and recombining them by using the masking scheme's structure, allows for the recovery of the probability mass function (p.m.f.) at the end of each branches.

With those two key results in mind, we aim to **integrate a belief propagation graph inside each branch** and verify that such strategy is beneficial for the overall learning. To do so, we take advantage of the structure of the (inverse) NTT by encoding the relationships between the different nodes, using a factor

graph like in belief propagation techniques, but at the difference of previous works, **use the factor graph during the training**. We have a special look at Kyber's inverse Number Theoretic Transform (iNTT), as a friendly graph can be constructed. In section 3 and 4, we introduce the part of Kyber's algorithm we target, along with the state-of-the-art strategies targeting the iNTT, and the optimal belief propagation update rule. We introduce the concept of multi-task learning in section 6. Finally, we describe the modeling for our example, along with the experiments we perform to check that information is indeed propagated, showcasing for the first time the propagation of information on multiple belief propagation nodes without access to the random shares during profiling. We provide the code and dataset on the following links[3] and summarize our contributions in the following way:

- We provide empirical evidence that one can propagate information using belief propagation during the training of a neural network, **on the masked shares**, while not having access to the internal randomness of the profiling device.
- We show that including such relationships inside the deep learning model is beneficial for the overall training and performance.

2 Related Works

In this section, we regroup related works that might be useful to better understand our work or its contributions.

2.1 Belief propagation attacks on NTTs

Primas et al. [28] was the first work to utilize the power of belief propagation against Kyber's NTT. The authors use the decryption process of Kyber to demonstrate the potential of an attack based on a belief propagation graph that takes advantage of the linearity of the butterflies used in the NTT. They manage to recover the private key in a single-trace attack on a real device (ARM Cortex M4F), which implements the reference implementation of Kyber even in the presence of some noise.

A follow-up work, from Pessl and Primas [26], improves the butterfly nodes inside the belief propagation to fix some problems of redundancy. The authors of this paper focus on encryption, which makes the attack strategy different but just as powerful for three reasons. The first one is simply because the secret key is not involved in the operation. Therefore, the target of the attack is the

[3]
- Dataset
- Github

ephemeral keys that are supposed to be generated by the key exchange mechanism. The second reason is also linked to the previously formulated fact. The Fujisaki-Okamoto CCA-2 transform [7] necessitates re-encrypting the message. Therefore, the attack can also be mounted on the receiving end of the key exchange mechanism. Finally, the NTT used in the encryption involves inputs with a special distribution that can be utilized to facilitate the belief propagation, and therefore improve the success rate under much noisier setups.

Since those two foundation works, Hamburg et al. [8], proposed a further study of the impact of a novel chosen ciphertext strategy when targeting the decryption process and therefore the secret key. The authors utilize the ciphertexts to introduce a known amount of zero values during the belief propagation, which improves the performance and helps the attacker succeed in the presence of more noise. This strategy is then improved in a follow-up work to deal with shuffled implementation in [9].

Rodriguez et al. [32] take an interest in a masking countermeasure introduced in [29]. This countermeasure, called *local masking*, aims to increase the security of the NTT butterflies by adding a mask that can be understood as a multiplicative mask, after each end node. Rodriguez et al. [32] show in their work that one can adapt the belief propagation to effectively remove the impact of the multiplicative mask on the performances of the belief propagation as long as the amount of shares stays relatively low (< 8).

2.2 Belief propagation fused with Deep learning

There have been a few other strategies in the deep learning literature attempting to fuse deep learning models with a factor graph. For example, Kuck et al. [15] propose Belief Propagation Neural Networks (BPNNs), a class of message-passing neural networks where the model learns some sort of message scheduling. The main idea behind this is to replace fixed dampening strategies. In the same realm, Zhang et al. [36] propose Factor Graph Neural Networks (FGNNs), which differ from BPNNs because they try to learn the update rules instead of message scheduling. Another interesting approach is the one from Lucibello et al. [17]: they employ message-passing algorithms based on BP to train deep neural networks. Their framework formulates the network's weights, activations, and data as variables in a graphical model and iteratively updates them using belief propagation enhanced with reinforcement terms.

The trend showcased in the previous works is rather to improve belief propagation using some sort of neural network, which is tasked to learn a better-than-human-designed update rule. This is not what we aim to achieve in this work, as we try to use belief propagation to guide the learning of a deep learning model. Such an approach is unlikely to bring positive results anywhere other than in the side-channel domain, as the computation cost is heavy. However, because of

the countermeasures, learning is extremely difficult in our domain, which makes approaches such as this worth consideration.

3 Crystals Kyber and the Number Theoretic Transform

After being chosen as one of the next cryptographic standard by the National Institute of Standards and Technology (NIST), the public key cryptographic algorithm named CRYSTALS-Kyber [1] has received considerable attention from cryptographers and side-channel practitioners. While its selection as the new standard makes it a prime candidate for deployment, it is crucial to continue rigorous investigations of its different components in order to circumvent or highlight flaws in the implementations of the algorithm.

The NTT operation. The Number-Theoretic Transform (NTT) is a specialized version of the FFT that operates over finite fields $\mathbb{Z}_q[x]/(x^n+1)$, with q a prime number, operating on polynomials instead of complex numbers. The goal of the NTT is to project the elements of the ring $\mathbb{Z}_q[x]/(x^n+1)$ in a multiplication-friendly domain which allows them to be multiplied in a pointwise way.

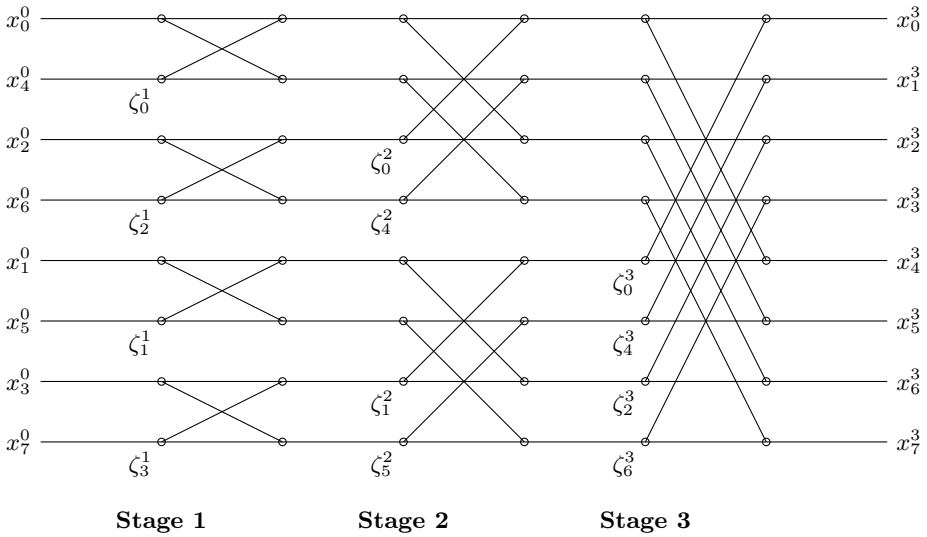


Fig. 1: Example of a 3-stage NTT Butterfly

A NTT butterfly. In the context of a finite field where n is a power of 2, such as Kyber, the NTT is implemented by successive simple modular equations 1 and 2 called *butterflies*, where each stage l manipulates a different pair of

coefficients k and $k+p_l$, as illustrated in Figure 1, and a twiddle factor ζ_k^l which is specific to the stage and the pair. This implementation is called "Cooley-Tukey" algorithm and was introduced by its eponymous authors in [5]. The efficiency of the operation comes from the possibility of breaking down the computation into many smaller operations. Their role is to follow a divide-and-conquer strategy in which each butterfly deals only with local information to compute the next stage.

$$\begin{cases} x_k^l &= x_k^{l-1} + x_{k+p_l}^{l-1} \pmod q & (1) \\ x_{k+p_l}^l &= \zeta_k^l (x_{k+p_l}^{l-1} - x_k^{l-1}) \pmod q & (2) \end{cases}$$

At the end of the scheme, after the operation from Equation 1 and 2 is repeated for all layers l. We note the obtained transform as $\tilde{x}$.

3.1 Masking (i)NTTs

To protect NTTs and especially in our context, inverse NTTs, one can think of using arithmetic masking [8,9], which is simply modular addition. This strategy is particularly well-suited for the modular nature of the operations. Let's assume a sensitive variable a with a two-share arithmetic scheme noted as $a_1 = a - m \ (mod\ q)$, and the mask $a_2 = m$. Due to the linearity of the NTT operations (same for the inverse NTT) :

$$NTT(a) = NTT(a_1) + NTT(a_2) \ (mod\ q)$$

Assuming we want to mask the secret s (or its NTT version $\tilde{s}$) with a randomly chosen mask m (or in NTT $\tilde{m}$), the flow of the inverse NTT operation is represented in the following Figure 2 :

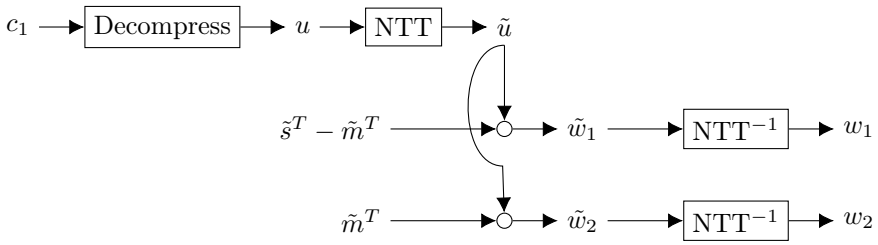


Fig. 2: Arithmetic masking flow around the inverse NTT operation

Summing w_1 and w_2 is equivalent to recovering w, the value that would have been computed using the secret key s directly.

4 Belief propagation against the Number Theoretic Transform

Belief propagation has been introduced for the first time in the side-channel community by Veyrat-Charvillon et al. [33]. The authors laid the foundations for what is now the most efficient multi-target attack in the literature, SASCA. The basic idea is to define a belief-propagation graph according to the intermediates that are leaking inside the power traces at our disposal. Then, the output scores are fed from a template or a deep learning model that is specialized in one intermediate. The belief propagation will create a sort of joint distribution over all intermediates, which will be, depending on the graph, helpful in recovering the key.

The optimal update rule. The computation of the NTTs using the Cooley-Tukey technique is unfortunately very sensitive to side-channel attacks. As you are indeed breaking down the computations, you are creating many intermediate states that are linked to secret information, either directly with the secret key in the case of inverse NTTs, or linked to the ephemeral key in the forward NTT. Previous works [28,26,8,9] showed that the NTT operation creates a belief propagation-friendly graph that allows for performing very powerful attacks.

One can define a few different strategies to model the nodes' update rules, but previous work agrees on an optimal strategy that maximizes the strength of belief propagation by ensuring the graph is tree-like [26,8,9]. To define this update rule, we reformulate the equations in 1 and 2 in the following :

$$\begin{cases} x_k^{l-1} &= (1-\zeta_k^l)^{-1}(-(1+\zeta_k^l)x_{k+p_l}^{l-1} + x_k^l + x_{k+p_l}^l) \pmod q & (3)\\ x_{k+p_l}^{l-1} &= (1+\zeta_k^l)^{-1}(-(1-\zeta_k^l)x_{k+p_l}^{l-1} + x_k^l + x_{k+p_l}^l) \pmod q & (4)\\ x_k^l &= (1-\zeta_k^l)x_k^{l-1} + (1+\zeta_k^l)x_{k+p_l}^{l-1} - x_{k+p_l}^l \pmod q & (5)\\ x_{k+p_l}^l &= (1-\zeta_k^l)x_k^{l-1} + (1+\zeta_k^l)x_{k+p_l}^{l-1} - x_k^l \pmod q & (6) \end{cases}$$

Using the equations 3, 4, 5, 6, we can define the messages sent to each part of the butterfly as a function of the three others. Since the belief propagation operates over score vectors, one needs to make modifications to the operations used in the equations. Let $S_t(x_i)$ the probability-like score vector representing the state of the intermediate x_i at iteration t, which can be represented as :

$$S_t(x_i) = \begin{bmatrix} Pr(x_i = 0) \\ Pr(x_i = 1) \\ \vdots \\ Pr(x_i = q) \end{bmatrix}$$

With the function f_b^l, the mapping function derived from the equations 3, 4, 5, 6, we have the following update rule for a single node $\mathbf{x}_m = \{x_k^{l-1}, x_{k+p_l}^{l-1}, x_k^l, x_{k+p_l}^l\}$:

$$S_t(x_i) = S_{t-1}(x_i) \cdot \sum_{\mathbf{x}_m \setminus x_i} f_b^l(\mathbf{x}_m) \cdot \prod_{x_n \in \mathbf{x}_m \setminus x_i} S_{t-1}(x_n) \tag{7}$$

Belief propagation requires more than Equation 7 to be fully described; however, since we focus on a single node due to computational complexity, we do not describe the others.

5 *d*-branch neural networks

A very successful trick in deep learning literature is to include **domain-specific** knowledge in the modeling of the neural network. The intuition behind such a method is that if you hardwire into the network some piece of knowledge that the network should have learned anyway, then you are considerably reducing the profiling complexity required to obtain a performant model.

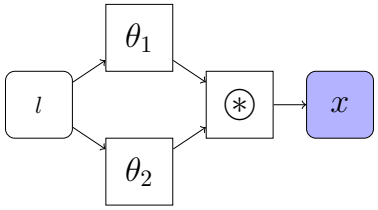


Fig. 3: example of a d-branch neural network, with $d = 2$

In our context, side-channel analysis against masked implementations, including domain-specific knowledge, means to include information one might have about the masking scheme, which is an acceptable assumption, as in some cases evaluators might have access to that knowledge, but not the internal randomness, and attackers might make assumptions about the masking scheme. One way this principle can materialize in our context is to split the network into d-branches, one for each share, and then using either parametric or non-parametric layers to recombine the branches. Such strategies have been very successfully utilized in our community [23,21,22]. We give a visualization of such modeling strategy in Figure 3, and we give an example of an equation that can describe the custom layer in Equation 8 with an arbitrary two-share ($d = 2$) masking scheme $*$, where $\hat{\theta_1}$, $\hat{\theta_2}$ and $\hat{x}$ are respectively the output of the branches θ_1, θ_2 and x from Figure 3, being all vectors of the same size n.

$$\hat{x}[k] = \sum_{j}^{n} \hat{\theta_1}[j] \times \hat{\theta_2}[k * j] \tag{8}$$

6 Multi-task learning

Multi-task learning in the context of deep learning is a model learning multiple related tasks. This can be achieved by providing multiple ground truths to the model. The key reason behind including multiple sources of knowledge is the hope that those sources are related and that learning them jointly helps the model to leverage shared representations. According to Caruana [4], the core idea is that information contained in the inductive bias of related tasks can be mutually beneficial when learned together, as opposed to learning each task independently (which we call *single-task* learning).

Formally, consider a set of n_τ supervised learning tasks $\{x_1, x_2, \ldots, x_{n_\tau}\}$, where each task x_i is associated with a dataset $\mathcal{D}_i = \{(l_j^{(i)}, y_j^{(i)})\}_{j=1}^{n_i}$ ($l_j^{(i)}$ can be the same). The model's goal is to learn the tasks x_1, ..., x_{n_t}. With θ, the set of all parameters within the model, composed of the shared parameters $\theta_\forall$, and each task-specific parameter θ_{x_i}. Each task-specific loss $\mathcal{L}_{x_i}$, such that $\theta = \{\theta_\forall, \{\theta_{x_i}\}_{\forall i}\}$, and the weight of the loss λ_i is used to represent the global multi-task objective :

$$\mathcal{L}(\theta) = \sum_{i=1}^{n_t} \lambda_i \mathcal{L}_{x_i}(\theta_\forall, \theta_{x_i}) \tag{9}$$

6.1 In the context of SCA

In the side-channel community, Maghrebi [18] was the first to pick up on the idea of multi-task learning. The paper presents a multi-task approach to enhance the efficiency of deep learning as distinguishers in the context of side-channel attacks. Traditional methods until then were only targeting small subsets of the cryptographic key during the training phase, and required a model for each subset. Maghrebi proposes a new profiling methodology leveraging multi-task learning, enabling the targeting of larger key chunks without incurring additional learning time overhead. The models deployed in this paper typically reuse the common idea of sharing layers at a higher level, as Caruana imagined. This methodology not only maintains attack efficiency comparable to traditional approaches but also allows simultaneous training on multiple intermediate operations. Experimental validation on simulated traces and publicly available datasets demonstrates the practical advantages of this approach, highlighting its potential for efficient and robust security evaluations in contexts requiring high attack potential. Similar ideas are developed later in Fukuda et al. [34] and Deng et al. [6].

Masure and Strullu [24] continue with the same ideas with a complete characterization of the ASCAD-v2 database. The paper revisits and extends prior

attacks using multi-task learning, as introduced by Maghrebi, to efficiently target several intermediate computations simultaneously. The key findings highlight the efficacy of deep learning-based templates in overcoming these countermeasures. The authors demonstrate that the shuffling countermeasure provides limited resistance against deep learning models despite their complexity. They also provide attacks defeating the multiplicative mask of the affine scheme. However, they did not succeed in defeating all countermeasures. Despite the very limited improvement over the models of Maghrebi, they underscore the potential of multi-task learning in the context of side-channel attacks.

Bursztein et al. [2] present SCANET, a multi-task architecture defeating multiple countermeasures against protected implementations of Elliptic Curve Cryptography (ECC). Using the most complex (in terms of parameters) multi-task model deployed in the literature and the full traces, they successfully recovered the targeted key. However, they still stay within a reasonable distance from the models of Caruana and, therefore, the models of the two other papers. They also complete their investigation with a study of the relationships between tasks while targeting a protected ECC implementation and by discussing the performance of their architecture on the ASCAD-v2 dataset with full knowledge of the countermeasures during profiling.

Marquet and Oswald [21,22] provide extensive experiments showcasing the superiority of multi-task learning over single-task learning in very specific scenarios. The authors show that in the context of masked AES implementations (using the public datasets ASCAD-r, ASCAD-v2 and CHESCTF-2023), when the attacker/evaluator does not have access to the internal randomness of the profiling device, multi-task models achieve more consistent performances across hyperparameters and initialization seeds. The abilities of multi-task models to overcome *the plateau effect* are far superior to single-task models. The reason provided by the authors to explain such success can be found in the increased data efficiency of multi-task models because of the implicit regularization created by the sharing of parameters.

Finally, Liao et al. [16] propose a Transformer-based neural network called Switch-T, designed specifically for cross-device side-channel attacks where the goal is to build one model that works on multiple devices or attack scenarios. The Switch-T architecture uses multi-task learning to handle data from different devices as different "tasks" and incorporates mechanisms (like task-specific parameters or EWC regularization) to retain performance across devices. By training the network on multiple devices' power traces (tasks) at once, the model can generalize and "switch" its inference to new devices with minimal performance loss. The authors demonstrate that multi-task learning enables more portable deep-learning side-channel attacks, achieving high key recovery rates on devices not seen during the initial training.

7 Experiments

Prior works targeting masking implementations (described in 2.1) assumed an attacker could observe or control the masked values m or $\tilde{m}$ during the profiling phase, used this to build templates for each share, and then, during the attack phase, run belief propagation independently on those shares. In some realistic settings, the attacker cannot access the internal randomness (the mask) to build the templates, so this approach fails. In that scenario, one might first train classical templates or deep-learning models to recover the unmasked intermediates and only then perform belief propagation. But this is entirely dependent on one's ability to build reliable templates in the first place.

In this section, we aim to push what can be achieved by multi-task learning models once you include domain knowledge by further leveraging the observation made in Masure et al. [23], that at the end of each branch, one can observe the true probability distribution. In Marquet et Oswald [21,22], multi-task learning was already using this concept to share masks across different intermediates. In this work, we aim to discuss the inclusion of a belief propagation graph **before the recombination of the shares**. The reasons why one might want to do so are the following:

- Defeating the "plateau" effect. As investigated in [21,22], including domain knowledge within a deep learning model, and especially within a multi-task learning model, significantly improves the ability of said network to capture information consistently across training runs.
- Increasing overall attack performance. Obviously, the attack is more powerful if the template quality is higher. By including belief propagation during training, we ensure the quality of the templates.

Therefore, we investigate the following question :

- Can multi-task models propagate information about the different shares within a complex relationship graph, such as Kyber's NTT, even when access to those shares is not assumed?

7.1 Dataset acquisition

The dataset acquired for the purpose of targeting Kyber is targeting the decryption algorithm, as we wish to recover the long-term secret key of the KEM procedure. To recover the secret key, we observe the inverse NTT operation as noted in 1.

The Kyber implementation we target is the pqm4 implementation [12] from Kannwischer et al., actually the masked version of the implementation, which is named mkm4. It has to be noted that a significant part of the mkm4 code base is taken out of the pqm4 repository and simply adapted to multiple shares, i.e.

Algorithm 1 Kyber.CPAPKE.Dec(sk, c)

Input: Secret key $sk \in \mathbb{B}_{12 \cdot k \cdot n/8}$
Input: Ciphertext $c \in \mathbb{B}_{du \cdot k \cdot n/8 + dv \cdot n/8}$
Output: Message $m \in \mathbb{B}_{32}$
$u := \text{Decompress}_q(\text{Decode}_{du}(c), du)$
$v := \text{Decompress}_q(\text{Decode}_{dv}(c + du \cdot k \cdot n/8), dv)$
$\hat{s} := \text{Decode}_{12}(sk)$
$m := \text{Encode}_1(\text{Compress}_q(v - \text{NTT}^{-1}(\hat{s}^T \circ \text{NTT}(u)), 1))$ ▷
$m := \text{Compress}_q(v - s^T \cdot u, 1)$
return m

the NTT part is simply repeated d times. We use this implementation because it allows compatibility with the chipwhisperer platform. Therefore, we perform our own acquisition using the ChipWhisperer Lite (CWLITE) and target the implementation on an STM32F303RCT7 microcontroller with an Arm Cortex M4. It is the classic target device for the chipwhisperer platform. The sampling rate for the acquisition is 105 MS/s. We capture the beginning of the inverse NTT operation for each share (up to the 5th layer) because the buffer size of the CWLITE is limited. We acquire 500k traces using random keys and random ciphertexts, but do not use all of them in our experiments. Finally, we acquire the attack dataset using a chosen ciphertext strategy, with a total of 10k traces, breaking down the acquisition into two for each share.

7.2 Modeling

Due to the high computational complexity involved (see Table 1), we restrict our design to only four intermediates, denoted x_k^l, $x_{k+p_l}^l$, x_k^{l-1}, and $x_{k+p_l}^{l-1}$. Here, k and $k + p_l$ are the target coefficient indices, and l refers to the butterfly layer selected for modeling. We illustrate the model in Figure 4 with $d = 2$ shares.

We begin by creating d-branches for each intermediate node in the graph, i.e. one branch per share. In the case of the intermediate x_k^l, we note the corresponding branches $\theta_{1_k}^l$ and $\theta_{2_k}^l$. The grey nodes represent non-parametric layers, whereas the blue nodes are the outputs. The layers f_b^l correspond to the operations required to perform the butterfly computations at layer l, as derived from Equation 7, but are different in the following ways :

- $S_t(x_i)$ is the logits of the dense layers at the end of the corresponding branches, for example $\theta_{1_k}^l$ when $x_i = x_{1_k}^l$. Therefore, the values within $s_t(x_i)$ evolve between $-\infty$ and $+\infty$. However, the logits are normalized by removing the maximum value, to make sure the values behave under the exponential.
- We make the assumption that the logits at the end of each branch follow logarithmic rules. Even though it is not a true logarithm, it remains a reasonable

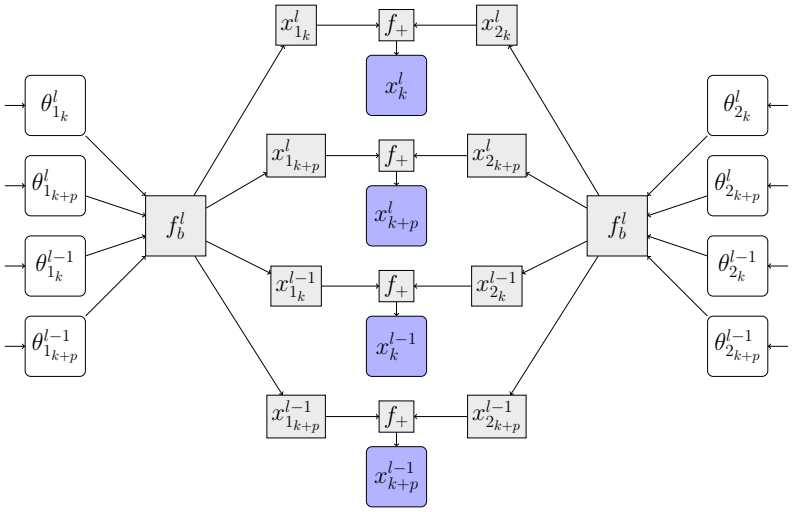


Fig. 4: Output connections of a multi-task model with d = 2, and one butterfly node.

assumption because of the softmax used for the application of the categorical cross-entropy loss, which can be easily empirically verified. Therefore, for numerical stability, one can express the expression in a "log-like" world :

$$S_t(x_i) = S_{t-1}(x_i) \;\; + \;\; \log\left[\sum_{\mathbf{x}_m \setminus x_i} \exp\Big(f^l_b(\mathbf{x}_m) \;\cdot \sum_{x_n \in \mathbf{x}_m \setminus x_i} S_{t-1}(x_n) \Big) \right]$$

The layer f_+ defines the combination of two shares using the arithmetic masking modulo q, as detailed in the abstract layer definition in Equation 8, where $* = +$ mod q. To effectively reduce the needs in computation power and efficiency, we note that such an operation is actually similar to a FFT convolution (from Masure et al [23]). Therefore, this layer is implemented as an FFT convolution.

Why should this work? We want to point out that the inverse NTT operation is linear and therefore creates a scenario where relationships between the different intermediates can be expressed, similar to the shared mask scenario in [21,22]. In the case of Kyber, the 128 masked coefficients of the input polynomial are used in the subsequent layers. To take example from Figure 4, $x^l_{1_k}$ depends on 4 sets of hyperparameters : ($\theta^l_{1_k}$, $\theta^l_{1_{k+p_l}}$, $\theta^{l-1}_{1_k}$ and $\theta^{l-1}_{1_{k+p_l}}$). Same is true for $x^l_{1_{k+p_l}}$, $x^{l-1}_{1_k}$ and $x^{l-1}_{1_{k+p_l}}$. Training the models using such links enforce constraints on the model, which will guide the learning and prevent the gradient from falling in valleys that do not satisfy all the tasks.

7.3 Validation Loss achieved with increasing profiling power

To observe if the idea is correct, we train the model from Figure 4 with different profiling sizes along with a baseline trained in a single-task manner, which can be seen as a subset of the multi-task model. We try to make the comparison as fair as possible by giving as input the same samples (all intermediates leak in a relatively close window) and keeping the same hyperparameters. In total, we train three models (two multi-task and the single-task baseline) described in the following way:

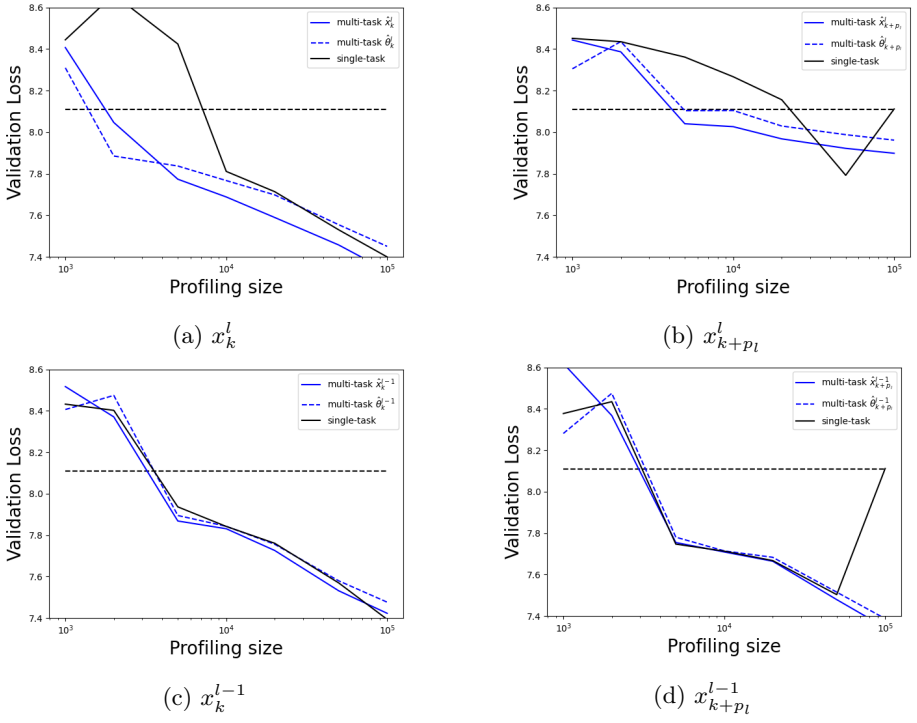
- **single-task**: Trained four different times to learn the different intermediates x_b^a. Corresponds to a model built using hyperparameters $\theta_{1_b}^a$ and $\theta_{2_b}^a$, connected using $\hat{\theta}_b^a = f_+(\hat{\theta_{1_b}^a}, \hat{\theta_{2_b}^a})$, where $\hat{\theta}$ is the prediction observed at the output of θ. The training loss is cross entropy using as labels the values of the intermediate x_b^a and the predictions $\hat{\theta_b^a}$. Represented in black in Figure 5.
- **multi-task** using an extra loss: Corresponds to the model from 4, but possesses two cross-entropy losses per intermediate x_b^a (using the same labels). First, a loss using $\hat{x}_b^a$, the predictions observed at the blue nodes of Figure 4. Additionally, another loss, added as regularization using $\hat{\theta}_b^a = f_+(\hat{\theta_{1_b}^a}, \hat{\theta_{2_b}^a})$. Represented in blue in Figure 5 and 6.
- **multi-task**: The same as the one before, but without the extra loss. Represented in red in Figure 6. The idea is to observe the impact of the extra loss, and if it is needed at all.

We train those models on only one initialization seed because of the heavy computation costs required to train the multi-task models. As a reference, we provide the epoch times on our machine (Nvidia A6000 Ada with 48GB of VRAM) for the different profiling sizes in Table 1. The goal of our experiment is to **observe $\hat{x}_b^a$ can possess more information than $\hat{\theta}_b^a$** about the intermediate x_b^a. Using hand-picked hyperparameters and only one training seed is reasonable to make such a claim, as in this context, multi-task models are expected to be more performant and more stable to initialization seeds than single-task models [21,22].

Profiling size	16.10^3	16.10^4	16.10^5
Epoch time	0:05:38	0:56:15	9:22:02

Table 1: Average epoch time, with a batch size of 50

With only one node, only a few layers are good candidates to obtain results with the four intermediates leaking in the trace. We choose the last layer of the NTT $l = 7$, with $k = 0$ and $p_l = 128$.



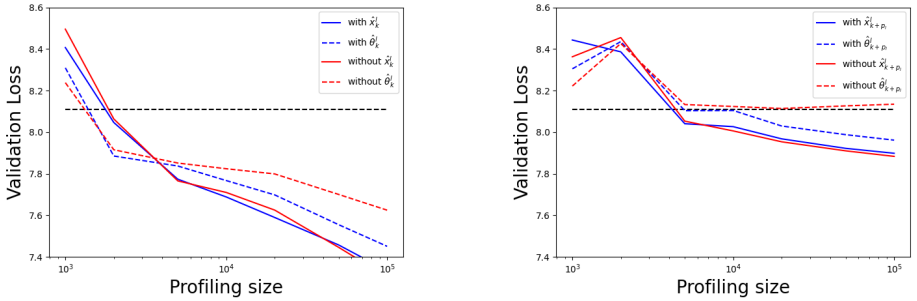
(a) x_k^l (b) $x_{k+p_l}^l$

(c) x_k^{l-1} (d) $x_{k+p_l}^{l-1}$

Fig. 5: Validation loss achieved on different intermediates ($l = 7$, $k = 0$), highlighting the differences between a multi-task model and the single-task models.

From Figures 5, we can suggest that the benefits of including the tasks' relationships inside the neural network model allow us to get the same benefits as in the case of the masked AES implementation, i.e. better performances and better stability. Looking at Figures 7a and 7b, we see that the output of the multi-task model achieves a slightly better validation loss than the single-task model, except in one occurrence in Figure 7b. We can attribute this to luck, as the following training run, which has more training traces, fails to recover more information than a random predictor. **Even if the difference is small, we can observe a clear trend**. Moreover, for the same reasons, and confirmed in Figure 7d, we can see that multi-task models are more stable.

Looking at the difference between the plain and the dotted blue lines in Figures 5, we can also observe a clear trend. The plain blue lines, representing the output after the f_b^l layer, achieve a better validation loss than the dotted lines, representing the output before. This indicates that the **information does propagate** through the relationship graph.



(a) x^l_k (b) $x^l_{k+p_l}$

(c) x^{l-1}_k (d) $x^{l-1}_{k+p_l}$

Fig. 6: Validation loss achieved on different intermediates ($l = 7$, $k = 0$), highlighting the differences between the two multi-task models.

During the inference, we collect for both multi-task models the information contained in $\hat{\theta^a_b} = f_+(\hat{\theta^a_{1_b}}, \hat{\theta^a_{2_b}})$. We plot the result in Figures 6, as we want to observe the difference between the model where a loss was applied on $\hat{\theta^a_b}$ (in blue), and the model where this loss was not applied (in red). We can see that the extra loss is not needed, as the model, even without this extra incentive, is effectively learning a significant part of the correct intermediate on $\hat{\theta^a_b}$. The ultimate learning of each intermediate (seen on $\hat{x^a_b}$ is marginally better since there is less regularization (from the extra loss). This allows the model to learn more from the combination of the intermediates than from the branches that are supposed to be directly related to x^a_b.

7.4 Key Recovery Attack

In the previous section Figures 5 and 6, we discussed the propagation of the information, using as the quantity of interest, the validation loss. While this quantity can be linked to actual attack performances, it is not always the case as can be shown in [10,11]. Therefore, in this section, we propose to perform an attack **using the templates trained in the previous section**. The goal of

this attack is to compare the templates, not to be realistic. The attack will allow us to individually judge the templates and conclude about their quality.

Attack Strategy. Due to the structure of the Number Theoretic Transform (NTT), straightforward key recovery is challenging. This is because the operation involved (polynomial multiplication) causes each output coefficient (our intermediate values) to depend on multiple key indices. To overcome this, we use a chosen ciphertext attack (CCA) that manipulates the inputs so that each intermediate value depends only on a few selected key indices, making the attack feasible. We utilize constant polynomial such that $\mathbf{u} = (\alpha_1, \ldots, \alpha_k)^\intercal \in \mathbb{Z}_q^k$. This trick allows us more control over the conversion between predictions on intermediates to actual key predictions. Another key difficulty introduced by the scheme is the compression of the ciphertext, which is not bijective. Therefore, one has to carefully choose "fixed points" of the decompression, such that without loss of generality, we have that $\mathbf{u} = \textsc{Decode}_{d_u}(\mathbf{c}_1)$. With those assumptions, the output of the inverse NTT in the decryption is then given by :

$$\sum_{i=1}^{m} c_i \cdot \mathbf{s}_i = \mathbf{w} \tag{10}$$

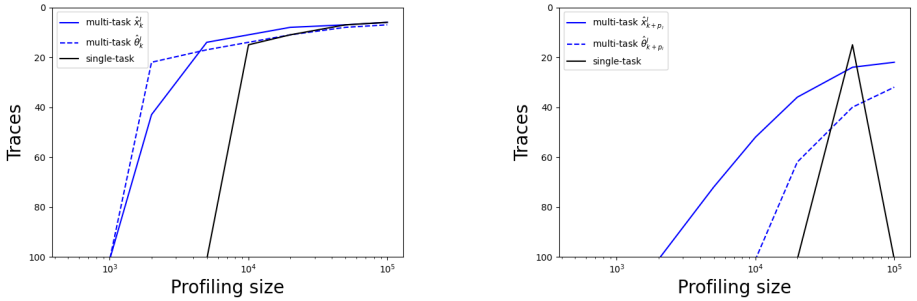
$$\Longleftrightarrow$$

$$\left(\sum_{i=1}^{m} c_i \cdot s_{i,k} \right)_{0 \leq k \leq n-1} = (w_k)_{0 \leq k \leq n-1} \tag{11}$$

Recall that $s_{i,k} \in [-\eta_1, \eta_1]$, thus only $(2 \cdot \eta_1 + 1)^m$ many values indexed by $(s_{1,k}, \ldots, s_{m,k})$ are attainable in each coefficient sum on the left-hand side. Using the ciphertexts $c_1, \ldots, c_m$ and exhaustive enumeration of the possible values for the m-tuplet of key coefficients $(s_{1,k}, ..., s_{m,k})$, we compute the attainable classes for w_k and extract scores of the value of the k-tuplet from the predictions on w_k ($\hat{x}_k^{l=7}$). We then either accumulate predictions on the value of the m-tuplet using a max-loglikelihood distinguisher.

Performing a key recovery with the output layer intermediates is straightforward using this strategy; however, for the one on the layer $l-1$, we simply give perfect knowledge about the other side of the butterfly to transform predictions about layer $l-1$ into predictions about layer l. In this context, the key corresponds to $s_{k=0}$ for Figure 8a and 8c, or $s_{k+p_l=128}$ for Figure 8b and Figure 8d. For each key recovery, we perform 50 experiments using 100 traces randomly picked from the fixed key attack dataset. We note the results of the comparisons between single-task models and multi-task models with the extra loss before the butterfly in Figure 7. Finally, we note the comparison between a model that uses this extra loss and a model without. We plot the results in Figure 8.

From Figures 7 and 8, we can observe similarities between the validation loss and the key recovery results. We see that multi-task models outperform



(a) x_k^l (b) $x_{k+p_l}^l$

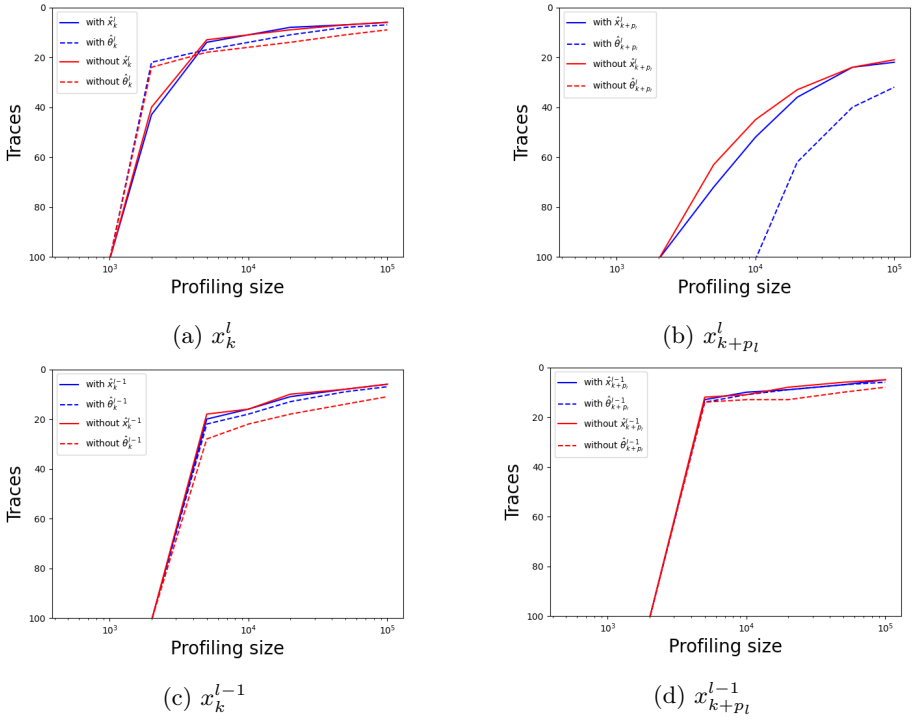
(c) x_k^{l-1} (d) $x_{k+p_l}^{l-1}$

Fig. 7: Subkey recovery results using individually the scores of different intermediates ($l = 7$, $k = 0$), highlighting the differences between a multi-task model and the single-task models.

the single-task models significantly at lower profiling sizes on x_k^l and $x_{k+p_l}^l$, but those differences are less significant when using the scores from the $l-1$ layer. In general, the key recovery benefits from the relationship graph, which confirms that information propagates between the different intermediates within the task graph. Finally, we see that the model trained with an extra loss (in blue), is negatively impacted by the extra regularization, hinting again towards the fact that the model does not need this loss and can learn directly from the relationship graph.

8 Conclusion

In this work, we introduced a new multi-task learning framework for profiled side-channel analysis that explicitly embeds the algebraic relationships between intermediates into the training process, using said relationships as a guide towards agreeable loss landscapes. We use Kyber's inverse Number Theoretic Transform (iNTT) as an example to showcase the idea. Our approach is different from previous work in that we integrate a belief propagation graph directly within the



(a) x_k^l (b) $x_{k+p_l}^l$

(c) x_k^{l-1} (d) $x_{k+p_l}^{l-1}$

Fig. 8: Subkey recovery results using individually the scores of different intermediates ($l = 7$, $k = 0$), highlighting the differences between the two multi-task models.

network, taking advantage of the shared randomness even in a context where one has no control over the countermeasures of the profiling device.

- First demonstration of belief propagation during training without internal randomness: We provide empirical evidence that it is possible to perform **belief propagation directly on masked shares during training**.
- Explicit modeling of task relationships within a neural network: We show that one can leverage the structure of the iNTT from within a neural network and therefore improve both **training stability** and **attack performance**.

While our method incurs substantial computational overhead, the results highlight the practical value of combining domain knowledge, belief propagation, and multi-task learning for profiling side-channel attacks. We believe this work opens a new direction for integrating structured prior knowledge into deep learning models for side-channel analysis and beyond.

8.1 Remark: Using FFTs for f_b^l

Since we are using FFTs to reduce computation complexity for the recombination layer f_+, one accurate remark is, why not use this for the relationship graph too? After all, the relationships are also modular additions and are very good candidates for acceleration using such a strategy. Using the equations 3, 4, 5, and 6, one can define the butterfly relationships using FFTs by reindexing the score vectors according to the constants that are multiplied or the signs present in the operations. This reindexing is low-cost in terms of computations. This makes the implementation of the butterfly layers very efficient, and can easily represent the full NTT graph on a top-tier GPU. While it is not significantly faster than the previous implementation, it is significantly more memory efficient and allows the use of larger batch sizes that speed up the computation. We described the speedups in Table 2.

Profiling size	16.10^3	16.10^4	16.10^5
Epoch time (bs = 50)	0:02:18	0:21:51	3:50:40
Epoch time (bs = 500)	0:00:16	0:2:25	0:24:29

Table 2: Average epoch time, with a batch size of 50 and 500

Problem. We performed the same experiments as previously with both batch sizes. Although the models did learn something, it was **not successful in propagating the information**. Unfortunately, the FFT implementation of the butterfly layers seems to struggle to propagate the knowledge of one node to the other. The training and validation losses on $\hat{\theta_b^a}$ are within a margin of error close to the losses after the graph $\hat{x_b^a}$. Therefore, **using FFTs to implement the relationship graph has been unsuccessful**. We can think of two issues that might have caused the problems. First, it may be that the increase in the batch size reduces the propagation. Training the models with a batch size of 50 shows some sign of propagation, but is still significantly inferior to the previous implementation, while it negates the speed up (but not the potential increase in graph size). The most likely issue is numerical stability. While two multiplications might be fine (since we are using FFTs to perform the f_+ operation also in the previous experiment), compounding the numerical errors might influence the propagation of information. From our experience with such models and belief propagation, numerical stability issues are very common.

9 Acknowledgements

Thomas Marquet has been supported in part by the Austrian Science Fund (FWF) 10.55776/F85 (SFB SpyCode). Elisabeth has been supported in part by the enCrypton project (Grant Agreement No 101079319).

References

1. Joppe Bos, Leo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 353–367, 2018.
2. Elie Bursztein, Luca Invernizzi, Karel Král, Daniel Moghimi, Jean-Michel Picod, and Marina Zhang. Generic attacks against cryptographic hardware through long-range deep learning, 2023.
3. Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. pages 45–68, 08 2017.
4. Rich Caruana. Multitask learning. In Sebastian Thrun and Lorien Y. Pratt, editors, *Learning to Learn*, pages 95–133. Springer, 1998.
5. James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.
6. Lianrui Deng, Lang Li, Yu Ou, Jiahao Xiang, and Shengcheng Xia. Tripm: a multi-label deep learning sca model for multi-byte attacks. *International Journal of Machine Learning and Cybernetics*, 16:4945–4960, 2025. Published 31 January 2025.
7. Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography – PKC 1999*, volume 1560 of *Lecture Notes in Computer Science*, pages 53–68, Heidelberg, Germany, March 1999. Springer.
8. Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked cca2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):88–113, Aug. 2021.
9. Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting belief propagation to counter shuffling of ntts. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):60–88, Nov. 2022.
10. Akira Ito, Rei Ueno, and Naofumi Homma. Perceived information revisited: New metrics to evaluate success rate of side-channel attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):228–254, Aug. 2022.
11. Akira Ito, Rei Ueno, and Naofumi Homma. Perceived information revisited ii: Information-theoretical analysis of deep-learning based side-channel attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):450–474, Dec. 2024.
12. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM cortex-m4. Cryptology ePrint Archive, Paper 2019/844, 2019. `https://eprint.iacr.org/2019/844`.
13. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology — CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
14. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology — CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
15. Jonathan Kuck, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon. Belief propagation neural networks. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, 2020.

16. Jiale Liao, Huanyu Wang, Junnian Wang, and Yun Tang. Switch-t: A novel multi-task deep-learning network for cross-device side-channel attack. *Journal of Information Security and Applications*, 93:104146, 2025.
17. Carlo Lucibello, Enrico Gabriele, Marco Gori, Federico Parisi, and Carlo Baldassi. Deep learning via message passing algorithms based on belief propagation. *Machine Learning: Science and Technology*, 3(3):035005, 2022.
18. Houssem Maghrebi. Deep learning based side-channel attack: a new profiling methodology based on multi-label classification. Cryptology ePrint Archive, Paper 2020/436, 2020.
19. Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. *IACR Cryptol. ePrint Arch.*, 2016:921, 2016.
20. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
21. Thomas Marquet and Elisabeth Oswald. A comparison of multi-task learning and single-task learning approaches. In *Applied Cryptography and Network Security Workshops*, pages 121–138, Cham, 2023. Springer Nature Switzerland.
22. Thomas Marquet and Elisabeth Oswald. Exploring multi-task learning in the context of masked aes implementations. In Romain Wacquez and Naofumi Homma, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 93–112, Cham, 2024. Springer Nature Switzerland.
23. Loïc Masure, Valence Cristiani, Maxime Lecomte, and François-Xavier Standaert. Don't learn what you already know: Scheme-aware modeling for profiling side-channel analysis against masking. 2023:32–59, Nov. 2022.
24. Loïc Masure and Rémi Strullu. Side-channel analysis against anssi's protected aes implementation on arm: end-to-end attacks with multi-task learning. *Journal of Cryptographic Engineering*, 13:1–19, 03 2023.
25. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of power analysis attacks on smartcards. In *Cryptographic Hardware and Embedded Systems (CHES'99)*, volume 1717 of *Lecture Notes in Computer Science*, pages 151–161. Springer, 1999.
26. Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *Progress in Cryptology – LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings*, page 130–149, Berlin, Heidelberg, 2019. Springer-Verlag.
27. Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):209–237, Nov. 2018.
28. Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 513–533, Cham, 2017. Springer International Publishing.
29. Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On Configurable SCA Countermeasures Against Single Trace Attacks for the NTT: A Performance Evaluation Study over Kyber and Dilithium on the ARM Cortex-M4. In *Security, Privacy, and Applied Cryptography Engineering (SPACE 2020)*, volume 12586 of *Lecture Notes in Computer Science*, pages 123–146. Springer, 2020.

30. Mathieu Renauld and Francois-Xavier Standaert. Algebraic side-channel attacks. Cryptology ePrint Archive, Paper 2009/279, 2009.
31. Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the aes: Why time also matters in dpa. volume 5747, pages 97–111, 09 2009.
32. Rafael Carrera Rodriguez, Florent Bruguier, Emanuele Valea, and Pascal Benoit. Cracking the Mask: SASCA Against Local-Masked NTT for CRYSTALS-Kyber. *IACR Communications in Cryptology*, 2(2):1–22, 2025.
33. 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*, pages 282–296, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
34. Hisashi HASHIMOTO Kunihiro KURODA Takeshi FUJINO Yuta FUKUDA, Kota YOSHIDA. Profiling deep learning side-channel attacks using multi-label against aes circuits with rsm countermeasure. *IEICE TRANSACTIONS on Fundamentals*, E106-A(3):294–305, March 2023.
35. Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.
36. Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8577–8587. Curran Associates, Inc., 2020.