Order vs. Chaos: A Language Model Approach for Side-channel Attacks

Praveen Kulkarni[1,2], Vincent Verneuil[1], Stjepan Picek[2] and Lejla Batina[2]

[1] NXP Semiconductors Germany GmbH
[2] Radboud University, Nijmegen, Netherlands

Abstract.
We introduce the Order vs. Chaos (OvC) classifier, a novel language-model approach for side-channel attacks combining the strengths of multitask learning (via the use of a language model), multimodal learning, and deep metric learning. Our methodology offers a viable substitute for the multitask classifiers used for learning multiple targets, as put forward by Masure et al. We highlight some well-known issues with multitask classifiers, like scalability, balancing multiple tasks, slow learning, large model sizes, and the need for complex hyperparameter tuning. Thus, we advocate language models in side-channel attacks.
We demonstrate improvements in results on different variants of ASCAD-V1 and ASCAD-V2 datasets compared to the existing state-of-the-art results. Additionally, we delve deeper with experiments on protected simulated datasets, allowing us to control noise levels and simulate specific leakage models. This exploration facilitates an understanding of the ramifications when the protective scheme's masks do not leak and allows us to further compare our approach with other approaches. Furthermore, with the help of unprotected simulated datasets, we demonstrate that the OvC classifier, uninformed of the leakage model, can parallelize the proficiency of a conventional multi-class classifier that is leakage model-aware. This finding implies that our methodology sidesteps the need for predetermined a leakage model in side-channel attacks.

Keywords: side-channel · masking · language model · deep metric learning · late fusion · multimodal · multitask · NLP · NPLM · siamese network · order vs. chaos

1 Introduction

In side-channel attacks (SCA), adversaries target cryptographic devices that emit measurable physical leakages, such as power consumption [KJJ99, KJJR11], processing time [Koc96], and electromagnetic emanation [GMO01] based on manipulated data and/or executed operations. Assessing cryptographic implementations' security is crucial, and profiled attacks are vital in determining the worst-case security level of such implementations [SMY09]. In profiled attacks, evaluators use a test device to construct an attack model ($\mathcal{M}$) to estimate sensitive variables' conditional distribution. Then, they exploit a target device containing secret information to predict the sensitive variable and reduce the entropy of the secret, ultimately undermining the device's security through side-channel leakage.

Background

The initial profiled SCA, known as *template attacks* (TA), was first introduced by Chari et al. in their seminal work [CRR02]. They assumed that the actual leakage model coincides perfectly with the deterministic portion of the leakage alongside a Gaussian noise assumption. Later, Schindler et al. presented what is termed as *stochastic attacks* (SA) that try to approximate the actual leakage model better [SLP05]. Today, these two methodologies are considered classical profiled SCA, and both can be thought of as generative classifiers as they model the joint probability of the input and the output.

However, these classical techniques grapple with the curse of dimensionality [LPMS18] and often struggle with translation invariance [CDP17]. In contrast, deep learning techniques are known to address these issues via the use of stacks of convolution, pooling, and dense layers [LBH15, GBC16]. The deep learning techniques used for SCA are generally posed as discriminative classifiers, which aim to learn a decision boundary directly from the input features to classify data points into different categories. In this paper, we refer to this approach as the *standard classifier* approach (denoted by the symbol $\mathcal{M}_{Stn}$). It is by far the most widely explored approach for side-channel attacks in recent years [MPP16, CDP17, KPH+19, ZBHV20, WAGP20, WPP20, RWPP21, PWP22] and is shown to outperform classical techniques. Apart from this, more recently, a deep learning-based generative learning approach, namely conditional variational autoencoder, is also proposed for SCA in [ZBC+23].

When it comes to deep learning-based SCA (DLSCA), the two variants (fixed key and variable key) of `ASCAD-V1`[1] datasets introduced in [BPS+20] are used prominently in literature. The standard classifier ($\mathcal{M}_{Stn}$) approaches have successfully attacked `ASCAD-V1` datasets, but their efficacy on the more challenging variable key `ASCAD-V2`[2] dataset, introduced in [MS23], remains unproven. Interestingly, the new line of thinking as put forward by Masure et al. [MS23] is to use *multitask classifiers* (denoted by the symbol $\mathcal{M}_{MTask}$) for targeting multiple intermediate values. The core idea here is to set a primary task akin to a standard classifier, e.g., learning the S-Box output. In contrast, other auxiliary tasks aim at learning targets like different masks and permutation indices. The rationale is that learning auxiliary tasks in tandem with the primary one could enhance the latter's performance. This strategy reportedly succeeded in attacking the `ASCAD-V2` dataset using just 60 traces, with the use of masks knowledge limited to the profiling phase [MS23]. A noteworthy addition to this discourse is the work of Marquet et al. [MO23], which showcased improved outcomes on the `ASCAD-V2` dataset. It is essential to note the distinction in datasets: Masure et al. use a dataset[2] that has 15,000 point-of-interests extracted from raw dataset, while Marquet et al. use a dataset[3] that has 7,181 points-of-interest extracted from raw dataset. For fair comparisons in this paper, we align with Masure et al.'s results, given that we use the same dataset as them with 15,000 points of interest.

Problems with Multitask Classifiers

Multitask classifiers, which aim to handle multiple tasks simultaneously, exploit commonalities among tasks to potentially improve performance. However, there are known challenges associated with multitask classifiers, and we will focus only on a few of those that our approach can tackle.

- *Scalability issues:* If we want to experiment with all the bytes while trying more than one intermediate value with accompanying masking schema-related information, then the number of tasks needed will quickly explode. As the number of tasks grows, the complexity of managing them concurrently escalates. Handling multiple tasks often necessitates a more intricate model architecture, increased memory overhead for storing intermediate activations, and can strain computational resources [ZY22]. This

[1]`https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1`
[2]`https://github.com/ANSSI-FR/ASCAD/tree/master/STM32_AES_v2`
[3]`https://zenodo.org/record/7885814`

eventually leads to scalability issues where multitask classifiers become impractical to use.

- *Balancing multiple tasks:* One of the most critical challenges in multitask learning is ensuring that all tasks are learned equitably. If one task's loss dominates, it can overshadow others, leading to underperformance on non-dominant tasks. This interplay between tasks, known as task interference, can also lead to the negative transfer where learning one task adversely impacts another [Car97, KGC18]. In the context of SCA, we are not sure which intermediate value is easy or difficult to attack and have no means to decide the weight for each task. There are some ways to tackle this issue, like the one introduced in [KGC18], but that adds to the extra effort of finding such weights.
- *Slow learning:* Incorporating multiple tasks can slow down the overall training process. This sluggishness is attributed to the increased complexity, optimization challenges from competing objectives, and additional overheads of evaluation across tasks. The gradient updates from one task could counteract those from another, causing training instability and potentially slower convergence [Rud17].
- *Large model sizes:* To accommodate the diverse needs of multiple tasks, multitask models often require a significant number of parameters. While a portion of these parameters is shared, task-specific layers or heads add to the model's overall size.
- *Complex hyperparameter tuning:* Multitask learning exacerbates the already challenging problem of hyperparameter tuning. Each task might require its own optimal learning rate, batch size, or regularization. Balancing these hyperparameters to ensure stable and efficient learning across all tasks becomes a substantial challenge, often requiring sophisticated strategies or search algorithms [DWH+15].

In conclusion, while multitask classifiers hold promise for a wide range of applications, their deployment necessitates careful consideration of the associated challenges. Achieving a balance between tasks, managing model size, and ensuring efficient learning are active research areas in this domain. In Section 5.4, we will provide insights on how the architecture of our approach can alleviate these problems compared to multitask classifiers.

Motivation for Employing Language Models

Our motivation for employing language models in side-channel attacks is twofold:

1. *Incorporation of NLP advances into side-channel attack:* Traditional profiled attacks primarily utilize variations of the original TA [CRR02] or adopt a multi-class classifier approach [RN20, Vap98, NJ01]. Yet, recent breakthroughs in language models [JM23], particularly transformer-based ones [VSP+17], made significant strides in Natural Language Processing (NLP) tasks ranging from summarization and translation [LLG+20], text generation [RWC+19], and question answering [DCLT19]. These strides have extended beyond NLP, influencing audio classification [BZMA20], Automatic Speech Recognition (ASR) [BZMA20], image classification [DBK+21, LMW+22], object detection [CMS+20], image segmentation [CMS+22], depth estimation [KGA+22], and more. Such developments prompt the question: ***"Can advances in the field of NLP benefit side-channel attacks?"***. To incorporate advancements in NLP into side-channel attacks, the setting of side-channel attacks needs to be framed to include language models. This work addresses this challenge and is the first attempt to utilize language models for side-channel attacks. Our approach, which we term the *Order vs. Chaos* (OvC) classifier, has the potential to spur discussions on the feasibility of using language models for side-channel attacks, thereby adding to the existing array of attack models for profiled attacks.

2. *Addressing multi-target learning challenges in SCA:* While multitask learning is useful in learning multiple targets in the field of SCA, it is not without its challenges. Our approach, anchored in language models, seeks to alleviate some of these issues, paving the way for enhanced multi-target learning.

Contributions

Our main contributions are as follows:

1. We introduce the first language model based approach for side-channel attack.

2. Our approach combines the strengths of multitask learning (via the use of a language model), multimodal learning, and deep metric learning.

3. For a side-channel attack, it is usually challenging to find an effective leakage model to generate labels for profiling. However, based on results from unprotected simulated datasets, we believe our method can effectively address this issue.

4. Our approach presents a unified and simple loss function to learn multiple targets during a side-channel attack. It facilitates implicit weighted learning for each target during the profiling phase. By unifying all targets for binary classification, we ensure a lower parameter count and less hyperparameter tuning demands compared to a multitask classifier.

5. We report new best results on `ASCAD-V1` (both fixed key and variable key) and `ASCAD-V2` dataset.

2 Related Works

Xiangjun et al.'s work [LZC+21] stands out for their utilization of transformer networks, originally designed for NLP tasks. However, we argue that their application of transformer networks cannot solely be viewed as a language-modeling approach. Essentially, Xiangjun et al.'s strategy can be seen as substituting the conventional convolution and pooling layers with a sophisticated transformer-based neural network, all while training with side-channel measurements. Contrarily, our language model trains on specific cryptographic tokens like plaintext, key, and ciphertext, as well as potential intermediary values we aim to target, rather than relying on side-channel measurements. We do, however, incorporate raw side-channel measurements through a distinct, parallel branch in a multimodal learning framework. In short, our language model leans towards learning an array of targets used to label side-channel measurements, contrasting with Xiangjun et al.'s strategy which does not capitalize on multi targets.

Additionally, it is worth drawing attention to the efforts of Hettwer et al. [HGG18]. They enhanced CNNs with domain knowledge neurons, which provide the network with added information, like plaintext, thereby delivering a marked advantage over traditional profiling attacks. In contrast, our methodology leverages a language model to infuse domain knowledge into the neural network, while utilizing embedding layers which are more appropriate for such task [MCCD13]. Moreover, our approach employs binary labels, emphasizing the learning of correct and wrong sequences of the multiple targets under training, enabling the understanding of contextual relationships among these targets. Complementing this, we integrate multimodal and deep metric learning techniques to relate what we learned from language models with the side-channel measurements.

3 OvC Classifier

This section presents the *Order vs. Chaos* (OvC) classifier. First, we explore the key NLP concepts that are vital for understanding our methodology in Section 3.1. Subsequently, in Section 3.2, we disclose the five core ideas underpinning the OvC classifier. Section 3.3 offers a detailed explanation of the OvC classifier's architecture. Wrapping up in Section 3.4, we revisit the well-known profiling and attack phases of the standard classifier and outline the changes needed to these phases when adopting the OvC classifier approach.

Before diving further, it is imperative to introduce some mathematical notations that will aid our discussion in the ensuing subsections. For a profiled attack, it is essential to produce labels ($\mathbf{Y}$) to train the attack model, as illustrated in Eq. (1) below.

$$\mathbf{Y}_i = f_{lk}(f_{iv}(\mathbf{P}_i, \mathbf{K}_i)) \quad \forall i \in \{1, ..., N_m\}, \tag{1}$$

with:

$\boldsymbol{N_m}$	— The number of side-channel measurements (i.e., traces) used for profiling.
$\mathbf{P}$	— A plaintext vector of length N_m. Note that we use only one byte.
$\mathbf{K}$	— A secret vector, i.e., key of length N_m. Note that we use only one byte.
$\boldsymbol{f_{iv}}$	— An intermediate value generator function that gives us target labels using public information (e.g., a plaintext $\mathbf{P}_i$) and the secret $\mathbf{K}_i$ which is part of the secret that the attacker wants to retrieve for i_{th} measurement. If the target intermediate value is 8-bit, then the label can take one of 256 (i.e., 2^8 values).
$\boldsymbol{f_{lk}}$	— A leakage model mapping function that is applied to intermediate targets (when considered as an integer set) obtained from f_{iv}. Here are some examples for an 8-bit intermediate target integer set (a domain with 256 values): - Identity (`ID`): This mapping preserves the values as they are and maps them to a codomain with 256 values. - Hamming weight (`HW`): This mapping calculates the Hamming-weight of each value and maps them to a codomain with nine values. - Least significant bit (`LSB`): This mapping looks at only the least significant bit of each value and maps them to a codomain with two values.
$\mathbf{Y}$	— A target vector of length N_m we want to use for the side-channel attack and is defined by Eq. (1) above. The number of unique values this vector has depends on the choice of surjective mapping function f_{lk}.

3.1 Natural Language Processing (NLP) Preliminaries

This section briefly explains concepts like language model, n-gram, integer tokenization, and the sentence completion task that are related to NLP, which are essential for understanding our approach. For more details on these concepts, refer to Jurafsky and Martin's textbook [JM23].

Language model: In NLP, a language model offers a way to assign probabilities to sequences of words or characters. It understands the common patterns in a language, such as how often words appear and their context, to estimate the probability of a specific sequence. Language models are essential for various NLP tasks like machine translation, speech recognition, text summarization, and autocomplete systems. They help predict the next word or character in a sequence based on the context from previous words or characters, allowing for a better understanding of the text and improved performance in NLP tasks.

N-gram: An *n-gram* is a contiguous sequence of n items from a given text, where an "item" can be a word, character, or any other unit of text. For example, in the sentence "The cat in the hat" the 2-grams (bigrams) include "The cat", "cat in", "in the", and "the hat" while the 3-grams (trigrams) comprise "The cat in", "cat in the", and "in the hat". Multiple n-grams are obtained from text corpora to train language models, capturing context and improving NLP task performance. Such language models are also called N-gram Language model. Given any sequence of these n-grams, a n-gram language model assigns a probability $P(w_1, w_2, ..., w_n)$ to the whole sequence. In short, it learns the joint distribution of n word occurrences.

Integer tokenization: Integer tokenization, or indexing, is a critical NLP step that assigns unique numerical identifiers to each token obtained during tokenization. This process of converting tokens into integer representations benefits neural network approaches in NLP [JM23]. For example, given the sentence "The cat in the hat" tokenization results in the following tokens: "The", "cat", "in", "the", and "hat" Integer tokenization assigns unique numerical identifiers to each token, e.g., 1 for "The", 2 for "cat", 3 for "in", and 4 for "hat". This transformation allows for efficient text processing, enabling neural networks to analyze and learn from textual data effectively. This step is vital for preparing textual data for machine learning algorithms, particularly deep learning architectures like recurrent neural networks (RNNs) and transformers, which rely on numerical inputs for predictions and output generation.

Sentence completion task: A language model uses the frequency of n-grams in a corpus of text to predict the next word. For instance, consider the incomplete sentence "The cat is sitting on the _____". Based on the trigram "cat is sitting" the language model might predict "mat" instead of "moon" as the next word, as "The cat is sitting on the mat" is more common than "The cat is sitting on the moon" in the text corpus used for training. This is an example of how language models can be used to complete sentences by predicting the most probable next word.

3.2 Five Key Ideas

In this section, we present the five ideas that underlie the design of the OvC classifier. The architecture of the OvC classifier, to be elaborated upon in Section 3.3, comprises three fundamental blocks: the language model, the deep metric learning, and the preprocessor. The initial three ideas, explored in Sections 3.2.1, 3.2.2, and 3.2.3, facilitate a better understanding of the language model block. Subsequently, the ideas presented in Sections 3.2.4 and 3.2.5 assist in comprehending the deep metric learning block. The final block, the preprocessor block, is akin to the trunk or stem of a standard classifier. Since it is not exclusive to the OvC classifier's design, we will not go into its details here but will explore it in Section 3.3.

3.2.1 Forming Sentences (introducing n-cryptograms)

In Section 3.1, we discussed how n-grams are extracted from sentences of arbitrary lengths to train language models. In the context of side-channel attacks, we will explain how to get n-grams essential to train our language model.

Let us start by explaining how to form sentences. For a side-channel attack the measurement can be labeled in many ways, like plaintext, ciphertext, key, masks, or output from an S-Box. These labels can be categorized into two groups: data tokens or intermediate values. Data tokens might include elements such as plaintext, ciphertext, key, and masks from a masking scheme. On the other hand, intermediate values represent the outcomes after executing cryptographic operations on these data tokens, like the output from an S-Box, i.e., `sbox-output`. Let us assume that we have five such labels as given by an example tuple `(key, sbox-output, plaintext, ciphertext, mask)`. For every given measurement, we can have such a tuple, which can be thought of as a "sentence", while the labels act as "words". For these sentences to be meaningful, it is vital they contain intermediate values, as they provide a contextual link to other labels. Without these intermediate values, a sentence comprised only of data tokens will appear unrelated, making it challenging for a language model to derive insights. Hence, incorporating one or more intermediate values in our sentences is crucial.

Unlike NLP sentences with arbitrary lengths and words occurring in any position within a n-gram, these side-channel attack sentences have two unique properties. First, they have a fixed length. Second, a word (i.e., a value for some data token or intermediate value) occurs in a fixed position based on the data token or intermediate value it represents. For example, the word `plaintext=5` always occurs in position 3 in the example tuple. Given these two unique properties of sentences in the context of side-channel attacks, we propose using these sentences directly as n-grams, eliminating the need for an n-gram extraction phase. Furthermore, since words can appear in fixed positions, fewer possible sentence combinations exist, making it possible to train the language model faster. We call these special n-grams "n-cryptograms".

Finally, we will discuss integer tokenization or indexing (see Section 3.1) that benefits neural network-based language models. In NLP, the vocabulary size (V) is the total number of unique tokens appearing in all sentences from the text corpora on which we train the language model. For side-channel attacks, using the example n-cryptogram tuple above, the vocabulary size is $V = 1280$ (i.e., $5 * 256$), where 256 represents the number of unique values a cryptogram can have, assuming each cryptogram represents an 8-bit integer with 256 possible values. To make each word unique, we add a number to its integer value. For example, in our example n-cryptogram tuple, if we have a word with an integer value *value* related to the first cryptogram (i.e., `key`), its index (i.e., the value after integer tokenization) is $(value + 256 * 0)$. If it is the second cryptogram (i.e., `sbox-output`), its index is $(value + 256 * 1)$. If it is the last cryptogram (i.e., `mask`), its index is $(value + 256 * 4)$. Thus, for a given measurement with a n-cryptogram tuple `(key=065, sbox-output=024, plaintext=129, ciphertext=032, mask=234)`, the integer tokenization or indexing results in the n-cryptogram `(0065, 0280, 0641, 0800, 1258)`. We use such integer sequences to train the language model of our OvC classifier approach.

3.2.2 *Order vs. Chaos*

Labels, determined by choices for f_{lk} and f_{iv} as described by Eq. (1), may not truly reflect the real leakage. This discrepancy can result in ambiguous labeling [KV22], which can compromise the accuracy if applied directly to standard classifiers ($\mathcal{M}_{Stn}$). In our preceding research [KV22], we presented the *'Multi-trunk Order Vs. Chaos'* classifiers, which were based on the "*Order vs. Chaos*" concept. This approach indirectly utilizes these ambiguous labels by converting the challenge into a binary classification task, thus producing more appropriate binary labels. In the similar lines, this subsection delves into adapting this concept for the OvC classifier. Simply put, for the OvC classifier, half of the training samples are termed *"Order"* when every gram in the n-cryptogram tied to each side-channel measurement is left unchanged. Conversely, the remaining half is labeled *"Chaos"* when certain grams in the n-cryptogram linked to each side-channel measurement undergo changes.

Let us begin by defining training tuples, which are composed of input-output pairs for a standard classifier ($\mathcal{M}_{Stn}$), as shown in Eq. (2), followed by the OvC classifier ($\mathcal{M}_{OvC}$) in Eq. (4). For a typical standard classifier, the input consists of measurements ($\mathbf{T}$), and the output corresponds to a label ($\mathbf{Y}$), as specified in Eq. (1).

$$\Big(\mathbf{T}_i, \mathbf{Y}_i\Big) \quad \forall i \in \{1, \cdots, N_m\}, \tag{2}$$

with:

$\boldsymbol{N_m}$ — The number of side-channel measurements (i.e., traces) used for profiling.

$\boldsymbol{N_s}$ — The number of samples in side-channel measurements (i.e., trace).

$\mathbf{T}$ — The side-channel measurements 2D vector with a dimension of $N_m \times N_s$.

$\mathbf{Y}$ — A target vector of length N_m we want to use for the side-channel attack and is defined by Eq. (1) above. The number of unique values this vector has depends on the choice of surjective mapping function f_{lk}.

For the OvC classifier, we initially define the training tuple as presented in Eq. (3). Here, the *"Ordered"* samples are designated by the label '✓', while the *"Chaotic"* (or unordered) samples carry the label '✗', each represented in distinct sets. Subsequently, we combine these two sets in Eq. (4) to yield the unified training tuple for the OvC classifier.

The input segment of the training tuple for each set is composed of n-cryptograms and side-channel measurements ($\mathbf{T}$). The n-cryptogram vector of size $(N_m \times (P+Q))$ is formed by joining two vectors: the *guess-gram* ($\mathbf{GG}$/$\mathbf{\ddot{G}G}$) with dimension $(N_m \times P)$ and the *fix-gram* ($\mathbf{FG}$) with dimension $(N_m \times Q)$ as defined in Eq. (3) below. For the ordered samples, the guess-gram reflects the actual values computed for the known correct key, represented by $\mathbf{GG}$. On the other hand, for the chaotic samples, the original guess-gram ($\mathbf{GG}$) undergoes mutations, producing entirely incorrect values. This results in a modified guess-gram, represented by $\mathbf{\ddot{G}G}$.

Given the above details, it is evident that the distinction in the set representing chaotic examples emerges from the mutations we perform on the actual guess-gram. Yet, it is crucial to note that the fix-gram and side-channel measurements stay unchanged across both sets.

$$\left.\begin{matrix} \Bigg(\bigg(\Big(\underbrace{\mathbf{GG}_{i,1}, \cdots, \mathbf{GG}_{i,K}, \mathbf{FG}_{i,1}, \cdots, \mathbf{FG}_{i,L}}_{n-cryptogram}\Big), \mathbf{T}_i\bigg), \checkmark\Bigg) \\ \Bigg(\bigg(\Big(\underbrace{\mathbf{\ddot{G}G}_{i,1}, \cdots, \mathbf{\ddot{G}G}_{i,K}, \mathbf{FG}_{i,1}, \cdots, \mathbf{FG}_{i,L}}_{n-cryptogram}\Big), \mathbf{T}_i\bigg), \text{✗}\Bigg) \end{matrix}\right\} \forall i \in \{1, \cdots, N_m\}, \tag{3}$$

with:

$\boldsymbol{P}$ — The number of guess-grams in the n-cryptogram.

$\boldsymbol{Q}$ — The number of fix-grams in the n-cryptogram.

$\mathbf{GG}$ — The guess-gram 2D vector with dimension $N_m \times P$, part of the n-cryptogram 2D vector. Typically comprised of targets dependent on the key ($\mathbf{K}$) as given by Eq. (1). Even the key itself can be used. Basically, these are the targets that are guessed, based on the key guess during the attack phase of the side-channel attack.

$\mathbf{\ddot{G}G}$ — The mutated guess-gram 2D vector with dimension $N_m \times P$, part of the n-cryptogram 2D vector. It is an altered version of guess-gram ($\mathbf{GG}$) which is obtained by mutating each of its value.

$\mathbf{FG}$ — The fix-gram 2D vector with dimension $N_m \times Q$, part of the n-cryptogram 2D vector. Generally made up of targets that are independent of the key ($\mathbf{K}$). They consist of targets like plaintext, ciphertext, masks, etc. which do not have key related component that needs to be guessed during the attack phase of the side-channel attack.

Now that we have defined fix-gram (**FG**), guess-gram (**GG**) and mutated guess-gram ($\ddot{\mathbf{GG}}$) let us examine it with reference to data token tuple (n-cryptogram) example mentioned in Section 3.2.1, specifically `(key, sbox-output, plaintext, ciphertext, mask)`. The first two data tokens are key-dependent (i.e., guess-gram related), while the last three data tokens are key-independent (i.e., fix-gram related). Therefore, in our example, the n-cryptogram has $n = 5$ elements, with $P = 2$ representing the guess-gram tokens and $Q = 3$ indicating the fix-gram tokens.

Furthermore, for the OvC classifier, we merge the two sets as outlined in Eq. (3). It is important to note that, when generating the chaotic examples set, there's a vast array of potential examples that can be generated, as introducing various random mutations can produce distinct guess-grams. However, since we're training the OvC classifier — a binary classifier — we limit ourselves to creating only N_m chaotic examples to maintain a balanced dataset. Consequently, the merged dataset will encompass $2N_m$ samples. Additionally, with each training epoch, we can introduce distinct mutations to craft the chaotic examples, ensuring that the chaotic part of the merged dataset remains unique for every epoch. Finally, we also randomize the consolidated dataset using a varying random shuffle seed (R) for each epoch. This ensures a uniform distribution of both ordered and chaotic samples throughout the dataset, and each epoch presents a different shuffle sequence of these samples. All these operations result in merged vectors $\mathbf{GG}'$, $\mathbf{FG}'$, $\mathbf{T}'$, and $\mathbf{Y}'$, as defined in the equation below.

$$\Bigg(\bigg(\underbrace{\Big(\mathbf{GG}'_{i,1},\cdots,\mathbf{GG}'_{i,P},\mathbf{FG}'_{i,1},\cdots,\mathbf{FG}'_{i,Q}\Big)}_{n-cryptogram},\mathbf{T}'_i\bigg),\mathbf{Y}'_i\Bigg) \quad \forall i \in \{1,\cdots,2N_m\}, \tag{4}$$

with:

GG′ — Created using **GG** and $\ddot{\mathbf{GG}}$ (defined in Eq. (3)), with the first half elements using guess-gram (**GG**) for ordered examples and the remaining half using mutated guess-gram ($\ddot{\mathbf{GG}}$) for chaotic examples. This is followed by a shuffle using the random shuffle seed R.

FG′ — The fix-gram 2D vector with dimension $2N_m \times Q$, part of the n-cryptogram 2D vector. Created using **FG** (defined in Eq. (3)) by duplicating it twice. This is followed by a shuffle using the random shuffle seed R.

T′ — The side-channel measurements 2D vector with a dimension of $2N_m \times N_s$. Created using **T** (defined in Eq. (2)) by duplicating it twice. This concatenated vector is then shuffled with random shuffle seed R.

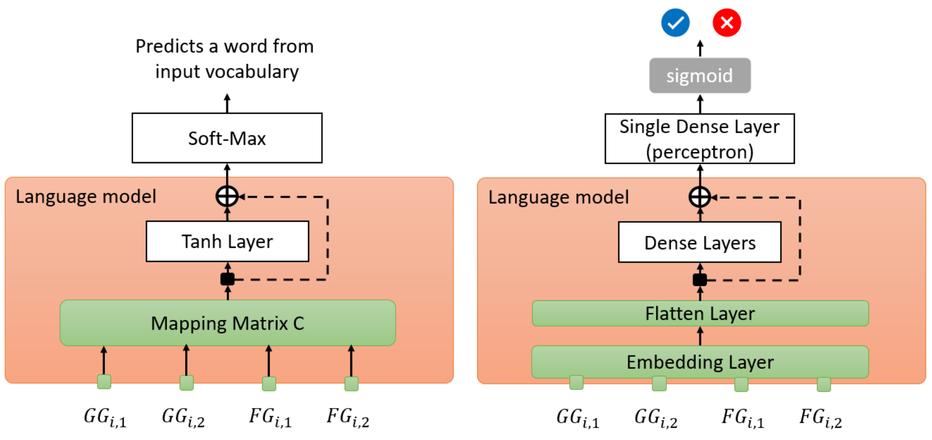
Y′ — A vector of length $2N_m$ that we want to use as label for training the OvC classifier. Note that, unlike **Y** (defined in Eq. (1)), it has double the length and can only have two unique values, representing *Order* and *Chaos*. Created by concatenating two vectors of length N_m, where first vector is set with true (✓) and second vector is set with false (✗). This concatenated vector is then shuffled with random shuffle seed R.

3.2.3 Binary Language Model (BLM)

The task here is to map $2N_m$ n-cryptograms from our training tuple, as detailed in Eq. (4), to their corresponding binary labels representing *Order vs. Chaos*. Although the training tuples input part also includes measurements ($\mathbf{T}'$) as input, we disregard that for now and address it in Section 3.2.5. Note that for simplicity in Figure 1, we show n-cryptogram for i_{th} example with $P = 2$ and $Q = 2$ as input to our language model.

One might consider using simple neural networks comprised of dense layers to learn the mapping from n-cryptograms to binary labels. However, this approach presents challenges because our input, i.e., n-cryptograms, is not composed of real numbered features but rather categorical features, with each feature having 256 possible values (if the target is 8-bit integer). Using them as real-valued features can result in issues such as ordinal assumptions, non-uniform scaling, and sparse representation [GBC16, MCCD13, GB16]. To tackle these challenges, categorical features are commonly converted into more suitable representations, such as one-hot encoding or dense continuous embeddings (for instance, using embedding layers). One-hot encoding generates binary vectors, with each category represented as a separate dimension while embedding maps each category into a lower-dimensional continuous space. Both methods allow neural networks to learn meaningful representations of categorical features without assuming ordinal relationships or dealing with scaling issues.

In our approach, we favor generating meaningful learnable embeddings with embedding layers over using one-hot encoding before feeding the data into downstream dense layers, as depicted in Figure 1. The rationale for avoiding one-hot encoding is: first, it leads to high dimensional input vectors; second, it is computationally inefficient due to unnecessary calculations, as most values in one-hot representation are zero; and finally, dense layers with one-hot encoded inputs treat each input category as independent, without capturing any relationships between them. Conversely, embedding layers learn continuous representations that capture semantic and syntactic relationships between categories, making it easier for the model to generalize and learn from the input data [BDVJ03, MCCD13, PSM14].


Predicts a word from input vocabulary
Soft-Max
Language model
Tanh Layer
Mapping Matrix C
$GG_{i,1}$ $GG_{i,2}$ $FG_{i,1}$ $FG_{i,2}$
sigmoid
Single Dense Layer (perceptron)
Language model
Dense Layers
Flatten Layer
Embedding Layer
$GG_{i,1}$ $GG_{i,2}$ $FG_{i,1}$ $FG_{i,2}$

(a) NPLM [BDVJ03] **(b)** BLM

Figure 1: NPLM vs. BLM

We suggest treating the task of mapping n-cryptograms to binary labels as a sentence completion task, where the objective is to predict the next word (a binary label) based on the input sequence of words. We utilize the simplest version of a neural network-based language model called Neural Probabilistic Language Model (NPLM), as introduced by Bengio et al. in [BDVJ03] and shown in Figure 1a. More advanced language models based on recurrent neural networks [KF17] or transformer-based models [VSP$^+$17] could be explored but we leave it as future work.

Our modified NPLM architecture, which we call Binary Language Model (BLM) as it has a binary output, is shown in Figure 1b and involves three modifications. First, we use embedding layers to map input words to feature vectors instead of using mapping matrix C. Second we substitute the "tanh" layer with a stack of dense layers for learning hierarchical representations. Third we use a perceptron layer with sigmoid loss at the output for binary classification rather than using a softmax layer that was used by NPLM for predicting multiple words. Our proposed binary language model has fewer parameters to learn at the output layer, as it does not require predicting a word from the input vocabulary, like NPLM, and has a single output.

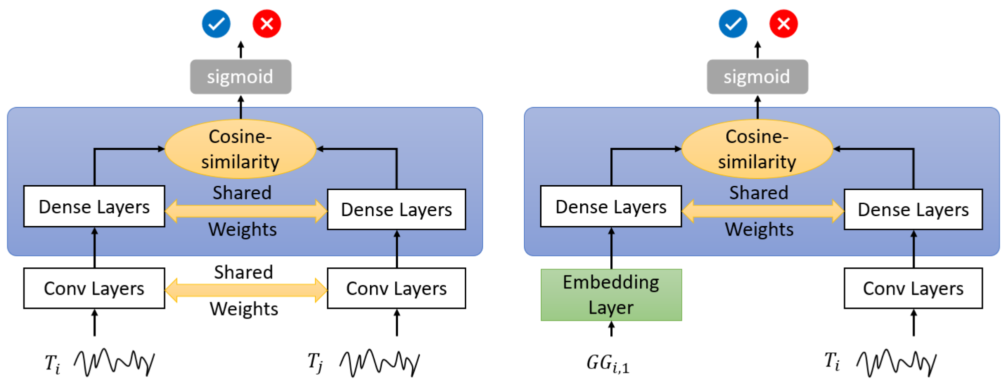
Lastly, we maintain using *skip-connections* from the original NPLM network, as shown by dotted lines in Figure 1. We concatenate the input embeddings (by flattening the embedding layer) and the output of dense layers before feeding them to the downstream perceptron layer. The first advantage of using these connections is that the model can learn direct associations between input and output words, capturing some of the simpler relationships in the data without relying on intermediate hidden layers. This helps the model to better generalize to unseen or rare word combinations [BDVJ03]. The second advantage of direct connections is that they help maintain a more stable gradient flow during training. By skipping the intermediate layers, the gradient information can propagate more effectively through the network, enhancing the training process and making it easier for the model to learn from the data [HZRS16].

3.2.4 Deep Metric Learning (DML)

Deep Metric Learning (DML), a burgeoning subfield of machine learning, leverages deep neural networks to learn a distance function or similarity metric that operates on input pairs [HCL06]. The core concept is to generate a representation, often referred to as an *embedding*, for each input so that the embeddings of similar inputs are close together and those of dissimilar inputs are farther apart in the learned embedding space [KB19]. Although DML is rooted in the principles of distance metric learning, it takes advantage of deep learning models' ability to generate complex, non-linear representations of input data [BCV13, HA15]. The multiple layers of these models enable the learning of data representations at different levels of abstraction, thereby augmenting the model's ability to detect and comprehend intricate patterns in the data.

Regarding the network architecture, Siamese neural networks, first introduced by Bromley et al. [BGL+93] in 1993, have garnered significant attention in the field of DML. These networks consist of two or more identical subnetworks or "branches" that process individual data points. The same architecture and shared weights across these branches allow the network to learn a single set of parameters that can generalize well across different pairs of inputs [Chi21]. This weight sharing reduces the parameter space, thus decreasing the risk of overfitting and improving generalization. Each pair of samples is fed into the Siamese network, with each sample processed by a distinct subnetwork. The output is a single value representing the similarity between the two inputs according to the learned metric. Figure 2a demonstrates a typical Siamese network employed for metric learning. This network is trained using two side-channel measurements ($\mathbf{T}$). If the measurements belong to the same category, that is, the corresponding intermediate value under attack is the same, then they are deemed similar; otherwise, they are considered dissimilar.

The selection of an appropriate distance metric or similarity function is a crucial factor influencing the overall performance of DML [Kul13, BHS13]. The literature describes various prevalent similarity functions, including Euclidean distance, cosine similarity, and Mahalanobis distance, among others. The suitability of these functions can differ based on the nature of the task in question. For example, in text-related tasks, cosine similarity is often preferred, as it emphasizes the orientation or angle of high-dimensional vectors, typically word embeddings, over their magnitude [MCCD13]. Conversely, in image-related tasks where the absolute difference in pixel intensities is crucial, Euclidean distance is commonly used. In the context of our work, which involves word embeddings (as discussed in Section 3.2.5), we favor using the cosine similarity function to train our OvC classifier.



(a) Typical **(b)** Modified

Figure 2: Deep Metric Learning

3.2.5 Multimodal Learning

This subsection introduces a modification to the standard Siamese network, depicted in Figure 2a. The proposed alteration, illustrated in Figure 2b, involves feeding one subnetwork with word embeddings, which are dense vector representations, and the other with side-channel measurements.

For simplicity, the figure shows a single guess-gram as an input. However, in actual scenarios, we use an n-cryptogram, as outlined in Eq. (4) and further detailed in Subsection 3.3.2. We treat the n-cryptogram and the side-channel measurement as distinct modalities as they come from different sources. However, they complement each other, promoting multimodal learning. To ensure the downstream deep metric learning block functions properly, the output dimensions of both modalities must align. Within this framework, we capitalize on the information that links a specific n-cryptogram to its associated side-channel measurement. Importantly, when the example is *"Ordered"* with the label as ✓, the corresponding word embedding is considered similar to the side-channel measurement. If the example is *"Chaotic"* with the label as ✗, the respective word embedding is considered dissimilar to the side-channel measurement. Given the stark contrast between the two modalities, we suggest that the adapted architecture we propose facilitates multimodal learning.

Multimodal learning, which harnesses the unique and complementary insights offered by various data types, combines different kinds of data to produce more accurate predictions or richer representations. This approach advocates that blending different data types provides a broader context, thereby enhancing the accuracy of results. When this strategy is paired with Siamese networks, the efficiency of multimodal learning is significantly improved [NKK$^+$11]. In this setup, the two subnetworks of the Siamese structure handle different data types separately and then merge and compare their outputs to detect patterns or similarities. Consequently, each data type undergoes independent processing for the majority of the network's operation, with their representations consolidated only in the final stages, a concept known as "late fusion" [KTS$^+$14]. The combination of multimodal learning with Siamese networks not only uncovers hidden connections between different data types that might be overlooked when analyzed separately but also improves the model's robustness against noise or errors, as the presence of multiple data types adds a layer of redundancy. More details on multimodal learning can be found in [NKK$^+$11, BAM19].

3.3 Architecture

In this section, we explain the OvC classifier's architecture, which combines all the ideas from Section 3.2. As illustrated in Figure 3, the architecture of the OvC classifier consists of three essential blocks: the language model, deep metric learning, and the preprocessor. This section is organized into three parts. The first part delves into the preprocessor block. The second part describes how altering and merging BLM with DML forms the language model and deep metric learning blocks. The last part widens the design to incorporate masking scheme-aware profiling, where mask information is present during the profiling stage but not available during the attack stage.

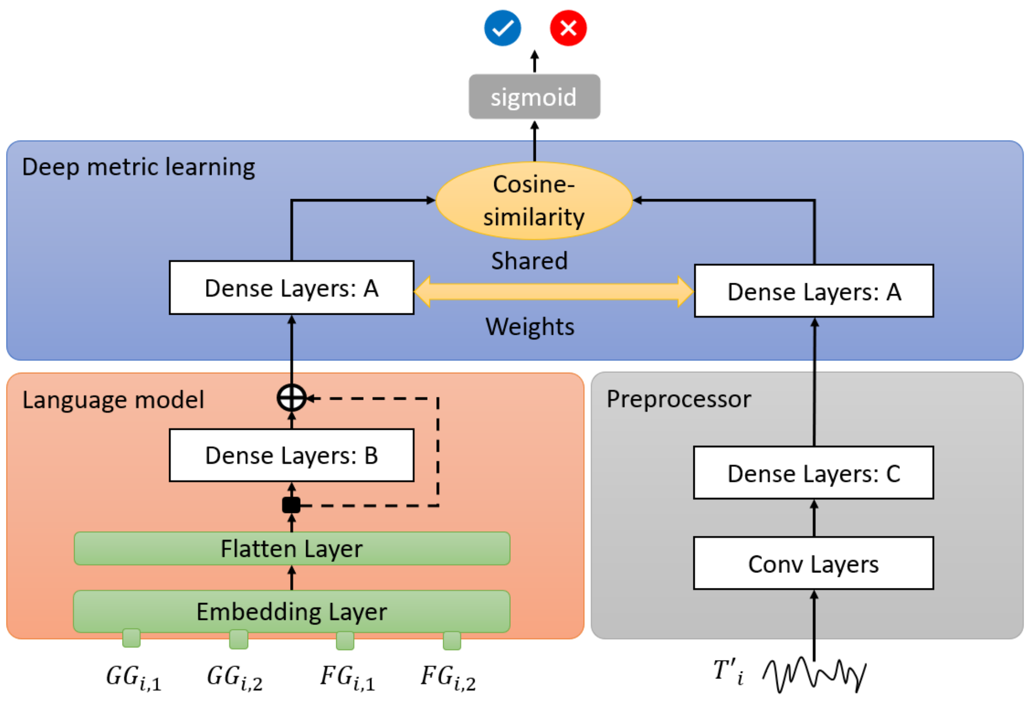


Figure 3: OvC Classifier

3.3.1 The Preprocessor Block

The preprocessor block functions in a manner akin to the stem of a standard classifier. It is designed to carry out preprocessing tasks leading to dimensionality reduction and translation invariance while also learning feature maps. To simplify, we opt for the same neural network architectures for the preprocessor block that are applied in the standard classifiers for the corresponding commonly used public datasets. Our only alteration is to discard the softmax layer found in the standard classifier, substituting it with a dense layer. The neuron count in this layer matches the output dimensionality of the language model block, facilitating the alignment of the embeddings from both the language model and preprocessor blocks. It is important to highlight that unlike the standard classifier, which aims to learn discriminative embeddings that aid in distinguishing different intermediate target values, the preprocessor block strives to learn embeddings that differentiate between an n-cryptogram and a mutated n-cryptogram, with support from the deep metric learning block and language model block.

3.3.2 Integration of BLM and DML

The BLM and DML blocks strive to learn different aspects. The BLM aims to differentiate between n-cryptograms and mutated n-cryptograms, whereas the DML seeks to bring similar embeddings together and separate dissimilar ones. Interestingly, they can both utilize the same training labels. For instance, a false label would be classified as a mutated n-cryptogram by the BLM, while the DML would consider input embeddings from two modalities as dissimilar. On the contrary, for a true label, BLM classifies it as an n-cryptogram, while DML treats input embeddings from two modalities as similar.

We leverage this shared usage of labels between the two tasks to merge BLM and DML. As depicted in Figure 3, we use DML for the deep metric learning block, using its binary head to train the entire OvC classifier. For the language model block, we only consider the base part of the BLM, without the head, composed of a single dense layer followed by a sigmoid activation (as seen in Figure 1b). In the deep metric learning block, the output from the language model block becomes one modality, while the other modality is the output from the preprocessor block. The deep metric learning block's input dimension is defined by the language model block's output. This includes the combined total of output units from block 'Dense Layers: B' and the output units from the flattened n-cryptogram embedding layer, which are accessible because of skip-layer connections. As stated in Section 3.2.5, we ensure that the output units of block 'Dense Layers: C' align with the output dimensionality of the language model block.

Finally, the training gradients derived from the deep metric learning block play a crucial role in ensuring that the language model block can differentiate between n-cryptograms and their mutated counterparts. Conversely, the DML employs cosine similarity loss to attract the embeddings of the language model and preprocessor blocks closer together when the language model block is fed an n-cryptogram, and to repel the embeddings when the input is a mutated n-cryptogram.

3.3.3 Masking Scheme Aware Architecture

To enable our $\mathcal{M}_{OvC}$ classifier to leverage mask-related knowledge akin to the $\mathcal{M}_{MTask}$, we introduce a modification to the basic architecture shown in Figure 3. The updated architecture is presented in Figure 4. A simplistic method could incorporate mask-related cryptogram in n-cryptogram as fix-gram (**FG**) since they are not related to key-guess data tokens. However, this would require the presence of mask-related data during the attack. To bypass this requirement, we construct a neural network identical to the preprocessor block and add a softmax layer to execute multi-class classification with masks as target labels. This resulting trained network, referred to as a "pre-trained neural network" (PTNN), is shown in Figure 4. It is important to note that we assume the presence of two mask tokens, but this can vary depending on the dataset and the understanding of the cryptography implementation. The PTNN output is fed into the language model by concatenating with its input. The language model treats this as a fix-gram token, but the use of PTNN allows us to eliminate the necessity for masks during the attack phase.

3.4 Profiling and Attack Phase

In this section, we take a fresh look at the profiling and attack phases of the standard classifier. Additionally, we identify and explain the adjustments needed in these stages to ensure their compatibility with the OvC classifier.

3.4.1 Profiling Phase

In the profiling phase, we train both the standard classifier model, $\mathcal{M}_{Stn}$, and the OvC classifier model, $\mathcal{M}_{OvC}$, using their respective training tuples as outlined in Eq. (2) and

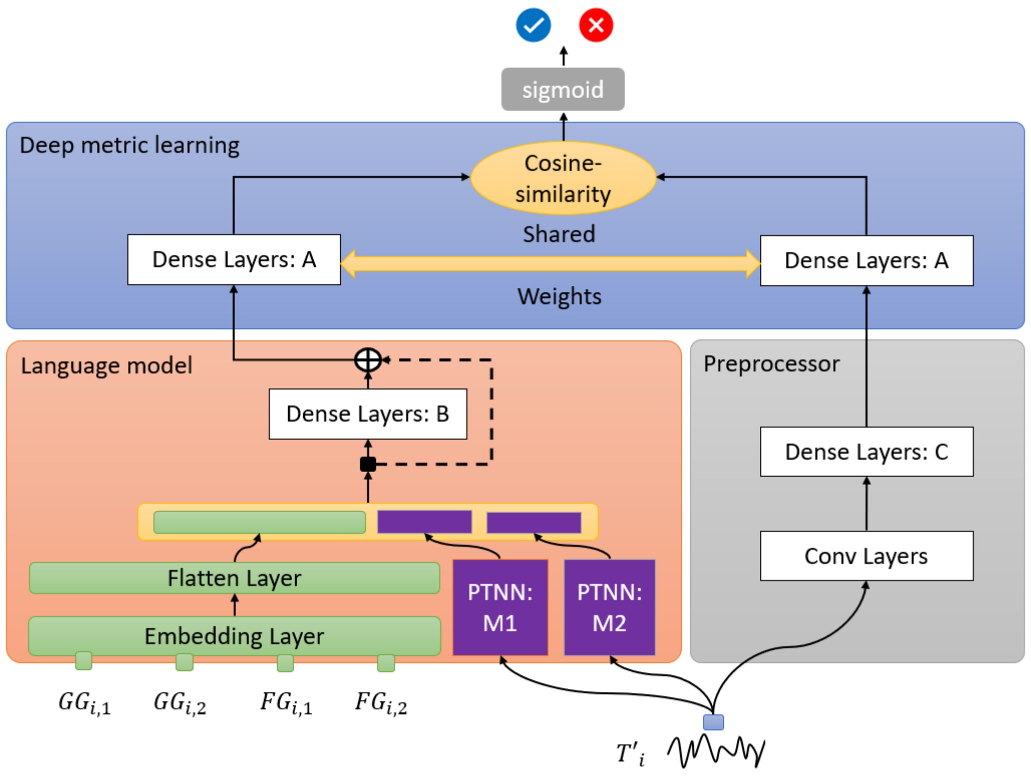


Figure 4: OvC Classifier: Schema Aware Extension

Eq. (4), respectively. It is worth mentioning that the training dataset for the OvC classifier consists of $2N_m$ examples, twice the size of the standard classifier dataset. This is due to our approach of generating negative examples. For every training epoch, the dataset for the standard classifier stays unchanged. However, for the OvC classifier, because of different mutations to guess-gram per epoch and due to different shuffle sequence per epoch we end up with unique dataset per epoch.

3.4.2 Attack Phase

To understand the attack phase, we first define attack tuples, i.e., input-output pairs used during the attack phase. We define them in Eq. (5) for the standard classifier, followed by Eq. (6) for the OvC classifier. Further, in Eqs. (7) and (8), we illustrate how probabilities obtained by feeding attack tuples to $\mathcal{M}_{Stn}$ and $\mathcal{M}_{OvC}$ can be consolidated to determine the sum of log probabilities over N_a side-channel measurements. By presenting comparative equations for both methodologies, our intention is to deepen the understanding of the OvC classifier in the context of the standard classifier. Concluding this section, we delve into T_{GE0}, a metric frequently employed to quantify the strength of a side-channel attack [SMY09]. This value represents the necessary number of side-channel measurements to achieve a guessing entropy of zero.

Attack tuple for $\mathcal{M}_{Stn}$: The attack tuple for the standard classifier mirrors the structure of the training tuple employed during profiling. The only difference is that there are N_k sets of labels corresponding to each potential key guess. The following illustrates the attack tuple for the standard classifier under the assumption that the key guess is k.

$$\left(\mathbf{T}_i, \mathbf{Y}_i^k\right) \quad \forall i \in \{1, \cdots, N_a\} \; and \; k \in \{0, \cdots, N_k - 1\}, \tag{5}$$

with:

$\boldsymbol{N_a}$ — The number of side-channel measurements (i.e., traces) used for an attack.

$\boldsymbol{N_k}$ — The number of guess values the key (k) can take. The value can be any number between 0 and $N_k - 1$. For a 8-bit target N_k is equal to 256.

$\mathbf{T}$ — This represents a 2D vector of side-channel measurements with a dimension of $N_a \times N_s$. Like in Eq. (2), where $\mathbf{T}$ stands for the side-channel measurements during the profiling phase, here it's reutilized to depict the side-channel measurements in the attack phase. Please bear in mind that unlike profiling phase these traces are typically tied to a specific key, which remains undisclosed during an attack.

$\mathbf{Y}^{\boldsymbol{k}}$ — A target vector of length N_a which is generated using Eq. (1), uses the key guess (k) in place of the correct key (k^*). This substitution is necessary as the correct key remains undisclosed during the attack phase.

Attack tuple for $\boldsymbol{\mathcal{M}_{OvC}}$: The attack tuple of the OvC classifier somewhat mirrors the structure of the training tuple used during the profiling phase, albeit with certain differences. This procedure necessitates calculating the guess-gram – a key-dependent segment of the n-cryptogram – N_k times for every possible key guess. Hereafter $\mathbf{GG}^k$ denotes the guess-gram associated with key guess k. It is worth noting that, unlike in the case of the standard classifier, the key guess doesn't influence the OvC classifier's labels. In fact, the attack phase doesn't require the generation of negative examples. The attack tuple is created N_k times, each construction corresponding to a potential key guess. We proceed with the assumption that all attack tuples, for every plausible key guess, are true (i.e., *"Ordered"*), and hence indicated by an all-ones label $(\vec{\mathbf{1}})$. If the OvC classifier has learned effectively during the profiling phase, it is expected to yield larger probabilities for the correct key guess. Conversely, for incorrect guesses, the input tuple appears as a negative example for the OvC classifier, thereby generating smaller probabilities. The following equation showcases the attack tuple for the OvC classifier, for a given key guess k.

$$\begin{gathered}\left(\left(\underbrace{\left(\mathbf{GG}_{i,1}^k, \cdots, \mathbf{GG}_{i,P}^k, \mathbf{FG}_{i,1}, \cdots, \mathbf{FG}_{i,Q}\right)}_{n-cryptogram}, \mathbf{T}_i\right), \vec{\mathbf{1}}_i\right) \\ \forall i \in \{1, \cdots, N_a\} \; and \; k \in \{0, \cdots, N_k - 1\},\end{gathered} \tag{6}$$

with:

$\mathbf{FG}$ — This denotes the fix-gram 2D vector with dimension $N_a \times Q$, which forms part of the n-cryptogram 2D vector and comprises Q fix-grams. While it shares similarity with the $\mathbf{FG}$ term as outlined in Eq. (3), this $\mathbf{FG}$ term is redefined specifically for the attack phase and is independent of key guess assumptions.

$\mathbf{GG}^{\boldsymbol{k}}$ — This denotes the guess-gram 2D vector with dimension $N_a \times P$, which forms part of the n-cryptogram 2D vector and comprises P guess-grams. While it is similar to the term **GG** as defined in Eq. (3) for the profiling phase when the correct key is known, in the attack phase, the correct key (k^*) remains unknown. Consequently, we employ the key guess (k) to compute $\mathbf{GG}^k$.

$\vec{\mathbf{1}}$ — This is a target 1D vector of length N_a with all elements set to one (i.e., the label for positive examples). Please note that the $\mathcal{M}_{OvC}$ is binary with a singular output. During the attack phase, the key-dependent part is represented by $\mathbf{GG}^k$ for the hypothesized key guess (k) instead of the classifier labels. As such, for every assumed key guess, we use the probabilities derived directly from this model. To obtain probabilities for each key guess (k), it's necessary to compute $\mathbf{GG}^k$ for each key guess k, then formulate the attack tuple with the corresponding $\mathbf{FG}$ and $\mathbf{T}$ and feed them into the $\mathcal{M}_{OvC}$.

Cumulative sum of log probabilities (S) for $\mathcal{M}_{Stn}$: In the profiling phase, we prepare the standard classifier to estimate the intermediate value target $\mathbf{Y}$, as defined in Eq. (1). In the attack phase, when we feed side-channel measurements corresponding to an undisclosed fixed key into $\mathcal{M}_{Stn}$, it produces a 2D probability vector ($\mathbf{Pr}$) for possible intermediate values, as demonstrated at the start of Eq. (7). Assuming a key guess k, the intermediate value target vector is labeled $\mathbf{Y}^k$. Extracting from vector $\mathbf{Pr}$ using $\mathbf{Y}^k$ (specifically, $\mathbf{Pr}_{\mathrm{i},\mathbf{Y}_i^k}$ for the i^{th} measurement), we gather probabilities for the key guess k, providing a measure of the model's confidence about a particular key usage. Then, the log of these probabilities is typically computed, followed by the aggregation of log probabilities across multiple side-channel measurements, as seen in the lower part of Eq. (7). This leverages the statistical independence of different side-channel measurements, translating multiplication in probabilities into addition in the logarithmic domain – a more computationally efficient and numerically stable process. The key guess boasting the largest sum of log probabilities – or the highest rank – is then deemed the most likely correct key. To track the progress of the correct key rank over N_a measurements, where the contribution of each measurement is incrementally added, we compute and save the cumulative sum. The following equation showcases the computation of the cumulative sum of log probabilities over N_a measurements for the standard classifier, given that the key guess is k.

$$\begin{aligned} &\mathbf{Pr}_i = \mathcal{M}_{Stn}\Big(\mathbf{T}_i\Big) && \forall i \in \{1, \cdots, N_a\}, \\ &\mathbf{S}_{j,k} = \sum_{i=1}^{j} log\Big(\mathbf{Pr}_{\mathrm{i},\mathbf{Y}_i^k}\Big) && \forall j \in \{1, \cdots, N_a\} \textit{ and } \forall k \in \{0, \cdots, N_k - 1\}, \end{aligned} \tag{7}$$

with:

Pr — This denotes a 2D probability vector with dimensions $N_a \times N_k$, produced by the standard classifier ($\mathcal{M}_{Stn}$). It's important to note that this 2D vector encompasses probabilities for all key guesses. In order to extract probabilities for a specific key guess (k), it's required to index this 2D vector with the target label $\mathbf{Y}^k$.

S — This represents a 2D vector with dimensions $N_a \times N_k$, that stores the cumulative sum of log probabilities from the standard classifier for all potential key guesses.

Cumulative sum of log probabilities (†S) for $\mathcal{M}_{OvC}$: Contrasting with the standard classifier, the guessed key does not directly influence the labels in the OvC classifier. Instead, it influences the guess-gram, which also serves as one of the inputs for $\mathcal{M}_{OvC}$. With an assumed key guess k in mind, we calculate the corresponding $\mathbf{GG}^k$ and create an attack tuple for side-channel measurements tied to an undisclosed fixed key. Introducing this attack tuple for key guess k into $\mathcal{M}_{OvC}$ results in a direct yield of a 1D probability vector ($^\dagger\mathbf{Pr}^k$), as indicated in the initial segment of Eq. (7). Subsequently, we typically compute the log of these probabilities and aggregate the log probabilities across various side-channel measurements, as displayed in the lower part of Eq. (8). The key guess with the largest sum of log probabilities - or the highest rank - is then considered the most probable correct key. To monitor the correct key rank's development over N_a measurements, with incremental additions from each measurement, we compute and store the cumulative sum. The following equation demonstrates this cumulative sum's calculation over N_a measurements for the OvC classifier, assuming the key guess is k.

$$\begin{aligned}
{}^{\dagger}\mathbf{Pr}_i^k = \;& \mathcal{M}_{OvC}\Big(\big(\mathbf{GG}_{i,1}^k, \cdots, \mathbf{GG}_{i,P}^k, \mathbf{FG}_{i,1}, \cdots, \mathbf{FG}_{i,Q}\big), \mathbf{T}_i\Big) \\
& \forall i \in \{1, \cdots, N_a\} \; and \; k \in \{0, \cdots, N_k - 1\}, \\
{}^{\dagger}\mathbf{S}_{j,k} = \;& \sum_{i=1}^{j} log\Big({}^{\dagger}\mathbf{Pr}_i^k\Big) \quad \forall j \in \{1, \cdots, N_a\} \; and \; \forall k \in \{0, \cdots, N_k - 1\},
\end{aligned} \tag{8}$$

with:

${}^{\dagger}\mathbf{Pr}^{\boldsymbol{k}}$ — Represents a 1D probability vector with a length of N_a, generated by the OvC classifier ($\mathcal{M}_{OvC}$). Note that it only contains probabilities for specified key guess (k), and this process needs to be repeated N_k times to acquire probabilities for all possible key guesses.

${}^{\dagger}\mathbf{S}$ — This represents a 2D vector with dimensions $N_a \times N_k$, that stores the cumulative sum of log probabilities from the OvC classifier for all potential key guesses.

Metric $\boldsymbol{T_{GE0}}$: T_{GE0} is a commonly used metric in side-channel attacks, indicating the necessary number of side-channel measurements to achieve a guessing entropy GE of zero. GE denotes the average ranking of the secret key (k^*) across a multitude of experiments. In this study, we conducted 100 separate experiments using unique sets of N_a measurements. For each trained model ($\mathcal{M}$), we employed these sets to compute 100 distinct cumulative sums of log probabilities, as denoted by $\mathbf{S}$ and ${}^{\dagger}\mathbf{S}$ in Eqs. (7) and (8) for the standard and OvC classifiers, respectively. Within each cumulative sum of log probabilities, each i^{th} row represents the probability of a particular guessed key being correct given i attack measurements. Knowing the secret true key k^* allows us to position that key in the sorted i^{th} row, which indicates the rank of the real key when i attack measurements are used. We further average the rank of the secret key (k^*) over multiple experiments (about hundred times) where we use a different set of attack measurements per experiment, which gives us GE. Consequently, T_{GE0} can be conceptualized as any minimum value for i ($\leq N_a$) for which the probability of k^* is highest, meaning $GE = 0$. If, after all N_a attack measurements, k^* fails to attain the highest rank, we consider the attack as failed.

4 Results

This section presents the results from our study, split into four parts: a review of the datasets we utilized, the choices made in the network design of our OvC classifier, the results for the real datasets, and finally, the results for the simulated datasets.

4.1 Datasets

To present the datasets that we used for our experiments, we need to introduce the following notations:

`SBO` = S-Box[`PLAINTEXT` $\oplus$ `KEY`] (also referred to as S-Box Output)

`UI` = *`SBO`* (unprotected implementation to be simulated which is devoid of masking countermeasure)

`MI` = `MASK` (mask implementation to be simulated which is used by the masking countermeasure)

`PI` = *`SBO`* $\oplus$ `MASK` (protected implementation to be simulated which is inclusive of a masking countermeasure)

Unless otherwise mentioned, *SBO* is chosen as the f_{iv} to generate labels as per Eq. (1) for profiling and attack across all the datasets. Meanwhile, *UI*, *PI*, and *MI* are the targets used by the simulated datasets to simulate leakages.

4.1.1 Real Datasets

For our real dataset experiments, we employ both ASCAD datasets. The first, ASCAD-V1, is detailed in [BPS+20] and captures measurements from an ATMega8515 executing a first-order Boolean-masked AES implementation. It is available in two distinct variants: one with a fixed key and another with variable keys. In contrast, ASCAD-V2, introduced in [MS23], provides enhanced security over ASCAD-V1. Derived from readings of an STM32 running an affine-masked and shuffled AES implementation, this latter version exclusively features a variable key setup.

To keep the number of model parameters reasonable, the authors of the ASCAD dataset chose to use a limited number of trace samples. Nonetheless, studies [LZC+21, MBC+20] have shown that using entire raw traces is more conducive to achieving the best results. However, these complete traces necessitate a greater number of network parameters and extended training durations. Therefore, Perin et al. [PWP22] turned to subsampling, especially for datasets with a sufficiently high Signal-to-Noise Ratio (SNR) and minimal translation invariance. This is particularly relevant for ASCAD-V1 datasets, prompting us to explore both limited sample range and full raw traces with subsampling. In the case of the ASCAD-V2 dataset, using the complete traces becomes challenging due to their huge sample size, at about 1 000 000. Our subsampling efforts for these were not successful, leading us to rely solely on the limited 15,000 samples, as mentioned in [MS23]. This study uses five real-world datasets that are primarily derived from the ASCAD-V1 and ASCAD-V2 datasets, the details of which are as follows.

ASCAD-V1-FK: The fixed-key variant of the ASCAD-V1 dataset has 100,000 samples per trace. For this dataset, we selected 700 samples that contain information on the first round of S-box (*SBO*) processing for the third byte, as recommended in [BPS+20]. We use a 'desync=0' version, that is, traces with no simulated desynchronization. Besides, we use 50,000 traces for profiling, with a 90%–10% split for training and validation. For the attack, we use 10,000 traces. Note that this dataset uses the same fixed key for the profiling and attack phases.

ASCAD-V1-FK-F: All the settings for this dataset are the same as for ASCAD-V1-FK. The only difference is that we consider the entire raw trace and perform average subsampling with a window of size 40 and a 50% overlap. This reduces the sample size from 100,000 to 5,000.

ASCAD-V1-VK: The variable-key variant of the ASCAD-V1 dataset has 250,000 samples per trace. For this dataset, we selected 1400 samples that contain information on the first round of S-box (*SBO*) processing for the third byte, as recommended in [BPS+20]. We use a 'desync=0' version as previously, and we employ 200,000 traces for profiling, with a 90%–10% split for training and validation. Although there are 100,000 traces available for an attack, we use only 10,000 in the attack phase.

ASCAD-V1-VK-F: All the settings for this dataset are the same as for ASCAD-V1-VK. The only difference is that we consider the entire raw trace and perform average subsampling with a window of size 40 and a 50% overlap. This reduces the samples from 250,000 to 12,500.

ASCAD-V2: While the initial raw dataset comprises 1,000,000 samples, we adhere to the reduced sample set recommended in [MS23], bringing it down to 15,000 samples for each trace. Our analysis targets the first key byte. As previously, we split the 500,000 traces available for profiling 90%–10% between training and validation; in the attack phase, we use 10,000 traces.

4.1.2 Simulated Datasets

Within actual public datasets gathered from real devices, the precise nature of the leakage remains uncertain. Thus, we introduce simulated datasets that allow us to evaluate targets with known leakages paired with noise that we can control within the readings. Two distinct versions of these simulated datasets exist: unprotected and protected, utilizing *SBO* as a target. For a successful attack, it becomes necessary to simulate leakages in our readings correlated with this target. For an unprotected version, the simulation of leakage is solely tied to *UI* (identical to *SBO*), while simulating leakages for *PI* and *MI* is unnecessary (as they are related to protected implementations). As exhibited in the first column of Table 3, only the leakage related to *UI* is simulated for a particular choice of f_{lk}. In contrast, for the protected variant, the simulated leakages of both *PI* and *MI* must be present for the evaluation, as it is the protected implementation that is subject to leakage. The first columns of Table 2 show which of *PI* and/or *MI* are simulated for a specific choice of f_{lk}. Finally, in Tables 2 and 3, "--" symbolizes the absence of leakage for the corresponding target.

In the simulated experiments, we add random noise generated with a normal distribution and a unit standard deviation. This translates to 68.2% of data points within each cluster not overlapping upon the adjacent cluster. As a final step, we perform standard normalization across each dimension, thus ensuring a mean of zero and a unit variance. It is important to recognize that the dimensionality of a measurement can be either one or two, as the leakages do not simultaneously involve all three targets. For unprotected implementations, the leaks relate solely to *UI*, while for protected ones, they concern either *PI* or a combination of *PI* and *MI*.

4.2 Results for Real Datasets

In this section, we report results for publicly available datasets. We evaluate three architectural setups: the standard classifier ($\mathcal{M}_{Stn}$), the multitask classifier ($\mathcal{M}_{MTask}$), and our newly introduced OvC classifier ($\mathcal{M}_{OvC}$). For all datasets, $\mathcal{M}_{MTask}$ results are possible to obtain, but we only conducted such experiments for ASCAD-V2 due to additional work of finding suitable network architectures. Thus, $\mathcal{M}_{MTask}$ architecture results for the other datasets are not available (NA) in Table 1.

All results feature the *Acc* metric, which represents the highest observed validation accuracy during profiling. First, for the standard classifier, *Acc* is calculated for labels derived following Eq. (1), with f_{iv} = *SBO* (as it is S-Box Output) and f_{lk} = ID. Second, for the multitask classifier, *Acc* is displayed as "<*Acc* for primary task>/<*Acc*'s for secondary tasks>". The primary task mirrors the standard classifier, and secondary tasks are intended to learn masks or other beneficial information that can enhance the primary task's learning. In the case of ASCAD-V2 dataset, the secondary task is to learn $\mathsf{r_m}$, β, and $\mathsf{p}[i]$ as suggested in [MS23]. Lastly, for the OvC classifier, the reported *Acc* metric represents the binary accuracy. In addition, we also report T_{GE0}, as defined in Section 3.4.2 for both the standard and OvC classifiers. Conversely, when it comes to a multitask classifier, the computation of T_{GE0} parallels that of a standard classifier, taking advantage of probabilities derived from the primary task, which uses *SBO* as labels.

Reviewing the data presented in Table 1 provides the following insights. In the case of the ASCAD-V1-FK dataset, the OvC classifier consistently outperforms the standard

Table 1: Results for protected real datasets

Dataset	*Acc*			T_{GE0}		
	$\mathcal{M}_{Stn}$	$\mathcal{M}_{MTask}$	$\mathcal{M}_{OvC}$	$\mathcal{M}_{Stn}$	$\mathcal{M}_{MTask}$	$\mathcal{M}_{OvC}$
`ASCAD-V1-FK`	0.009	NA	1.000	87 [PWP22]	NA	1
`ASCAD-V1-VK`	0.009	NA	0.721	78 [WPP22]	NA	61
`ASCAD-V1-FK-F`	0.972	NA	1.000	1 [PWP22]	NA	1
`ASCAD-V1-VK-F`	1.000	NA	1.000	1 [PWP22]	NA	1
`ASCAD-V2`	0.004	0.016/*	0.742	✗ [MS23]	60 [MS23]	47

$* = (0.992, 0.211, 0.889)$: accuracies for $(\mathsf{r_m}, \beta, \mathsf{p}[i])$ respectively as reported in [MS23].
NA: results are not available

classifier. However, the advantage narrows for the `ASCAD-V1-VK` dataset, with only a slight edge. When examining the `ASCAD-V1-FK-F` and `ASCAD-V1-VK-F` datasets, which encompass full traces (possible due to subsampling strategy), both the standard classifiers and the OvC classifier have same performance leading to a successfull attack with one trace. This disparity implies that datasets like `ASCAD-V1-FK` and `ASCAD-V1-VK` being limited by the number of samples used, may omit critical data found in their full-version counterparts.

Turning to the `ASCAD-V2` dataset, the standard classifier's shortcomings become evident. In comparison, while the multitask classifier necessitates 60 traces for a successful attack, the OvC classifier improves the attack, achieving success with only 47 traces. The full-version `ASCAD-V1` datasets, due to the benefit of subsampling, may come across as less intimidating than the `ASCAD-V2` ones. Still, it is clear that the original `ASCAD-V1` datasets, which used limited samples to train the neural network models as described in [BPS+20], remain quite challenging. This observation hints at the potential benefit of expanding the `ASCAD-V2` dataset beyond its current 15,000 samples. Our present subsampling methods on `ASCAD-V2` raw traces have not been fruitful, indicating a notable loss of information. Therefore, it becomes crucial that more efforts be made to identify more potent data points or samples for the `ASCAD-V2` dataset.

4.3 Results for Simulated Datasets

In this section, we present an analysis of the results obtained from simulated datasets. We commence our discussion with Table 2, which outlines the results for protected simulated datasets. The format of the results in Table 2 aligns with that of Table 1. However, in the context of the $\mathcal{M}_{MTask}$ model applied to the protected simulated dataset, there exists a solitary secondary task, namely *`MI`*. Hence, we showcase *Acc* as "<*Acc* for *`SBO`*>/<*Acc* for *`MI`*>" for the $\mathcal{M}_{MTask}$ model targeting the protected simulated dataset. It is noteworthy that, just as in Table 1, all the results discussed here rely on label utilization where f_{lk} is set to `ID`. To recap, for simulated datasets, the targets *`UI`*, *`PI`*, and *`MI`* (italicized) undergo leakage simulation in side-channel measurements, while *`SBO`* serves as the training target for our neural network models. Notably, for the protected simulated datasets, we refrain from simulating leakage for *`UI`*; instead, leakage is simulated for *`PI`* and optionally for *`MI`*. The insights drawn from Table 2 are as follows:

1. In protected setups, when information leakage linked to masking strategies, exemplified by *`MI`*, is absent, all three attack models fail to target *`SBO`*. This observation is clear from the first three rows of the table, which indicate the ineffectiveness of attack strategies when no simulated leakage for *`MI`* is present. However, successful attacks are possible when both *`PI`* and *`MI`* leakages are present in side-channel evaluations, as evidenced by the results in the last four rows. This underscores that the target like *`SBO`* (which

does not leak in side-channel measurements for protected implementations, i.e., *UI*), can lead to successful attacks if both the protected target (*PI*) and the associated protection-based masks (*MI*) exhibit leakage.

2. Focusing on the fourth and fifth rows, it is evident that a higher number of traces are required for a successful attack under `HW` leakage simulation compared to `ID` leakage simulation. This discrepancy arises due to the presence of 256 clusters simulated within samples under the `ID` leakage model, facilitating easier correlation with the intended target. In contrast, the `HW` leakage model simulates only nine clusters within samples, making it more challenging to correlate with a target featuring 256 potential parts. This suggests that the datasets where leakages can correlate to each part of the intended target uniquely need fewer traces for successful attack.

3. The OvC classifier consistently outperforms both the standard and multitask classifiers, especially when dealing with noisy protected simulated datasets. Moreover, the multitask classifier slightly edges out the standard classifier in performance.

Table 2: Results for protected simulated datasets

Leakage			*Acc*			T_{GE0}		
UI	*PI*	*MI*	$\mathcal{M}_{Stn}$	$\mathcal{M}_{MTask}$	$\mathcal{M}_{OvC}$	$\mathcal{M}_{Stn}$	$\mathcal{M}_{MTask}$	$\mathcal{M}_{OvC}$
--	ID	--	0.005	0.005/0.005	0.509	✗	✗	✗
--	HW	--	0.006	0.005/0.006	0.509	✗	✗	✗
--	L3B	--	0.006	0.005/0.005	0.508	✗	✗	✗
--	ID	ID	0.113	0.071/0.654	0.962	4	4	4
--	HW	HW	0.014	0.015/0.037	0.851	13	13	11
--	L3B	HW	0.009	0.009/0.038	0.761	80	65	57
--	HW	L3B	0.009	0.009/0.037	0.758	71	66	56

For completeness, we present in Table 3 results on unprotected simulated datasets. Note that we omit to report results for the $\mathcal{M}_{MTask}$ model, as its viability hinges on the availability of secondary tasks related to the employed masking scheme. Nevertheless, the $\mathcal{M}_{OvC}$ model can be applied to unprotected datasets. The absence of secondary tasks simply means skipping the use of PTNN blocks. This results in using an architecture similar to Figure 3 instead of Figure 4. Additionally, we present findings for $\mathcal{M}^{\dagger}_{Stn}$, which is similar to the $\mathcal{M}_{Stn}$ classifier, the only difference being that it uses labels generated by applying the same leakage model used for simulating the *UI* leakage in side-channel measurements to *SBO*. In other words, $\mathcal{M}^{\dagger}_{Stn}$ shows the performance of an attacker using $\mathcal{M}_{Stn}$ exactly tailored to the underlying leakage model of the target device (an ideal case). We highlight two points:

1. In terms of attack results, the OvC classifier matches the results of the ideal case of a standard classifier with a known leakage model while not making any assumption on the leakage model. This suggests that the OvC design is more appropriate to solve the task at hand where the real leakage model is unknown than the standard classifier design.

2. The accuracy of the $\mathcal{M}^{\dagger}_{Stn}$ increases when the number of classes decreases. This is expected since designing a multi-class classifier for a large number of classes (256 in the case of `ID`) poses challenges in maintaining balanced representation and high accuracy [DHS01,Bis07]. However, note that the opposite is true for $\mathcal{M}_{OvC}$: it achieves near-to-one accuracy in the `ID` leakage model, while the accuracy drops to about 0.75

with the binary `LSB` leakage model. We believe that this behavior is inherent to the design of the OvC classifier, as *"Chaos"* examples used in the training phase can be accidentally equal to *"Order"* ones with a probability inversely proportional to the number of leakage classes.

Table 3: Results for unprotected simulated datasets

Leakage			*Acc*			T_{GE0}		
UI	*PI*	*MI*	$\mathcal{M}_{Stn}$	$\mathcal{M}^{\dagger}_{Stn}$	$\mathcal{M}_{OvC}$	$\mathcal{M}_{Stn}$	$\mathcal{M}^{\dagger}_{Stn}$	$\mathcal{M}_{OvC}$
ID	--	--	0.884	0.884	0.958	5	3	2
HW	--	--	0.037	0.954	0.923	20	8	7
L3B	--	--	0.034	0.940	0.900	15	4	5
LSB	--	--	0.010	0.957	0.751	32	13	13

5 Discussion

In this section, we delve into the merits of the OvC classifier and reference relevant reported results to support our assertions.

5.1 First Language Model-based Approach

Traditionally, techniques for side-channel attacks revolve around modified versions of the original TA or supervised learning-based classifier strategies. However, acknowledging the influential strides of NLP techniques in advancing other domains within machine learning and artificial intelligence (elaborated in Section 1), we introduce a novel approach. This approach brings language models to the forefront of side-channel analysis.

While analyzing an intermediate value for profiling, conventional methods like TA or supervised learning classifiers often overlook the rich knowledge embedded in other data tokens, such as plaintext, ciphertext, key, masks, and other potential intermediate values. In contrast, both the multitask classifier and the OvC classifier are adept at harnessing this additional knowledge. Specifically, the OvC classifier achieves this by creating an n-cryptogram that feeds into the language model block. Importantly, if we want to increase the number of targets, OvC classifiers demonstrate superior scalability compared to multitask classifiers — a topic further delved into in Section 5.4. Hettwer et al.'s work [HGG18] similarly aims to leverage this knowledge. However, the distinctions between our approach and theirs are detailed in Section 2.

What distinguishes the OvC classifier from other multi-target strategies (like the multitask classifier) is its foundation on language models. It can learn the contextual relationships between multiple targets, constructing a joint distribution over n-cryptograms. Given their training to understand and predict word or token sequences, language models intrinsically grasp the context binding these elements. On the other hand, multitask classifiers are designed to solve multiple classification problems simultaneously but do not inherently capture the sequential or contextual relationships between inputs unless explicitly designed to do so (e.g., combining with architectures like LSTMs) [DCLT19].

5.2 A Multitask and Multimodal Approach

Language models inherently function as unsupervised multitask learners [RWC^{+}19]. By employing n-cryptogram in training the language model within the OvC classifier, every gram in n-cryptogram takes on a task role, making our method inherently multitask.

Multitask learning is a technique that enhances generalization by utilizing domain-specific knowledge present in the training signals of related tasks as an inductive bias. By learning tasks simultaneously using a shared representation, it can improve the learning of each task through knowledge gained from other tasks [Car98]. Therefore, in the context of side-channel attacks, learning multiple relevant intermediate-value targets, plaintext, key, and masks concurrently can improve the performance of each individual task.

Additionally, as indicated in Section 3.2.5, our methodology is multimodal. Multimodal learning offers several advantages over unimodal learning. By integrating information from multiple sources, it can enhance the accuracy and robustness of machine learning models, particularly in complex and uncertain environments. It can also be beneficial in cases where a single modality may be inadequate or unreliable, such as in noisy or ambiguous data. Furthermore, multimodal learning can facilitate more efficient and effective learning by reducing the amount of data required for training and enabling transfer learning across different tasks and domains [SS14]. Therefore, in the context of side-channel attacks, if the labels are ambiguous or uncertain, or if the measurements do not precisely reflect the leakage corresponding to the label, multimodal learning can be advantageous.

5.3 Getting Away from Assuming the Leakage Model Function (f_{lk})

For side-channel attacks, assigning appropriate labels to measurements is vital. Typically, labels for such attacks are determined by Eq. (1), dependent on two critical components built on assumptions: the intermediate-value generator function (f_{iv}) and the leakage model function (f_{lk}). In real-world applications, f_{lk} choice hinges on the dataset and f_{iv} assumptions, mandating heuristic-based or experimental selection. Although the identity (`ID`) function is the go-to for $\mathcal{M}_{Stn}$, alternate f_{lk} choices have also been explored [Tim19, WPP22, PWP22, KWPP22]. Notably, there is no conclusive evidence suggesting $f_{lk} = \texttt{ID}$ outperforms other options for a standard classifier. However, the OvC classifier, without the knowledge of an explicit simulated leakage model, showcases performance akin to the standard classifier equipped with knowledge of an explicit simulated leakage model (denoted as $\mathcal{M}^{\dagger}_{Stn}$). This adaptability shown by OvC classifiers indicates the possibility of bypassing f_{lk} choices. This finding can benefit researchers seeking to narrow down the search for label-generating strategies for side-channel attacks.

Our observation in Section 4.3 (the third observation for Table 3) alludes to the potential of bypassing f_{lk} assumptions. While these insights originate from simulated, unprotected datasets and call for further exploration on real datasets, it is essential to interpret these results with caution. Due to the unknown nature of leakage models in real datasets, framing the experiments posed challenges, which led us to use simulated datasets. Similarly, Wu et al. [WAR$^+$23] emphasized the importance of removing leakage model assumptions while offering experiments on real datasets. They conducted experiments using the same neural network architectures across various leakage models for a fair comparison, demonstrating that their multi-bit approach outperforms others. In future work, we aim to explore using analogous or enhanced approaches to evaluate our approach on real datasets. However, currently, our conclusions pertain solely to simulated datasets.

5.4 Address Problems with Multitask Classifiers

In the introduction, we highlighted problems with multitask classifiers. Next, we will discuss how they could be addressed. Both the multitask classifier and the OvC classifier aim to leverage additional information such as masks, permutation indices, various intermediate values, and data tokens like key, plaintext, or add-round-key. The OvC classifier employs a single binary head, whereas the multitask classifier requires a separate head for each task. As the number of targets rises, the multitask classifier faces scalability challenges due to its multiple heads. In contrast, the OvC classifier's design, with its singular head, offers scalability, allowing for the addition of more interesting targets. Next, we dive deeper into how parameter complexity is reduced, how the balancing learning across tasks is achieved, and how the hyperparameter tuning complexity gets reduced.

Parameter complexity: The number of targets (N_t) directly influences the parameters used in the networks we are considering. When comparing parameter needs between the multitask and OvC classifiers, we can simplify by breaking down the parameter complexity discussion into three parts. First, both the multitask classifier's common trunk and the OvC classifier's preprocessor block have the same structure. The parameters they require scale linearly with the sample count (N_s from side-channel measurement, leading to a complexity of $\mathcal{O}(N_s)$ for both. Second, the parameters of the language model block scale with the target count N_t, resulting in a complexity of $\mathcal{O}(N_t)$, while the multitask classifier lacks the language model block. Third, when considering the multitask classifier's heads and the OvC classifier's deep metric learning block, we can assume that the output size for both the multitask classifier's common trunk and the OvC classifier's preprocessor block is a constant N_o. If each head in the multitask classifier and deep metric learning block in the OvC classifier share the same structure, their complexity is $\mathcal{O}(N_o)$. The multitask classifier has N_t heads, leading to $\mathcal{O}(N_t \times N_o)$ complexity, while the OvC remains at $\mathcal{O}(N_o)$. Summing it up, the multitask classifier's complexity is $\mathcal{O}(N_s + (N_t \times N_o))$, and the OvC classifier has a complexity of $\mathcal{O}(N_s + N_t + N_o)$.

For clearer notation, let us omit the constant term N_s, which dictates the size of either the multitask classifier's common trunk or the OvC classifier's preprocessor block. It is essential to understand that N_o is not constant for the OvC classifier and is influenced by N_t, resulting in an OvC classifier complexity of $\mathcal{O}(2N_t)$. For a multitask classifier, N_o can be assumed to be constant if it is large enough, or else it should scale with N_t. Having a large N_o is vital for a larger N_t to avoid information bottlenecks, as it might not carry enough information for all the upstream heads. Moreover, it provides flexibility and can capture the nuances of more complex and diverse tasks. Thus, the multitask classifier's complexity could be either $\mathcal{O}(N_t \times N_o)$ or $\mathcal{O}(N_t^2)$, contingent on how N_o is interpreted. In conclusion, this analysis suggests that the number of parameters needed by the OvC classifier is less than that needed by the multitask classifier, especially when the number of targets (N_t) is large.

Balanced learning across tasks: In the context of side-channel attacks, multitask classifiers presented in [MS23] employ multiple targets during training. These targets are treated with equal weight during loss backpropagation. Yet, this strategy may not always be optimal, as highlighted in [Car97, Rud17], primarily due to two reasons. The first concern is the risk of negative transfer: when tasks are not closely related or exhibit negative correlation, shared representations might deteriorate performance rather than enhance it. The second challenge arises in optimization: managing multiple loss functions proves difficult. Should one task dominate others due to its scale or inherent difficulty, it might slow down the learning of the remaining tasks.

In contrast, the OvC classifier inherently addresses these concerns. Each target undergoes a transformation into a uniform, dense input feature vector thanks to the embedding layers. During backpropagation, the unified binary task evaluates these input features via the language model block. Over multiple training iterations, it can prioritize each target based on its influence on the classification loss. In essence, by training collaboratively on dense target representations and corresponding side-channel measurements, each target gets an implicit weight.

Hyperparameter tuning complexity: In the domain of machine learning, using multitask classifiers with multiple heads, each having its own unique loss function, often leads to extended learning periods. This subsequently increases the model search time, given the need to navigate a myriad of hyperparameters. Each head contributes its own hyperparameters, ranging from the number of layers and units within those layers to distinct loss functions, optimizers, etc. All these elements compound to expand the hyperparameter search space, intensifying the optimization's duration and computational demands. On the other hand, employing a singular-head strategy, as exemplified by the OvC classifier, significantly narrows down the hyperparameter tuning landscape. This streamlined approach also tends to improve model convergence due to the reduced parameter complexity. We have detailed our network choices in Appendix A.

6 Conclusion

We have introduced the OvC classifier, a novel approach merging principles from NLP and deep metric learning to execute side-channel attacks. This classifier is a notable alternative to multitask classifiers, especially when handling multiple targets and executing masking scheme-aware profiling. Drawing strengths from both the multitask and multimodal learning paradigms, our proposed architecture holds several advantages. Leveraging an integrated language model allows it to learn contextual relationships between numerous targets. This approach relaxes assumptions on the leakage model and narrows the search for possible targets to be used for side-channel attacks. Additionally, its single-head approach with unified and simple loss minimizes parameter requirements, ensures balanced learning across multiple tasks, and shrinks the hyperparameter exploration space. Finally, our approach is scalable if needed to add more targets to the attack.

However, every advantage comes with its own set of challenges. Our model's reliance on PTNN blocks, geared towards deciphering masking scheme targets, requires an initial pre-training phase before training the OvC classifier. Eliminating this pre-training step is one of our future goals. Additionally, our observation that assumptions related to f_{lk} can be relaxed is derived from simulated datasets, not real-world ones, primarily because simulated datasets offer more control over pertinent leakages. This finding warrants further examination using real datasets, a direction we plan to undertake in subsequent research.

References

[BAM19] Tadas Baltrusaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy. IEEE Trans. Pattern Anal. Mach. Intell., 41(2):423–443, 2019.

[BCV13] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell., 35(8):1798–1828, 2013.

[BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. J. Mach. Learn. Res., 3:1137–1155, 2003.

[BGL$^+$93] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a siamese time delay neural network. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993], pages 737–744. Morgan Kaufmann, 1993.

[BHS13] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. CoRR, abs/1306.6709, 2013.

[Bis07] Christopher M. Bishop. Pattern Recognition and Machine Learning, 5th Edition. Information Science and Statistics. Springer, 2007.

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

[BZMA20] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. Wav2vec 2.0: A framework for self-supervised learning of speech representations. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual, 2020.

[Car97] Rich Caruana. Multitask learning. Mach. Learn., 28(1):41–75, 1997.

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

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, volume 10529 of Lecture Notes in Computer Science, pages 45–68. Springer, 2017.

[Chi21] Davide Chicco. Siamese neural networks: An overview. In Hugh M. Cartwright, editor, Artificial Neural Networks - Third Edition, volume 2190 of Methods in Molecular Biology, pages 73–94. Springer, 2021.

[CLH$^+$23] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms. CoRR, abs/2302.06675, 2023.

[CMS+20] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I, volume 12346 of Lecture Notes in Computer Science, pages 213–229. Springer, 2020.

[CMS+22] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022, pages 1280–1289. IEEE, 2022.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers, volume 2523 of Lecture Notes in Computer Science, pages 13–28. Springer, 2002.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics, 2019.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern Classification, 2nd Edition. Wiley, 2001.

[DWH+15] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers, pages 1723–1732. The Association for Computer Linguistics, 2015.

[GB16] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. CoRR, abs/1604.06737, 2016.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. Deep Learning. Adaptive Computation and Machine Learning. MIT Press, 2016.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings, volume 2162 of Lecture Notes in Computer Science, pages 251–261. Springer, 2001.

[HA15] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In Aasa Feragen, Marcello Pelillo, and Marco Loog, editors, Similarity-Based Pattern Recognition - Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015, Proceedings, volume 9370 of Lecture Notes in Computer Science, pages 84–92. Springer, 2015.

[HCL06] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA, pages 1735–1742. IEEE Computer Society, 2006.

[HGG18] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Profiled power analysis attacks using convolutional neural networks with domain knowledge. In Carlos Cid and Michael J. Jacobson Jr., editors, Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers, volume 11349 of Lecture Notes in Computer Science, pages 479–498. Springer, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society, 2016.

[JM23] Daniel Jurafsky and James H. Martin. Speech and Language Processing (3rd Ed. Draft). 2023.

[KB19] Mahmut Kaya and Hasan Sakir Bilge. Deep metric learning: A survey. Symmetry, 11(9):1066, 2019.

[KF17] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. IEEE Trans. Pattern Anal. Mach. Intell., 39(4):664–676, 2017.

[KGA$^+$22] Doyeon Kim, Woonghyun Ga, Pyunghwan Ahn, Donggyu Joo, Sewhan Chun, and Junmo Kim. Global-local path networks for monocular depth estimation with vertical CutDepth. CoRR, abs/2201.07436, 2022.

[KGC18] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pages 7482–7491. Computer Vision Foundation / IEEE Computer Society, 2018.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, volume 1666 of Lecture Notes in Computer Science, pages 388–397. Springer, 1999.

[KJJR11] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. J. Cryptogr. Eng., 1(1):5–27, 2011.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings, volume 1109 of Lecture Notes in Computer Science, pages 104–113. Springer, 1996.

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

[KTS+14] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014, pages 1725–1732. IEEE Computer Society, 2014.

[Kul13] Brian Kulis. Metric learning: A survey. Found. Trends Mach. Learn., 5(4):287–364, 2013.

[KV22] Praveen Kulkarni and Vincent Verneuil. Order vs. Chaos: Multi-trunk classifier for side-channel attack. In Jianying Zhou, Sridhar Adepu, Cristina Alcaraz, Lejla Batina, Emiliano Casalicchio, Sudipta Chattopadhyay, Chenglu Jin, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Weizhi Meng, Stjepan Picek, Jun Shao, Chunhua Su, Cong Wang, Yury Zhauniarovich, and Saman A. Zonouz, editors, Applied Cryptography and Network Security Workshops - ACNS 2022 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Rome, Italy, June 20-23, 2022, Proceedings, volume 13285 of Lecture Notes in Computer Science, pages 218–232. Springer, 2022.

[KWPP22] Maikel Kerkhof, Lichao Wu, Guilherme Perin, and Stjepan Picek. Focus is key to success: A focal loss function for deep learning-based side-channel analysis. In Josep Balasch and Colin O'Flynn, editors, Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings, volume 13211 of Lecture Notes in Computer Science, pages 29–48. Springer, 2022.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. Nat., 521(7553):436–444, 2015.

[LLG+20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020, pages 7871–7880. Association for Computational Linguistics, 2020.

[LMW+22] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022, pages 11966–11976. IEEE, 2022.

[LPMS18] Liran Lerman, Romain Poussier, Olivier Markowitch, and François-Xavier Standaert. Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: Extended version. J. Cryptogr. Eng., 8(4):301–313, 2018.

[LZC+21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2021(3):235–274, 2021.

[MBC+20] Loïc Masure, Nicolas Belleville, Eleonora Cagli, Marie-Angela Cornelie, Damien Couroussé, Cécile Dumas, and Laurent Maingault. Deep learning side-channel analysis on large-scale traces - A case study on a polymorphic AES. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part I, volume 12308 of Lecture Notes in Computer Science, pages 440–460. Springer, 2020.

[MCCD13] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013.

[Mis20] Diganta Misra. Mish: A self regularized non-monotonic activation function. In 31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020. BMVA Press, 2020.

[MO23] Thomas Marquet and Elisabeth Oswald. A Comparison of Multi-task learning and Single-task learning Approaches. IACR Cryptol. ePrint Arch., page 611, 2023.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings, volume 10076 of Lecture Notes in Computer Science, pages 3–26. Springer, 2016.

[MS23] Loïc Masure and Rémi Strullu. Side-channel analysis against ANSSI's protected AES implementation on ARM: End-to-end attacks with multi-task learning. J. Cryptogr. Eng., 13(2):129–147, 2023.

[NJ01] Andrew Y. Ng and Michael I. Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada], pages 841–848. MIT Press, 2001.

[NKK+11] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In Lise Getoor and Tobias Scheffer, editors, Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011, pages 689–696. Omnipress, 2011.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A Meeting of SIGDAT, a Special Interest Group of the ACL, pages 1532–1543. ACL, 2014.

[PWP22] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2022(4):828–861, 2022.

[RN20] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach (4th Edition). Pearson, 2020.

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

[RWC$^+$19] Alec Radford, Jeff Wu, Rewon Child, D. Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. In Technical Report, OpenAi, 2019.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2021(3):677–707, 2021.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings, volume 3659 of Lecture Notes in Computer Science, pages 30–46. Springer, 2005.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings, volume 5479 of Lecture Notes in Computer Science, pages 443–461. Springer, 2009.

[SS14] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep Boltzmann machines. J. Mach. Learn. Res., 15(1):2949–2980, 2014.

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2019(2):107–131, 2019.

[Vap98] Vladimir Vapnik. Statistical Learning Theory. Wiley, 1998.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017.

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

[WAR$^+$23] Lichao Wu, Amir Ali-pour, Azade Rezaeezade, Guilherme Perin, and Stjepan Picek. Breaking Free: Leakage Model-free Deep Learning-based Side-channel Analysis, 2023.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. IACR Cryptol. ePrint Arch., page 1293, 2020.

[WPP22] Lichao Wu, Guilherme Perin, and Stjepan Picek. The best of two worlds: Deep learning-assisted template attack. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2022(3):413–437, 2022.

[ZBC$^+$23] Gabriel Zaid, Lilian Bossuet, Mathieu Carbone, Amaury Habrard, and Alexandre Venelli. Conditional variational AutoEncoder based on stochastic attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2023(2):310–357, 2023.

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

[ZY22] Yu Zhang and Qiang Yang. A survey on multi-task learning. IEEE Trans. Knowl. Data Eng., 34(12):5586–5609, 2022.

A Network Design Choices

We elaborate in this section on the various network design choices for the OvC classifier. Note that we have not performed hyperparameter optimizations, and most of the parameters are heuristically selected.

A.1 Language Model Block

The input dimension of the language model block depends on the number of data tokens in n-cryptogram and the output size of PTNNs. It is essential to note that PTNNs are geared towards learning targets emerging from the specific protection scheme in use, such as masks and permutation indices, among others. Illustratively, in Figure 4, the size of the n-cryptogram is four where the number of fix-gram tokens (Q) and guess-gram tokens (P) are two. Every data token in n-cryptogram is an integer that is transformed into a dense vector of dimension five using an embedding layer. Thus, the input size due to n-cryptogram equates to 4×5, totaling 20. The subsequent input comprises the output from a pair of PTNN blocks. Assuming these blocks are trained for 8-bit masks, they yield 256 outputs. Thus, the two PTNN blocks collectively account for 512 input dimensions. Consequently, the aggregate input dimension for the language model block becomes 20 + 512, resulting in 532. This aforementioned input size relates specifically to the figure and might vary based on the dataset in question.

The number of grams in n-cryptogram for fixed key datasets we are considering is two, with $Q = 1$ symbolizes plaintext, and $P = 1$ signifies *SBO*. In contrast, for variable-key ASCAD-V1 datasets, there is the added advantage of leveraging key-related data and using it as a guess-gram. This leads to a n-cryptogram size of three, where $P = 2$. This results in input sizes of 10 (2×5) and 15 (3×5), respectively. The subsequent input comprises the output from the PTNN blocks. For ASCAD-V1 datasets, we have two PTNN blocks âĂŞ one trained for the state mask r_i for the particular i^{th} byte, and the second for mask r_{out}. The number of unique values for both r_i and r_{out} is 256, resulting in an input size of 512 for two PTNN blocks. Furthermore, for ASCAD-V2 datasets, we have three PTNN blocks designed to learn affine masks and permutation indices $\mathsf{r_m}$, β, and $\mathsf{p}[i]$, as introduced in [MS23]. The number of possible values is 255 for $\mathsf{r_m}$, 256 for β, and 16 for $\mathsf{p}[i]$. The output size of each corresponding PTNN block reflects these values and totals 527. Finally, for the unprotected simulated dataset, we do not use any PTNN blocks while for the protected simulated dataset, we have only one PTNN block for the mask with the output of size 256. In summary, the input size for ASCAD-V1 fixed key datasets is 10 + 512 = 522. For ASCAD-V1 variable key datasets, it is 15 + 512 = 527. For the ASCAD-V2 dataset, which is variable key, it is 15 + 527 = 542. For the unprotected variable key simulated dataset, it is 15. For the protected variable key simulated dataset, it is 15 + 256 = 271.

As can be seen in the language model block of Figure 4, the output from flattened embedding layers and PTNN blocks is concatenated, and the resulting input is fed to the “Dense Layers: B” block and also concatenated to its output via skip layer connections. The “Dense Layers: B” block comprises three dense layers, with the number of units reduced by a fraction of 0.2 from that of the previous layer. If this reduction results in a fractional number, we simply round up to the nearest integer. The first dense layer size is dictated by the dataset used and is specified in the previous paragraph.

A.2 Preprocessor Block

In the preprocessor block, we employ a dense network of six layers. The initial layer comprises units that are 70% the size of the input samples (rounded up if necessary). Following this, we have five layers, each packed with 512 units. However, for simulated datasets, where the sample points per trace can be just one or two (which is relatively low), we adapt and equip all six layers with 10 units each.

Subsequent to this, we introduce an additional dense layer. The unit count in this layer is adjusted to correspond with the output from the language model block. The design of the language model block and the selected dataset dictate this number. Considering the three layers in the language model block and noting that each subsequent layer contains 0.8 times the units of the preceding one, the output size of the “Dense Layers: B” is 0.64 times the initial input size. We then round this number up if necessary. It is also vital to factor in the units originating from the skip layer connections. Therefore, if we label the input size as N_{in}, the output size of the language model block is $N_{in} + round(0.64 \times N_{in})$. Based on this formula, the resulting number of units in the seventh dense layer for `ASCAD-V1` fixed key datasets is 856. For `ASCAD-V1` variable key datasets, it is 864. For the `ASCAD-V2` dataset, it is 889. For the unprotected variable key simulated dataset, it is 25. For the protected variable key simulated dataset, it is 444.

A.3 Deep Metric Learning Block

In the deep metric learning block, the “Dense Layers: A” comprises three dense layers. The initial input layer size is determined as 80% of the output from the language model block. Successively, each of these three layers is reduced by a factor of 0.2 compared to its predecessor. The network’s weights are shared across both modalities. Deep metric learning is facilitated by the employment of multiple layers. At the terminal stage, a cosine similarity loss is utilized to carry out metric learning between the distinct modalities.

A.4 Miscellaneous Settings

Across our network structures featuring dense layers, we consistently employ the ‘mish’ activation function as described in [Mis20]. Each model undergoes training for a total of 100 epochs, utilizing the ‘lion’ optimizer [CLH^{+}23]. We have set the learning rate at 5-e4 and maintain a batch size 32.