The Best of Two Worlds: Deep Learning-assisted Template Attack

Lichao Wu[1], Guilherme Perin[1] and Stjepan Picek[1]

Delft University of Technology, The Netherlands

Abstract. In the last decade, machine learning-based side-channel attacks became a standard option when investigating profiling side-channel attacks. At the same time, the previous state-of-the-art technique, template attack, started losing its importance and was more considered a baseline to compare against. As such, most of the results reported that machine learning (and especially deep learning) could significantly outperform the template attack. This does not mean the template attack does not have certain advantages even when compared to deep learning. The most significant one is that it does not have any hyperparameters to tune, making it easier to use. We take another look at the template attack, and we devise a feature engineering phase allowing template attacks to compete or even outperform state-of-the-art deep learning-based side-channel attacks. More precisely, we show how a deep learning technique called the triplet model can be used to find highly efficient embeddings of input data, which can then be fed into the template attack resulting in powerful attacks.

Keywords: Side-channel Analysis · Similarity learning · Triplet network · Deep learning · Template attack.

1 Introduction

Side-channel attacks (SCA) exploits weaknesses in the physical implementation of cryptographic algorithms rather than the algorithms themselves [MOP06]. One common division of SCAs is based on the assumed attacker power and divides SCAs into direct (non-profiling) and two-stage (profiling) attacks. Profiling side-channel attacks consider the worst-case security evaluation. There, the attacker has access to a clone device used to build a model of a device under attack. After a model is built, the attacker uses it to attack an identical (or at least similar) copy of that device and obtain the secret information. The first proposed profiling SCA is the template attack [], and while this attack is the most powerful one from the information-theoretic perspective [LPB+15], many real-world settings showed its limitations. Indeed, protected targets and dataset limitations often result in settings where machine learning and especially deep learning techniques significantly outperform template attack [MPP16, PHJ+17, PHJ+18].

Although template attack cannot compete against deep learning in attack performance, it still has certain advantages. Since it has no hyperparameters to tune, this makes the template attack easier to deploy. In contrast, deep learning techniques commonly have many hyperparameters, and failing to tune them properly could result in poor attack performance. As such, a large part of the deep learning-based SCA research is oriented toward hyperparameter tuning and finding efficient neural network architectures [ZBHV19, WAGP20, RWPP21]. An additional perspective that requires consideration in the context of template attack is the points of interest selection (i.e., feature engineering). Since the template attack requires a relatively small number of features (or a very large number of measurements), one needs to select (craft) the most informative features carefully. This step naturally leads to information loss compared to deep learning that works with raw signals.

Consequently, it sounds intuitive that any improvement in template attack performance concerning the feature engineering phase, especially making it a competitive choice compared to deep learning, would be highly relevant. While during the years variants of template attacks appeared (e.g., pooled template attack [CK13]), they were commonly not aimed at improving the attack performance but making the template attack more stable. The current state-of-the-art template attack uses PCA for feature engineering [LPB+15], which, while powerful, cannot compete with the deep learning approaches.

This work proposes a novel combination of deep learning and template attack that we denote *deep learning-assisted template attack*. More precisely, we use a deep learning approach called similarity learning to find the most relevant data embedding (transformed features) and then use such data as the input to a template attack. Our main contributions are:

1. We propose a similarity learning-based approach capable of finding an efficient representation of traces in the latent space. We use the triplet model for this goal, and we obtain a compact data representation resulting in a very successful attack.
2. We propose a deep learning-assisted template attack capable of breaking protected implementations. What is more, our approach offers comparable or better results than state-of-the-art deep learning architectures.
3. Since we use template attack, there are no hyperparameters to tune, simplifying the attack process and making our approach much more straightforward to deploy, without requiring any special expertise in deep learning.

We conduct experiments on two publicly available datasets and two leakage models and show our approach to be a highly viable technique capable of outperforming state-of-the-art neural networks. To foster the reproducibility of our research, we will make our source code available upon the acceptance of the paper.

2 Preliminaries

In this section, we start by introducing the notation we follow. Next, we discuss the profiling side-channel analysis and provide details about the template attack and deep learning-based SCAs. Finally, we provide details on how to evaluate the performance of side-channel attacks.

2.1 Notation

We use calligraphic letters like $\mathcal{X}$ to denote sets. The corresponding upper-case letters denote random variables (X) and random vectors ($\mathbf{X}$) over $\mathcal{X}$. The corresponding lower-case letters (x, $\mathbf{x}$) represent realizations of X and $\mathbf{X}$, respectively.

A dataset represents a collection of side-channel measurements denoted as $\mathbf{T}$, with each trace $\mathbf{t}_i$ associated with an input value (plaintext or ciphertext) $\mathbf{d}_i$ and a key $\mathbf{k}_i$. We denote with k a key candidate that takes its value from the keyspace $\mathcal{K}$, where k^* represents the correct key. Each trace (measurement) $\mathbf{t}_i$ consists of F features (samples, points of interest). We divide the dataset into a training set of size N, a validation set of size V, and an attack set of size Q.

For the deep learning techniques we consider, $\boldsymbol{\theta}$ denotes the vector of parameters learned in a profiling model (e.g., the weights in neural networks), and $\mathcal{H}$ denotes the set of hyperparameters defining the profiling model.

2.2 Profiling Side-channel Analysis

We consider a scenario where a powerful attacker has a clone device that is identical (or at least similar) to the device to be attacked. The attacker uses N measurements from the profiling device to build a model, and then Q measurements from the device to be attacked to infer the secret information. A general principle of the profiling attack is depicted in Figure 1. Depending on the profiling technique, one builds different types of profiling models. The two most common types are a template for the template attack and machine learning models.

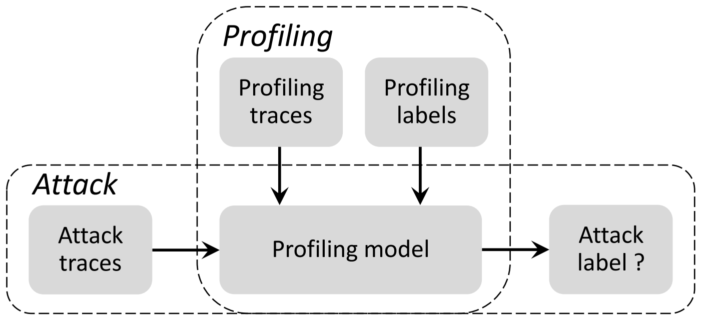


Figure 1: Profiling side-channel attack.

2.2.1 Template Attack

The best-known profiling attack is the template attack (TA) that uses Bayes theorem to obtain predictions, dealing with multivariate probability distributions as the leakage over consecutive time samples is not independent [CRR02]. In the state-of-the-art, template attack relies mostly on a (multivariate) normal distribution and is parameterized by the mean and covariance matrix. It consists of two phases: the offline phase during which the templates are built and the online phase where the matching between the templates and unseen power leakage happens.

In practice, the covariance matrices' estimation for each class value of y can be ill-posed mainly due to insufficient traces for each class. Choudary and Kuhn proposed to use one pooled covariance matrix to cope with statistical difficulties and thus lower efficiency [CK13]. Some related works showed that the pooled TA could be more efficient, in particular for a smaller number of traces in the profiling phase [CK13, PHJ+17].

2.2.2 Deep Learning-based SCA

As commonly done in the state-of-the-art [ZBHV19, ZBD+21], we consider supervised machine learning and the classification task. More precisely, the goal is to learn a function f mapping an input to the discrete output ($f : \mathcal{X} \rightarrow Y$)) based on examples of input-output pairs. The number of classes c for the discrete output depends on the leakage model and cryptographic algorithm.

The function f is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^n$, where n represents the number of trainable parameters. The goal of the profiling phase is to learn the parameters $\boldsymbol{\theta}'$, minimizing the empirical risk represented by a loss function L on a dataset of size N. In the attack phase, the goal is to predict classes (more precisely, the probabilities that a certain class would be predicted) y based on the previously unseen set of traces $\mathbf{x}$ of size Q and the trained model f.

2.3 Evaluating the Attack Performance

Once a profiling attack is finished, the result is a two-dimensional matrix P with dimensions equal to $Q \times c$. Then, it is common to use the maximum log-likelihood distinguisher, which is a probability $S(k)$ for any key candidate k:

$$S(k) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,v}). \tag{1}$$

The value $\mathbf{p}_{i,v}$ represents the probability that for a key k and input d_i, the result is class v (derived from the key and input through a cryptographic function C and a leakage model l).

The result of an attack is a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ calculated for Q traces in the attack phase. This vector contains the key candidates in decreasing order of probability: g_1 is the most likely, and $g_{|\mathcal{K}|}$ is the least likely key candidate. To reduce the effect of selected measurements (as commonly, one evaluates the attack performance for different subsets of Q measurements), it is usual to estimate the effort to obtain the secret key k^* with the guessing entropy (GE) metric [SMY09]. Guessing entropy represents the average position of k^* in $\mathbf{g}$.

3 Related Works

Chari et al., in their seminal work in 2002, proposed the template attack, representing the beginnings of profiling SCA [CRR02]. This attack is the most powerful one from the information-theoretic perspective and has no hyperparameters to tune. The main drawbacks of this attack are unrealistic assumptions (unlimited number of traces, noise following the Gaussian distribution [LPB+15]). Afterward, Schindler et al. proposed stochastic models [SLP05] and Choudhary and Kuhn pooled template attack [CK13]. For a number of years, those techniques represented state-of-the-art for profiling SCA. Besides good attack performance, those techniques also do not have hyperparameters to tune, making them easier to use. At the same time, to reduce the attack complexity, one needs to conduct a feature engineering phase to reduce the number of points of interest by either using feature selection [PHJB19] or dimensionality reduction like PCA [BHvW12].

Several years later, machine learning-based SCA became popular due to many results where such techniques surpassed the performance of the template attack. The most common examples of the machine learning methods are support vector machines [HGM+11, HZ12, PHJ+17], random forest [LMBM13, MPP16], Naive Bayes [PHG17, HPGM16], and multilayer perceptron [GHO15, MZ13]. While the results for machine learning techniques were in general favorable compared to the previous ones (e.g., template attack), the complexity of running such attacks was higher. Indeed, most machine learning techniques have hyperparameters that one needs to tune to reach the best attack performance. At the same time, to reduce the computational complexity, it was common to use feature engineering techniques as before.

Finally, in the last few years, profiling SCA mostly moved toward deep learning techniques that provided even better results than machine learning or template attack [CDP17, KPH+19]. Additionally, deep learning does not require feature engineering, making the attack preparation simpler. Unfortunately, deep learning algorithms have significantly more hyperparameter to tune compared to other techniques in profiling SCA, which again increased the complexity of running those attacks. The first significant progress was showcasing that convolutional neural networks can efficiently break targets [MPP16]. Additionally, the authors showed that deep learning works well with raw traces (or at least many more points of interest than before), removing the need for feature engineering. Cagli et al. demonstrated how deep learning could break implementations protected with a jitter countermeasure [CDP17]. Additionally, they introduced the data augmentation approach to profiling SCA. Kim et al. designed a deep learning architecture that gave excellent results for several publicly available datasets [KPH+19]. While the developed architectures differ due to different dataset dimensions (number of features), we can recognize a common design principle used for all experimental settings.

While the performance of the first deep learning-based side-channel attacks was very good, the SCA community quickly realized it could be further improved by following a careful hyperparameter tuning phase. Benadjila et al. investigated hyperparameter tuning for the ASCAD dataset and proposed several well-performing neural network architectures [BPS+20]. Zaid et al. proposed the first methodology to tune the hyperparameters related to the size (number of learnable parameters, i.e., weights and biases) of layers in convolutional neural networks [ZBHV19]. Improving up the work from Zaid et al. [ZBHV19], Wouters et al. [WAGP20] showed how to reach similar attack performance with even smaller neural network architectures. Perin et al. investigated deep learning model generalization and demonstrated how ensembles of random models could perform better than a single carefully tuned neural network model [PCP20]. Rijsdijk et al. explored the reinforcement learning paradigm to find small neural networks that perform well [RWPP21]. While their approach requires a significant tuning effort (computational time), the authors significantly improved state-of-the-art results.

All those works have in common that they design a specific architecture for each dataset and leakage model. For some works like [KPH+19], the difference in the architecture design is a natural consequence of different dataset shapes (the number of features in each trace). For others, e.g., [ZBHV19, RWPP21], the architectures are finely tuned for each experimental setting, and they differ significantly.

4 Experimental Setup

This section provides information about the datasets and leakage models we investigate, as well as information about the environment setup.

4.1 Datasets

The ASCAD datasets represent a common target for profiling SCA as they contain measurements protected with masking and settings with fixed or random keys [BPS+20]. More precisely, the ASCAD datasets contain the measurements from an 8-bit AVR microcontroller running a masked AES-128 implementation. Currently, there are two versions of the ASCAD dataset. The datasets are available at `https://github.com/ANSSI-FR/ASCAD`.

The first dataset version has a fixed key and consists of 50 000 traces for profiling and 10 000 for the attack. Each trace has 700 features (preselected window corresponding to the processing of key byte 3, the first masked key byte). We denote this dataset as **ASCAD_F**. From 50 000 traces in the profiling set, we use 45 000 traces for profiling and 5 000 for validation.

The second version has random keys, with 200 000 traces for profiling and 100 000 for the attack. Each trace has 1 400 features (preselected window corresponding to the processing of key byte 3, the first masked key byte). We denote this dataset as **ASCAD_R**. We use 45 000 traces for profiling and 5 000 traces from the attack set for validation (the attack set has a fixed but a different key from the profiling set).

4.2 Leakage Models

Our work investigates two leakage models:

1. **The Hamming weight (HW) leakage model.** In this leakage model, the attacker assumes the leakage is proportional to the sensitive variable's Hamming weight. For the AES cipher, this leakage model results in nine classes for a single intermediate byte.
2. **The Identity (ID) leakage model.** In this leakage model, the attacker considers the leakage in the form of an intermediate value of the cipher. For the AES cipher, this leakage model results in 256 classes for a single intermediate byte.

4.3 Environment

The machine learning models were implemented in python version 3.6, using TensorFlow library version 2.0. The model training algorithms were run on a cluster made out of Nvidia GTX 1080 and GTX 2080 graphics processing units (GPUs), managed by Slurm workload manager version 19.05.4.

5 Triplet Attack

In this section, we introduce the general concept of the triplet model, and afterward, we discuss how we adapt it to the context of profiling side-channel analysis.

5.1 Triplet model

Similarity learning belongs to supervised machine learning, where the goal is to learn a similarity function that measures how similar or related two objects are. One option for this task is to use a triplet network model to learn useful data representations by distance comparisons. Triplet network was evolved from the Siamese network [MKR16, GFZ+17] and is first proposed by Wang *et al.* [WSL+14] in 2014. Then, based on the triplet network, Schroff *et al.* developed the well-known Facenet network for face recognition and clustering [SKP15].

A depiction of a triplet network is shown in Figure 2. An input pair consists of three samples: positive, anchor, and negative. Among them, positive and anchor samples have the same label, while that label is different from the negative samples. By training the deep network with the shared weights, three embeddings (Emb_p, Emb_a, and Emb_n), corresponding to their input are outputted by the deep network and used for the triplet loss calculation.

An embedding represents a (relatively) low-dimensional space into which high-dimensional vectors can be translated. Embedding allows easier machine learning tasks when the inputs are large. Ideally, an embedding would capture some input semantics by placing semantically similar inputs close together in the embedding space. An embedding can be learned and reused across different models. A triplet model aims to extract these features while enlarging their inter-class differences. The triplet loss function is defined in Eq. (2).

$$loss = \mathbf{max}(d\,(a,p) - d\,(a,n) + margin, 0), \tag{2}$$

where d denotes the Euclidean distance between two feature vectors; a, p, and n stand for anchor, positive (with a label same as the anchor), and negative samples (with a label different from the anchor); $margin$ is enforced between positive and negative pairs. Based on the definition of the loss, there are three categories of triplets:

- Easy triplets: $d(a,p) + margin < d(a,n)$.
- Hard triplets: $d(a,n) < d(a,n)$.
- Semi-hard triplets: $d(a,p) < d(a,n) < d(a,p) + margin$.

Clearly, $margin$ defines the boundary between the three types of triplets. When $margin$ reaches zero, only easy and hard triplets exist. From the feature learning perspective,

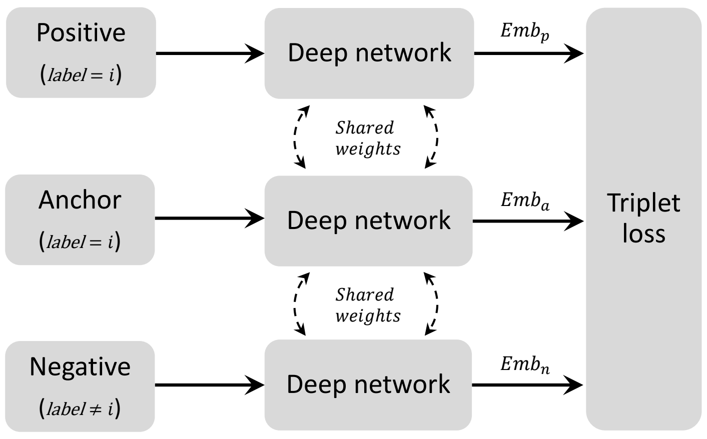


Figure 2: The structure of the triplet network.

training on easy triplets could easily reach a low loss value as p and n are easy to distinguish. However, it may result in the model converging to the local optima and struggling in differentiating the samples belonging to the different clusters but with a close Euclidean distance [1]. Training directly on the hard triplets whose negative sample is closer to the anchor than the positive may also lead the model to stop learning or collapse (the embedded output collapses to one feature) [SKP15]. On the other hand, training on semi-hard triplets increases the learning difficulties but in a reasonable range, thus leading to more representative extracted embeddings features. We set the margin to 0.2 for all of the following experiments, enabling us to choose a random semi-hard negative (the negative lies inside the *margin*) for every pair of anchor and positive and train on these triplets. The influence of this parameter is discussed in Section 6.1.7.

5.2 Attack Scheme

The correctly trained triplet model outputs embeddings with a larger Euclidean distance between each cluster than the raw inputs. Our attack scheme can be divided into two steps: 1) train a triplet model and extract the embeddings features for the profiling and attack traces, 2) launch standard profiling attacks using these embeddings features. A demonstration of the attack scheme is shown in Figure 3.

Compared with the traditional dimensionality reduction methods such as PCA [WEG87, APSQ06] or autoencoders [VLL$^+$10, WP20, RAD20], the triplet network is more task-specific: the label information utilized by the triplet network forces the network to focus on differentiating leakages (or point of interests), which is directly beneficial for the SCA attack. Techniques like PCA and autoencoders are unsupervised, which means they do not utilize the label information.

Additionally, since it is based on the construction of a Probability Density Function (PDF), a template attack can benefit from using the extracted features as the input. First, the small triplet embeddings size reduces the computation complexity of the template attack. Second, the triplet network outputs embeddings with a greater inter-class difference. Then, using those embeddings as the input would build more separated PDFs. As a result, it would help to retrieve the key with fewer attack traces. Compared to most of the other profiling attack methods, template attacks have no tunable hyperparameters. Consequently, there is

[1]The Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.

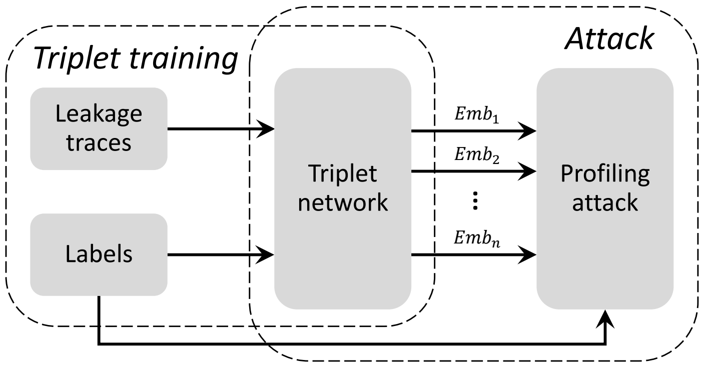


Figure 3: Triplet-assisted profiling attack.

no algorithmic randomness that can affect the profiling and attack phases. Therefore, after training the triplet network, we use the extracted embeddings and the corresponding label to perform the template attack, where the attack does not change for different datasets.

5.3 Neural Network Architectures

Several deep learning architectures, including ResNet [ZS20, HZRS16], encoder part of denoising autoencoder [WP20], CNN_best [BPS+20], and VGG 16 [SZ14] have been tested and benchmarked with SCA datasets. Consequently, we decide to design the main body of the triplet network based on the VGG neural network and follow the design principle from SCA literature [KPH+19, BPS+20]. The neural network tuning is based on the grid search of different hyperparameter combinations. The search space is listed in Table 1.

Table 1: Hyperparameters search space for triplet network.

Hyperparameter	Options
Convolution layers	5 to 13 in a step of 1
Convolution size	64 to 512 in a step of 32
Pooling Type	max pooling, avg pooling
Pooling size/stride	2 to 10 in a step of 1
Embedding size	16 to 128 in a step of 16
Learning Rate	1e-3, 5e-4, 1e-4, 5e-5, 1e-5
Activation function (all layers)	ReLU, Tanh, ELU, or SeLU
Loss function	RMSProb, Adam
Batch size	32 to 512 in a step of 32
Trainig epoch	1 to 25 in a step of 1

Since the goal of the triplet network is to extract useful embeddings from side-channel leakages, several adjustments were needed. First, the large dense layers and the final classification layer are replaced with a single embedding (dense) layer as the goal is feature extraction but not classification. Note that the size of the embeddings layer is essential for the triplet network: either too large or too small embeddings size may have side effects on the extracted embeddings, influencing the attack performance. Here, we set the size of the embedding to 32 based on the grid search results. The influence of this parameter is discussed in Section 6.1.6. Besides, we use average pooling as it performs better for the tested datasets [BPS+20, WP21]; the corresponding size and stride are set to two. *SeLU* is used as the activation function to avoid vanishing and exploding gradient problems [KUMH17]. The batch size is set to 512 for all experiments. The detailed implementation of the network is listed in Table 2.

Table 2: Triplet architectures used in the experiments.

Layer	kernal number/size	neurons
Conv*2+AvgPooling	64/3	-
Conv*2+AvgPooling	128/3	-
Conv*3+AvgPooling	256/3	-
Conv*3+AvgPooling	512/3	-
Conv*3+AvgPooling+flatten	512/3	-
Dense	-	32

6 Experimental Results

This section systematically evaluates our attack scheme from two aspects: side-channel attack performance and triplet parameters' influence. The analysis is conducted on two publicly available datasets described in Section 4.1. We consider both the HW and ID leakage models. In terms of model training, we set the training epoch to 20. Still, it is worth noting that the original traces can even be broken with one-epoch training. The detailed discussion is in Section 6.1.4. The optimizer is *Adam* with a learning rate of 1e-4.

To evaluate the attack performance, we consider two metrics:

- Guessing entropy (GE): the averaged key rank after applying the maximum number of attack traces.
- T_{GE0}: the number of traces required to reach GE equal to zero.

GE and T_{GE0} metrics are derived from key rank, aiming at evaluating the key recovery capacity of profiling models by setting a limited number of attack traces. Specifically, the second metric (T_{GE0}) is designed for cases where the models require fewer traces (than the maximum number of attack traces) to retrieve the secret key. In this case, even if GE equals zero for different settings, we can better estimate the attack performance by evaluating the number of attack traces to reach it.

6.1 Performance Evaluation

First, we show results for the original (as publicly available) datasets, and afterward, we provide results for traces with included simulated countermeasures.

6.1.1 Original Datasets

We compare the attack performance of our models with the state-of-the-art models [ZBHV19, WPP20, RWPP21, PCP20] and different dimensionality reduction techniques such as PCA (PCA-TA) and autoencoder (AE-TA) with the same feature size (32). The implementation details for the autoencoder are listed in Appendix A. Note that the model from [ZBHV19] was optimized for the ID leakage model, and we adjust the output layer by reducing the number of classes from 256 to 9 and test with the HW leakage model as well. The benchmarks with ASCAD_F and ASCAD_R with the T_{GE0} metric are shown in Tables 3 and 4. The best results are denoted in bold style.

Interestingly, although the models from considered literature are optimized for specific datasets and leakage models, our triplet attack is still comparable as it reaches the best performance in two out of four scenarios. Meanwhile, compared with PCA and AE, the usage of the label information significantly increases the quality of the extracted features by the triplet network, thus leading to a better attack performance. We also note that for the ASCAD_R dataset and the ID leakage model, the best result is obtained from an ensemble of profiling models. Our approach gives the best results when considering only a single model performance.

Table 3: Benchmark the attack performance (T_{GE0}) with the ASCAD_F dataset.

	[ZBHV19]	[WPP20]	[RWPP21]	PCA-TA	AE-TA	This work
HW	1 346	965	906	542	539	**370**
ID	191	**155**	202	337	862	372

Table 4: Benchmark the attack performance (T_{GE0}) with the ASCAD_R dataset.

	[PCP20]	[WPP20]	[RWPP21]	PCA-TA	AE-TA	This work
HW	470	496	911	>10 000	2 220	**424**
ID	**105**	1 568	490	8 300	6 085	297

Note that we used template attack as the final stage of triplet attacks, which could also be switched to other profiling attack methods. However, we believe a template attack represents a robust and straightforward solution as it does not require additional tuning of the neural network. In addition, we also tested the pooled template attack on features extracted by the triplet network. However, this technique works significantly worse than the template attack (failed to break the target with 10 000 traces), so we do not include the corresponding results in the paper.

6.1.2 Add Gaussian Noise

The Gaussian noise represents the most common type of noise existing in side-channel traces. It can be introduced by several sources such as the transistor, data buses, the transmission line to the record devices such as oscilloscopes, or even the work environment. Gaussian noise can also be used as hiding countermeasures by implementing parallel operations or a dedicated noise engine. In terms of side-channel attacks, the addition of noise reduces the signal-to-noise ratio (SNR), thus degrading the attack performance. To demonstrate the influence of the Gaussian noise, we add a normal-distributed random value with zero mean and different standard deviations to each point of the original trace. Since the triplet network is trained with noisy traces, we could evaluate its resilience towards the perturbation of the Gaussian noise. The pseudocode for constructing traces with Gaussian noise is shown in Algorithm 1 [WP20].

Algorithm 1 Add Gaussian Noise.

```
1: function ADD_GAUSSIAN_NOISE($trace, mean, variance$)
2:     $new\_trace \leftarrow []$
3:     $i \leftarrow 0$
4:     while $i < len(trace)$ do
5:         $level \leftarrow$randomNumber($mean, variance$)
6:         $new\_trace[i] \leftarrow traces[i] + level$          ▷ add noise to the trace
7:         $i \leftarrow i + 1$
8:     end while
9:     return $new\_trace$
10: end function
```

In terms of test settings, four standard deviation levels (1/3/5/8) were tested. For each selected noise value, we independently train 20 triplet models and perform the template attack. The averaged model (the model with attack performance averaged from 20 attacks) represents the attack performance with the current test setting. In this way, the effect of the random weight initialization can be compensated, and we can better estimate the true performance of the model.

The averaged results are shown in Figure 4. Besides, in Table 5, we list the attack performance of the best-performing models to demonstrate the maximum attack capability of our model. As expected, the addition of the Gaussian noise degrades the attack performance. Still, except for one case for the ASCAD_F dataset (Figure 4a, standard deviation equals eight), the attacks performed on the extracted embeddings lead to converging GEs. Interestingly, up to standard deviation level five, from Table 5, the best performing model can break the target within 10 000 attack traces, indicating strong resilience against the Gaussian noise.

Note that Gaussian noise with a standard deviation of eight can also be partially removed with a denoising autoencoder [WP20] training with noisy-clean traces pairs. When comparing their TA performance with ours, our model requires less training effort but could reach a comparable attack performance with the same level of noise (Figure 4b). In addition, we also tested PCA-TA and AE-TA[2] for noisy traces. As a result, besides the results testing on ASCAD_F with noise level 1 for PCA-TA, none of those techniques lead to GE converging to zero with 10 000 traces. Indeed, both techniques only use the traces information (and not the label information) to extract the features. Thus, the addition of the noise has a stronger side-effect on the template construction.

Table 5: The T_{GE0} of best-performing triplet attacks with different level of Gaussian noise.

Noise level	ASCAD_F HW	ASCAD_F ID	ASCAD_R HW	ASCAD_R ID
1	738	1 283	799	1 007
3	1 731	2 390	1 653	1 547
5	5 883	5 122	2 759	4 767
8	>10 000	>10 000	>10 000	7 822

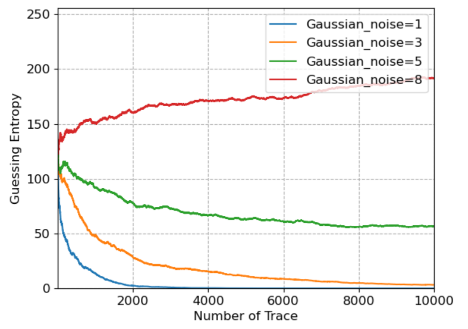
Meanwhile, from Table 5, the best attacks with the HW leakage model require fewer attack traces than the ones with the ID leakage model. Knowing that both ASCAD_F and ASCAD_R work well with both leakage models, this behavior could be explained with the cluster number's difference: the triplet model may need additional training effort (i.e., more training epochs or training traces) to extract useful embeddings from 256 class (ID) than nine classes (HW). The same observation is also shown when adding time randomness to the traces in the next section.

6.1.3 Add Desynchronization

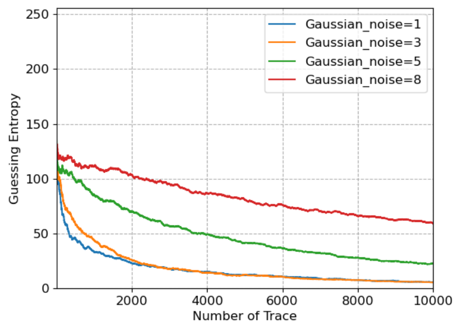
The well-synchronized traces can significantly improve the correlation of the intermediate data and trace values. The alignment of the traces is, therefore, an essential step for the side-channel attack. Different from the Gaussian noise, the desynchronization adds randomness to the time domain. To show the effect of trace desynchronization, we perturb the traces with different levels of desynchronization. The pseudocode for constructing traces with desynchronization is shown in Algorithm 2 [WP20].

Three desynchronization levels (20/50/100) were simulated and tested in this section. Aligned with the previous section, we show the results from 20 averaged models (Figure 5). The attack performances of the best models are listed in Table 6. Again, we observe the positive relation between the desynchronization level and attack performance. However,

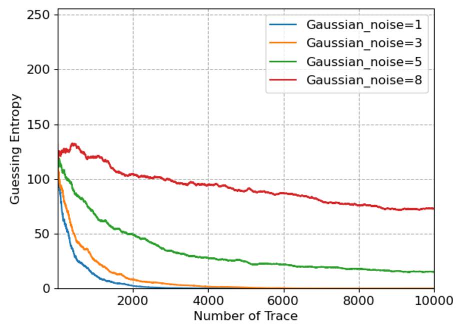
[2]We train AE with noisy-noisy traces pair as our goal is to test the feature extraction capability of AE. Training with noisy-clean traces pairs (as done in denoising autoencoder approach) is not fair to compare. This training setting is also applied to the next section with desynchronization.



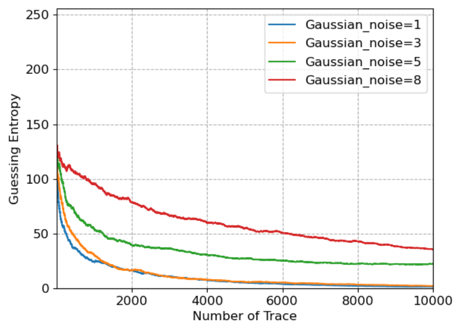
(a) GE: ASCAD_F with the HW leakage model.



(b) GE: ASCAD_F with the ID leakage model.



(c) GE: ASCAD_R with the HW leakage model.



(d) GE: ASCAD_R with the ID leakage model.

Figure 4: The effect of Gaussian noise: attack on ASCAD_F and ASCAD_R with the HW and ID leakage models.

compared with the Gaussian noise, the triplet model is more robust towards the randomness in the time domain: the best performing models can retrieve the key within 10 000 traces in three out of four cases even with the maximum desynchronization level of 100 (Table 6). Indeed, literature [CDP17, ZBHV19] indicates that convolution blocks are resilient to trace misalignment. Benefiting from this property, our triplet network can extract useful features from data with high time randomness. We also tested PCA-TA and AE-TA, but again, both work significantly worse than our method.

Table 6: The T_{GE0} of best-performing triplet attacks with different level of desynchronization.

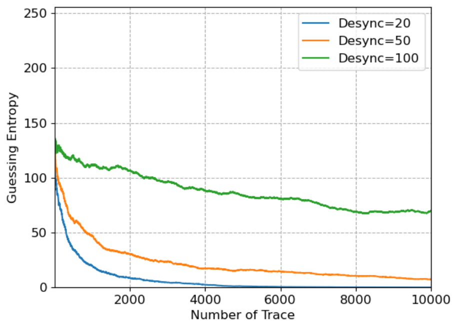
Noise level	ASCAD_F HW	ASCAD_F ID	ASCAD_R HW	ASCAD_R ID
20	580	1 201	536	1 203
50	747	1 229	689	2 077
100	>10 000	6 993	3 811	3 495

Knowing the attack performance of the triplet network, one could be curious about 1) the features the triplet learns and 2) the factors that influence the attack performance. To cover the first aspect, we consider the traces with desynchronization level 20 and investigate the averaged embedding per cluster. The random seed is fixed to avoid the effect of the random weight initialization. The results are shown in Figure 6.

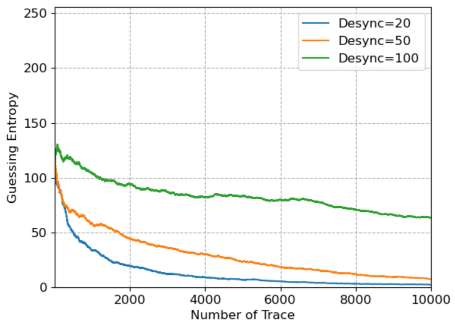
Interestingly, the triplet network can consistently extract the robust features as well as

Algorithm 2 Add Desynchronization.

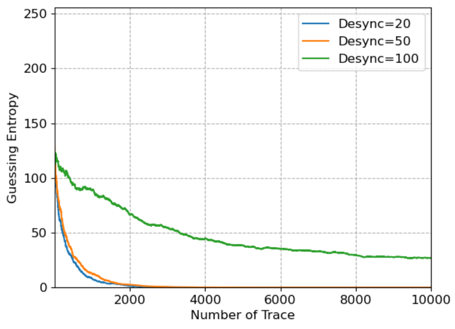
```
1: function ADD_DESYNC($trace, desync\_level$)
2:     $new\_trace \leftarrow []$
3:     $level \leftarrow$ randomNumber($0, desync\_level$)
4:     $i \leftarrow 0$
5:     while $i + level < len(trace)$ do
6:         $new\_trace[i] \leftarrow traces[i + level]$          ▷ shift the trace
7:         $i \leftarrow i + 1$
8:     end while
9:     return $new\_trace$
10: end function
```



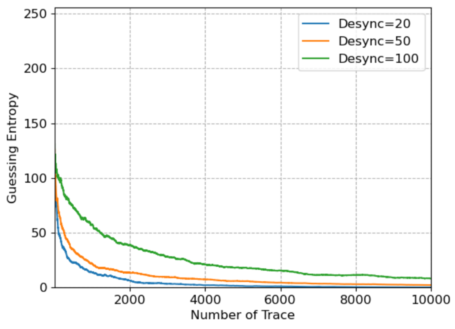
(a) GE: ASCAD_F with the HW leakage model.



(b) GE: ASCAD_F with the ID leakage model.



(c) GE: ASCAD_R with the HW leakage model.

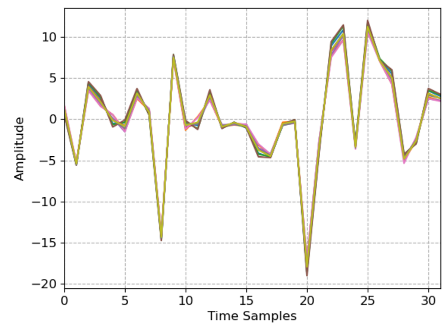


(d) GE: ASCAD_R with the ID leakage model.

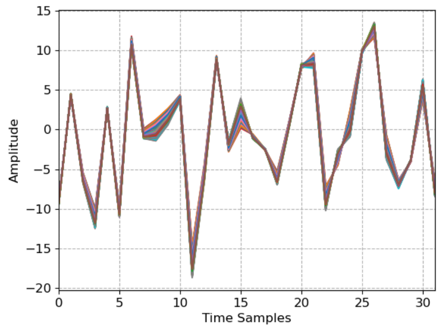
Figure 5: The effect of desynchronization: attack on ASCAD_F and ASCAD_R with the HW and ID leakage models.

find variations between clusters from the input. Compared with the original traces per cluster (see Appendix B), the extracted features significantly help the template attack in building distinguishable PDFs, thus boosting the attack performance. Meanwhile, one could notice the embedding difference between the HW and ID leakage models, indicating that the extracted features are based on the selected leakage model.

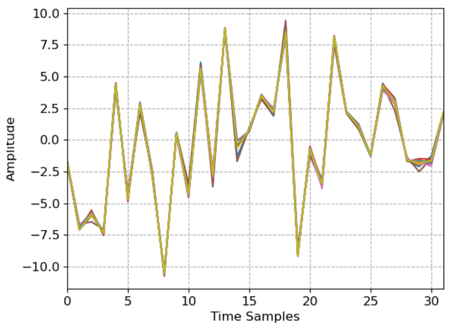
For the second aspect, we concentrate on several hyperparameters that could influence the behavior of the triplet network model. The detailed evaluations are presented in the following sections.



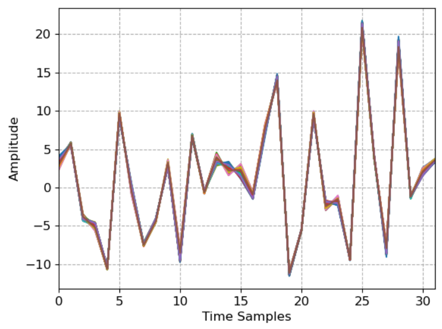
(a) ASCAD_F with the HW leakage model.



(b) ASCAD_F with the ID leakage model.



(c) ASCAD_R with the HW leakage model.



(d) ASCAD_R with the ID leakage model.

Figure 6: Averaged embedding per cluster: extracted from ASCAD_F and ASCAD_R with the HW and ID leakage models.

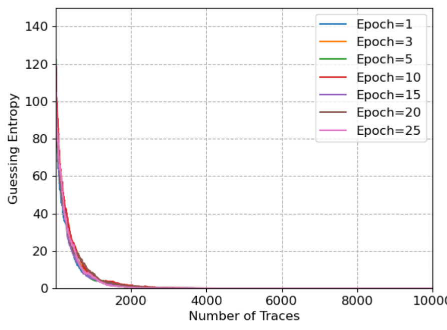
6.1.4 The Number of Training Epochs

Accuracy is one of the core metrics to evaluate a deep learning model in the deep learning domain. Most of the individual examples must be correctly classified to reach high accuracy. As a consequence, when using the triplet network to extract the features, a high training effort is required to extract meaningful embeddings (1 000 to 2 000 CPU hours according to [SKP15]).

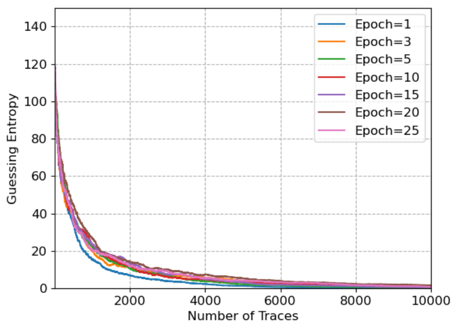
In the SCA domain, instead of focusing on a single data, the evaluator commonly focuses on the accumulated probability of the samples in clusters. Since the requirement for the classification accuracy is reduced, the triplet network for SCA may require fewer training epochs, and the secret information can be successfully retrieved. In this section, we explore the influence of the number of training epochs on attack performance. Same as in the previous sections, the results are averaged over 20 independently trained models.

As shown in Figure 7, training epochs have limited influence on the attack performance, indicating that the leakage-related features have been extracted during the first training epochs. In other words, our attack could even be finished with one-epoch training and exhibit the best attack performance (Figures 7a and 7b). Compared with other deep learning attacks that normally required more than 50 training epochs [BPS+20, KPH+19, ZBD+21], our attack scheme dramatically speeds up the feature learning process.

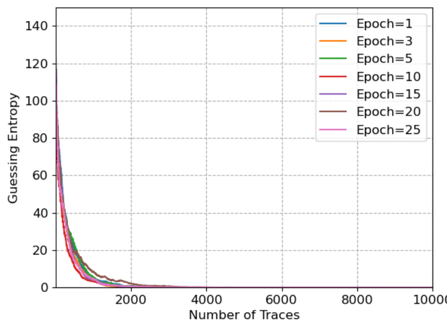
Similar to the other deep learning attacks, our attacks also suffer from overfitting. Specifically, from Figure 7b, more training epochs lead to worse attack performance. Fortunately, the effect of overfitting has limited effect with the considered attack scenario.



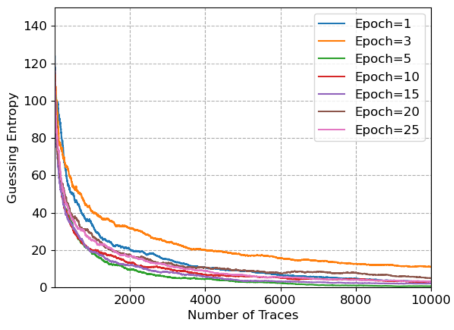
(a) GE: ASCAD_F with the HW leakage model.



(b) GE: ASCAD_F with the ID leakage model.



(c) GE: ASCAD_R with the HW leakage model.



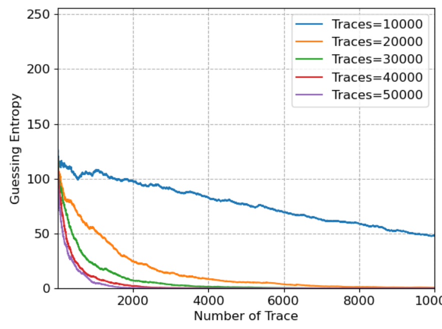
(d) GE: ASCAD_R with the ID leakage model.

Figure 7: The effect of training epochs: attack on ASCAD_F and ASCAD_R with the HW and ID leakage models.

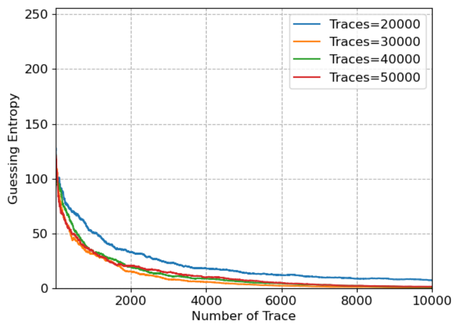
6.1.5 Training Set Size

Like other deep learning models, the triplet model can require large quantities of data to perform well. However, the training sets for the triplet network have more constraints: 1) a single training set for a triplet network consists of three individual samples (anchor, positive, and negative) and 2) the selection of the positive and negative samples is limited by the *margin* value. Following this, the triplet network training may require more traces than the conventional deep learning attacks. To investigate the relation between the number of training traces and the attack performance, we vary the number of the profiling traces from 10 000 to 50 000 with a step of 10 000 traces. The results are shown in Figure 8. Note that for the ID leakage model, due to the lack of training data, training with 10 000 traces (and 20 000 traces for ASCAD_R) always leads to an unsuccessful template attack (singular matrix) so the results are not presented.

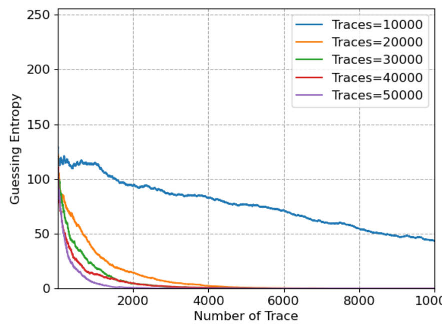
In both test scenarios, more training traces lead to better attack performance. Besides, a significant performance leap can be observed when the number of training traces increases from 10 000 to 20 000 for the HW leakage model. Indeed, with a 10 000 training set, the smallest cluster only has 35 samples for the HW leakage model (class imbalance) and 20 for the ID leakage model. Knowing that not all of them can form a triplet due to triplet margin restriction, the triplet network cannot generalize well for the minority clusters, thus leading to bad performance or even attack failure for both HW and ID leakage models. On the other hand, the performance increases slowly when the traces are above 20 000, indicating that the triplet model reaches its maximum feature extraction capability.



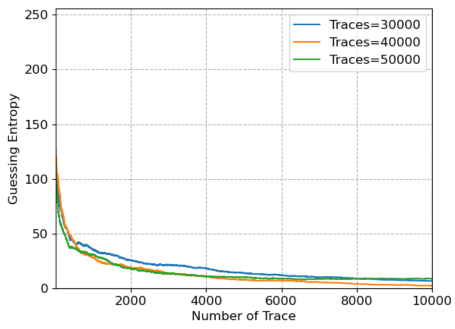
(a) GE: ASCAD_F with the HW leakage model.



(b) GE: ASCAD_F with the ID leakage model.



(c) GE: ASCAD_R with the HW leakage model.



(d) GE: ASCAD_R with the ID leakage model.

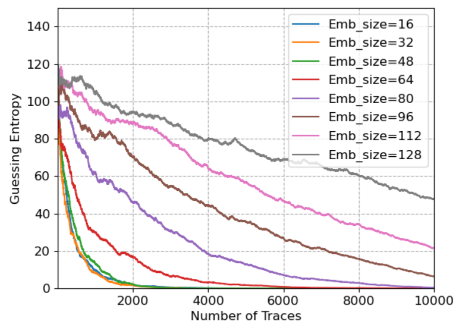
Figure 8: The effect of training traces: attack on ASCAD_F and ASCAD_R with the HW and ID leakage models.

6.1.6 Embedding Size

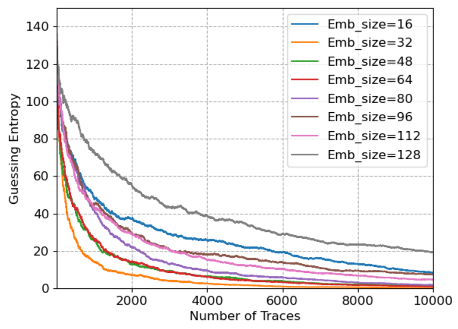
As mentioned, the embedding size directly impacts the template attack performance as it determines the dimension of the extracted features. In this section, we tune this hyperparameter and analyze its effect on the attack performance. Similar to the previous sections' test setting, we independently train 20 models and present the averaged performance for each embedding size. The tuning range is from 16 to 128 in a step of 16. We set the maximal embedding size to 128, as there are only around 140 measurements for the least represented class, so higher values would trigger a singular matrix problem, and the template attack would fail.

The embedding tuning results are shown in Figure 9. From the results, a larger embedding size could actually lead to worse attack performance. Indeed, the additional features introduced by a larger embedding size may be harmful to the overall attack performance as they may contain noise learned from the irrelevant raw features. Moreover, more embedding features would dilute the features extracted by the triplet model, thus reducing the attack performance.

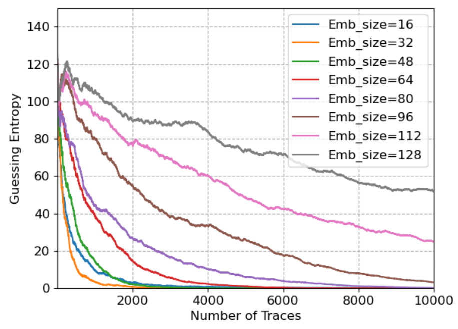
When evaluating smaller embedding sizes, size 32 performs comparable (Figure 9a) or even better than size 16. As expected, an overly small embedding size would not have enough dimensions to represent the characteristic of the raw features. Although the optimal embedding size would be different when testing other datasets, we believe the relationship between the embedding size and attack performance follows the observations in this section. In any case, since it is common to conduct feature engineering when



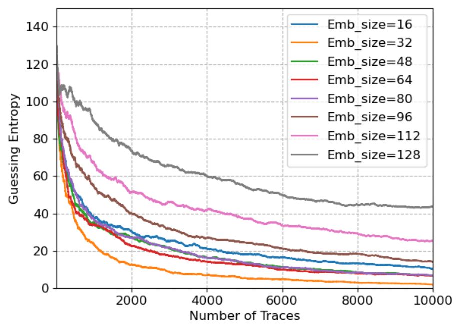
(a) GE: ASCAD_F with the HW leakage model.



(b) GE: ASCAD_F with the ID leakage model.



(c) GE: ASCAD_R with the HW leakage model.



(d) GE: ASCAD_R with the ID leakage model.

Figure 9: The effect of embedding size: attack on ASCAD_F and ASCAD_R with the HW and ID leakage models.

running the template attack, we do not consider the effort required to tune the embedding size more substantial.

6.1.7 Triplet Margin

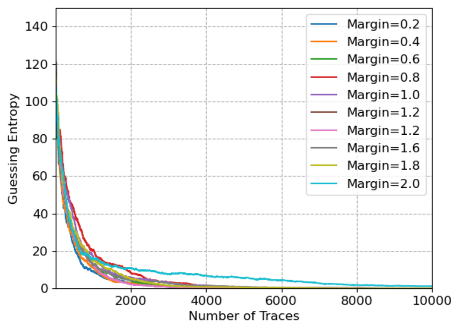
In this section, we vary the triplet margin and investigate its influence on the attack performance. The test setting is the same as the previous sections. In the experiments, we set *margin* ranges from 0.2 (same as [SKP15]) to 2.0 in a step of 0.2 to explore the *margin* effect.

The experimental results are shown in Figure 10. From the results, the increase of the *margin* value slightly degrades the attack performance. As mentioned before, when the margin becomes too large, since too many simple triplets are involved in training, the model can easily converge to the local optima and stop learning. Meanwhile, from Figures 10b and 10d, smaller *margins* does not necessary mean better attack performance. Still, compared with the effect of the size of the embedding shown in Figure 9, the *margin* has a limited effect on the attack performance.

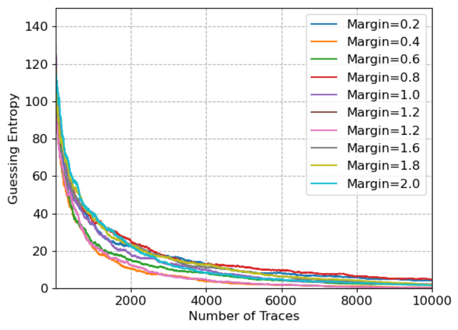
6.2 General Observations

Based on the conducted experiments, we provide several general observations:

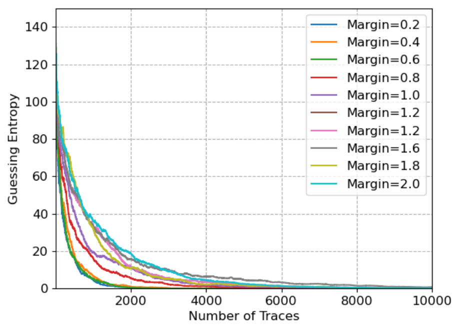
- When attacking the original datasets (without noise addition), triplet-assisted template attack performs comparably or even better than the state-of-the-art models



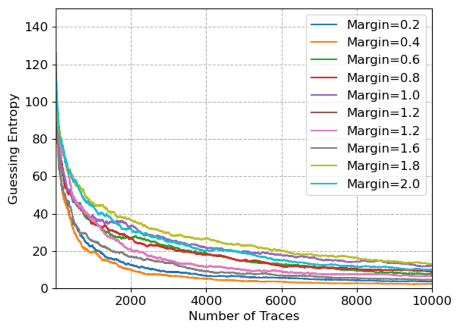
(a) GE: ASCAD_F with the HW leakage model.



(b) GE: ASCAD_F with the ID leakage model.



(c) GE: ASCAD_R with the HW leakage model.



(d) GE: ASCAD_R with the ID leakage model.

Figure 10: The effect of triplet margin: attack on ASCAD_F and ASCAD_R with the HW and ID leakage models.

from the literature.

- Triplet-assisted template attack is more resilient to noise in horizontal and vertical dimensions than other commonly used feature engineering techniques (PCA and autoencoders).
- The triplet network can be highly efficient in extracting the leakage-related features. One-epoch training is sufficient to train a triplet network for the evaluated datasets.
- The embedding size has a dominant influence on the attack performance. Either too large or too small embedding size would lead to the degradation of the attack performance.
- The number of training traces has a significant impact on the attack performance.
- The embeddings are different for different leakage models, so changing a leakage model requires re-training a triplet model to find a new embedding.
- Increasing *margin* will increase the triplet loss and provide additional capability to the triplet network to learn from the data. However, since the semi-hard triplet selection is also based on the *margin* value (the negatives lie inside the *margin*), greater *margin* would also include more easy triplet being formed. In general, the triplet network still has a high tolerance towards the variation of the *margin* value.

7 Conclusions and Future Work

This work investigates how to improve the performance of template attacks. To accomplish this, we use the concepts of triplet networks that have a task to find a well-performing embedding based on the input traces. We conduct experiments on two publicly available datasets and two leakage models and show that our deep learning-assisted template attack can even outperform state-of-the-art deep learning-based SCAs. This result is especially significant as we compared with a number of deep learning architectures that were specifically tuned for different experimental settings. Additionally, we show that our approach is rather resilient to Gaussian noise and desynchronization and significantly better performing than related techniques for feature engineering (PCA and autoencoders). Finally, we show that both the profiling set size and the embedding size significantly influence the attack performance.

We plan to extend our approach to raw traces (and not preselected windows as used now). Since the raw traces are significantly larger (e.g., 100 000 features vs. 700 features), it is interesting to explore the limitations of triplet networks. Additionally, this work showed how template attack works well when combined with triplet networks. We aim to explore the combinations of triplet networks with simpler machine learning techniques like a random forest or support vector machines in future work.

A Autoencoder Implementation

An autoencoder consists of two parts: encoder (ϕ) and decoder (ψ). As shown in Eq. (3), the encoder transfers the input with more features to its latent space ($\phi : \mathcal{X} \rightarrow \mathcal{F}$), while the goal of the decoder is to reverse this process ($\psi : \mathcal{F} \rightarrow \mathcal{X}$). The goal of an autoencoder is to transform inputs into outputs with the least possible amount of distortion [Bal12]. Naturally, the most representative features of the input data are learned and kept in the latent space (embedding) of the network:

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \, \|X - (\psi \circ \phi)X\|^2. \tag{3}$$

The design of the autoencoder used in this work follows the related work [WP20]. We simplify the original denoising autoencoder as our goal is not to remove noise from the traces but to extract the embeddings from the latent space. Intuitively, the latent space can be considered as the representation of compressed data. For an autoencoder, the dimension of the latent space is normally defined by the layer in the middle. The implementation details are listed in Table 7 (BN stands for batch normalization layer). We use *SeLU* as the activation function, the loss function is *Adam* with a learning rate of 1e-4, and the number of training epochs is set to 75.

B Averaged Traces per Cluster for the Original Traces

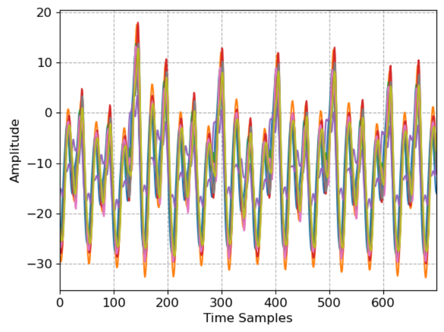
This section shows the averaged traces per cluster for the original traces (triplet input). The large variation between clusters is caused by the introduced desynchronization (recall that we use traces with added desynchronization of 20).

References

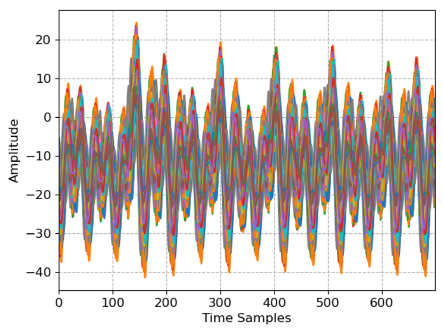
[APSQ06] Cédric Archambeau, Eric Peeters, F-X Standaert, and J-J Quisquater. Template attacks in principal subspaces. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–14. Springer, 2006.

Table 7: Autoencoder used in the experiments.

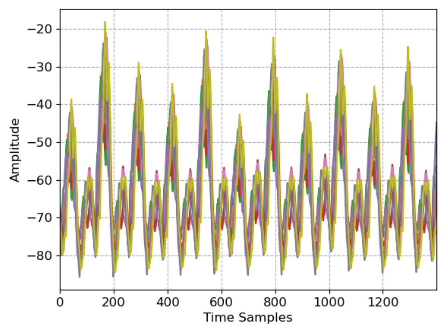
Layer	kernel number/size	neurons
Conv*2+BN+AvgPooling	256/2	-
Conv*2+BN+AvgPooling	256/2	-
Conv+BN+AvgPooling	128/2	-
Conv+BN+AvgPooling	128/2	-
Conv+BN+AvgPooling	64/2	-
Dense	-	32
Deconv+BN+UpSampling	64/2	-
Deconv+BN+UpSampling	128/2	-
Deconv+BN+UpSampling	128/2	-
Deconv*2+BN+UpSampling	256/2	-
Deconv*2+BN+UpSampling	256/2	-
Deconv	1/2	-
Crop	-	-



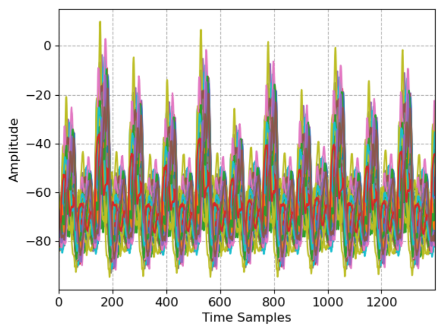
(a) ASCAD_F with the HW leakage model.



(b) ASCAD_F with the ID leakage model.



(c) ASCAD_R with the HW leakage model.



(d) ASCAD_R with the ID leakage model.

Figure 11: Averaged traces per cluster: extracted from ASCAD_F and ASCAD_R with the HW and ID leakage models.

[Bal12] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[BHvW12] Lejla Batina, Jip Hogenboom, and Jasper G. J. van Woudenberg. Getting more from pca: First results of using principal component analysis for extensive power analysis. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, pages 383–397, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[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. Cryptographic Engineering*, 10(2):163–188, 2020.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 45–68, Cham, 2017. Springer International Publishing.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *LNCS*, pages 253–270. Springer, 2013.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay (Redwood City), USA.

[GFZ$^+$17] Qing Guo, Wei Feng, Ce Zhou, Rui Huang, Liang Wan, and Song Wang. Learning dynamic siamese network for visual object tracking. In *Proceedings of the IEEE international conference on computer vision*, pages 1763–1771, 2017.

[GHO15] R. Gilmore, N. Hanley, and M. O'Neill. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111, May 2015.

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

[HPGM16] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In Gerhard P. Hancke and Konstantinos Markantonakis, editors, *Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers*, volume 10155 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2016.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[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 Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2013. Berlin, Germany.

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

[MKR16] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. Siamese network features for image matching. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 378–383. IEEE, 2016.

[MOP06] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, http://www.dpabook.org/.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[MZ13] Z. Martinasek and V. Zeman. Innovative method of the power analysis. *Radioengineering*, 22(2), 2013.

[PCP20] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):337–364, Aug. 2020.

[PHG17] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template attack versus bayes classifier. *J. Cryptogr. Eng.*, 7(4):343–351, 2017.

[PHJ$^+$17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 4095–4102, 2017.

[PHJ$^+$18] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):209–237, Nov. 2018.

[PHJB19] S. Picek, A. Heuser, A. Jovic, and L. Batina. A systematic evaluation of profiling through focused feature selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2802–2815, 2019.

[RAD20] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. SCARL: side-channel analysis with reinforcement learning on the ascon authenticated cipher. *CoRR*, abs/2006.03995, 2020.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):677–707, Jul. 2021.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[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*, pages 30–46, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[SMY09] François-Xavier Standaert, Tal G. 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*, pages 443–461, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[VLL$^+$10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):147–168, Jun. 2020.

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):389–415, Aug. 2020.

[WP21] Lichao Wu and Guilherme Perin. On the importance of pooling layer tuning for profiling side-channel analysis. *IACR Cryptol. ePrint Arch.*, 2021:525, 2021.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2020/1293, 2020. https://eprint.iacr.org/2020/1293.

[WSL$^+$14] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1386–1393, 2014.

[ZBD$^+$21] Gabriel Zaid, Lilian Bossuet, François Dassance, Amaury Habrard, and Alexandre Venelli. Ranking loss: Maximizing the success rate in deep learning side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):25–55, 2021.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

[ZS20] Yuanyuan Zhou and François-Xavier Standaert. Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks. *J. Cryptogr. Eng.*, 10(1):85–95, 2020.