IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 3, pp. 413–437. DOI:10.46586/tches.v2022.i3.413-437

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

Lichao Wu[1], Guilherme Perin[1] and Stjepan Picek[2,1]

[1] Delft University of Technology, The Netherlands
[2] Radboud University, The Netherlands

Abstract. In the last decade, machine learning-based side-channel attacks have become 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. Nevertheless, the template attack still has certain advantages even compared to deep learning. The most significant one is that it has only a few hyperparameters to tune, making it easier to use.
We take another look at the template attack, and we devise a feature engineering phase allowing the template attack to compete or even outperform state-of-the-art deep learning-based side-channel attacks. More precisely, with a novel distance metric customized for side-channel analysis, we show how a deep learning technique called similarity learning can be used to find highly efficient embeddings of input data with one-epoch training, 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) exploit weaknesses in the physical implementation of cryptographic algorithms rather than the algorithms themselves [MOP06]. One standard division of SCAs is based on the assumed attacker power and divides SCAs into direct (non-profiling) and two-stage (profiling) attacks. Depending on the profiling attack settings, profiling side-channel attacks could 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 building a model, 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 [CRR02, LPB+15], while many real-world settings showed its limitations. Indeed, protected targets often result in settings where machine learning and deep learning techniques significantly outperform template attack [MPP16, PHJ+17, PHJ+18, WP20].

However, certain advantages of the template attack cannot be ignored. Template attack is easier to deploy as it has only a few hyperparameters compared to the deep learning-based approaches. Indeed, the main hyperparameters for the template attack are the number of features to use and the technique to select those features. On the other hand, a large part of the deep learning-based SCA research is oriented toward hyperparameter tuning and finding efficient neural network architectures [ZBHV19, WAGP20, RWPP21, LZC+21]. While those works manage to find powerful neural network architectures, the search can take significant time, and the architectures are commonly dataset-specific. Those drawbacks can result in difficulties for the evaluators/attackers when deploying such methods in real-world

Received: 2022-01-15 Accepted: 2022-03-15 Published: 2022-06-08

applications. However, the fact that the template attack has only a few hyperparameters does not necessarily mean it becomes trivial to tune them. Certainly, a perspective that requires consideration in the context of the template attack is the points of interest selection (i.e., feature engineering). Since the template attack requires a relatively small number of features (or many side-channel 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 [LZC+21, PWP21a].

Consequently, it sounds intuitive that any improvement in the 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 the template attack appeared (as discussed in Section 3), they were commonly not aimed at improving the attack performance but at making the template attack more stable and efficient. The current state-of-the-art template attack uses Principal Component Analysis (PCA) [BHvW12, LPB+15], Linear Discriminant Analysis (LDA) [SA08], Sum of Squared Pairwise T-differences (SOST) [GLRP06] for feature engineering, which, while powerful, struggles when dealing with protected leakages [BPS+20].

This work proposes a novel combination of deep learning and template attack that we denote *deep learning-assisted template attack*. 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 extracting an efficient embedding of side-channel traces in the latent space. We use the triplet model for this goal, and we obtain a compact data representation resulting in an outstanding attack performance.
2. We propose a novel distance metric, denoted the Hybrid Distance, that takes into consideration both embedding distance and label distance. This new distance metric significantly improves the quality of extracted features from the attack perspective.
3. We validate the time efficiency and attack performance on dynamic attack settings (datasets, leakage models, traces desynchronization). As a result, with one epoch training (around 20 seconds on a GPU in our setting), our results are comparable to or better than state-of-the-art deep learning architectures and feature reduction techniques in all scenarios.
4. We systematically evaluate the influence of several critical hyperparameters in the proposed attack scheme, which can serve as a guideline for potential evaluators/attackers.

We conduct experiments on three publicly available datasets and three leakage models. The source code is available in the Github `https://github.com/AISyLab/Triplet-attack`.

The rest of this paper is organized as follows. In Section 2, we discuss the notation we follow, profiling side-channel analysis, and how to evaluate the attack performance, datasets, and leakage models. Section 3 provides an overview of related works. Section 4 introduces the triplet attack, the novel distance we consider, and the neural network architectures used in the experiments. Section 5 presents experimental results with various datasets and leakage models. Additionally, we provide a detailed evaluation of various hyperparameters and general observations. Finally, in Section 6, we conclude the paper and provide several possible future research directions.

2 Preliminaries

This section starts by introducing the notation we follow. Next, we discuss the profiling side-channel analysis and give details about the template attack and deep learning-based SCAs. Then, we provide details on how to evaluate the performance of side-channel attacks. Finally, we discuss the datasets and leakage models considered in this work.

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 side-channel dataset $\mathbf{T}$ represents a collection of side-channel measurements, with each measurement (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 taking its value from the keyspace $\mathcal{K}$. k^* denotes the correct key. Each trace $\mathbf{t}_i$ consists of a number of features (samples, points of interest). We divide the dataset into a training (profiling) set of size O and an attack (test) set of size Q.

2.2 Profiling Side-channel Analysis

We consider a scenario where a powerful attacker has a clone device identical (or at least similar) to the device to be attacked. The attacker uses O 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.

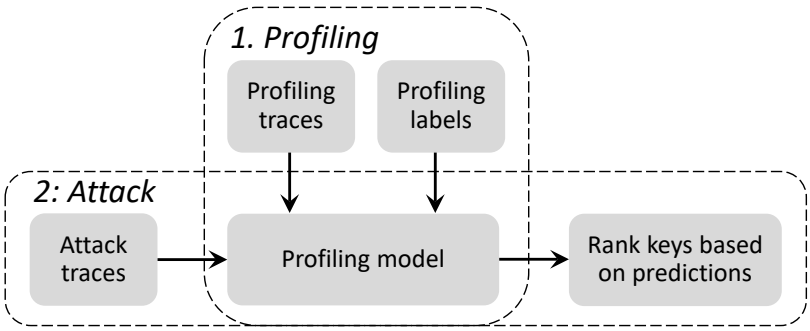


Figure 1: Profiling side-channel attack.

2.2.1 Template Attack

The best-known profiling attack is the template attack (TA) that uses the 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. The template attack 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 y can be ill-posed mainly due to insufficient traces for each class. To prevent this issue, it is possible to combine all covariance matrices into a single one, reaching the version of the template attack commonly known as the pooled template attack. 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 ($f : \mathcal{X} \to Y$)) based on examples of input-output pairs. The number of classes c for the discrete output depends on the leakage model and the cryptographic algorithm.

The function f is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^z$, where z represents the number of trainable parameters and $\boldsymbol{\theta}$ denotes the vector of parameters learned in a profiling model. The profiling phase aims to learn the parameters $\boldsymbol{\theta}$, minimizing the empirical risk represented by a loss function on a dataset of size O. 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 with dimensions equal to $Q \times c$. Then, it is common to use the maximum log-likelihood distinguisher, which is a cumulative sum $S(k)$ for any key candidate k:

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

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

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 $\mathbf{g}$.

2.4 Datasets

2.4.1 ASCAD

The ASCAD datasets represent a common target for profiling SCA as they contain measurements protected with a masking countermeasure 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.

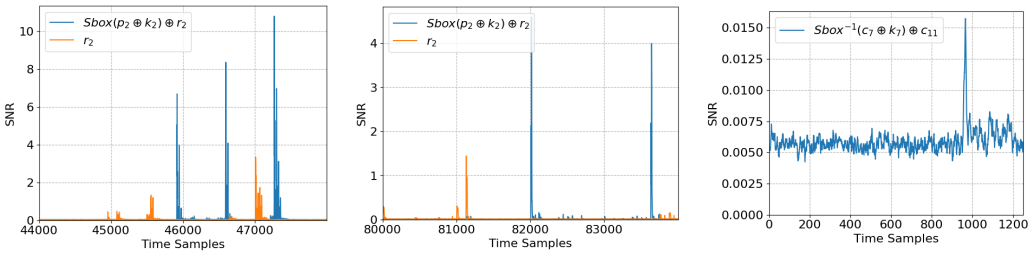
ASCAD_F: This dataset version has a fixed key and consists of 50 000 traces for profiling and 10 000 for the attack. Note that traces with 700 features (requires knowledge of r mask share) are commonly used in related works. To make our work closer to realistic settings, we increase the time window including the signal-to-noise ratio (SNR) peaks of both secret shares $s_{r,2} = Sbox(p_2 \oplus k_2) \oplus r_2$ and r_2 (shown in Figure 2a). Finally, we select a time window with 4 000 features, corresponding to the processing of key byte 3, the first masked key byte.

ASCAD_R: This dataset version has random keys, with 200 000 traces for profiling and 100 000 for the attack. Similarly, instead of attacking traces with 1 400 features that rely on knowledge of r mask share (commonly used in literature), we extend the pre-selected window to 4 000 features corresponding to the processing of third masked key byte based

on SNR of the *Sbox* output. The corresponding SNR is shown in Figure 2b.
We use 50 000 traces for profiling, and 10 000 traces for the attack for both datasets.

2.4.2 AES_HD

This dataset is first introduced in [KPH+19], targeting an unprotected hardware implementation of AES-128 written in VHDL in a round-based architecture. Side-channel traces were measured using a high sensitivity near-field EM probe, placed over a decoupling capacitor on the power line on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board. In this paper, the Hamming distance (HD) leakage model is used and it considers $Sbox^{-1}(c_7 \oplus k_7) \oplus c_{11}$ in the last AES round. 45 000 traces are used for profiling, and 5 000 traces are used for the attack. Each trace has 1 250 features. The SNR is shown in Figure 2c. The dataset is available at http://aisylabdatasets.ewi.tudelft.nl/aes_hd.h5.



(a) ASCAD_F. **(b)** ASCAD_R. **(c)** AES_HD.

Figure 2: SNR of the three datasets.

2.5 Leakage Models and the Number of Classes

Our work considers three leakage models:

1. **The Hamming Weight (HW) and Hamming Distance (HD) leakage models.** For the Hamming weight leakage model, the attacker assumes the leakage is proportional to the sensitive variable's Hamming weight. For the Hamming distance leakage model, the attacker assumes the leakage is proportional to the XOR of two sensitive variables' Hamming weights. These leakage models result in nine classes for a single intermediate byte for the AES cipher ($c = 9$).
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. This leakage model results in 256 classes for a single intermediate byte for the AES cipher ($c = 256$).

3 Related Works

Chari et al., in their seminal work in 2002, proposed the template attack, representing the beginnings of profiling SCA [CRR02]. While powerful, the template attack also relies on unrealistic assumptions: an unlimited number of profiling traces and noise following the Gaussian distribution [LPB+15]. Afterward, Schindler et al. proposed stochastic models where the authors approximated the real leakage function within a suitable vector subspace [SLP05]. Next, to resolve the issues stemming from the insufficient number of measurements per class for TA, it is also possible to pool all covariance matrices into a single one [JW02]. For instance, Choudhary and Kuhn investigated the pooled template attack and achieved performance improvements, both in terms of the extracted information and computational cost [CK13].

For a number of years, those techniques represented state-of-the-art for profiling SCA. Besides good attack performance, the limited hyperparameters make them easier to deploy in real-world cases. Nevertheless, it is commonly necessary to conduct a feature engineering phase to reduce the number of points of interest by using, e.g., machine learning-based feature selection [PHJB19], dimensionality reduction like Principal Component Analysis (PCA) [WEG87, APSQ06, BHvW12], Linear Discriminant Analysis (LDA) [SA08], or Sum of Squared Pairwise T-differences (SOST) [GLRP06].

Several years later, machine learning-based SCA became popular due to many results surpassing 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 generally favorable compared to the previous ones (e.g., template attack), the complexity of running such attacks was higher. Indeed, the best attack performance could only be achieved when hyperparameters in machine learning techniques are properly tuned. At the same time, feature engineering techniques are commonly required in the same manner as before to reduce computational complexity.

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 hyperparameters to tune than other techniques in profiling SCA, increasing the complexity of deploying 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), it is possible to 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]. Starting from the work from Zaid et al. [ZBHV19], Wouters et al. showed how to reach similar attack performance with data regularization and even smaller neural network architectures [WAGP20]. 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 improved state-of-the-art results.

Lu et al. made a significant step forward in the deep learning-based SCA as they investigated the performance of deep learning with raw traces (while the previous works actually considered pre-selected windows of features) [LZC+21]. Their results showed even better attack performance but at the cost of significantly more complex neural networks (e.g., having around 30 layers). Finally, Perin et al. showed how simple re-sampling of raw traces could result in extremely powerful attacks (requiring only a single attack trace)

while using simple neural networks with only a few hidden layers [PWP21b].

All those works have in common that they design a specific architecture for each dataset and leakage model. For some works, i.e., [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 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.

4.1 Similarity Learning and Triplet Network

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 [HA15]. Triplet network was evolved from the Siamese network [MKR16, GFZ+17] and was 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 3. A triplet input consists of three samples [1]: positive, anchor, and negative. Positive and anchor samples have the same label i, while that label is different from the negative samples. By training the deep network with the shared weights, three embeddings [2] (Emb_p, Emb_a, and Emb_n), corresponding to their input are outputted by the deep network and used for the triplet loss calculation. Weight vectors are updated using shared architecture during back-propagation. During training, we follow the online triplet mining method proposed in [SKP15], meaning that triplets are generated in real-time within a training batch. Compared with offline triplet mining, which fits the manually-created triplets to the network, the randomly-generated triplets increase the chance to find triplets with high triplet loss, thus speeding up the learning process.

An embedding represents a (relatively) low-dimensional space into which high-dimensional vectors can be translated. Ideally, an embedding would capture some input semantics by placing semantically similar inputs close together in the embedding space. A triplet model aims to extract these features while enlarging their inter-class differences. The conventional triplet loss function is defined in Eq. (2). The evaluation and benchmark between different loss functions is presented in Section 5.2.1. Among all of the considered loss functions, triplet loss performs the best.

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

where d denotes the Euclidean distance [3] between two feature vectors. a, p, and n stand for anchor, positive (with the label same as the anchor), and negative samples (with a label different from the anchor); $margin$ is enforced between the positive and negative pairs.

Based on the loss definition, there are three categories of triplets:

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

[1] For SCA, samples are leakage traces.

[2] For SCA, the embeddings are extracted features used for attacks.

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

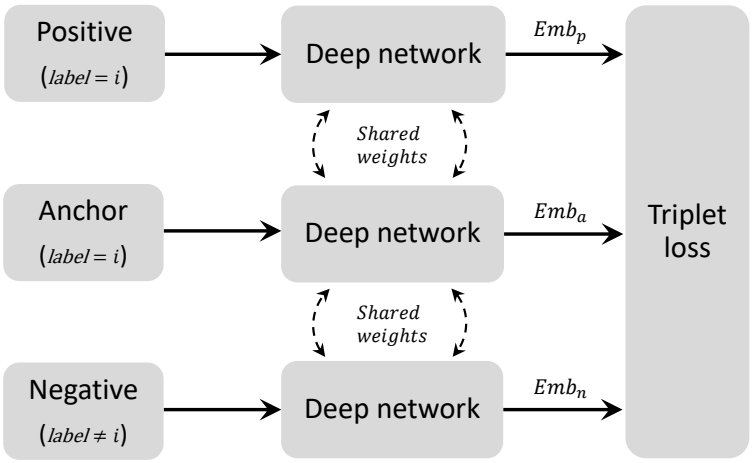


Figure 3: The structure of the triplet network. Each deep network is identical to the others. From the implementation perspective, any of these networks can be used to generate embeddings for anchor, positive, and negative inputs.

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, 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. 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 in a reasonable range, leading to more representative extracted embeddings features. We set the margin to 0.4 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 $margin$ is discussed in Section 5.2.3.

4.2 Triplet Loss with Hybrid Distance

Based on Eq. (2), once the anchor's label is set, the rest of the samples can be binary classified based on their label: positives and negatives. However, these embedding-based semi-hard triplets ignore the diversity of labels in negatives. Indeed, for a dataset with c classes, negatives contains $c-1$ classes. Within all embedding-based semi-hard triplets, if one can use negative's label information to find negatives that are potentially closer to the anchor than other negatives, the newly formed (semi-hard) triplets could include negatives that could be more 'difficult', thus leading to more efficient learning. From the classification perspective, focusing on differentiating with neighboring clusters would help in improving classification performance.

Unfortunately, for the triplet learning tasks such as images or audio feature extraction, it is challenging to judge the similarity between the anchor and negatives based on their labels [HA15, CR19]. For instance, imagine images with the labels 'Alice', 'Bob', and 'Eve'. One can hardly tell which two clusters are similar by only seeing the name. On the other hand, the correlation between label distance and embedding distance is stronger for SCA. Indeed, the SCA's labels are determined by intermediate data processed by the target, and we apply leakage models to these labels to simulate and correlate with measured leakages.

Naturally, a smaller label distance may indicate similar leakage traces (smaller feature distance). For instance, as defined in Section 2.5, the Hamming weight leakage model assumes that the leakage is proportional to the sensitive variable's Hamming weight. For a device that leaks byte-wise Hamming weight, intermediate values with closer Hamming weight values would generate similar leakages.

Following this, we optimize the distance metric (Euclidean distance) to enforce the triplet learning based not only on embedding distance but also on label distance. We denote it as Hybrid Distance. The newly proposed embedding distance calculation method is defined in Eq. (3).

$$Hybrid\ Distance = \frac{d(a_{l_a}, b_{l_b})}{\alpha^{d^{'}(l_a, l_b)}}, \alpha \in (0, 1]. \tag{3}$$

Here, $d(a_{l_a}, b_{l_b})$ stands for the squared Euclidean distance between embedding a and b with their corresponding labels l_a and l_b (determined by the used leakage model). $d^{'}(.)$ denotes the normalized Euclidean distance between labels (ranges from zero to one); α is a constant that need to be tuned (detailed evaluation in Section 5.2.1). Following Eq. (3), the Hybrid Distance ranges from $d(a_{l_a}, b_{l_b})$ to $d(a_{l_a}, b_{l_b})/\alpha$ based on the label distance. When α equals one, the squared Euclidean distance is calculated.

An illustration of the conventional and newly proposed embedding distance calculation methods is shown in Figure 4. a, p, and n are used to represent anchor, positive, and negative samples, respectively; the corresponding labels are denoted by their subscript i, $i+1$, and $i+2$. The *margin* range is highlighted in pink. As defined in Eq. (2), only negative samples within this range can be counted as the semi-hard triplet and used later for learning. The left graph indicates the conventional method where the embedding distance is purely based on the extracted features; the right graph takes into consideration the label distance so that n_{i+2} is pushed out of the *margin* range. Consequently, the triplet model will learn from n_{i+1} that is semi-hard both embedding-wise and label-wise.

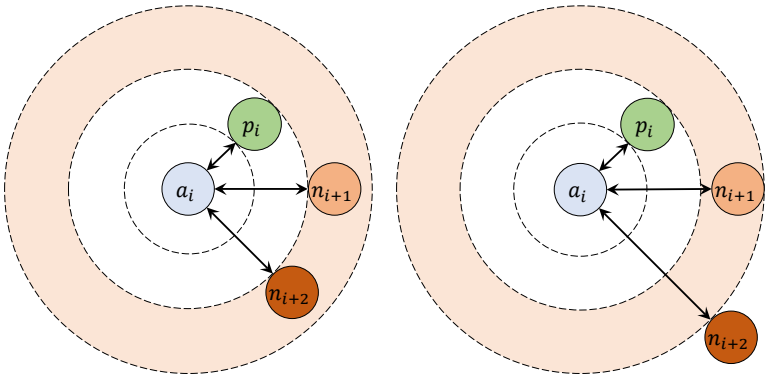


Figure 4: Comparison of two embedding distance calculation methods. Compared with the Euclidean distance (left), the Hybrid Distance (right) introduces a larger distance value when the label distance increases.

4.3 Attack Scheme

The correctly trained triplet model outputs embeddings with a larger 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 5.

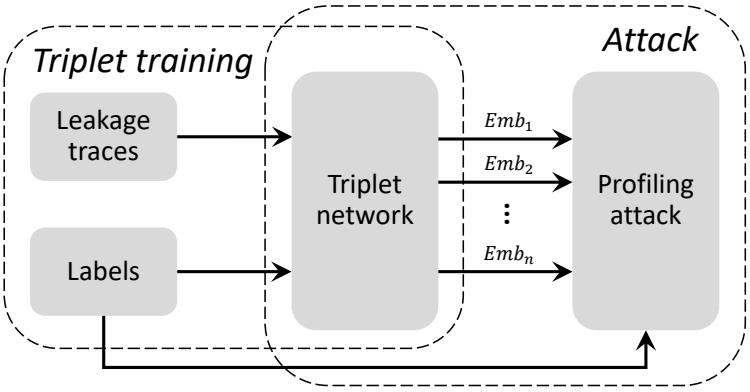


Figure 5: Triplet-assisted profiling attack.

Compared with the traditional dimensionality reduction methods such as PCA 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 helpful for the SCA attack. Considering LDA and SOST, the triplet network combines features in a nonlinear manner, which is beneficial when the leakage traces are noisy or protected by countermeasures.

Additionally, since it is based on constructing 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 Gaussian-distributed embeddings with a greater inter-class difference, thus leading to more separated PDFs. As a result, it can help to retrieve the key with fewer attack traces. Therefore, after training the triplet network, we use the extracted embeddings and corresponding labels to perform the template attack. Since our attack scheme is partially based on deep learning, better neural network tuning will help to achieve powerful attack performance. Still, with a single model, we demonstrate that our method is more robust than conventional DL-SCA toward hyperparameter and datasets changes, thus reducing the bar for launching such attacks. Besides, note that template attack is only one of many methods that can be used for attack. Still, we believe template attack is a more general attack method considering the difficulties of classifying the leakages protected by countermeasures and tuning the hyperparameters.

4.4 Neural Network Architectures

The main body of the triplet network is designed based on the VGG neural network [SZ14]. The design principle from related works [KPH+19, BPS+20] is applied to tune the specific hyperparameters. The neural network tuning is based on the combination of different hyperparameters to reach the best attack performance on all test settings (datasets, leakage models, noise resilience). The search space is listed in Table 1. Note that the architecture of the triplet model is flexible. In Section 5.1.2, we modify different state-of-the-art models to build triplet networks and reach outstanding performance with minimal training effort.

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 and 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 (see Section 5.2.3 for

Table 1: Hyperparameters search space for the triplet network.

Hyperparameter	Options
Convolution layers	1 to 13 in a step of 1
Convolution size	1 to 128 in a step of 1
Pooling size/stride	2 to 80 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
Margin	0.2 to 1 in a step of 0.2
Loss function	RMSProb, Adam
Batch size	32 to 512 in a step of 32
Training epoch	1, 5, 10, 15, 20, 25, 30

detailed discussion). We set the size of the embedding to 32 based on the grid search results (discussion in Section 5.2.2). Besides, we use average pooling as it performs better for the tested datasets [BPS+20, WP21]. *SeLU* is used as the activation function to avoid vanishing and exploding gradient problems [KUMH17]. To provide a sufficient number of valid triplets per batch, the batch size is set to 512 for all experiments. The optimizer is *Adam* with a learning rate of 5e-4. The detailed description of the neural network is listed in Table 2.

Table 2: Triplet architectures used in the experiments.

Layer	Kernel number/size	Pooling stride/size	Neurons
Conv+AvgPooling	64/15	15/15	-
Conv+AvgPooling	128/3	2/2	-
Dense	-	-	32

To verify that the reported attack performance is due to the proposed attack scheme and not just to a choice of neural network architecture that happens to suit the evaluated attack scenarios, the model presented in Table 2 is directly used for profiling by adding one additional prediction layer. As a result, the attack performance becomes significantly worse than state-of-the-art attack results.

4.5 The Environment

The machine learning models were implemented in Python version 3.6, using TensorFlow library version 2.4.1. The model training algorithms were run on an Nvidia GTX 1080 graphics processing unit (GPU), managed by Slurm workload manager version 19.05.4.

5 Experimental Results

This section evaluates our attack scheme from two aspects: side-channel attack performance and triplet hyperparameters' influence. The analysis is conducted on three publicly available datasets described in Section 2.4. We consider the HW, HD (for AES_HD), and ID leakage models. The training epoch is set to one, which requires around 20 seconds of training time. The detailed discussion about the required number of training epochs is in Section 5.2.4.

To evaluate the attack performance, we report the number of traces required to reach GE equal to zero, which is denoted as T_{GE0}. T_{GE0} metric is derived from guessing entropy, aiming at evaluating the key recovery capacity of profiling models by setting a limited

number of attack traces. Specifically, 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 guessing entropy equals zero for different settings, we can better estimate the attack performance by evaluating the required number of attack traces to reach it.

The algorithmic randomness stemming from the weight initialization for neural networks could have a significant impact on the attack performance [WPP21]. Besides, simulating noise (Section 5.1.2) with different random seeds would cause attack performance fluctuation. To provide representative results and a fair benchmark, all considered test scenarios (datasets, leakage models, deep learning models, dimensionality reduction techniques) in the following section are trained/executed and attacked 20 times independently. The *medium*-performing model is used to represent the attack efficiency in the following sections.

5.1 Performance Evaluation

First, we show results for the original (publicly available) datasets, and afterward, we test the perturbation resilience with different desynchronization levels.

5.1.1 Attack Capability

The attack performance of triplet attacks is benchmarked with the state-of-the-art MLPs [WPP20] and CNNs [ZBHV19, RWPP21, PCP20] models (SOTAs). Note that those neural networks are designed for the pre-selected windows of features (700 for ASCAD_F and 1 400 for ASCAD_R). Since our work adjusts their input layers with dimensions defined in Section 2.4 [4], the attack performance of the modified architectures does not correspond to the different numbers given in the respective works. Still, we expect targets can still be broken and even reach better attack performance [LZC+21]. The references are kept in the tables for readability.

Besides offering insight into how these networks perform on (much) longer traces, various dimensionality reduction techniques, such as PCA, LDA, SOST, and autoencoder (AE) [WP20] are considered in this paper. The feature size is set to be optimal [5]. The implementation details for the autoencoder are presented in Appendix A. The extracted features/latent space are then used for the template attack.

The benchmarks for all datasets with the T_{GE0} metric are shown in Tables 3, 4, and 5. Here, '-' indicates that GE does not reach zero with a given number of attack traces. The best values are denoted in **bold** font.

For ASCAD_F and ASCAD_R, the increased number of input features leads to similar or even significantly better performance compared to the original papers (SOTA model from [RWPP21] with the ID leakage model now requires only seven traces to break the target). Still, the proposed attack scheme generates the best performance in four out of five scenarios with a single model presented in Table 2, confirming the generality and transferability of the triplet model and the attack method. On the other hand, 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. Although LDA and SOST also consider the labels, the high sensitivity to the embedding size (i.e., they may only work with a specific embedding size setting), the linear combination of raw features, and the absence of the mask knowledge [BCS21] could be the reason

[4] We also evaluate the performance with the pre-selected windows of feature with sizes 700/1 400 to provide a better comparison with related works.

[5] We experimentally test multiple feature sizes ranges from 8 to 128. The one with the best attack performance is considered to be optimal. The detailed settings for each dataset and leakage model are listed as follows, and the results for the HW and ID leakage models are separated by '/'. ASCAD_F: PCA=16/16; LDA=8/128; SOST=32/64; AE=16/16. ASCAD_R: PCA=16/16; LDA=8/32; SOST=128/8; AE=16/16. AES_HD: PCA=8, LDA=8; SOST=8; AE=16.

for their mediocre performance. Finally, for [PCP20], the hyperparameter space used to generate ensembles could be non-optimal due to an increased number of input dimensions, thus leading to unsuccessful attacks.

Besides the results listed in the tables, we verify the generality of the proposed method by varying the input dimension size. Specifically, we attack ASCAD_F and ASCAD_R with commonly-used feature settings (700 and 1 400). As a result, for ASCAD_F, a median model requires 353 (HW) and 632 (ID) traces to break the target. For ASCAD_R, the required number of traces for two leakage models are 533 and 1 228. Consequently, we can observe that more attack traces are required when the input dimension is smaller. Still, the secret information can be retrieved with one-epoch training.

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

	[ZBHV19]	[WPP20]	[RWPP21]	PCA	LDA	SOST	AE	This work
HW	174	225	294	187	-	1 123	239	**159**
ID	191	160	**7**	193	-	5 294	183	64

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

	[PCP20]	[WPP20]	[RWPP21]	PCA	LDA	SOST	AE	This work
HW	-	864	519	416	-	-	686	**197**
ID	-	3 144	4 244	577	-	-	1 183	**188**

Table 5: Benchmark the attack performance (T_{GE0}) with the AES_HD dataset.

	[KPH+19]	[ZBHV19] [6]	PCA	LDA	SOST	AE	This work
HD	-	4 415	-	19 23	1 860	-	**1 768**

The template attack used as the final stage of triplet attacks could also be switched to other profiling attack methods. For instance, the trained triplet model can be used for transfer learning: adapting one or more hidden layers and a prediction layer with additional training epochs could also break the target. Still, we believe a template attack represents a robust and straightforward solution. In addition, we also tested the pooled template attack on features extracted by the triplet network. This technique fails to break the protected dataset (ASCAD_F and ASCAD_R) with the given number of attack traces but performs very well on AES_HD (reaches zero GE with around 600 attack traces). Indeed, pooled template attack can only reach a higher precision estimate if each cluster has a different mean but the same covariance matrix. The introduction of the masking countermeasure in ASCAD_F and ASCAD_R would break this assumption, thus leading to worse attack performance.

5.1.2 Perturbation Resilience

The well-synchronized traces significantly improve the correlation of the intermediate data and trace values. Therefore, the alignment of the traces is an essential step for the side-channel attack. Two desynchronization levels (50 and 100) were simulated and tested to show the effect of trace desynchronization. The model is trained for one epoch for

[6] Although using the same profiling model, our attack settings, such as round key and label calculation, are different from Zaid *et al.* [ZBHV19] for AES_HD (they assume the subkeys of the last round are all zeros due to the lack of plaintext and ciphertext). This causes performance variation when compared with the original paper. Since the intermediate data we used is the real data corresponding to the cryptographic calculation, we suggest using these result as a reference.

triplet attack, aligned with the previous section. To counter the added noise, the kernel size of the first convolution layer and the pooling size/stride of the first pooling layer is increased to 55 for all test settings based on grid search results. Finally, the methods that failed in the previous section are excluded from the experiments, as the addition of noise further increases the attack difficulties.

For AE, we train with noisy-noisy traces pair as our goal is to test the feature extraction capability of AE. [7]Consequently, this method failed in key recovery with all noise levels.

We modify SOTAs from [RWPP21, ZBHV19] to build triplet models and compare the noise resilience of conventional deep learning-based method and triplet-based training method. More specifically, all dense layers were replaced by the embedding layer with a size of 32. The rest of the settings are aligned with previous experiments.

The median attack performance over 20 independent training is listed in Tables 6, 7, and 8, the corresponding models are referred as median model. The perturbation in the time domain significantly reduces the attack performance with the conventional deep learning-based methods. Meanwhile, since the leakage traces are not perfectly aligned (common in realistic settings), valid features become more difficult to extract with the dimensionality reduction techniques. On the other hand, with only one-epoch training, the triplet-based method shows its perturbation resilience: the triplet-based SOTA attacks break the target in some attack scenarios, while their counterparts failed in all test cases.

In addition, we tested the noise resilience of the triplet attack with a reduced number of input features for ASCAD_F (700) and ASCAD_R (1 400). For both ASCAD_F and ASCAD_R, except for the desynchronization level 100 and the ID leakage model, the median model can retrieve the secret information within 10 000 traces. More precisely, for ASCAD_F and desynchronization 50, we require 751/6 097, while for desynchronization 100, we need 2 641/- attack traces. For ASCAD_R and desynchronization 50, we need 1 449/8 478 attack traces, and for desynchronization 100, 7 936/- attack traces. Therefore, we can confirm the superior perturbation resilience of the triplet-based attack method.

Table 6: Benchmark the attack performance (T_{GE0}) with the ASCAD_F dataset perturbed with desynchronization. Attack results for the HW and ID leakage models are separated by '/'.

Noise	[ZBHV19] [8]	[WPP20]	[RWPP21]	PCA	SOST	Triplet-[RWPP21]	This work
50	-/-	-/-	-/-	-/-	-/-	-/2 850	**251/191**
100	-/-	-/-	-/-	-/-	-/-	-/-	**382/582**

Table 7: Benchmark the attack performance (T_{GE0}) with the ASCAD_R dataset perturbed with desynchronization. Attack results for the HW and ID leakage models are separated by '/'.

Noise	[WPP20]	[RWPP21]	PCA	Triplet-[RWPP21]	This work
50	-/-	-/-	-/-	7 805/3 715	**2 251/3 385**
100	-/-	-/-	-/-	**-/-**	**6 386/9 932**

[7]Training with noisy-clean traces pairs (as done in denoising autoencoder approach [WP20]) may reach better attack performance. However, this method is not considered here because clean traces are difficult to obtain in realistic settings.

[8]For consistency, the same model from [ZBHV19] used in the previous benchmarks is considered here as well. In addition, we have also the models optimized for different levels of desynchronization. As a result, it reaches comparable performance with our attack method.

Table 8: Benchmark the attack performance (T_{GE0}) with the AES_HD dataset perturbed with desynchronization.

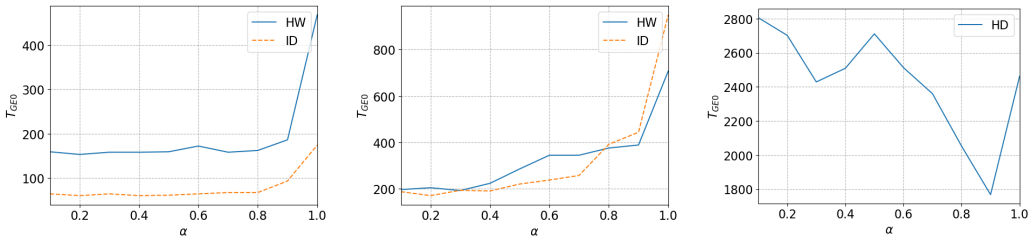
Noise	[ZBHV19]	LDA	SOST	Triplet-[ZBHV19]	This work
50	-	-	-	-	**4 662**
100	-	-	-	-	-

5.2 Hyperparameter Evaluation

This section concentrates on evaluating several critical hyperparameters for the triplet network model. Besides better understanding their influence on the attack performance, we hope the detailed evaluations could serve as guidelines for potential users to design their triplet models. This section considers the setting where each trace has 4 000 features for ASCAD both versions and 1 250 for AES_HD.

5.2.1 Loss Function

Recall that the proposed loss function introduces a new hyperparameter α. To better understand the effect of this hyperparameter, we tune α from 0.1 to 1 in a step of 0.1 and attack all considered datasets. Note that the Hybrid Distance is equivalent to the Euclidean distance when α equals one. The rest of the training settings are aligned with the previous sections.



(a) ASCAD_F. **(b)** ASCAD_R. **(c)** AES_HD.

Figure 6: The effect of α.

The attack results for three datasets are shown in Figure 6. First, we can confirm that introducing α and Hybrid Distance helps increase the attack performance. On the other hand, the optimal α varies for each dataset: the best α is 0.9 for AES_HD, while this value drops to 0.1 for the other two datasets. For AES_HD, the attack performance becomes worse than the default distance metric ($\alpha = 1$) when α is below 0.6. Indeed, although smaller α strengthens the influence of the label distance, it reduces the number of valid triplets to be learned. As a result, it causes quick overfitting (ASCAD both versions) or the degradation of attack performance (AES_HD). Note that each alpha parameter is averaged from 20 independent tests, and we expect limited performance fluctuation even with a greater resolution of the tested α value.

Next, we benchmark the attack performance of the proposed loss function with some other loss functions used for similarity learning. More specifically, we considered Contrastive loss [HCL06], Lifted Structure loss [OSXJS16], Pinball loss [SC11], and Hard triplet loss. Besides, the Semi-hard triplet loss with the default distance metric (Euclidean distance) is included in this benchmark. Loss functions that contain hyperparameters are tuned to be optimal [9]. The model and training hyperparameters are kept the same.

[9] We experimentally test multiple τ (for Pinball loss) and *margin* (for the rest except the Hard triplet loss) ranging from 0.2 to 1.0. The one with the best attack performance is considered to be optimal. The

Table 9: Benchmark different loss functions with T_{GE0}. Attack results for the HW and ID leakage models are separated by '/'.

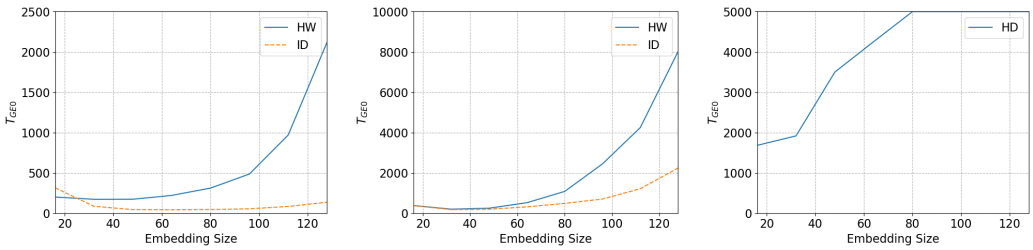
	Contrastive	Lifted Structure	Pinball	Hard	Semi-hard	This work
ASCAD_F	4174/230	744/324	432/332	-/8600	296/124	**159/64**
ASCAD_R	5376/904	999/1457	651/983	-/-	775/713	**197/188**
AES_HD	3 849	3 486	3 279	-	2 910	**1 768**

The attack results are shown in Table 9. Although the model trained with almost all loss functions can generate features that lead to zero guessing entropy within the given number of attack traces, our proposed loss function outperforms all considered loss functions in all attack scenarios. Specifically, one can observe a significant improvement in the attack performance from the Hard triplet loss to the Semi-hard triplet loss, indicating the importance of learning from the semi-hard triplets. On top of that, besides the embedding distance, we introduce label distance in the distance metric calculation. With the help of the Hybrid Distance metric proposed in this paper, our loss function reaches the best attack performance. The attack performance with other loss functions could increase with more training epochs or profiling traces, but the computation complexity is increased as a trade-off.

5.2.2 Embedding Size

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. 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 for the ASCAD dataset, so higher values would trigger a singular matrix problem, and the template attack would fail.

The embedding tuning results are shown in Figure 7. From the results, a larger embedding size could lead to worse attack performance. Indeed, the additional features introduced by a larger embedding size could harm the overall attack performance as they may contain noise learned from the irrelevant raw features. Moreover, more embedding features would either dilute the features extracted by the triplet model or require more training effort, thus reducing the attack performance.



(a) ASCAD_F. **(b)** ASCAD_R. **(c)** AES_HD.

Figure 7: The effect of embedding size.

When evaluating smaller embedding sizes, size 32 performs comparable (Figure 7b) or even better than size 16. As expected, an overly small embedding size would not

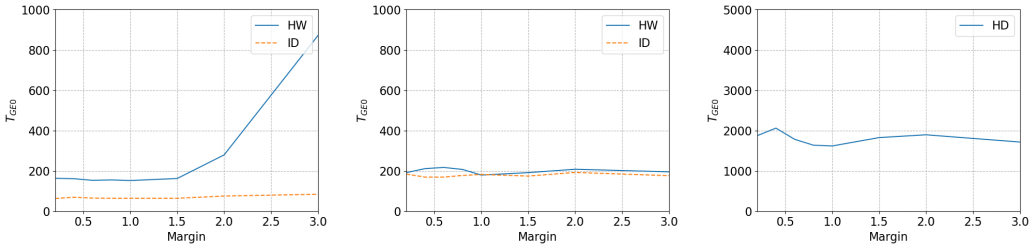
detailed settings for each loss function and leakage model are listed as follows, and the results for the HW and ID leakage models are separated by '/'. ASCAD_F: Contrastive=1.0/0.2; Lifted Structure: 0.4/0.2; Pinball: 0.2/0.2. ASCAD_R: Contrastive=1.0/0.6; Lifted Structure: 0.8/0.6; Pinball: 1.0/0.4. AES_HD: Contrastive=0.8; Lifted Structure: 0.8; Pinball: 1.0.

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. Since it is common to conduct feature engineering when running the template attack, we do not consider the effort required to tune the embedding size more substantial.

5.2.3 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 in the previous sections. The minimum *margin* is set to be 0.2, which is aligned with [SKP15].

The experimental results are shown in Figure 8. From the results, the increase of the *margin* value could slightly degrade the attack performance in some cases (Figure 8a). 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. Still, compared with the effect of the size of the embedding shown in Figure 7, the *margin* has a limited effect on the attack performance.



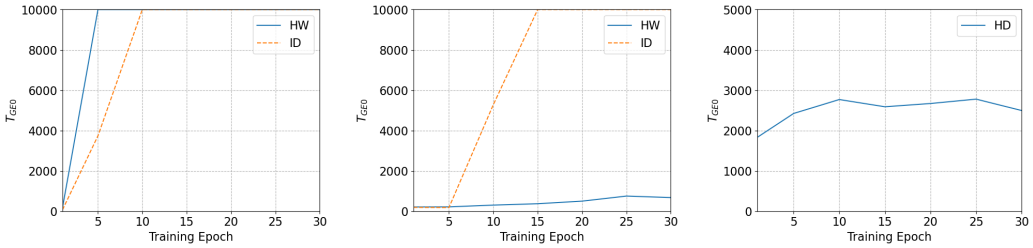
(a) ASCAD_F. **(b)** ASCAD_R. **(c)** AES_HD.

Figure 8: The effect of margin.

5.2.4 Training Epochs

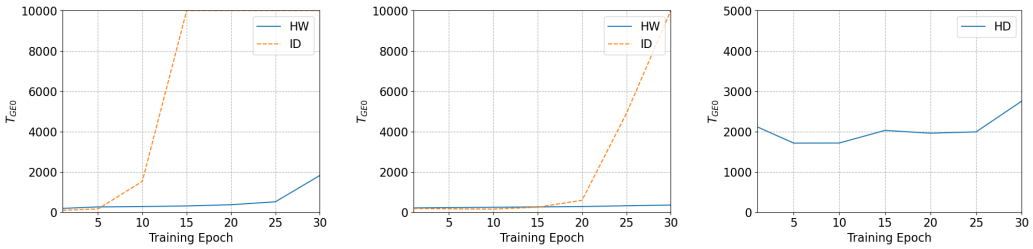
Accuracy is one of the core metrics to evaluate a deep learning model in the deep learning domain. Most of the individual examples must be correctly classified to reach high accuracy. As a consequence, when using the triplet network to extract the features, a high training effort is required to extract meaningful embeddings (1 000 to 2 000 CPU hours according to [SKP15]). This section explores 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 mentioned, each epoch training requires around 20 seconds with a single GPU.

As shown in Figure 9, more training epochs lead to worse attack performance for ASCAD_F and ASCAD_R, indicating they suffer more from overfitting. Indeed, although the introduction of α in the distance metric increases the difficulties of the selected triplets, it reduces the number of valid triplets that can be used for learning. If the profiling traces are limited or well-protected by countermeasures, a too small alpha could significantly contribute to the observed effect. The most straightforward approach to prevent/delay such an effect is to increase the pooling size/stride of the triplet model. The triplet model could focus on more general features from input by doing this. As a demonstration, we increase the pooling size/stride of the first pooling layer from 15 to 60 in Table 2. As shown in Figure 10, although overfitting still occurs, the performance remains stable with more training epochs without a performance loss.



(a) ASCAD_F. **(b)** ASCAD_R. **(c)** AES_HD.

Figure 9: The effect of training epoch.



(a) ASCAD_F. **(b)** ASCAD_R. **(c)** AES_HD.

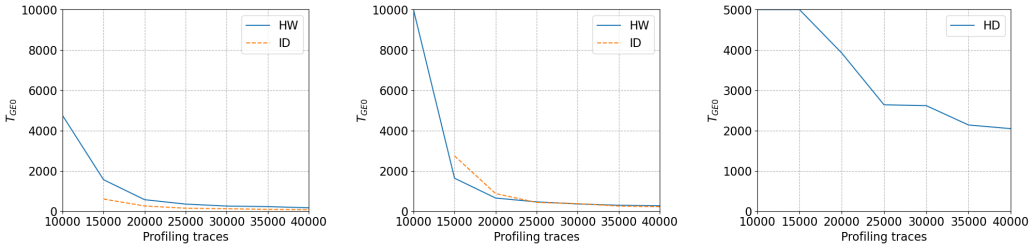
Figure 10: The effect of training epoch with the increased pooling size and stride.

Compared with other deep learning attacks that generally require more than 50 training epochs [BPS+20, KPH+19, ZBD+21], our triplet-based method dramatically speeds up the feature learning process. Indeed, the DL-based SCA attacks aim at training an efficient classifier that should work well in both efficient feature extraction and precise classification. To reach both goals, careful hyperparameter tuning and an increased training effort are required. On the other hand, the triplet-based method splits the feature extraction and classification into separate steps (same as the conventional profiling SCA attack method). Therefore, the task of the triplet model is much simpler and straightforward: transfer and combine raw features that can maximize the embedding distance between different clusters. The triplet model can extract meaningful features more efficiently with the SCA-optimized distance metric. For the same reason, the hyperparameter's flexibility in designing a triplet model is significantly increased.

5.2.5 Training Set Size

Similar to other deep learning-based methods, 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 5 000 traces. The results are shown in Figure 11. Note that for the ID leakage model, due to the lack of training data, training with 10 000 traces always leads to an unsuccessful template attack (singular matrix), so the results are not presented.

In all 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. For the HD leakage model, this trend



(a) ASCAD_F. **(b)** ASCAD_R. **(c)** AES_HD.

Figure 11: The effect of training set size.

extends to 25 000 profiling traces. Indeed, with a 10 000 training set, the smallest cluster has very limited samples (i.e., 35 for HW and 20 for ID). 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 poor performance or even attack failure for both leakage models. On the other hand, the performance increases slower when the traces are above 20 000, indicating that the triplet model reaches its maximum feature extraction capability.

5.3 What Can We Learn?

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

- When attacking the original datasets (without noise addition), with a single deep learning model, triplet-assisted template attack performs comparably to or better than the state-of-the-art models from the literature that consider pre-selected windows of features.
- With a single deep learning model, triplet-assisted template attack can be more resilient to noise in horizontal and vertical dimensions than deep learning-based attack and commonly used feature engineering techniques.
- The optimal α values are different for different datasets, but starting with a larger value (i.e., 0.9) would be a good starting point.
- The newly proposed Hybrid Distance helps the semi-hard loss function to outperform other loss functions for SCA tasks.
- 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.
- 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*), a 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.
- The number of training traces has a significant impact on the attack performance.
- The triplet network is highly efficient in extracting the leakage-related features. One-epoch training is sufficient to train a triplet network for the evaluated datasets.

6 Conclusions and Future Work

This work investigates how to extract useful features from side-channel leakages for efficient template attacks. To accomplish this, we use the concept of triplet networks that have the task of finding a well-performing embedding based on the input traces. We conduct experiments on three publicly available datasets and three leakage models, showing that our

deep learning-assisted template attack can effectively break targets (even with the addition of noise) with significantly reduced training effort. This result is especially significant as we compared it 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 desynchronization. Finally, we systematically investigate the influence of multiple hyperparameters in the proposed attack scheme, which could be helpful in future research.

We plan to extend our approach to raw traces (and not pre-selected windows as used now). Since the raw traces are significantly larger (e.g., 100 000 features vs. 4 000 features), it would be interesting to explore the limitations of triplet networks. Additionally, we aim to explore the combinations of triplet networks with simpler machine learning techniques like the random forest or support vector machines. Finally, it would be interesting to see whether continuous encoding of labels would be beneficial for the proposed method.

Acknowledgements

This work received funding in the framework of the NWA Cybersecurity Call with project name PROACT with project number NWA.1215.18.014, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). Additionally, this work was supported in part by the Netherlands Organization for Scientific Research NWO project DISTANT (CS.019).
The authors thank our anonymous reviewers and our shepherd, Alexandre Venelli, for their valuable comments and suggestions.

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.

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ascad with a single side-channel trace. *Cryptology ePrint Archive*, 2021.

[BHvW12] Lejla Batina, Jip Hogenboom, and Jasper GJ van Woudenberg. Getting more from pca: First results of using principal component analysis for extensive power analysis. In *Cryptographers' track at the RSA conference*, pages 383–397. Springer, 2012.

[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.

[CR19] Anurag Chowdhury and Arun Ross. Fusing mfcc and lpc features using 1d triplet cnn for speaker recognition in severely degraded audio signals. *IEEE transactions on information forensics and security*, 15:1616–1629, 2019.

[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] Richard Gilmore, Neil Hanley, and Maire 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.

[GLRP06] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 15–29. Springer, 2006.

[HA15] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, 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'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[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.

[JW02] Richard Arnold Johnson and Dean W. Wichern. *Applied multivariate statistical analysis*. Prentice Hall, Upper Saddle River, NJ, 5. ed edition, 2002.

[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.

[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 Transactions on Cryptographic Hardware and Embedded Systems*, pages 235–274, 2021.

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

[OSXJS16] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4004–4012, 2016.

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

[PWP21a] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2021/1414, 2021. https://ia.cr/2021/1414.

[PWP21b] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *Cryptology ePrint Archive*, 2021.

[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.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 411–425. Springer, 2008.

[SC11] Ingo Steinwart and Andreas Christmann. Estimating conditional quantiles with the help of the pinball loss. *Bernoulli*, 17(1):211–225, 2011.

[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.

[WPP21] Lichao Wu, Guilherme Perin, and Stjepan Picek. On the evaluation of deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2021/952, 2021. https://ia.cr/2021/952.

[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.

A Autoencoder Implementation

The design of the autoencoder used in this work follows [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 the representation of compressed data. For an autoencoder, the dimension of the latent space is usually defined by the layer in the middle. The implementation details are listed in Table 10 (BN stands for the 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 50.

Table 10: Autoencoder architecture 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	-	8 to 128
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	-	-