Autoencoder Assist: An Efficient Profiling Attack on High-dimensional Datasets

(Full verson*)

Qi Lei[1], Zijia Yang[1], Qin Wang[2], Yaoling Ding[3], Zhe Ma[1] and An Wang[3]

[1] *Bank Card Test Center (National Fintech Evaluation Center)*, Beijing, China
leiqiuq@outlook.com,zjyangzijia@outlook.com,ma.z@bctest.com
[2] *CSIRO Data61*, Sydney, Australia
qinwangtech@gmail.com
[3] *Beijing Institute of Technology*, Beijing, China
dyl19@bit.edu.cn,wanganl@bit.edu.cn

Abstract.

Deep learning (DL)-based profiled attack has been proved to be a powerful tool in side-channel analysis. A variety of multi-layer perception (MLP) networks and convolutional neural networks (CNN) are thereby applied to cryptographic algorithm implementations for exploiting correct keys with a smaller number of traces and a shorter time. However, most attacks merely focus on small datasets, in which their points of interest are well-trimmed for attacks. Countermeasures applied in embedded systems always result in high-dimensional side-channel traces, i.e., the high-dimension of each input trace. Time jittering and random delay techniques introduce desynchronization but increase SCA complexity as well. These traces inevitably require complicated designs of neural networks and large sizes of trainable parameters for exploiting the correct keys. Therefore, performing profiled attacks (directly) on high-dimensional datasets is difficult.

To bridge this gap, we propose a dimension reduction tool for high-dimensional traces by combining signal-to-noise ratio (SNR) analysis and autoencoder. With the designed asymmetric undercomplete autoencoder (UAE) architecture, we extract a small group of critical features from numerous time samples. The compression rate by using our UAE method reaches 40x on synchronized datasets and 30x on desynchronized datasets. This preprocessing step facilitates the profiled attacks by extracting potential leakage features. To demonstrate its effectiveness, we evaluate our proposed method on the raw ASCAD dataset with 100,000 samples in each trace. We also derive desynchronized datasets from the raw ASCAD dataset and validate our method under random delay effect. We further propose a 2^n-structure MLP network as the attack model. By applying UAE and 2^n-structure MLP network on these traces, experimental results show that *all* correct subkeys on synchronized datasets (16 S-boxes) and desynchronized datasets are successfully revealed *within* hundreds of seconds. This shows that our autoencoder can significantly facilitate DL-based profiled attacks on high-dimensional datasets.

Keywords: Side-channel Analysis · Deep Learning · Autoencoder · Multi-layer Perceptron · Convolutional Neural Networks

*This paper has been accepted by ICICS'22.

1 Introduction

Side-channel analysis (SCA) exploits the weakness of cryptographic algorithm implementations from the view of physical information such as timing [Koc96], power consumption [KJJ99], and electromagnetic emanations [QS01]. When running a cryptographic algorithm, intermediate states and operations that are closely dependent on secret keys may cause leakage and be detected through SCA observation, severely threatening the security of users. In SCA, attack methods can be categorized into *profiled attacks* and *non-profiled attacks*. Profiled attacks indicate that a powerful adversary has access to an open clone cryptographic device for secret keys, side-channel traces, and unlimited computational power. The adversary firstly characterizes all types of secret information from a clone device (a.k.a., profiling phase), and then recovers secret data from the target device by utilizing their learned models (a.k.a., attack phase). Typical profiled attacks covers template attack [CRR02] and machine learning-based attacks [HGM+11, BL12, HZ12, LMBM13, GHO15, MPP16]. Non-profiled attacks refer to attacks where an adversary can only collect physical information from a target device. The abilities of adversaries in this type are weaker than those in profiled attacks. The attacker uses statistical analysis to derive secret information from side-channel traces. This line of attacks include simple power analysis (SPA), differential power analysis (DPA) [KJJ99], correlation power analysis (CPA) [BCO04] and mutual information analysis (MIA) [GBPV09] [BGP+11].

One of the most powerful side-channel attacks is proven to be the deep learning (DL)-based profiled attacks. This type of attack outperforms other types of attacks in the context of attack traces needed to recover keys [LPB+15] [MPP16]. Also, DL-based profiled attacks have been explored on many datasets that cover both unprotected datasets and protected datasets [MPP16, CDP17, PSB+18, CCC+19, ZBHV20, WP20, MDP20]. These datasets have accurately located *points of interest* (POIs). However, in a practical secure implementation, each trace includes tens of thousands of samples due to applied countermeasures. Directly applying neural networks on high-dimensional datasets is impractical because the design inevitably requires a very complicated network architecture and a large size of trainable parameters to learn key information. To the best of our knowledge, deep learning-based profiled attacks are absent of applications on high-dimensional datasets. To bridge this gap, a preprocessing of locating leakage samples and extracting features becomes a necessity.

In advance to accelerate the attacks, preprocessing techniques for POI selection have been explored in profiled attacks in many solutions.For instance, principal component analysis (PCA), linear discriminant analysis (LDA), and kernel discriminant analysis (KDA) are investigated in template attacks [APSQ06, CK13, CDP15, CDP17, EPW10] to improve the performance attacks. Similarly, long short-term memory (LSTM) autoencoder in machine learning-based profiled attacks [RAD20] has been applied on unprotected AES implementation to extract features. Additionally, convolutional layers are also considered being a feature extraction tool [ZBHV20]. However, all these solutions are applied on small datasets containing only hundreds of samples. It is insufficient in practical scenarios where each trace consists of large size of samples. An efficient tool to locate and extract critical features when performing attacks is desperately necessary. We thereby propose a new approach to preprocess high-dimensional datasets.

Our approach utilizes the undercomplete autoencoder (UAE). UAE can learn a lower-dimensional representation of the data by an encoder and correspondingly decodes it into the original input space by a decoder. The structure of the encoder and decoder can be designed as fully connected layers, convolutional layers, LSTM layers, etc. Our proposed solution can outperform previous methods due to a specialized design. Specifically, our method introduces an asymmetric structure into autoencoder design to reduce the training complexity. This design differs from traditional autoencoder solutions which have symmetric autoencoder structures as in [WP20, RAD20]. In our asymmetric model, the encoder and decoder have different structures. This provides benefits with flexible parameter settings

and independent combinations of training models. Therefore, to accelerate deep learning-based attacks on high-dimensional traces, we design a preprocessing and attack model suite, which consists of signal-to-noise ratio (SNR) analysis, autoencoder, and multilayer perceptron (MLP). Our proposed model can extract key features from high-dimensional traces. To demonstrate its effectiveness, we give experiments under our model and several parallel models on the raw ASCAD datasets to compare attack results before and after the feature extraction. Experimental results indicate that our method has significantly improved the attack performance. In a nutshell, the contributions of this work are listed as follows.

- We propose a novel trace preprocessing architecture to compress high-dimensional datasets. The preprocessing tool is composed of SNR analysis and asymmetric undercomplete autoencoder (UAE) that can extract critical trace features.
- We design a 2^n-structure MLP structure to perform profiled attack. Experimental results show that the MLP can perform successful attacks while existing attack models cannot reveal all correct keys before and after the feature extraction. Meanwhile, training a 2^n-structure MLP takes only $\frac{1}{42} \sim \frac{1}{3}$ the time compared with the state-of-the-art models.
- The preprocessing and attack architecture is confirmed on both synchronized and desynchronized traces. We introduce convolutional layers in UAE to confront desynchronization effect and further exploit the secret key. The compression rate by using UAE reaches **40x** on synchronized datasets and **30x** on desynchronized datasets.

Paper Structure. Section 2 introduces the raw ASCAD dataset for evaluation and provide dimension reduction techniques applied in SCA. The architectures of the asymmetric UAE and 2^n-structure MLP are presented in Section 3. Section 4 details the hyperparameter selection and shows experimental results on synchronized and desynchronized datasets. Section 5 concludes this work.

2 Primitives and Components

In this section, we firstly introduce the high dimensional traces: raw ASCAD dataset. ASCAD contains a large number of samples served for profiling attacks. Then, we provide details of dimension reduction techniques in SCA, covering SNR and autoencoders. Finally, we present neural networks used as attack models in profiled attacks. These techniques are the basic components of our established scheme.

2.1 Raw ASCAD Dataset

The ASCAD dataset [PSB+18] provides power traces produced by a masked AES-128 implementation on ATMega8515 microcontroller. It is used as a benchmark to evaluate machine learning techniques applied in the side-channel context. The ASCAD dataset provides 4 subsets in 2 conditions: fixed key encryption (with random delay $N_D = \{0, 50, 100\}$) and random keys encryption. Each trace in fixed key version has 700 samples, and in random key version 1400 samples. The leakage model in the ASCAD dataset is chosen to be S-box output, which is defined as,

$$\textit{S-box}(p[i] \oplus k[i]),$$

where $i \in \{1, ..., 16\}$, p represents plaintexts while k is secret keys (both are measured in byte).

The ASCAD fixed key dataset is trimmed from the raw ASCAD dataset, in which it contains the operation information of all 16 S-boxes in the first round of the AES. The raw ASCAD dataset consists of 60,000 traces and each trace is composed of 100,000 samples. It is split into a training set with 50,000 traces and a test set with 10,000 traces. The raw ASCAD dataset has no label for training. We thereby use each S-box output (16 in total) to tag traces. Then, we obtain 16 subsets for further training and attacks.

Besides, desynchronized subsets in ASCAD are generated via random delay technique. To check the jitter effect on machine learning techniques, Benadjila et al. [PSB+18] introduced a jitter parameter to desynchronize trace. Specifically, assume window maximum jitter is N_D time samples and the point of interest interval on the raw ASCAD dataset is $[N, N + L]$. Firstly, they set a maximum random delay value N_D (50 or 100). Then a random number $r < N_D$ is generated to trim an interval $[N + r, N + L + r]$ for each trace. Finally, they generated 2 desynchronized datasets, i.e., ASCAD_desync50 and ASCAD_desync100. A similar desynchronization technique is applied in our work to generate high-dimensional desynchronized datasets.

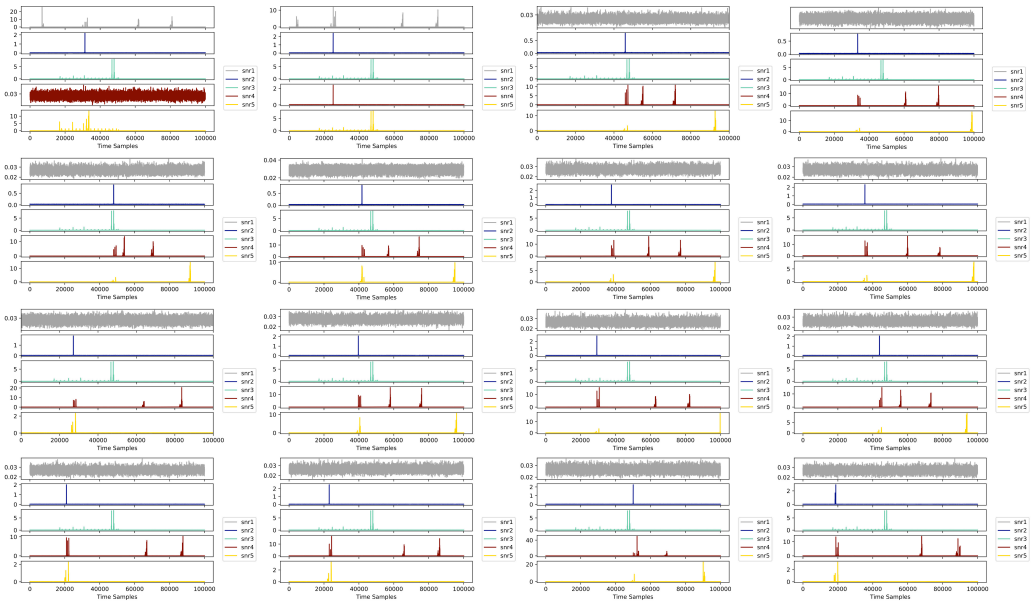

Figure 1: SNR of 16 S-boxes

2.2 Dimension Reduction Techniques in SCA

To find a possible leakage interval, SCA makes use of statistical tests (SNR, T-test) to detect leakage points and dimensional reduction skills (PCA, LDA, KDA, autoencoders) to reduce attack complexity. Our method combines SNR and autoencoder to address the high-dimensionality issue in SCA trace preprocessing. We introduce them as follows.

2.2.1 Signal-to-Noise Ratio Analysis

SNR has been applied to find POIs in the raw ASCAD dataset. We introduce five mainstream types of SNR defined in [PSB+18] (see Tab.1). We conduct the SNR analysis on the raw ASCAD dataset for 16 S-boxes separately (see Fig.1: Subgraphs from upper left to bottom right horizontally represent the 1^{st} to the 16^{th} S-box SNR analysis results.). Different types of SNR analysis results on each S-box can be used to infer a rough calculation

period for further attacks. The value of $snr1$ (unmasked S-box output) is generally low in most S-boxes, implying the first-order leakage is weak in tested traces. In contrast, $snr2$ (masked S-box output) and $snr4$ (masked S-box output in linear parts) have high correlation value that shows the real intermediates in AES traces. $snr3$, as a common mask constant, leads to high SNR peaks at the same position in 16 S-boxes. We can find that $snr3$ is a misleading signal when selecting the proper S-box operation period. $snr5$ shows high peaks of each S-box mask $r[i]$. Specifically, to illustrate how we locate POIs from original traces according to SNR analysis results, we use SNRs in the third S-box (see Fig.2) as an example. Firstly, we zoom in SNR analysis result to observe the same position SNR peaks excluding the misleading signal $snr3$. When $n, (n > 1)$ SNR show peaks at the same position (marked with a red square), we collect the highest values of each SNR type at the point $P_i, i = 1, ..., n$. Next, we use P_{middle}, where $P_{middle} = \frac{\sum P_i}{n}$, as the middle point to trim an interval for further analysis. In the third S-box, $P_{middle} = 45900$ and the trimmed interval (the number of samples denoted as L) is $(45900 - \frac{L}{2}, 45900 + \frac{L}{2})$.

Table 1: SNR Definitions

Name	Type	Target sensitive variables
$snr1$	unmasked S-box output	S-box$(p[3] \oplus k[3])$
$snr2$	masked S-box output	S-box$(p[3] \oplus k[3]) \oplus r_{out}$
$snr3$	common S-box output mask	r_{out}
$snr4$	masked S-box output in linear parts	S-box$(p[3] \oplus k[3]) \oplus r[3]$
$snr5$	S-box output mask in linear parts	$r[3]$

Here, we consider the POI selection from two folds. Firstly, the selected POI interval should be large enough to cover all operation information of one S-box. Secondly, a small number of POIs will help to reduce the subsequent analysis complexity. Therefore, we start attacks from a coarse sample number set $L = \{1000, 2000, 3000, 4000, 5000\}$ on the raw ASCAD dataset.

2.2.2 Autoencoders

The autoencoder is a type of neural network trained to represent the original information from a code. An autoencoder consists of two parts, namely *encoder* and *decoder*. An encoder $z = f(x)$ maps an input to the code while a decoder $x' = g(z)$ generates the reconstruction of original inputs. The autoencoder types that are widely adopted include undercomplete autoencoder (UAE), denoising autoencoder (DAE), and contractive autoencoder (CAE).

- *Undercomplete autoencoder (UAE).* Undercomplete means the dimension of the code is lower than the input. By using smaller-length vector representation, a UAE can force the network to learn the most significant features from original data. The training goal is to minimize the loss of $L(x, g(f(x)))$.

- *Denoising autoencoder (DAE).* Denoising autoencoder introduces noise to original data. It uses the corrupted data as input without corrupting the output. The goal of DAE is to minimize the loss function $L(x, g(f(\widetilde{x})))$, where $\widetilde{x} = x + \epsilon$ is the corrupted data. Noise ϵ is a type of regularization in autoencoder to maintain a robust reconstruction of $g(f(\widetilde{x}))$. DAE has been applied in trace preprocessing [WP20] to reduce the impact of countermeasures when taking them as noises.

- *Contractive autoencoder (CAE).* Contractive autoencoder introduces a regularizer to autoencoder so that the representation $z = f(x)$ becomes robust as the training samples have a minor change. The loss function is $L(x, g(f(x))) + \lambda \left\| J_h(x) \right\|_F^2$, where the regularizer is the Frobenius norm of the jacobian matrix. The regularizer here assists to contract a neighborhood of input points into a smaller neighborhood of output points.

In this work, we focus on dimension reduction for profiling attacks. We adopt the UAE as our first choice to simplify attacks. This is because we have no extra noise on original traces (as DAE required) and no limitations on the output (as in CAE). As we noticed that implementing DAE [WP20] needs to generate extra corrupted data for robust training, we decide to design UAE in an asymmetric manner as a penalty to reconstruct a robust code. We would maximumly reduce hidden layers in the decoder part and use original data directly to perform an attack. In this way, we can reduce the training model complexity and shorten preprocessing time.

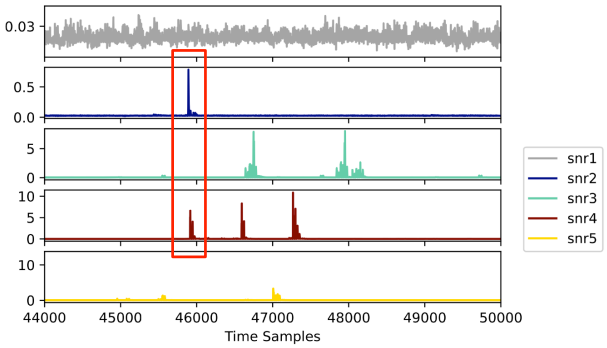

Figure 2: Same position SNR peaks of the third S-box

2.3 Neural Networks

A deep learning-based profiled attack is a task of classification that trains the neural network model for precisely predicting the correct keys. A neural network contains an *input layer*, *hidden layer(s)* and an *output layer*. The input layer in SCA receives the one-dimensional side-channel traces. Output layer provides key classification results. The number of nodes in the output layer is determined by the target space of keys, intermediates, or hamming weights. Hidden layers are changeable and can utilize the fully connected layer, convolutional layer, LSTM layer, etc. Multi-layer perceptron (MLP) and convolutional neural network (CNN) are mostly investigated in the SCA context.

In a neural network, parameters refer to variables that are automatically learned during the training process. And hyperparameters (model parameters and optimizer parameters) refer to variables being set before training. A neural network structure is defined by model hyperparameters. Model hyperparameters include the number of hidden layers, the number of neurons in each layer, filter size, the number of filters and the number of CNN blocks, etc. After set a model structure, we control the training process by specifying optimizer hyperparameters such as activation functions, learning rate, batch size, number of epochs, optimizer. Model hyperparameter and optimizer hyperparameters would influence the network attack performance. Grid search optimization [BBBK11] is widely used to select a better group of hyperparameters. Neural networks are trained by the backpropagation algorithm [Hec88]. Backpropagation computes the gradient of the loss function and then updates the weights to minimize loss. Next, we introduce MLP and CNN separately.

2.3.1 Multilayer Perceptron (MLP)

A multilayer perceptron network can be generally formalized as

$$\hat{f} = P \circ A_n \circ \lambda_n \circ \cdots A_2 \circ \lambda_2 \circ A_1 \circ \lambda_1 \circ I,$$

where the input layer is represented as an identity function I. A hidden layer is expressed as $A_i \circ \lambda_i$, where A_i is an activation function and λ_i is a linear function. Note that hidden layers gain non-linearity from activation functions such as sigmoid, rectified linear unit (ReLU), hyperbolic tangent (tanh). Hidden layers contain adjustable model hyperparameters, i.e., the number of hidden layers and the number of nodes in each layer. An output layer is defined as P to represent the prediction function used for classification.

In an MLP, all layers are fully connected and each connection has a weight value (parameter) that gets updated during backpropagation. We adopt the widely used categorical cross-entropy loss in prediction layer for MLP classification in SCA. Categorical cross-entropy is a combination of softmax activation and cross-entropy loss. Softmax is a normalized exponential function defined as $f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$. Cross-entropy, in general case, is defined as

$$Loss(\hat{y}, y, W) = -\sum_i^C y_i log(\hat{y}_i),$$

where C represents the number of classes, y is the target output, $\hat{y}$ means the estimated output through the network, W is the weight matrix. The categorical cross-entropy function calculates the probability of over C classes for each trace. In the end, the categorical cross-entropy loss is updated by

$$Loss(f\hat{(s)}, f(s), W) = -\sum_i^C f(s)_i log(f\hat{(s)}_i).$$

2.3.2 Convolutional Neural Networks (CNN)

A convolutional neural network is composed of three types of building blocks: *convolutional layers*, *pooling layers* and *fully connected layers*. Convolution layers play a key role in CNN to extract features. A CNN can be formalized as

$$\hat{f} = P \circ [A \circ \lambda]^{H_{FC}} \circ [\delta \circ [A \circ \gamma]^{H_{conv}}]^{H_{Block}},$$

where a CNN has H_{Block} convolutional blocks. Each convolutional block contains H_{conv} convolutional layers (denoted γ), an activation function (denoted A) and one pooling layer (denoted δ). A fully connected layer is represented by $A \circ \lambda$ where A is an activation function and λ represents a linear function. In the study of Zaid et al. [ZBHV20], they proposed to build an efficient CNN for side-channel attack. Here we present the convolutional layer and pooling layer that are useful to detect desynchronization. They assume that the maximum random delay in the original trace is N_D. In the convolutional block for feature extraction, filter size is set to $\frac{N_D}{2}$. Pooling stride is the same as filter size ($\frac{N_D}{2}$) to preserve information related to desynchronization while also maximizing dimension reduction. It is demonstrated that smaller filter sizes help to identify local features and larger filter sizes can extract global features but inevitably cause spreading of relevant information. Besides, they recommended a small number of filters {2, 4, 8} for fast training.

It is worth noticing that convolutional neural networks can be deconstructed into the convolutional part for feature extraction and a fully connected part for classification. In CNNs, fully connected layers are similar to MLP. As convolutional structures can be applied to undercomplete autoencoder for feature extraction, in this work, only MLP is used for classification.

3 Model Design

In this section, we propose our autoencoder-assisted model that combines SNR, UAE, and MLP. The model provides efficient reduction of large-size datasets.

3.1 Model Overview

The entire model (including feature selection and classification modules) consists of three parts: *SNR*, *UAE* and *MLP*. Fig.3 shows the trace processing path. Firstly, SNR analysis is performed on original traces. We use SNR middle position P_{middle} (see Sec.2.2.1) to trim traces of length L. The goal of this SNR exploration is to find a proper interval that can cover all possible leakage points and give good performance results. Then, the trimmed traces are fed into an undercomplete autoencoder. The autoencoder will compress L-sample traces by R times. SNR and UAE are the preprocessing part for feature extraction (POI selection). Each code with L/R samples becomes the input of the MLP attack model. MLP acts as the classification module and the attacking result is presented by the key rank (guessing entropy) of correct subkeys.

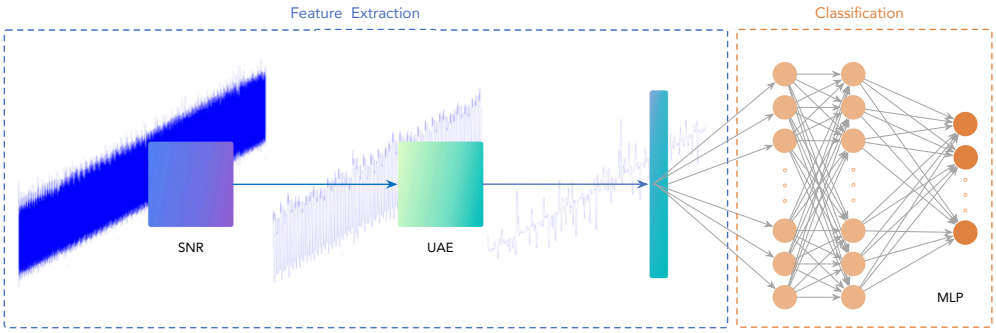

Figure 3: Preprocessing and Attack Model Structure

3.2 UAE Structure

Autoencoders are normally designed to be symmetric where the encoder and decoder have the same structure. SCA attack solutions that utilize autoencoders [WP20] [RAD20] also follow the standard symmetric structure. Heuristic thinking is that a symmetric structure can help to retrieve the original data. However, the symmetric structure increases the complexity of a UAE network. In this work, we introduce asymmetric UAE for feature extraction with higher efficiency. Asymmetric means the structures of the encoder and decoder are different and the encoder is more complicated than a decoder. Specifically, an encoder has one or more hidden layers. Whereas a decoder has fewer hidden layers or no hidden layer. The asymmetric structure can be viewed as a regularization that forces the network to learn more robust features without relying on a corrupted input or a regularizer. Formally, we define this asymmetric UAE model as

$$\hat{f} = I' \circ [\alpha]^{H_D} \circ C \circ [\alpha]^{H_E} \circ I, H_E > H_D,$$

where the input layer is represented as an identity function I, and the output layer I' means a reconstruction of input. The code (bottleneck layer) is denoted as C. A hidden layer is denoted as α. The number of hidden layers in encoder is H_E and decoder H_D. Here, the restriction $H_E > H_D$ shows asymmetric feature. A hidden layer α can apply structures such as a fully connected layer, a convolutional block, or an LSTM layer. Specifically, when α represents a fully connected layer, it is formalized as

$$A \circ \lambda,$$

where A is an activation function and λ represents a linear function. When α represents a convolutional block, it is

$$\delta \circ [A \circ \gamma]^{H_{conv}},$$

where a convolutional block contains H_{conv} convolutional layers (denoted γ), an activation function (denoted A) and one pooling layer (denoted δ).

In this work, we instantiate the asymmetric UAE model for synchronized traces and desynchronized traces. We test our proposed architecture on both synchronized and desynchronized high-dimensional datasets. As for synchronized traces, the hidden layers in UAE are configured to be fully connected layers for fast training. As for desynchronized traces, convolutional layers are introduced in UAE. Similar techniques applying CNN to deal with desynchronized cases are also demonstrated in [MPP16, ZBHV20].

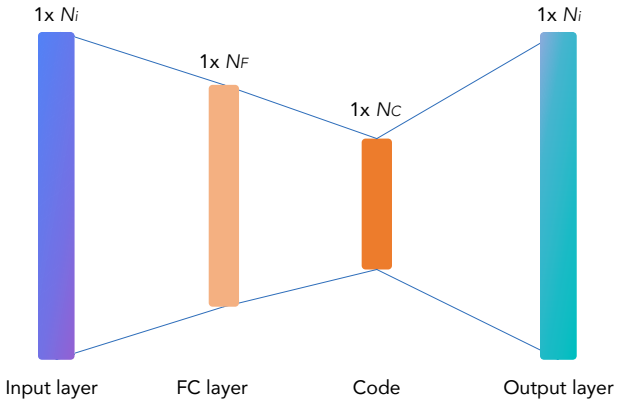

Figure 4: UAE structure for synchronized traces

3.2.1 UAE for Synchronized Traces

An asymmetric UAE for synchronized traces is presented as

$$\hat{f}_{sync} = I' \circ C \circ [A \circ \lambda]^{H_E} \circ I,$$

where $H_D = 0$ means that the decoder has no hidden layer. Notably, we remove the hidden layer in the decoder to maximally simplify the structure and reduce the complexity. This UAE structure is shown in Fig. 4. The input layer and output layer have N_I nodes. Each hidden layer has N_F nodes. The bottleneck layer has N_C nodes. In this way, an N_I-dimensional trace is compressed to a N_C-dimensional code. We denote the compress ratio from input dimension to a code dimension as $R = \frac{N_I}{N_C}$.

Here, we specify the trick we used to determine the number of nodes in each fully connected layer. Let us assume $N_F^{[j]}$ is the number of neurons in the jth fully connected layer. We define the number of nodes in j-th hidden layer is 2^n. The number of nodes in $(j-1)$-th hidden layer is twice the number of j-th layer, i.e., $N_F^{[j-1]} = 2 \times N_F^{[j]}$. We limit $N_F^{[1]} < N_I$ to make the number of nodes in each hidden layer is fewer than the input layer. We denote this structure as 2^n**-structure**. To set an asymmetric UAE structure, we only need to define the number of hidden layers H_E and the number of nodes $N^{[H_E]}$ in the last hidden layer. Using 2^n-structure, we perform grid search optimization for UAE hyperparameters (Tab.2) on the raw ASCAD dataset (synchronized traces).

3.2.2 UAE for Desynchronized Traces

Similarly, for desynchronized traces, a UAE is formalized as

$$\hat{f}_{desync} = I' \circ [A \circ \lambda] \circ C \circ [A \circ \lambda] \circ [\delta \circ [A \circ \gamma]] \circ I,$$

Table 2: Grid search optimization for UAE hyperparameters on synchronized traces

Parameter	Values
Activation function	{sigmoid, tanh, ReLU, SeLU}
Learning Rate	0.001 (default value)
Batch size	{128,256,512}
Epochs	{10,20,30,40}
n∘ of neurons $N^{[H_E]}$ (last hidden layer)	{64,128,256,512}
n∘ of hidden layers H_E (encoder)	{1,2,3}
Compress ratio R	{5,10,20,30,40,50}

where $H_D = 1, H_E = 2$ *and* $H_{conv} = 1$. During the UAE model design, we still attempt to design the most simplified UAE. As a flatten layer at the output layer of a convolutional block is required, we add a fully connected layer in both encoder and decoder. Therefore, in our UAE, the encoder consists of a convolutional block and a flatten layer (fully connected layer). Similarly, the decoder only has a fully connected layer whose nodes number is same as which in the encoder. The structure of the asymmetric UAE for desynchronized traces is depicted in Fig.5.

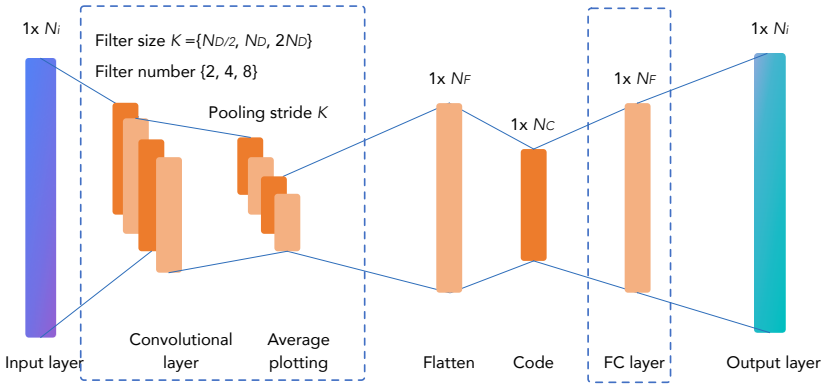

Figure 5: UAE structure for desynchronized traces

The input layer and output layer have N_I nodes. Regarding the method of building efficient CNN architectures [ZBHV20], we apply filter size $K = \{\frac{N_D}{2}, N_D, 2N_D\}$. Note that we extend the kernel size $\frac{N_D}{2}$ recommended in [ZBHV20] to $N_D, 2N_D$. As smaller kernels extract local features and larger kernels extract global features, we would like to find out the kernel size influence in UAE feature extraction. The pooling stride is the same as kernel size. We perform grid search on a small number of filters from $\{2, 4, 8\}$. After average pooling, the flatten layer has N_F nodes and the decoder hidden layer has the same number of nodes. The compress ratio from input dimension to a code dimension is $R = \frac{N_I}{N_C}$. Based on CNN structure, we perform grid search optimization for UAE hyperparameters (see Tab.3) on the desynchronized raw ASCAD dataset. *One-cycle policy* [Smi18] is applied to update learning rate.

3.3 MLP Structure

As for the attacking model, 2^n-structure is applied to build an MLP network. We denote the number of nodes in jth hidden layer as $N^{[j]}$. Here, $N^{[j]} = 2^n$ and $N^{[j+1]} = 2 \times N^{[j]}$, i.e., the next hidden layer has twice the number of nodes than the previous layer. The 2^n-structure in MLP is reverse to that in UAE because a UAE tries to compress a dataset. We assume that there are H hidden layers. Then an MLP under 2^n-structure is configured by H and $N^{[1]}$. The number of nodes in the input layer is denoted as N_C (code length)

Table 3: Grid search optimization for UAE hyperparameters on desynchronized traces

Parameter	Values
Activation function	{sigmoid, tanh, ReLU, SeLU}
Learning Rate	One-cycle policy
Weight initialization	He uniform
Batch size	{128,256,512}
Epochs	{10,20,30,40,50,60}
Filter size K	$\{\frac{N_D}{2}, N_D, 2N_D\}$
n∘ of filter	{2,4,8}
Compress ratio R	{5,10,20,30,40,50}

while the output layer as N_O. The architecture of a 2^n-structure MLP is shown in Fig.6.

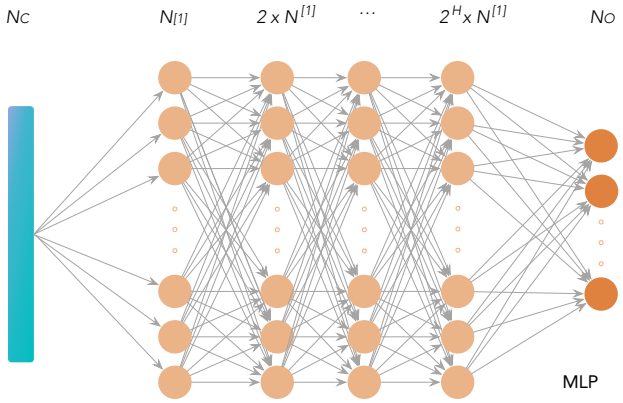

Figure 6: 2^n-structure MLP structure

With this model, we apply hyperparameters in Tab.4 to find a simple architecture so as to reveal S-box subkeys with fewer traces and shorter training time.

Table 4: Grid search optimization for the MLP hyperparameters

Parameter	Values
Activation function	{sigmoid, tanh, ReLU, SeLU}
Learning Rate	One-Cycle policy
Batch size	{50,100,200,300}
Epochs	{10,20,25,50,75,100}
n∘ of neurons $N^{[1]}$ (first hidden layer)	{64,128,256}
n∘ of hidden layers H	{2,3,4,5,6}

4 Experimental Results

In the following section, we apply our preprocessing and attack model suite on high-dimensional side-channel traces. The model suite is tested on synchronized and desynchronized raw ASCAD datasets. Firstly, we show the experiment settings and datasets preparation for profiled attacks. Then, we grid search proper UAE and MLP hyperparameters for each dataset. For a good model evaluation, the first priority is to compare the least number of traces for guessing entropy (GE) converge to 1. We also compared our model with the state-of-the-art SCA attack models from the perspective of training time and trainable parameters as in most SCA studies [PSB+18] [ZBHV20] [WP20].

4.1 Settings

For environment settings, our implementation of machine learning techniques is based on the Keras library and Tensorflow backend. We run training models on a laptop equipped with 8GB of RAM and Intel(R) Core(TM) i7-8750H CPU. The target leakage model is the S-box output S-box$(p[i] \oplus k[i])$. To validate our proposed processing and attack model, we have three types of datasets: a synchronized dataset (raw ASCAD traces of $100,000$ samples) and desynchronized datasets (50 and 100 samples separately window maximum jitter on raw ASCAD traces).

On synchronized raw ASCAD traces, we first select the third S-box as the preliminary experimental target. We perform a coarse searching of applicable parameters for an efficient attack. After finding the opportune parameters for this target S-box, further parameter fine-tuning will be implemented on the rest 15 S-boxes to exploit the key byte.

Since there is no desynchronized dataset of raw ASCAD traces, we generate random delay traces (samples window maximum of 50 and 100) in the same way as mentioned in Sec.2.1. Based on SNR observation, a leakage interval $[N, N+L]$ is selected for an S-box. Desynchronization parameter N_D is set to 50 and 100, respectively. Then a random integer $r < N_D$ is generated to trim each interval $[N+r, N+r+L]$ out as one item in the desynchronized dataset. In this way, we obtain the $N_D = 50$ and $N_D = 100$ desynchronized datasets.

Hyperparameters used in UAE and MLP are selected via a grid search optimization with finite sets of values. The final chosen value is obtained based on the best attack performances, namely, the least traces for the GE reaches 1. To perform a successful attack, three steps are conducted. Firstly, we propose a naïve UAE and MLP to explore a proper POI length based on SNR analysis. Then, we fix a non-optimized MLP to grid search UAE hyperparameters on the POI dataset. Finally, optimize MLP on UAE extracted features.

4.2 SNR Parameter

Since the state-of-the-art attack models use the third S-box in ASCAD dataset as the target of evaluation, we also aim at the same S-box for model training and comparison. POI intervals are trimmed from the length set $L = \{1000, 2000, 3000, 4000, 5000\}$ according to the SNR peak central point (see Sec.2.2.1). The trimmed dataset $\mathcal{D}^L_{S\text{-}box3}$ is labeled by $sbox(p[3] \oplus k[3])$. Each $\mathcal{D}^L_{S\text{-}box3}$ is split into three subsets: $45,000$ traces as *training* set, $5,000$ traces as *validation* set and $10,000$ traces as *test* set for attacks. We use 2^n-structure for both asymmetric UAE and MLP. For initial setting of UAE and MLP, we use non-optimized hyperparameters. We set the initial UAE structure (as in Fig.4) to have 2 hidden layers while the last hidden layer has $N_F^{[2]} = 256$ nodes (i.e., $N_F^{[1]} = 512 < N_I$). Compress ratio is set to 10. MLP is set to $H = 3$ and $N^{[1]} = 256$ (3 hidden layers and the first hidden layer has 256 nodes). For both UAE and MLP, activation function is ReLU. UAE is trained under 30 epochs with a batch size of 512 and learning rate of 0.001. MLP is trained under 50 epochs with a batch size of 200 and learning rate is updated under One-Cycle policy.

We use the same autoencoder and MLP attack model to compare the number of traces needed in exploiting the third byte. The attack results on the third S-box (Fig.7) show that POI intervals $L = \{2000, 3000, 4000\}$ can help to perform a successful attack (GE=1). The reason for the bad performance of $L = 5000$ dataset is that an autoencoder will extract a general feature and thus the useful leakage information is compressed. The interval with 3000 time samples requires the least traces. Thus, we fix the POI interval to 3000 time samples to investigate UAE and MLP architectures.

For the third S-box, we obtain the synchronized dataset $\mathcal{D}^{3000}_{S\text{-}box3}$ by trimming raw ASCAD dataset at position $(44400, 47400)$. We also derive desynchronization datasets ($N_D = 50$ and $N_D = 100$) based on this interval. We split each desynchronized dataset

into 3 subsets: 45,000 traces are used as the training set, 5,000 as validation set while 10,000 as the attack.

To test the performance of a classical attack model MLP_{best} [PSB+18] and the state-of-the-art models [ZBHV20] on ASCAD dataset, we perform a profiled attack using these models on $\mathcal{D}^{3000}_{S\text{-}box3}$ directly. We also evaluate the model we proposed on $\mathcal{D}^{3000}_{S-box3}$. Experimental results show that none of these models can reveal the correct keys before UAE feature extraction.

The attack results on the third S-box (Fig.7) show that POI intervals $L = \{2000, 3000, 4000\}$ can help to perform a successful attack (GE=1). We use the same autoencoder and MLP attack model to compare the number of traces needed in exploiting the third byte. The interval with 3000 time samples requires the least traces. Thus, we fix the POI interval to 3000 time samples to investigate UAE and MLP architectures.

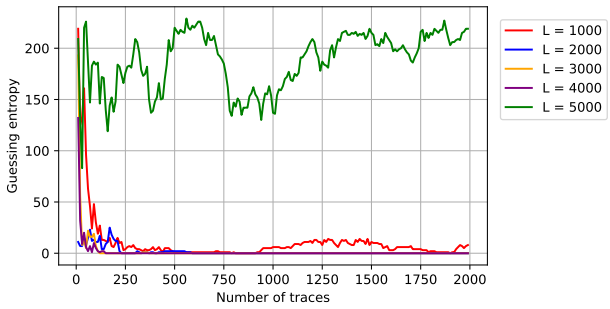

Figure 7: Comparison of different SNR intervals

For the third S-box, we obtain synchronized dataset $\mathcal{D}^{3000}_{S\text{-}box3}$ by trimming raw ASCAD dataset at position $(44400, 47400)$. We also derive desynchronization datasets ($N_D = 50$ and $N_D = 100$) based on this interval. We split each desynchronized dataset into 3 subsets: 45,000 traces as the training set, 5,000 as validation set and 10,000 for the attack.

To test the performance of a classical attack model MLP_{best} [PSB+18] and the state-of-the-art models [ZBHV20] on ASCAD dataset, we perform a profiled attack using these models on $\mathcal{D}^{3000}_{S\text{-}box3}$ directly. We also evaluate the model we proposed on $\mathcal{D}^{3000}_{S-box3}$. Experimental results show that none of these models can reveal the correct keys before UAE feature extraction.

4.3 Grid Search on UAE Hyperparameters

For synchronized and desynchronized datasets, we use corresponding UAE structures (Sec.3.2.1 and Sec. 3.2.2) to optimize asymmetric UAE hyperparameters. Mean squared error is used to measure loss and Adam [KB15] to be optimizer.

4.3.1 Synchronized Traces

We apply hyperparameters in Tab.2 to explore the best hyperparameters on synchronized dataset. Non-optimized MLP (right column in Tab.**??**) is used to evaluate UAE performance.

Among four activation functions, ReLU performs the best and takes less training time compared with SeLU function. Optimization is completed on a batch size of 512 and 30 training epochs with a learning rate of 0.001. The compress ratio is 40. The best UAE structure $H_E = 1$ and $N^{[H_E]} = 256$ (a UAE has one hidden layer and the last hidden layer has 256 nodes) make guessing entropy converge to 1 within 200 traces. The number of

Table 5: Hyperparameters for the best UAE on synchronized dataset

Hyperparameter	Values
Activation function	ReLU
Learning Rate	0.001
Batch size	512
Epochs	30
n ∘ of neurons (last hidden layers)	256
n ∘ of hidden layers (encoder)	1
Compress ratio	40

trainable parameters in this UAE is $787,531$ and it takes 104 seconds to reduce loss to 0.64 (Fig.8a).

Besides, according to our experimental results, we demonstrate that the number of hidden layers in the encoder has little impact on reducing loss and guessing entropy. On the contrary, the number of nodes in each layer can have a significant influence on attack results.

According to the best experiment result, we set the compress ratio to 40 in the third S-box. Equivalently, the code generated by UAE has 75 time samples. We apply this code in the next step, i.e., MLP optimization. The best asymmetric UAE hyperparameters for synchronized traces are summarized in Tab.5.

4.3.2 Desynchronized traces

Following our UAE architecture for desynchronized traces (Fig.5), we grid search an efficient UAE to extract critical features for key revealing in the existence of random delay effect. We evaluate our proposed architecture when $N_D = 50$ and $N_D = 100$. The best hyperparameters for UAE are grid searched on $N_D = 50$ and $N_D = 100$ datasets separately. By applying the UAE architecture in Sec.3.2.2, the filter size and pooling stride in convolutional layer applies not only $\frac{N_D}{2}$ in [ZBHV20], we extend the larger size of $\{N_D, 2N_D\}$ for each dataset. For both datasets, optimization is completed with SeLU activation function and He uniform weight initialization. The one-cycle policy is applied to update the learning rate for the convolutional autoencoder. For desynchronized traces, the naïve MLP adds one hidden layer (i.e., $H = 4$), which is more powerful for classification but can also cause overfitting.

(i). Random delay with $N_D = 50$. In $N_D = 50$ desynchronized dataset, the best compress ratio is 20. The UAE with 8 filters and filter size of $2N = 100$ requires the least traces to recover the correct key on the third S-box. It takes 2,741 seconds to run the training on a batch size of 512 within 50 epochs. Finally, the loss of UAE reaches 0.49 (Fig.8b). The number of trainable parameters in this UAE is $1,588,438$. 440 traces are needed for GE converging to 1.

The best UAE hyperparameters for $N_D = 50$ dataset are listed in Tab.6. It should be noted that in this experiment, the best filter size is $2N = 100$ instead of $\frac{N_D}{2}$ which uses the least number of traces. This indicates that global features extracted by a larger kernel also help distinguish the correct key candidate.

(ii). Random delay with $N_D = 100$. For a higher random delay dataset, it needs a filter size of 50 and 4 filters to find the correct key. The $N_D = 100$ dataset is compressed by 30 times on a batch size of 512 within 50 epochs. The training takes 2428 seconds to reach a loss of 0.75 (Fig.8c). The number of trainable parameters in this UAE is $771,544$. The best hyperparameters are summarized in Tab.6.

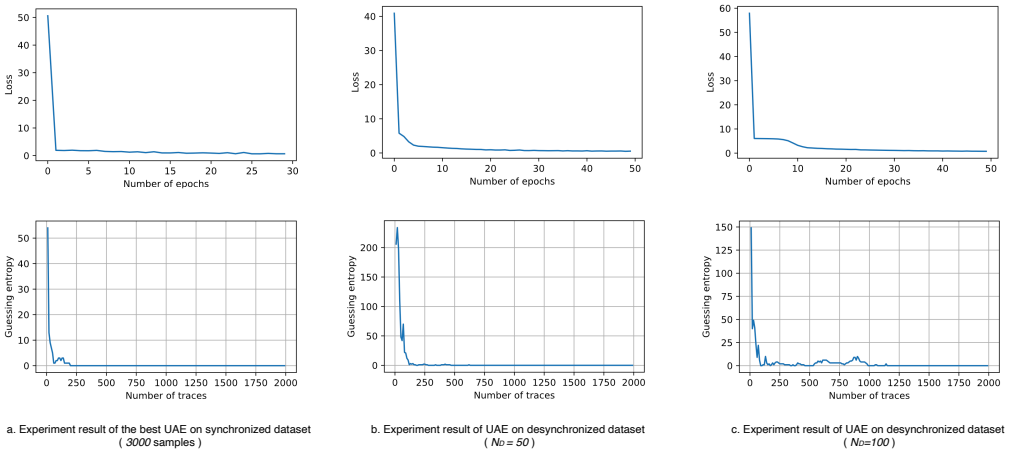

Figure 8: UAE Evaluations

Table 6: Hyperparameters for the best UAE on desynchronized datasets ($N_D = 50$ and $N_D = 100$ traces)

Hyperparameter	**Values** ($N_D = 50$)	**Values** ($N_D = 100$)
Activation function	SeLU	SeLU
Learning Rate	Once-Cycle policy	Once-Cycle policy
Batch size	512	512
Epochs	40	30
Filter size K	100	50
n ∘ of filters	8	4
Compress ratio	20	30

Here we discuss the selection of different UAE filter sizes and filter numbers in profiled attacks. On $N_D = 50$ dataset, filter length of $\{\frac{N_D}{2}, N_D, 2N_D\}$ all can make GE converge to 1. While on $N_D = 100$ dataset, only $\frac{N_D}{2}$-kernel size can help exploit the correct key. Based on our experiment on 2 desynchronized datasets, we demonstrate that although $\frac{N_D}{2}$ is not always the best parameter for filter size; it is more general in datasets with different random delays. Besides, increasing the number of kernels can provide more information about trace features.

4.4 Grid Search on MLP Hyperparameters

4.4.1 Results on Synchronized Traces

On synchronized datasets, we first present experimental results on the third byte. Then we extend the best UAE and MLP for synchronized datasets to the other 15 S-boxes.

(i). The third S-box. Grid search optimization is conducted for MLP on parameter set in Tab.4. We use categorical cross-entropy to evaluate guessing entropy. For fast convergence, the optimization is conducted by using Adam within 50 epochs and a batch size of 200. We also apply the one-cycle policy to update the learning rate in each epoch. ReLU is used as the activation function since it consumes less training time. Under 2^n-structure MLP, $H = 2$ and $N^{[1]} = 64$ deliver the best performance. The best hyperparameters of MLP are summarized in Tab.7.

Table 7: Hyperparameters for the best MLP on synchronized dataset

Hyperparameter	Values
Activation function	ReLU
Learning Rate	One-Cycle policy
Batch size	200
Epochs	50
n ∘ of neurons (first hidden layer)	64
n ∘ of hidden layers	2

Table 8: Comparison of performance on ASCAD 75 time samples (R=40) (synchronized dataset)

	MLP_best [PSB+18]	State-of-the-art [ZBHV20]	Our method
Complexity (trainable parameters)	227,456	4,432	46,208
Number of traces	-	-	140
Learning time (seconds)	98	160	46

We compare our network performance with MLP_{best} [PSB+18] and the state-of-the-art model [ZBHV20] for ASCAD dataset ($N_D = 0$) in Tab.8. The complexity of our network (in terms of trainable parameters) performs 5 times less than the MLP_{best} model. Training time is 3 times shorter than the convolutional network in [ZBHV20]. Guessing entropy reaches 1 within 120 traces in our MLP network while another two models cannot reveal the correct subkey within 2000 traces. In conclusion, our 2^n-structure MLP network has a good performance while all other models fail on 3000 time sample traces (as shown in Fig.10a). The least training time (autoencoder together with MLP) only takes around 200 seconds without any professional GPUs, which is proved to be satisfactorily efficient in a profiled attack.

(ii). All S-boxes. According to the grid search results of UAE and MLP on the third S-box, we obtain a group of initial hyperparameters (Tab.5 for UAE and Tab.7 for MLP) to perform an attack on the rest of 15 S-boxes. POI interval based on SNR analysis is set to 3000 time samples as in the third S-box. We detail the 16 S-boxes POI intervals on the raw ASCAD dataset in Appendix A. All these intervals are trimmed by finding P_{middle} point. The structure of an asymmetric UAE is designed to be $H_E = 1$ and $N_{H_E} = 256$, which means UAE has one hidden layer with 256 nodes in the encoder. In practice, we fine-tune UAE as $H_E = 1$ and $N_{H_E} = 512$ for S-box 16 when $N_{H_E} = 256$ cannot exploit the correct key. All other hyperparameters of UAE stay the same as in Tab.5. At first, MLP applies 2^n-structure with 2 hidden layers and the first hidden layer has 64 nodes. When we apply the 2-hidden-layer MLP to the other 15 S-boxes, profiled attacks fail on some S-boxes. Then, we add one hidden layer in MLP where the first hidden layer still has 64 nodes. Correct keys of 15 S-boxes (except S-box 1) can be revealed under the same MLP model, which means 3 hidden layers (2^n-structure) MLP performs well to most S-boxes. As for S-box 1, we apply $H = 3$ and $N_1 = 256$ MLP, namely the MLP has 3 hidden layers and the first layer has 256 nodes. Successful attacking results on 16 S-boxes are shown in Fig.9.

4.4.2 Results on Desynchronized Traces

In this part, we conduct experiments based on random delay $N_D = 50$ and $N_D = 100$ traces, respectively. Same as synchronized dataset, hyperparameters in Tab.2 are used for optimization.

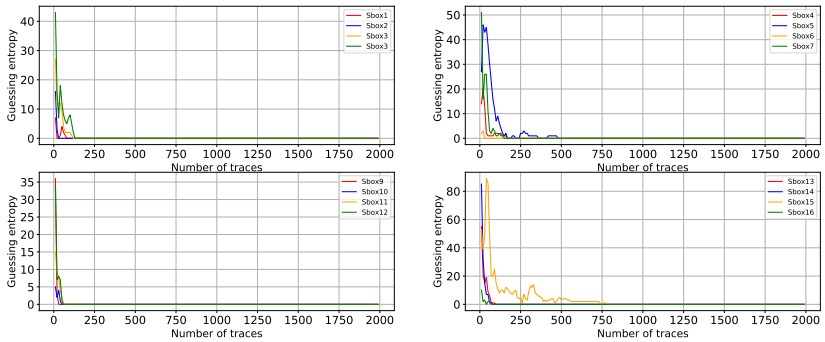

Figure 9: Guessing entropy result on 16 S-boxes

Table 9: Hyperparameters for the best MLP on desynchronized datasets

Parameter	**Values** ($N_D = 50$)	**Values** ($N_D = 100$)
Activation function	ReLU	ReLU
Learning Rate	One-Cycle policy	One-Cycle policy
Batch size	200	200
Epochs	20	30
n$\circ$ of neurons (first hidden layer) $\mathcal{L}_{1_M}$	128	256
n$\circ$ of hidden layers (encoder)	3	2
R	30	20

(i). Random delay with $N_D = 50$. Under 2^n-structure MLP, optimization is done on $H = 3$ and $N^{[1]} = 128$ within 20 epochs and a batch size of 200. The best hyperparameters of MLP are summarized in Tab.9.

We compare our network performance with the state-of-the-art model [ZBHV20] for ASCAD dataset ($N_D = 50$) in Tab.10. Since the input layer only has 150-time samples, the dimension size becomes negative at the third convolutional block of the existing CNNs [ZBHV20]. We update the third block pooling size and pooling stride from 4 to 3 to make the attack applicable. The complexity of our network (in terms of trainable parameters) is nearly 4 times more than the state-of-the-art model. However, our model training time is over 42 times less than the network in [ZBHV20]. GE reaches 1 within 170 traces in our MLP network while the state-of-the-art model requires 640 traces (shown in Fig.10b). Thus, our 2^n-structure MLP model is highly effective in profiled attacks.

(ii). Random delay with $N_D = 100$. Under 2^n-structure MLP, optimization is done on $H = 3$ and $N^{[1]} = 256$ MLP within 30 epochs. Similar to $N_D = 50$ case, MLP hyperparameters are summarized in Tab.9. We also compare our network performance with the state-of-the-art model [ZBHV20] for ASCAD dataset ($N_D = 100$) in Tab.11.

Guessing entropy reaches 1 within 1150 traces in our MLP network while the other model cannot reveal the correct subkey within 2000 traces (as shown in Fig.10c). Although the trainable parameters of our network are 7 times more than the state-of-

Table 10: Comparison of performance on random delay $N_D = 50$ dataset

	State-of-the-art [ZBHV20] ($N_D = 50$)	**Our method**
Complexity (trainable parameters)	82,991	315,264
Number of traces	640	170
Learning time (seconds)	1309	31

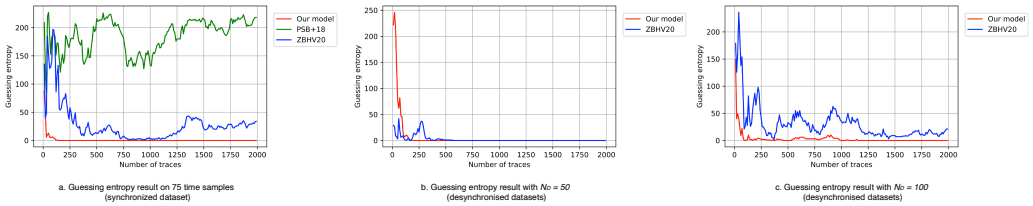

Figure 10: Guessing Entropy Evaluations

Table 11: Comparison of performance on random delay $N_D = 100$ dataset

	State-of-the-art [ZBHV20] ($N_D = 100$)	**Our method**
Complexity (trainable parameters)	136,476	945,152
Number of traces	-	1150
Learning time (seconds)	1120	245

the-art model, the training time is 4.5 times shorter than the convolutional network in [ZBHV20]. On random delay $N_D = 50$ and $N_D = 100$ datasets, the short learning time (31 seconds and 245 seconds, respectively) shows an adversary can efficiently perform an attack using UAE preprocessing. This tool saves exploiting effort in terms of time-division by extracting key features from original traces.

5 Conclusion

In this paper, we propose an efficient method to perform profiled attacks on high-dimensional datasets. We introduce the asymmetric undercomplete autoencoder to extract key features from high-dimensional leakage traces. To reduce the huge dimensionality, we combine SNR and UAE to reduce $100,000$ samples down to $3,000$ samples (by SNR) and further to 75 samples (by UAE) on the raw ASCAD dataset. We propose 2^n-structure UAE and MLP to perform the profiled attack and further investigate that our approach can work with desynchronization cases. We also equip UAE with convolutional layers to resolve the desynchronization problem as discussed in previous works.

To be noted, the state-of-the-art models are applied for profiled attacks on the raw ASCAD dataset before and after UAE compression, while none of them can reduce GE to 1. In contrast, by using our 2^n-structure MLP, we find out all the correct keys of 16 S-boxes successfully and efficiently. Further, 2^n-structure MLP can also function well towards random delay datasets with 50 and 100 time samples. We demonstrate that UAE is a necessary component in our approach to exploit leakage information. UAE can be a powerful preprocessing tool for an attacker, especially in a practical scenario with thousands of time samples. Although our experiments are conducted on laptops without any professional GPUs, a successful attack only takes around 200 seconds in total for UAE and MLP training. Experimental results indicate that our method is efficient in terms of both setup cost and training time. In the future, it is fascinating to study the model on secure implementations with complex countermeasures. Apart from high dimensional trace datasets, higher jittering and other masking schemes can also be an interesting research field full of opportunities.

References

[APSQ06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In Louis Goubin and

Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 2546–2554, 2011.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BGP+11] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *J. Cryptol.*, 24(2):269–291, 2011.

[BL12] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2012.

[CCC+19] Mathieu Carbone, Vincent Conin, Marie-Angela Cornelie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. Deep learning to evaluate secure RSA implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):132–161, 2019.

[CDP15] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Enhancing dimensionality reduction methods for side-channel attacks. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 15–33. Springer, 2015.

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

[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 *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013.

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

[EPW10] Thomas Eisenbarth, Christof Paar, and Björn Weghenkel. Building a side channel based disassembler. *Trans. Comput. Sci.*, 10:78–99, 2010.

[GBPV09] Benedikt Gierlichs, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Revisiting higher-order DPA attacks: Multivariate mutual information analysis. *IACR Cryptol. ePrint Arch.*, 2009:228, 2009.

[GHO15] Richard Gilmore, Neil Hanley, and Máire O'Neill. Neural network based attack on a masked implementation of AES. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015*, pages 106–111. IEEE Computer Society, 2015.

[Hec88] Robert Hecht-Nielsen. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[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.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2012.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

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

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

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES. 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 *Lecture Notes in Computer Science*, pages 61–75. Springer, 2013.

[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 Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 348–375, 2020.

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

[PSB+18] Emmanuel Prouff, Rémi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptol. ePrint Arch.*, 2018:53, 2018.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[RAD20] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. SCAUL: power side-channel analysis with unsupervised learning. *IEEE Trans. Computers*, 69(11):1626–1638, 2020.

[Smi18] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):389–415, 2020.

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

Appendix

5.1 POI selection of 16 S-boxes

Zooming in SNR analysis peaks where there are same position peaks shown in each S-box, intervals marked in red (Fig.11) are possible leakage positions. After finding middle point P_{middle} for 16 S-boxes, $L = 3000$ time samples are trimmed for each S-box. Specifically, we trim $L/2$ to the left of P_{middle} and $L/2$ to the right to derive new trace set for attack. Tab.12 is the trimmed 3000 time sample interval from the raw ASCAD dataset. Based on the trimmed interval, successful attacks are performed combing UAE and MLP.

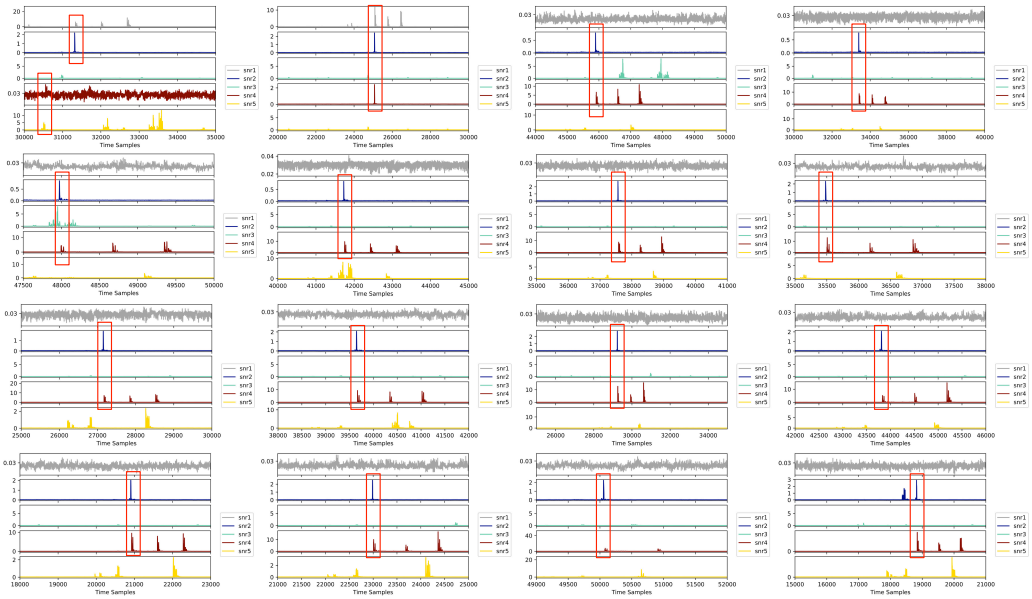

Figure 11: Detailed SNR of 16 S-boxes

Table 12: 3000 POIs trimmed for 16 S-boxes

S-box1	**S-box2**	**S-box3**	**S-box4**
(28500,31500)	(22500,25500)	(44400,47400)	(31930,34930)
S-box5	**S-box6**	**S-box7**	**S-box8**
(46450,49450)	(40230,43230)	(36080,39080)	(34000,37000)
S-box9	**S-box10**	**S-box11**	**S-box12**
(25650,28650)	(38170,41170)	(27750,30750)	(42330,45330)
S-box13	**S-box14**	**S-box15**	**S-box16**
(19430,22430)	(21500,24500)	(48580,51580)	(16970,19970)