Improving Non-Profiled Side-Channel Attacks using Autoencoder based Preprocessing

Donggeun Kwon[1], HeeSeok Kim[2], and Seokhie Hong[1]

[1] School of Cyber Security, Korea University, Seoul, Republic of Korea
donggeun.kwon@gmail.com , shhong@korea.ac.kr
[2] Department of Cyber Security, Korea University, Sejong, Republic of Korea
80khs@korea.ac.kr

Abstract. In recent years, deep learning-based side-channel attacks have established its position as mainstream. However, most deep learning techniques for cryptanalysis mainly focused on classifying side-channel information in a profiled scenario where attackers can obtain the label of training data. In this paper, we introduce a novel approach with deep learning for improving side-channel attacks, especially in a non-profiling scenario. We also propose a new method that trains autoencoder through the noise from real data using the noise-reduced-label. It notably diminishes the noise in a trace by adapting the autoencoder framework to the signal preprocessing. We show the convincing comparison from our custom dataset, which captured that our works outperform conventional preprocessing methods such as principal component analysis and linear discriminant analysis. Furthermore, we apply the method to realign desynchronized traces that applied hiding countermeasures, and we experimentally validate the performance of the proposal. Also, for masking countermeasures, we experimentally show that we can improve the performance of side-channel analysis by using an existing combining function or proposed method using domain knowledge.

Keywords: Side-channel attack · Non-profiling · Deep learning · Autoencoder · Preprocessing

1 Introduction

Side-channel analysis, which exploits physical leakage from a cryptographic device, was introduced by Kocher in 1996 [11]. For the successful side-channel analysis against cryptographic implementations, it generally consists of three steps. The first step is to collect side-channel information, such as power consumption or electromagnetic radiation, from the target cryptographic device, which is highly dependent on the performance of the measurement equipment. Second, preprocessing steps, such as noise reduction, trace alignment, dimensionality reduction, and feature selection, is required to extract meaningful information in the measurements. Finally, it consists of modeling and exploiting secret information on the preprocessed information to recover the correct key.

However, in the real world, attackers could fail to extract secret information, e.g., cryptographic key, from power traces obtained in the actual analysis environment, even if the analytical technique is performed correctly, because of noise and misalignment. In the context of side-channel analysis, several methods have been applied to preprocess the physical information leakages for reducing the attack complexity in terms of the number of traces needed. To briefly review the commonly used preprocessing techniques, averaging method, Singular Spectrum Analysis (SSA), Principal component analysis (PCA) and Linear discriminant analysis (LDA) with sliding window are used as preprocessing methods for denoising. To realign the desynchronized traces, cross-correlation for a matching pattern with sliding window [13] and Elastic alignment [20], which is based on Dynamic Time Warping (DTW), are introduced in side-channel context.

These methods have shortcomings that depend on the capability of the attacker and require to handle the parameter search manually. To overcome the difficulty of preprocessing, end-to-end deep learning based side-channel attacks have been significantly interested in recent years. The attacks have an advantage that it could obtain similar (or better) results without the preprocessing processes, which depend on the attacker's ability. Early research based on deep learning was studied regression that attempted to characterize the power model by Yang *et al.* [22]. However, subsequent studies using deep learning as a method of solving the classification problem were performed. In this case, assuming profiling attack scenario, the attacker trains a deep neuron network through the traces obtained from a profiling device, and then uses the network as the way of classifying the traces from a target device. Results of Maghrebi *et al.* [12] confirm that profiling attacks with deep neural networks such as Multi-Layer Perceptron, Convolutional Neural Network, Stacked Autoencoder, Long Short-Term Memory (LSTM), can be analyzed regardless of whether masking countermeasure is applied or not. Through the study of Cagli *et al.* [4], we could recover the secret information only through deep learning based side-channel attack without performing preprocessing techniques like trace alignment, when we use the convolutional neural network. Hettwer et al. [8] introduced a new architecture of convolutional neural network, and it show that additional input, Domain knowledge neurons which are concatenated with output of flatten layer, can improve the performance of convolutional neural networks.

While most of the studies have focused on applying deep learning to perform profiling attacks, Differential Deep Learning Analysis (DDLA), which can use the power of deep learning in the non-profiled context, is proposed by B. Timon[18]. DDLA is the method which uses deep learning as distinguisher, and it show that different training trends, such as loss, accuracy, sensitivity, appear depending on key guessed label. Using the training trends, the attacker distinguishes right key guessed label from wrong key label in non-profiling context. This study has shown that deep learning based side-channel attacks can be performed in non-profiling attack scenarios.

Recent methods, which is related to this paper's work, profiled correlation electromagnetic analysis using Correlation Optimization proposed by Robyns et

al. [17]. Correlation Optimization is a novel approach that improves side-channel attack by encoding the leakage so that the correlation coefficient is maximized. Also, as one of the studies similar with our works, a technique for preprocessing side-channel Measurements using autoencoders has been proposed [21], but it is the technique in the profiling attack environment using a convolutional autoencoder. However, by limitation of supervised learning, which can not perform without a label (in this case, the trace that easy to analyze), profiled deep learning based side-channel attacks are limited to research in the profiled context where training data and its labels can be obtained.

1.1 Our Contributions

Our main contributions of this paper can be summarized as follows.

- **Introducing a new approach of deep learning based techniques to improve non-profiled side-channel attacks.** To the best of our knowledge, side-channel attacks that apply the deep learning techniques in non-profiling context is only Differential Deep Learning Analysis, which proposed by Timon[18]. We introduce a novel approach based on deep learning to improve non-profiling side-channel attacks.

- **Presenting denoising preprocessing with neural network which can outperform the classic preprocessing methods.** We propose a new autoencoder which can reduce the noise by modifying a training principle to the context of side-channel analysis. Through experiments, we demonstrate that the proposed method outperforms conventional preprocessing methods.

- **Extending the proposed method to realign de-synchronized traces with convolutional autoencoder.** We show that the proposed method can help to re-synchronize the traces that misaligned by countermeasure such as random delay, jitter. We confirm that it is possible experimentally.

1.2 Organization

The structure of this paper is oranized as follows. Section 2 briefy describes non-profiled side-channel attacks, deep learning, Autoencoder and Denoising Auto-Encoder. In section 3, we introduce a novel approach of improving side-channel attacks with autoencoder, and propose a new methods that preprocess the traces by modifying autoencoder framework to the context of non-profiled side-channel analysis. Section 4 compares the performance of noise reduction and alignment between the classic preprocessing techniques and the proposed method from experiments performed on traces obtained from ChipWhisperer-Lite, random delays countermeasure dataset and first order masking countermeasure database, ASCAD. Finally, section 5 concludes this paper with conclusion and future works.

2 Preliminaries

2.1 Non-profiled Side-Channel Attack

Non-profiled attack is a part of side-channel analysis which performs in non-profiled context where measurements can be collected only from a target device which has a fixed key. Depending on the number of traces, there are Simple Power Analysis (SPA) which analyze through one or a few traces, and Differential Power Analysis (DPA) [10], Correlation Power Analysis (CPA) [3] that perform statistical analysis through a large number of traces. Especially, CPA, which is proposed by Brier et al in 2004 [3], is a power analysis using the correlation between the power consumption from the target device that performs cryptographic operation and the hypothetical power consumption value to be calculated. The leakage model is defined as the following:

$$Power = \delta + HW(Data) + Noise \tag{1}$$

where δ is a fixed constant offset, $HW(\cdot)$ is the Hamming Weight function, and $Noise$ is gaussian random noise centered in zero with a standard deviation σ. In order to perform CPA, first step is that attacker measures the power consumption of target device while it calculates the cryptographic operations. Next, attacker calculates the hypothetical consumption with guessing key, and then calculates a correlation between hypothesis and power consumption. The correlation coefficient between two values is calculated as follows:

$$\begin{aligned} \rho(X,Y) &= \frac{Cov[X,Y]}{\sqrt{Var[X] \cdot Var[Y]}} \\ &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \end{aligned} \tag{2}$$

It can be deduced that the hypothesis with the largest correlation coefficient with the measurements is the hypothetical power consumption which calculated by a right key. Thus, attacker can recover the correct key. Since the statistical analysis techniques are affected by the noise of the data, noise reduction is required for successful side-channel attacks.

2.2 Deep Learning

Deep Learning is a subset of machine learning that approximates a function using neural network and is used in various fields such as image, natural language and speech processing. Training is a process of modifying the parameters to approximate a neuron network with a function that we want to get. If label, which is output of function that the attacker want to approximate, is given, it is classified into supervised learning, and unsupervised learning if not.

When a neural network is a function $f(x)$ and function that want to approximate is $f^*(x)$, $f(X;\theta)$ is output of the neural network for input X with trainable

parameters θ. To approximate the function $f^*(x)$ means that it minimize a difference between the output of the neural network $f(x;\theta)$ and the output of the actual function, label $y(= f^*(x))$. The difference, called loss (or cost, error), is followed as:

$$Loss(X, Y) = L(f(X;\theta), Y) \tag{3}$$

In equation (3), $L(\cdot)$ is a loss function, which is also called the objective function, cost function and error function, and usually uses Mean Squared Error or Cross Entropy. By the training, the neural network learns the optimal parameters θ_{best} that satisfy (4). In other words, the neural network searches the parameters θ_{best} that minimize loss through the training.

$$\theta_{best} = \underset{\theta}{argmin}(L(Y, f(X;\theta))) \tag{4}$$

We usually use the Gradient Descent method to find the optimal parameters that minimize the loss. α is a learning rate that decides how much to change the parameters of neural network with respect the gradient. For scheduling the learning rate, there are various methods such as RMSProp and Adam [7].

A neural network is represented by composing three different type functions, called layers. The layers are organized in three types: an input layer corresponding to the input of data, an output layer corresponding to the output of the network, and the remaining hidden layer. The input layer and the output layer have a number of neurons corresponding to the dimension of the input data and the output data, respectively, and are fixed according to the training data. In the case of the hidden layer, there are parameters to be set by the attacker such as the number of hidden layers, the activation function, and the number of each hidden layer's neurons, and are called hyperparameters. The hyperparameters are not trainable parameter in neural network so that the attacker have to be carefully set for best results.

Multi-Layer Perceptron. Multi-Layer Perceptron (MLP), is also called Deep Neural Network (DNN) or Artificial Neural Network (ANN), is a basic model of neural network. Each hidden layer of MLP consists of a linear function and a nonlinear function. MLP consists of multiple hidden layers, and can be expressed as follows:

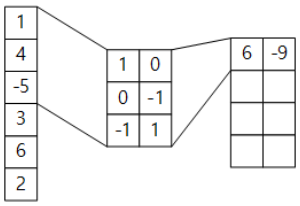
$$f(x) = s \circ \lambda \circ \sigma \circ \lambda \circ \cdots \circ \sigma \circ \lambda(x) \tag{5}$$

λ is called Fully-Connected layer and is the linear function that calculates as $WX + b$, W and b are the trainable parameters, called weight and bias. σ is called Activation layer and is the nonlinear function that usually uses sigmoid, ReLU(Rectified Linear Unit), SELU(Scaled Exponential Linear Unit) or Hyperbolic Tangent. s is a classification layer, that is a little different with activation layer, which is to re-normalize the output. It usually uses softmax function when neural network is multi-class classifier, sigmoid function when it is binary classifier and output cell is one unit. According to universal approximation theorem, it show that a multi-layer perceptron with single hidden layer can approximate

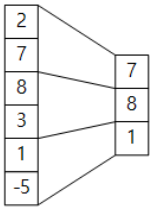
arbitrary continuous functions. In deep learning based side-channel analysis, the most of researchs study method to approximate a function that outputs a intermediate value (or its hamming weight, least significant bit) by inputting measurements.

Convolutional Neural Network. Convolutional Neural network (CNN) is a particularized class of neural networks that additionally contain convolutional layers and pooling layers. Convolutional layers are linear layers that share weights and apply convolution operation to the input. The weights of the convolutional layer are called kernels or filters, which can detect a feature in the input. The kernels also are optimized by the gradient decent method. Similar to the MLP, convolution layers have an activation function, which is a nonlinear operation followed by convolution operations.

The other kind of layer, pooling layers are usually performed after the convolution layers. Pooling layers performs down-sampling on the input dimension to output the reduced volume by averaging local or sub-sampling maximum. By pooling steps, small changes in input do not have a large effect on the feature extraction. For example, eyes, ears and mouth positions are slightly different for each person, but when using max pooling, this difference will not have a big impact on recognizing people.



Convalutional Layer with 2 kernels and kernel size 3.



Max Pooling Layer with pooling size 2.

Fig. 2: Example of (a) Convolutional Layer and (b) Pooling Layer.

2.3 Basic Autoencoder

Autoencoder is a unsupervised learning model of neural network that the output of neural network is similar to the input, and used for pre-learning of neural network, compression of input data and denoising. In earlier studies where training about the deep layers was difficult, autoencoder was usually used for initialization and pre-training of the network's parameters. After learning the weights of each layer by pre-learning using autoencoder to best express the input data, the network is learned by adjusting the overall weight through fine-tuning. However, it is not well used due to the inconvenience of learning time and design, and the new initialization techniques are proposed such as xavier and he initializer [].

Secondly, autoencoder can be used as a way to compress the dimensions of input data. Autoencoder used for dimensionality reduction basically consists of an encoder part for compressing the dimension of the input data and a decoder part for reconstructing the compressed data through the encoder into the original input data. Figure 3 is the basic architecture of autoencoder.

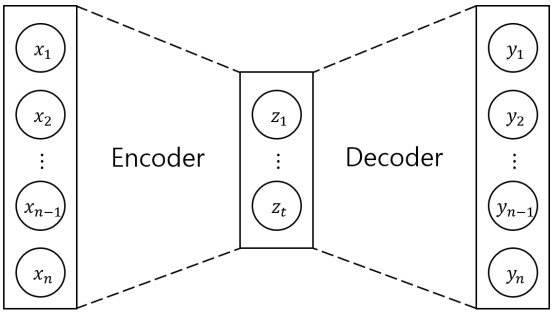


Fig. 3: Basic Architecture of Autoencoder

In Fig. 3, $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ is the input of autoencoder, $Z = (z_1, z_2, ..., z_t) \in \mathbb{R}^t$ is called Code that is data compressed by the encoder of autoencoder, and $Y = (y_1, y_2, ..., y_n) \in \mathbb{R}^n$ is the output of autoencoder. A neural network consisting of hidden layers between input and Code is called the encoder. Also, a neural network consisting of hidden layers between Code and output is called the decoder. Generally, the dimension of the code t is smaller than the dimension of the input n, and if the autoencoder satisfy the condition, it is called an undercomplete autoencoder. If not, it is called an overcomplete autoencoder. The operation of autoencoder in which encoder and decoder are each composed of one layer is calculated as follows:

$$z_i = \sigma(\sum_{j=1}^{n} weight_{(j,i)}^{enc} x_j + bias_i^{enc}) \tag{6}$$

$$y_i = \sigma(\sum_{j=1}^{t} weight_{(j,i)}^{dec} z_j + bias_i^{dec}) \tag{7}$$

$$Loss_{AE} = L(X, g(f(X;\theta))) \tag{8}$$

$$\theta_{best-AE} = \underset{\theta}{argmin}(L(X, g(f(X;\theta)))) \tag{9}$$

When the encoder is a function $f()$ and the decoder is a function $g()$, the loss of autoencoder is defined as (8). If training is successful and the output Y is the same for input X, then $X = g(f(X)) = g(Z)$, $X = g(Z) : \mathbb{R}^t \rightarrow \mathbb{R}^n$. It means that the compressed data, code can be reconstructed to the original through the decoder function g, while the dimension of data is smaller than the dimension of

input data. Therefore, the code has all of the features of the input, but is also low-dimensional data.

2.4 Denoising Autoencoder

Autoencoder which reduce the noise, called Denoising Autoencoder (DAE), is proposed by Vincent et al [19] in 2008. The structure of DAE and autoencoder are the same, but the main difference lies in the input data used for training. Unlike autoencoder that uses input data as it is, DAE is trained through randomly added noise by an attacker. Fig. 4 is the basic architecture of denoising autoencoder.

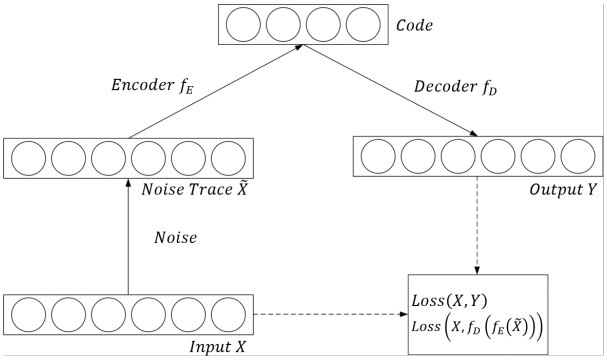


Fig. 4: Denoising Autoencoder Architecture

As shown in Fig. 4, the attacker adds random noise to data X, which collected, to generate new data $\widetilde{X}$ and use it as training data. The learning is performed to minimize the loss which calculated using the output of the neural network Y and the original data X before adding the noise as a label. It trains noisy data to recover the original undistorted input, and the new training principle for autoencoder can make the neural network to remove the noise.

$$Loss_{DAE} = L(X, g(f(\widetilde{X}; \theta))) \tag{10}$$

Equation (10) represents the loss of the DAE. There are two ways to add noise to input data in the DAE: adding gaussian noise to the data, or zeroing some elements of the data randomly. By adding random noise, the model learns with $\widetilde{X}$ to project them back into original X, it can make to decide the data in the close range as the same data. Also, by making some elements zero, the model can be learned about the whole data, not just by focusing on specific parts of the data. Using this method, the autoencoder can train to output noise-reduced data.

3 Side-Channel Preprocessing using Autoencoder

3.1 Conventional Methods and Traditional Autoencoder in Side-Channel Analysis

In side-channel analysis, PCA and LDA based noise reduction methods are mainly used. In terms of dimensional reduction, PCA and LDA are the methods that project data to linear hyperplane, but autoencoder is the method that project data to non-linear hyperplane, like Isomap, and is the deep learning based technique with the advantage that the more data, the higher the dimension, the better the performance. When t is smaller than n, the decoder is linear layer and loss function is mean squared error, an autoencoder learns to span the same subspace as PCA [7]. Therefore, conventional techniques can be replaced by autoencoder theoretically. For similar reasons, denoising autoencoder is applicable, and according to previous studies, better performance can be expected when applying the denoising autoencoder.

Initialization method using autoencoder is called stacked autoencoder that Maghrebi at el [12] had used in side-channel analysis to classfy the power traces. Through their experiments, it show that stacked autoencoder is better than MLP in same case.

However, there are two disadvantages to using the denoising autoencoder in the side-channel analysis due to the differences from the image processing field. First, if noise is added to the collected traces and used as training data, the noise already existing in the collected traces is further added to the noise, that may make class classification more difficult and fail to learn. Assuming that the power model is Equation (1) as described above, Equation (11) is the denoising autoencoder's loss when the input data is the power traces.

$$\begin{aligned} Loss_{DAE} &= L(g(f(\tilde{X};\theta), X) \\ &= L(g(f(\delta + HW(D) + Noise + Noise')), \delta + HW(D) + Noise) \end{aligned} \tag{11}$$

When the value of the noise $Noise$ of the actual traces is low, it is not necessary to remove the noise for the waveform. On the contrary, when the value of $Noise$ is high, the weight of the noise in the training data $\delta + HW(D) + Noise + Noise'$ calculated by adding the new noise $Noise'$ becomes larger than before. This make it difficult to learn. If the attacker set the $Noise'$ too low to train, the denoising autoencoder will only train about low noise, reducing the effect of noise reduction. In addition, in the context of side-channel analysis, the operation to be attacked is performed only at a certain point in time. When training data is generated with a random sample point of 0, training data may be generated in which sample related to the secret key is excluded. Therefore, this method is not suitable for the side-channel analysis environment. Thus, it is difficult to apply denoising autoencoder easily due to the problems. Also, when using this approach, other features may appear larger in the dataset and may not focus on data which is meaningful to the attacker. In this paper, we propose a new autoencoder model modified to solve the problems.

3.2 Side-Channel Autoencoder for Denoising

In this section, we introduce our approach to preprocess the measurements by modifying the training principle of autoencoder to the context of side-channel analysis, which is called Side-Channel Autoencoder (SCAE). Figure 5 is the basic architecture of autoencoder proposed in this paper. The proposed model is similar to the basic structure of the autoencoder. However unlike the denoising autoencoder, the input data is used as training data. Also, by using preprocessed data as the label, the autoencoder can be learned about the real noise to output noise-reduced traces.

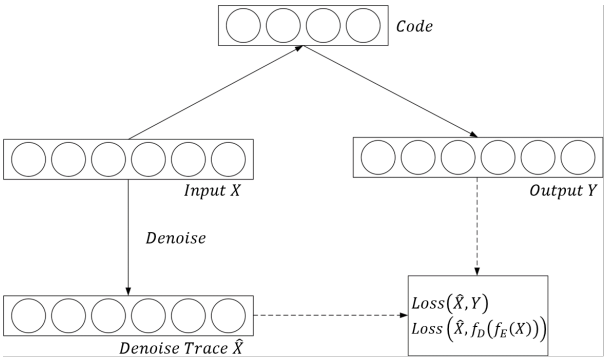


Fig. 5: Side-Channel Autoencoder Architecture

As you can see in Fig. 5, input X is used as the input of the autoencoder, and denoise trace $\hat{X}$ is used as the label for the input data. The loss of the proposed autoencoder is as follows.

$$Loss_{SCAE} = L(\delta + HW(D), g(f(\delta + HW(D) + Noise))) \tag{12}$$

In the conventional autoencoder which denoises, the network is trained to remove the newly added noise which is added by attacker, but the proposal is trained to remove the noise in the collected traces. In contrast to the loss of the denoising autoencoder in Equation (11), we calculate the loss as the difference between output Y obtained by inputting X and denoise trace $\hat{X}$.

There are many ways to preprocess side-channel traces to perform the proposed method, but we use the simplest and reasonable approach 'average'. By maximum likelihood estimation in equation (1), expectation value is an average value of the traces with the same intermediate value [2]. If the key K is a fixed value, the intermediate value $D = Sbox(P \oplus K)$ is determined according to the plaintext P, so that the traces performed with the same plaintext P have the same intermediate value D. Since the average trace for the same plaintext is the average trace for the same intermediate value, thus, the proposed preprocessing technique can be performed even in a nonprofiling attack environment

in which the intermediate value is not known. The label for each trace can be set to the average trace corresponding to the plaintext of the trace. Algorithm 1 summarizes the proposed method to perform with averaging technique. After the preprocessing step, secret key can be exploited by using side-channel attack such as DPA, CPA.

Algorithm 1 Label preprocessing for denoising

Input: Traces $(T_n)_{0 \leq n \leq N}$ with corresponding plaintexts $(P_n)_{0 \leq n \leq N}$, when $0 \leq P_n \leq p$.
Output: Label traces Y

for $i = 0...p$ **do**
    Calculate mean traces $R_i \leftarrow mean(\{X_j | P_j = i\})$
end for
for $i = 0...N$ **do**
    Set label trace as $Y_i = R_j$, when $j = P_i$
end for
return Y

Such a method is difficult to perform in an image processing context but can be performed due to differences in data in the context of side-channel analysis. For example, in an image processing implementation that classifies handwritten digits like MNIST database, the number of classes that the attacker has to classify is 10, and the samples that attacker must be analyzed are separated into several samples in 784 (28×28) samples. Therefore, two different data of the same class can have features at different points in samples, thus, average value of the image with same digit is a meaningless value. We easily expect that if we use the method with mean trace, the traces for a particular plaintext are always output as the same trace (label trace). Nevertheless, such a situation is not easy to occur, unless overfitting.

3.3 Side-Channel Autoencoder for Hiding Countermeasure

When the alignment of the traces are disturbed by side-channel countermeasures such as random delay and jitter, the point of samples as the attack target is different for every trace. Therefore, it is difficult to acquire the noise-reduced traces through the averaging, and to apply the above-described proposed method. In this case, preprocessing is required to align the traces rather than the noise reduction in order to apply the conventional side-channel attack. In this subsection, by modifying the proposed labeling technique, we propose a simple labeling preprocessing to encode the traces into aligned data.

In the previous description, the method of obtaining a representative trace of each class from which 256 noises have been removed is described, the following description is for a method of collecting an aligned representative trace of each class (intermediate value). Algorithm 2 summarizes the labeling method to obtain the realigned traces in de-synchronized traces.

Algorithm 2 Label preprocessing for alignment

Input: Traces $(T_n)_{0 \le n \le N}$ with corresponding plaintexts $(P_n)_{0 \le n \le N}$, when $0 \le P_n \le p$.

Output: Label traces Y

Set first representative trace $R_0 = T_i$ where $P_i = 0$
for $i = 0...p$ **do**
    $R_i = T_j$ where $j = argmax(corr(R_0, T_k)), k \in \{n | P_n = i\}$
end for
for $i = 0...N$ **do**
    Set label trace as $Y_i = R_j$, when $j = P_i$
end for
return Y

Similar to the method for noise reduction, a representative label for each plaintext is selected in de-synchronized traces having the same intermediate value. First, one traces is selected at random in some plaintext (like 0), and the correlation coefficient is calculated with traces having different plaintexts (1 255). Next, one of the traces with the highest correlation coefficient is selected for each plaintext set and used as a label trace of each set. Thereafter, additional alignment can be performed using an conventional alignment technique for 256 traces. In this way, it is possible to obtain labels by not performing the alignment, or by performing the alignment only on a small number of traces, 255.

3.4 Side-Channel Autoencoder for Masking Countermeasure

In the implementation applied masking countermeasure, the intermediate values are changed by masking value, which is the unknown, so that the proposed methods described above can not be used. Therefore, we introduce a new autoencoder with domain knowledge neurons. The domain knowledge (DK) neurons, which is proposed by Hettwer et al. [8] in 2018, provide the plaintext or ciphertext as additional information into neural network to learn the leakage in regard to the secret key. The research of Hettwer et al. show that better results can be obtained when using side-channel traces with domain knowledge. We also get better results when using the domain domain knowledge in the autoencoder.

Furthermore, in our experiments, we use one byte of the plaintext as domain knowledge, however, we encode the plaintext into bit-encoding, not one-hot encoding. Bit-encoding represent the plaintext as vector of 8 variables like binary representation, where one-hot encoding encodes the plaintext into vector of 256 variables. The bit-encoding can represent data in a smaller dimension than one-hot encoding, and also represent vector of binary variables. The basic architecture of autoencoder with domain knowledge can be shown in Fig. 6.

The methods described in section 3.1, 3.2 are similar to autoencoder with domain knowledge. When domain knowledge provide additional information to the input, the methods described above were processed using additional information, plaintext on the label.

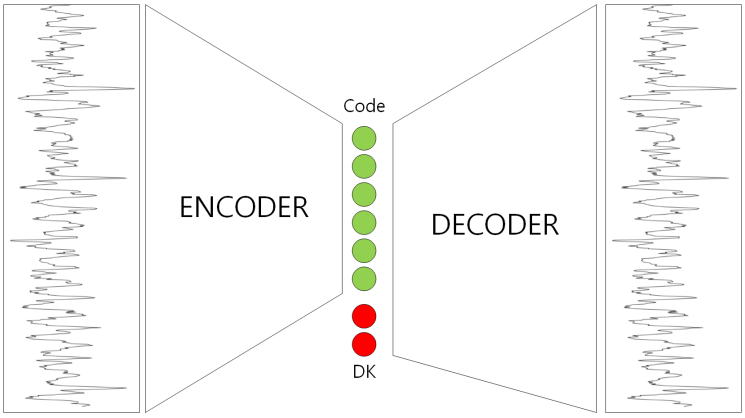


Fig. 6: Autoencoder with domain knowledge

4 Experiment Results

In this section, we validate the performance of the proposed methods through experiments and compare with conventional preprocessing methods. All experiments have been performed with TensorFlow (Version 1.13.1) [1] and Keras (Version 2.2.4-tf) [5] library from Python on a single NVIDIA GeForce GTX 1080 8GB.

4.1 Implementation Result of Unprotected AES

XMEGA. In order to analyze the noise reduction performance of proposed approach, we capture the power traces of the AES-128 implementation without side-channel countermeasures. We gether 10,000 side-channel traces from the first round of the software AES implementation on the ChipWhisperer-Lite platform [14], the target board is an Atmel XMEGA128 with a fixed clock frequency of 7.37MHz. The power consumption traces, which contain 800 samples, are captured with 29.538 MS/s sampling rate (4 points-per-cycle).

To validate the performance of the proposed method, we compared the Signal-to-Noise Ratio (SNR) of the traces according to the preprocessing methods. The results are illustrated in Fig. 7. In our implementation, PCA with sliding window technique showed the best results in window size 24, components 2, and LDA showed in window size 23, components 21. The maximum value of Both SNR results 9.5489, 5.9725 are higher than the original traces' result 5.1782, but, as presented in Fig. 7, the maximum value of SNR is 20.4902 in $SNR_{proposal}$. Comparing preprocessing methods, the results indicate that, proposed method outperforms classic preprocessing methods, PCA and LDA.

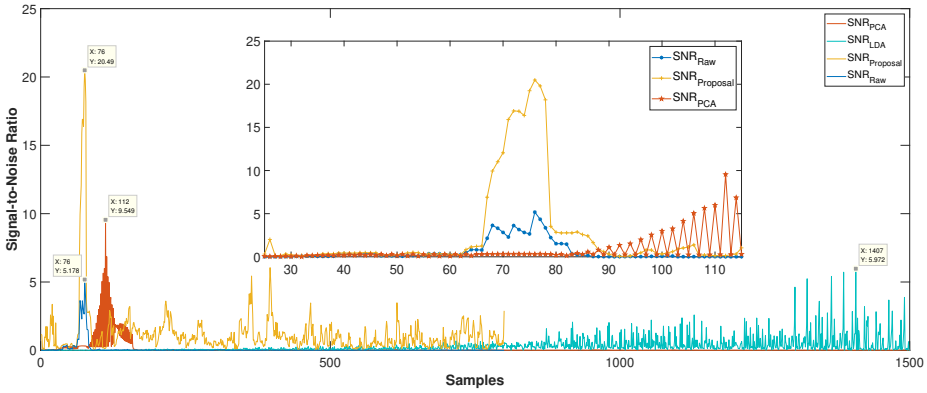


Fig. 7: Comparison of signal-to-noise ratio results for preprocessing method

4.2 Implementation Result of AES Protected by Random Delay

RandomDelay. In order to validate the performance of Realignment, we use a protected software AES implementation obtained from an 8-bit Atmel ATmega16 AVR microcontroller. The implementation of AES is protected by random delay countermeasure which is proposed by Coron et al. [6]. The measurements were performed with a LeCroy WaveRunner 104MXi DSO equipped with ZS1000 active probe, and the details of the measurement setup and the implementation are in [9]. [3] we normalize the traces by minmax scaling $X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$. The dataset contains 50,000 traces of 3,500 samples each, but we only use 25,000 traces as training set.

To validate the performance of alignment of the proposed method, we compared the absolute correlation coefficient of the traces according to the preprocessing methods. The results are illustrated in Fig. 9. In Fig. 7, the x-axis presents the number of traces used in the attack, the y-axis presents the absolute correlation coefficient, and the gray lines are the correlation coefficient for the wrong key, the red line is the correlation coefficient for the correct key. Starting from 100 traces, the absolute correlation coefficient was calculated using up to 5000 traces while increasing by 100 traces. Side-channel autoencoder with CNN encoder means that CNN is used for the encoder part of the autoencoder, and the decoder part of side-channel autoencoder is MLP. As showed in Fig. 8a, the CPA on the raw traces failed. The maximum value of absolute correlation coefficient is in side-channel autoencoder with CNN encoder, but the noise level is highest. However, considering the number of traces required for CPA, the attack can be successful with the fewest traces using the proposed technique. This results imply that the proposed methods can perform alignment.

In order to visually confirm the results, the 100 traces according to alignment technique are illustrated in Fig. 11. The results from Simple Power Analysis can

[3] The dataset is available at Kizhvatov's github 'randomdelays-traces' page, `http://github.com/ikizhvatov/randomdelays-traces`.

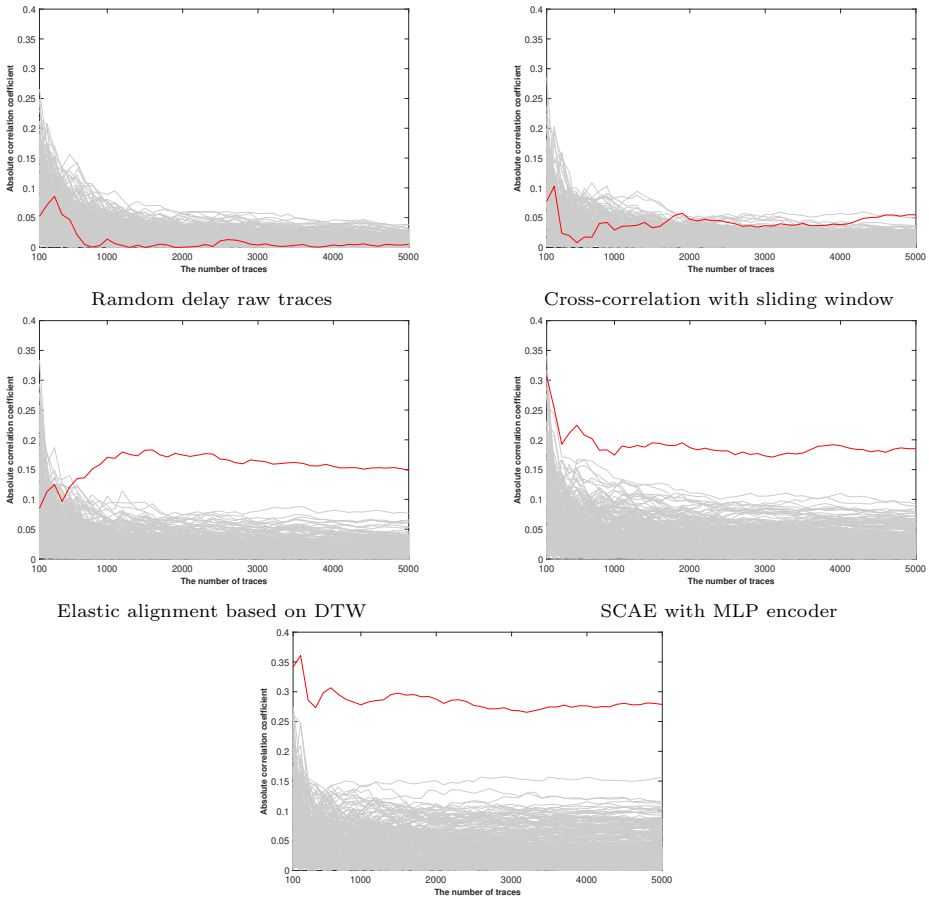


Fig. 9: Comparison of absolute correlation coefficient for preprocessing method

not demonstrate exactly, but it is clear that the raw traces and Cross-correlation with sliding window based realigned traces did not align well. Dynamic Time Warping based realigned traces 10c and proposed method based realigned traces 10d are better aligned than the two results 10a, 10b.

4.3 Implementation Result of AES Protected by First-Order Masking

ASCAD. In order to analyze the performance of proposed method, we use a software Masked AES implementation obtained from an ATMega8515 device. The dataset called ASCAD (ANSSI SCA Database) is introduced by Prouff et al. [16] to provide a benchmarking reference in side-channel analysis, like MNIST database in machine learning. The dataset *ASCAD.h5* contains 60,000 traces of

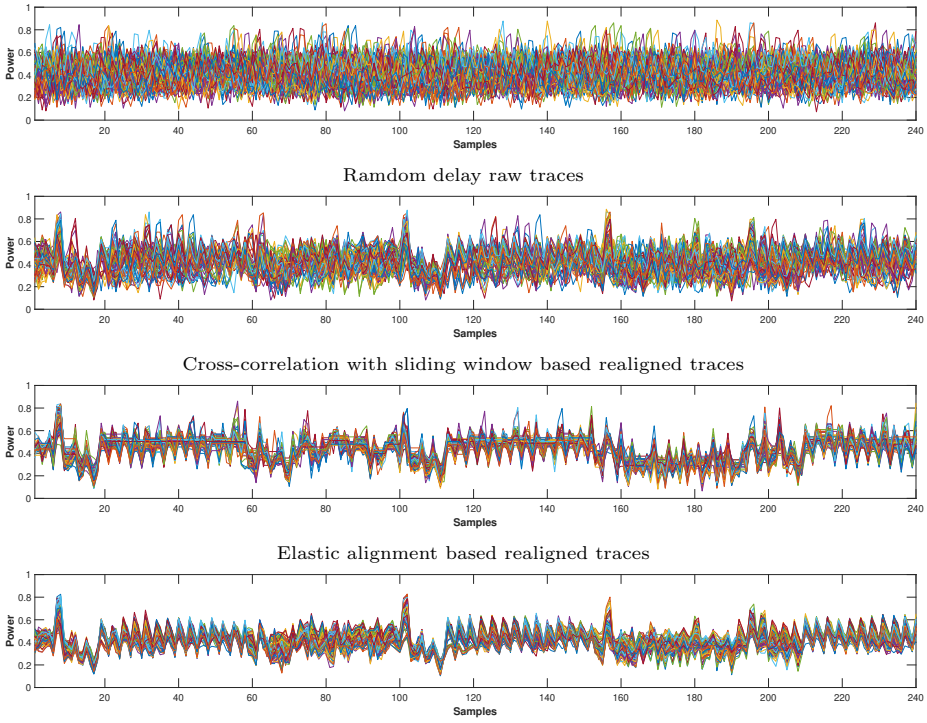


Fig. 11: Comparison of power traces for preprocessing method

700 samples each, but we only use 50,000 traces as training set. The implement of AES is protected by the masking countermeasure with different masking value for each byte. The ASCAD dataset is available at https://github.com/ANSSI-FR/ASCAD. We also normalize the traces by feature scaling, and newly add gaussian random noise centered in zero with a standard deviation 0.1 for noise reduction experiments.

In our experiments, we apply a product combining Second Order CPA with the improved product combining function [15]:

$$C_{prod}(L(t_1), L(t_2)) = (L(t_1) - E[L(t_1)]) \times (L(t_2) - E[L(t_2)]) \tag{13}$$

We combine 140 to 190 point as masking value into t_1 and 490 to 540 point as Subbytes value into t_2, and the length of combining traces is 2601. In result of raw traces, the maximum value of the absolute correlation coefficient is 0.109672 at 539 point, and the maximum value of the difference between the correlation of correct key and highest correlation in wrong keys is 0.076478. The maximum value of the correlation with our proposal is 0.193304 at 900 point, and the maximum value of the difference between the correlation of correct key and highest correlation in wrong keys is 0.136384, roughly twice higher than the

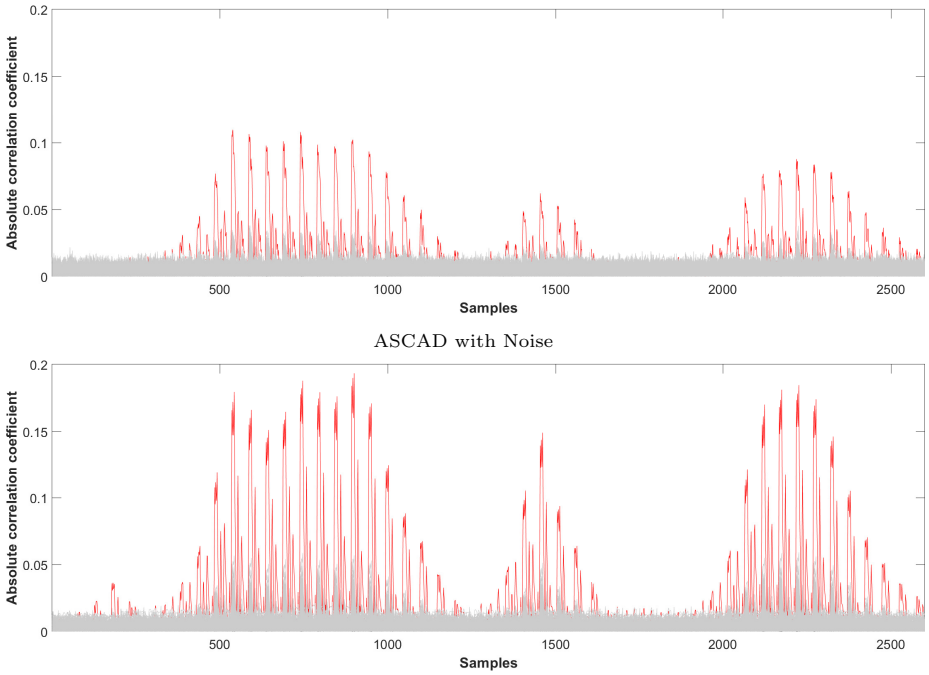


Fig. 13: Comparison of Second Order CPA results

result of raw traces. Also, we can not confirm the leakage at the result from the raw traces at 180 point, however, the correlation of correct key is higher than all correlation of wrong key in Fig. 12b. This results show that the proposed methods can improve the conventional side-channel analysis, even if the masking countermeasure is applied in implementation.

5 Conclusion

One of the reasons why the study on the deep learning based side-channel attacks are noticed is that it is possible to analyze without performing preprocessing step, which were required in the conventional side-channel analysis, regardless of whether or not the countermeasures are applied. However, end-to-end attack which perform preprocessing and analysis steps at a time can only be trained when attacker already know intermediate values of traces. Therefore, there are limitations that it performs only in profiling attack context, or that training is required as many as the estimated number of secret key. In this paper, we can perform side channel analysis using deep learning in nonprofiling attack as well as profiling attack environment by separating preprocessing and analysis steps. Furthermore, it can improve the performance of conventional side channel analysis, and we confirmed the performance experimentally. In this paper, we only

have focused on side-channel analysis in non-profiling attack environment, but we expect that the performance of profiling attack may be improved through the proposed techniques. Also, proposed method may be used to stack autoencoder to initialize a neural network for improving profiled attack, like Maghrebi's stacked autoencoder. The proposed training principle of autoencoder model in this paper is not applicable in all situations, however it can improve the performance of side-channel attacks without compromising the constraints in the non-profiling context. In addition, it can be said that the proposed techniques have given us a new approach to how deep learning can be applied to side-channel analysis, not simply classify the side-channel information.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Bishop, C.M.: Pattern recognition and machine learning. springer (2006)
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International workshop on cryptographic hardware and embedded systems - CHES 2004. pp. 16–29. Springer (2004)
4. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2017. pp. 45–68. Springer, Cham (2017)
5. Chollet, F., et al.: Keras. https://keras.io (2015)
6. Coron, J.S., Kizhvatov, I.: An efficient method for random delay generation in embedded software. In: International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2009. pp. 156–170. Springer (2009)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
8. Hettwer, B., Gehrer, S., Güneysu, T.: Profiled power analysis attacks using convolutional neural networks with domain knowledge. In: International Conference on Selected Areas in Cryptography - SAC 2018. pp. 479–498. Springer (2018)
9. Kizhvatov, I.: Physical Security of Cryptographic Algorithm Implementations. Ph.D. thesis, University of Luxembourg (2011)
10. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology - CRYPTO '99. pp. 388–397. Springer (1999)
11. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Advances in Cryptology - CRYPTO '96. pp. 104–113. Springer (1996)
12. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering - SPACE 2016. pp. 3–26. Springer, Cham (2016)

13. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards, vol. 31. Springer Science & Business Media (2008)
14. NewAE: Chipwhisperer-lite
15. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Transactions on Computers **58**(6), 799–811 (June 2009)
16. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. Cryptology ePrint Archive, Report 2018/053 (2018)
17. Robyns, P., Quax, P., Lamotte, W.: Improving cema using correlation optimization. IACR Transactions on Cryptographic Hardware and Embedded Systems - CHES 2018 **2019**(1), 1–24 (Nov 2018)
18. Timon, B.: Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems - CHES 2018 **2019**(2), 107–131 (Feb 2019)
19. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on Machine learning - ICML 2008. pp. 1096–1103. ACM (2008)
20. van Woudenberg, J.G., Witteman, M.F., Bakker, B.: Improving differential power analysis by elastic alignment. In: Cryptographers' Track at the RSA Conference. pp. 104–119. Springer (2011)
21. Wu, L., Picek, S.: Remove some noise: On pre-processing of side-channel measurements with autoencoders. Cryptology ePrint Archive, Report 2019/1474 (2019), https://eprint.iacr.org/2019/1474
22. Yang, S., Zhou, Y., Liu, J., Chen, D.: Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations. In: International Conference on Information Security and Cryptology - ICISC 2011. pp. 169–185. Springer (2011)

A Experiments over Hyperparameters

We do not claim that our hyperparameters of neural network are not optimal answer for proposal. Although the hyperparameters presented in the paper are not always the correct answer for side-channel analysis, we believe that our experiments can be used as a reference for future research. It means that the hyperparameters, used in this paper, is not essential. Therefore, through the appendix, we experimentally show that how hyperparameters affect the performance of the proposed technique.

A.1 Experiments over number of hidden layer's node using our method

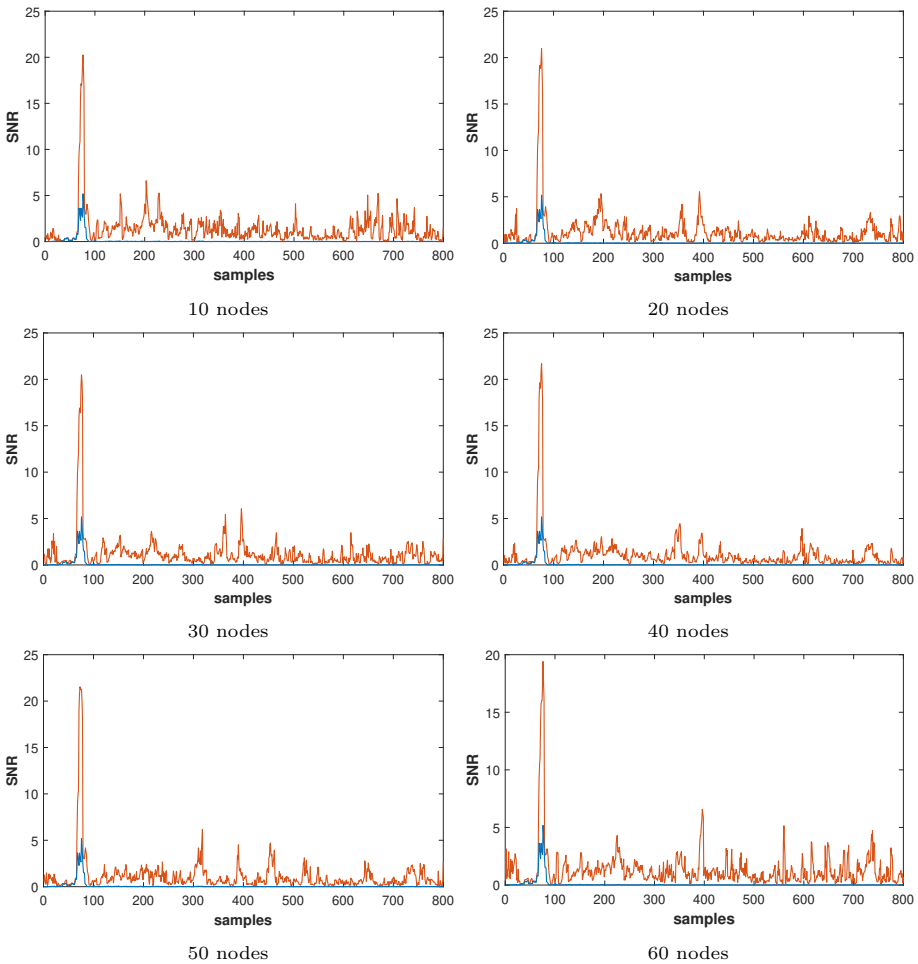


Fig. 15: Result of SNR over number of hidden layer's node using our method (1)

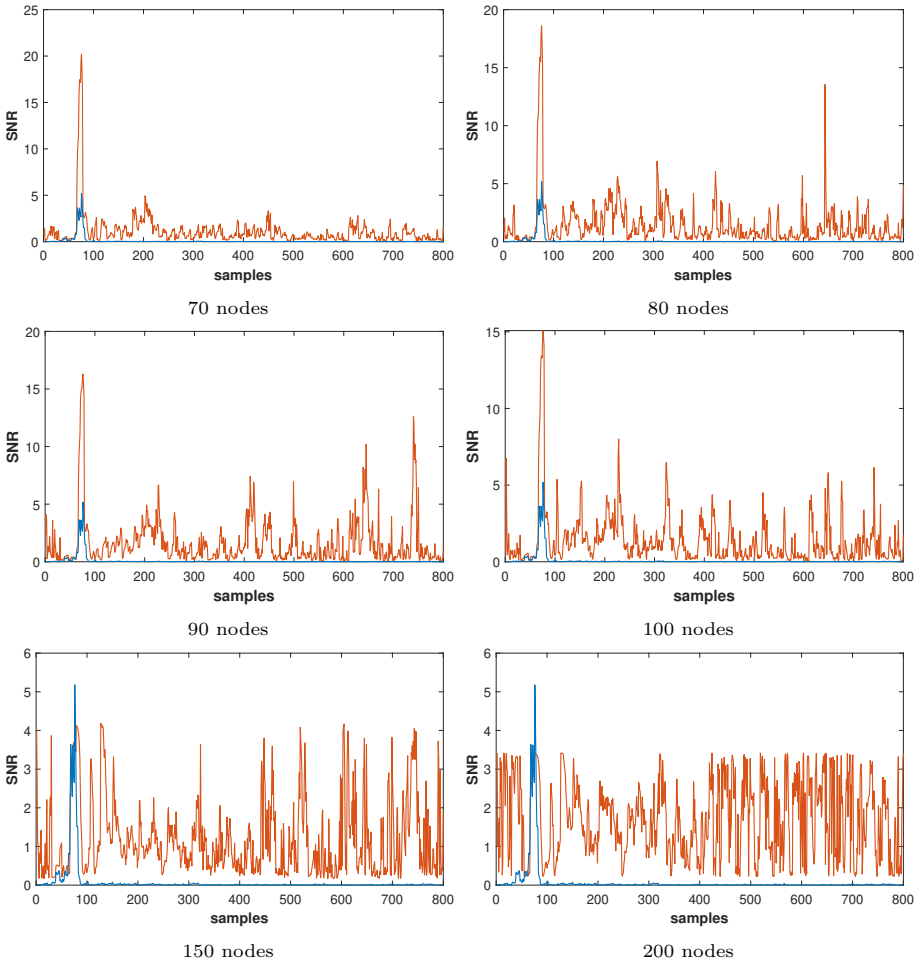
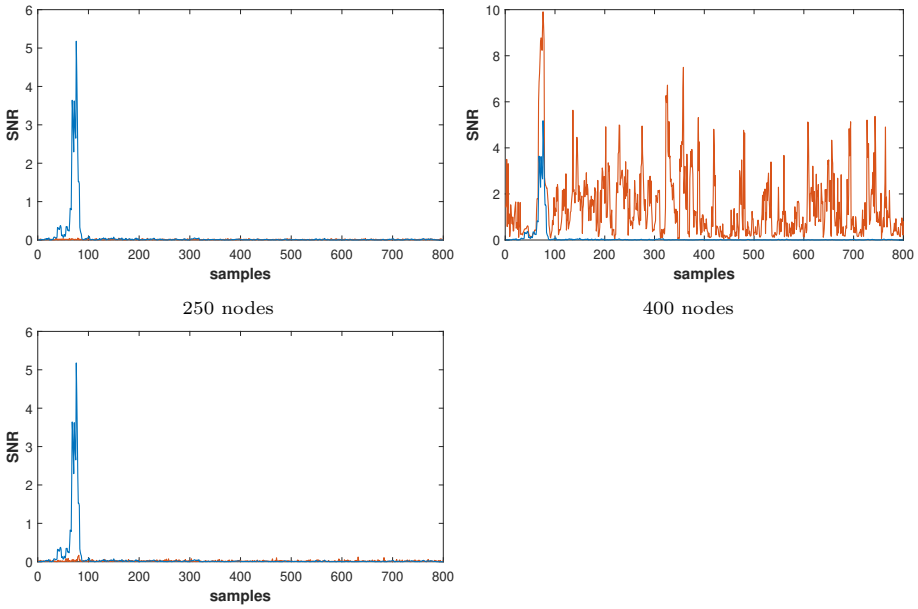


Fig. 17: Result of SNR over number of hidden layer's node using our method (2)



250 nodes

400 nodes

790 nodes

Fig. 19: Result of SNR over number of hidden layer's node using our method (3)

A.2 Experiments over hidden layer's activation function using our method

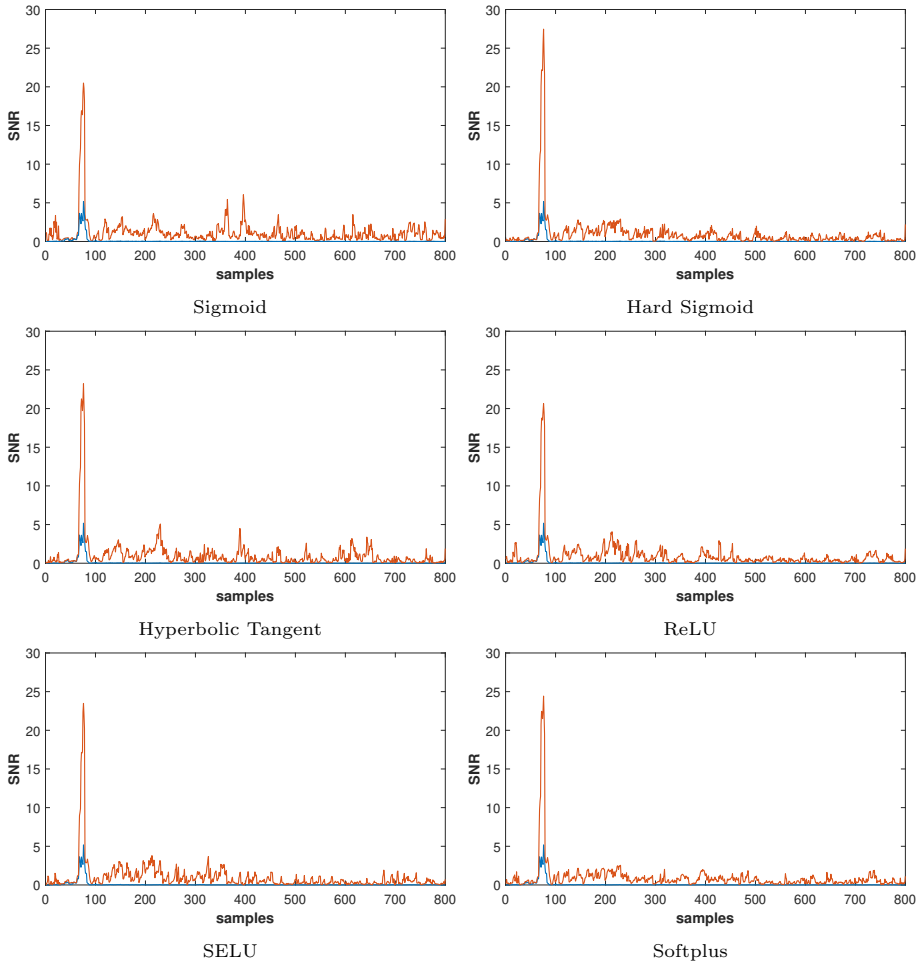


Fig. 21: Result of SNR over hidden layer's activation function using our method

A.3 Experiments over each byte using our method

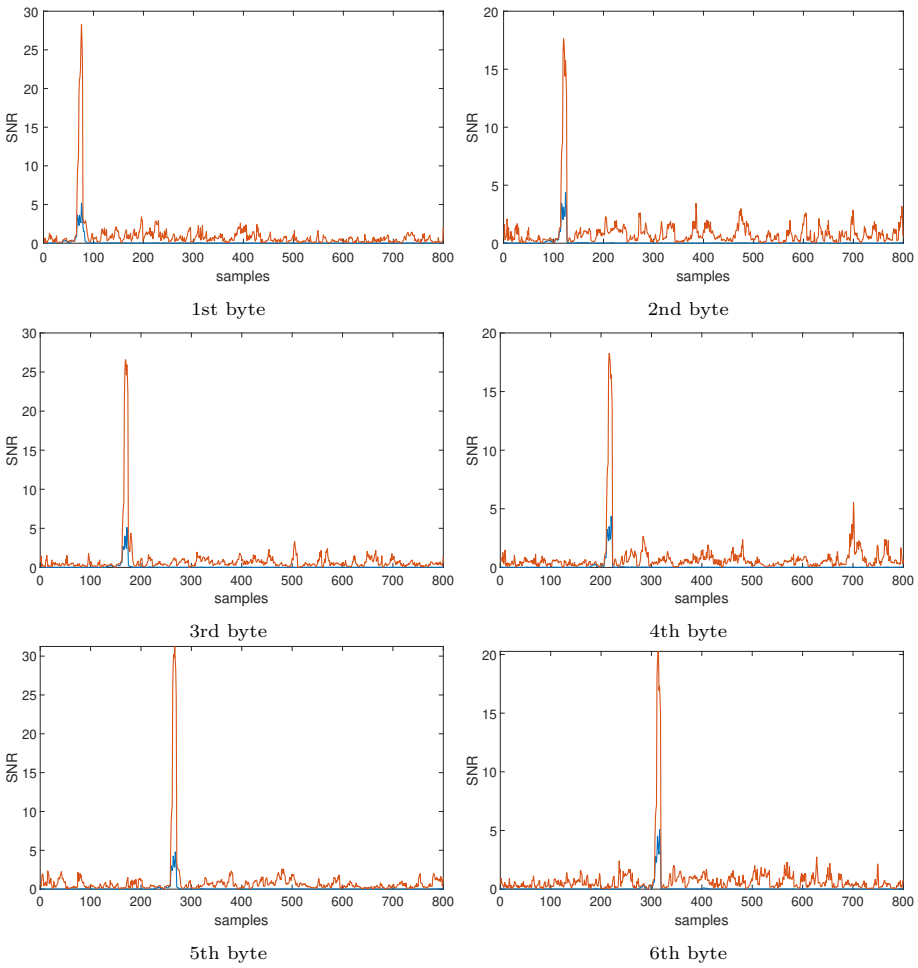


Fig. 23: Results of SNR over each byte using our method (1 6)

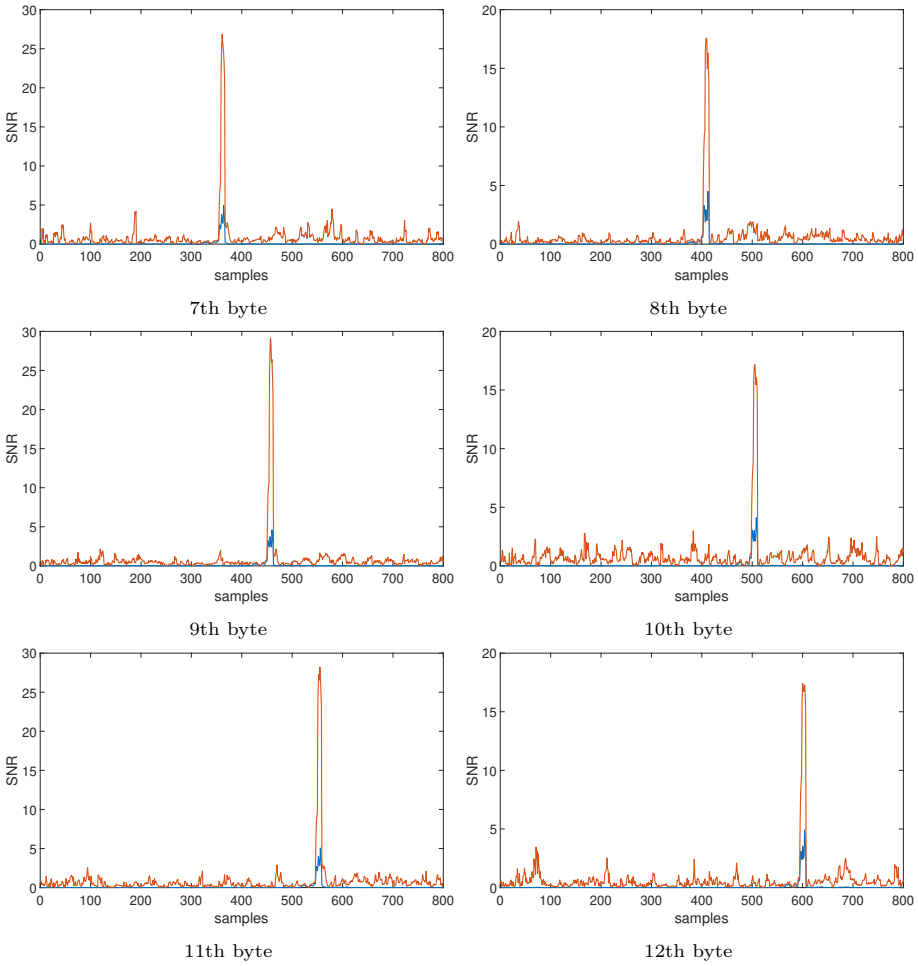


Fig. 25: Results of SNR over each byte using our method (7 12)

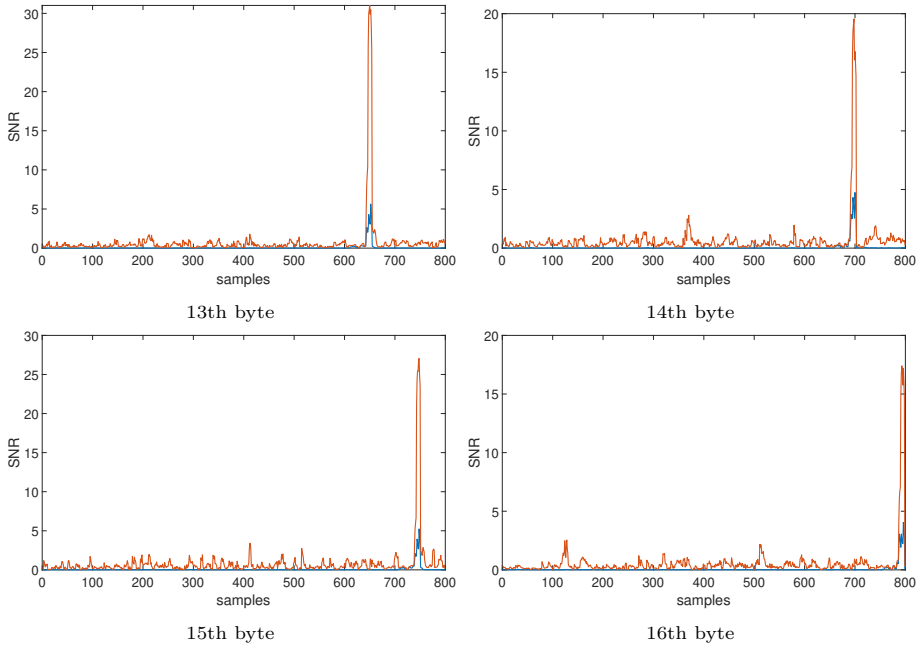


Fig. 27: Results of SNR over each byte using our method (13 16)

B Performances of Dimensionality Reduction

In order to compare the performance of dimensionality reduction, SNR results are performed according to the preprocessing techniques, proposed methods with code size 30, PCA and LDA with component 30.

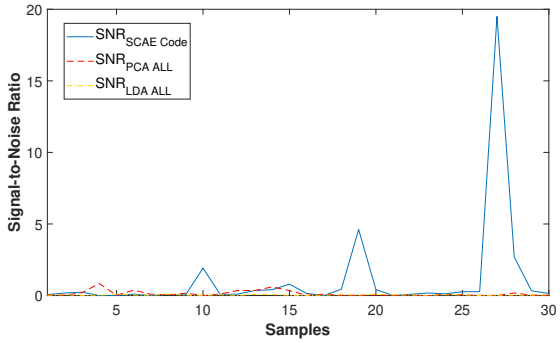


Fig. 28: Comparison of compressed traces of signal-to-noise ratio results