IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 29, 2021, accepted December 25, 2021, date of publication January 5, 2022, date of current version January 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3140446

Optimizing Implementations of Non-Profiled Deep Learning-Based Side-Channel Attacks

DONGGEUN KWON[1], SEOKHIE HONG[1], (Member, IEEE), AND HEESEOK KIM[2]
[1]Institute of Cyber Security and Privacy (ICSP), Korea University, Seoul 02841, Republic of Korea
[2]Department of Cyber Security, College of Science and Technology, Korea University, Sejong 30019, Republic of Korea

Corresponding author: Heeseok Kim (80khs@korea.ac.kr)

This work was supported in part by the Military Crypto Research Center throughthe Defense Acquisition Program Administration (DAPA) and the Agency for Defense Development (ADD) under Grant UD210027XD.

ABSTRACT The differential deep learning analysis proposed by Timon is the first non-profiled side-channel attack technique that uses deep learning. The technique recovers the secret key using the phenomenon of deep learning metrics. However, the proposed technique made it difficult to observe the results from the intermediate process, while the neural network had to be retrained repeatedly, as the cost of learning increased with key size. In this paper, we propose three methods to solve the aforementioned problems and any challenges resulting from solving these problems. First, we propose a modified algorithm that allows the monitoring of the metrics in the intermediate process. Second, we propose a parallel neural network architecture and algorithm that works by training a single network, with no need to re-train the same model repeatedly. Attacks were performed faster with the proposed algorithm than with the previous algorithm. Finally, we propose a novel architecture that uses shared layers to solve memory problems in parallel architecture and also helps achieve better performance. We validated the performance of our methods by presenting the results of non-profiled attacks on the benchmark database ASCAD and for a custom dataset on power consumption collected from ChipWhisperer-Lite. On the ASCAD database, our shared layers method was up to 134 times more efficient than the previous method.

INDEX TERMS Side-channel attacks, deep learning, non-profiled attack, differential deep learning analysis, efficient implementations.

I. INTRODUCTION

As attackers can access physical equipment, they require cryptographic algorithms embedded in the equipment for security against not only mathematical cryptanalysis but also against physical attacks. A side-channel attack is a physical attack that breaks a security system using side-channel information gained from cryptographic devices, such as operation time, sound, temperature, power consumption, or electromagnetic radiation. Cases of successful attacks through side-channel attacks on actual devices, such as mobile phones and transit cards, are rising [1], [2].

Side-channel attacks can be categorized into profiled and non-profiled attacks, depending on an attacker's environment. A profiled attack is an example of a side-channel attack, performed using a profiling device, which is the same (but not equal to) as the attack target device with a fixed secret key, such as Template Attack [3] and Stochastic Attack [4]. Attackers use profiling devices to characterize the leakage of the target device and then use the information collected to analyze the target device. Conversely, a non-profiled attack is part of a side-channel attack in an environment without profiling devices. Attackers collect side-channel information only from a target device and analyze secret keys using statistical techniques with the measurements captured from the target. These attacks involve recovering the secret key using only the side-channel information collected from the target devices without pre-calculated templates. Non-profiled attacks include Differential Power Analysis (DPA) [5] and Correlation Power Analysis (CPA) [6].

Recently, research on side-channel attacks combined with deep learning, which demonstrated high performance in various fields, has been introduced [7]–[9]. Most studies have focused on profiling attack scenarios. Similar to conventional profiling attacks, an attacker trains a neural network using side-channel information acquired through a

The associate editor coordinating the review of this manuscript and approving it for publication was Yongming Li.

profiling device. Next, the secret key is recovered by classifying the side-channel information gained from the target device using a trained neural network. Studies have shown that masking and hiding countermeasures can be exploited without preprocessing when the attacker uses deep learning-based side-channel attacks [10]–[12]. However, profiled deep learning-based side-channel attacks have a limitation because it is necessary to know the correct label for the training data to train the neural network. Therefore, this approach has been restricted to research into profiling attacks where training data and labels can be obtained.

With a focus on research in the profiling environment, Timon proposed a side-channel analysis using deep learning in a non-profiling context, called differential deep learning analysis [13]. Timon's method uses the metrics of deep learning as a distinguisher instead of conventional metrics, such as the correlation between side-channel information and intermediate values, such as the estimated power consumption or electromagnetic radiation. The attacker uses deep learning metrics, such as loss, accuracy, and gradient, as the distinguisher. The method takes advantage of the fact that an intermediate value calculated with the right key is faster to train than a value calculated with wrong keys, and the various training metrics can distinguish this. To the best of our knowledge, Timon's work is the first to demonstrate that deep learning can be applied in non-profiling attacks.

However, performing differential deep learning analysis requires training the same neural network the same number of times as the number of key guesses k. For example, if a single network needs to be trained for e epochs, then the total training time is $k \times e$ epochs. This total number of epochs required is relatively expensive compared to profiled deep learning attacks. In addition, the cost of I/O operations is not included in the algorithm and cannot be ignored because the adversary needs to initialize k identical neural network parameters to the same value. Still, determining the total number of epochs needed to distinguish the right key before an attack is an enormous challenge. In order to monitor the metrics in every epoch, the computational cost, including I/O operations, must increase. More complex neural networks may be needed to succeed in a non-profiled deep learning attack, compared to a profiled attack. Thus, higher time complexity is required. This implies that the cost of performing differential deep learning analysis is prohibitive.

Differential deep learning analysis is different from conventional non-profiled side-channel attacks such as CPA or DPA, which can obtain fixed results with a single execution. Because random factors such as initial weights and hyperparameters affect deep learning network training, analysis results must be obtained through several repetitions and then analyzed. For example, the correlation coefficient between the measurements and hypothetical intermediate values was fixed unless the data changed. However, even if the training data and their labels are the same, the training results vary from performance to performance owing to random factors, such as hyperparameters and weighted initial values. In addition, it is difficult to set hyperparameters in non-profiling environments where the attacker cannot obtain the correct labels. Therefore, it is important to reduce the attack time to obtain results for more hyperparameters or to reduce probabilistic factors through repetition attacks. Therefore, unlike in conventional attacks, the faster the attack, the better the results.

According to the previous algorithm, if the attacker performs differential deep learning analysis, the attacker will set up an initial epoch and then perform an attack. However, a non-profiled deep learning attack can fail at the initial set value. If more training is needed, the attacker must start the training from the beginning. When the attacker modifies the algorithm and stored each model, they can load the stored training parameters and perform the attack again using these parameters; however, in this case the added cost of storing models as many as the number of key guesses must be considered. Therefore, the attacker needs to monitor the results of each epoch to decide when the attack stops learning. It makes that the attack can be trained at less than the value set up at the beginning. This called an early stopping. Early stopping [14], which is widely used in deep learning, is clearly necessary and has been applied in profiled deep learning side-channel attacks [12], [15], [16]. Non-profiled attacks are no exception. Prior research recognized these problems and suggested reducing complexity by monitoring, but failed to suggest a practical method to achieve this. In this paper, we propose novel methods that can monitor training metrics that reflect the intermediate training process and help perform faster than the previous approach.

A. OUR CONTRIBUTIONS

The contributions of this paper can be summarized as follows.

1) *Modifying a Timon's algorithm to apply an early stopping technique that can prevent overfitting*
   In Timon's algorithm [13], an attacker cannot observe metrics, such as accuracy, loss, and gradient in the intermediate process of training. Therefore, it is difficult for the attacker to set an epoch and check the result or terminate it in advance; instead, they must wait for all the training to be performed. In this paper, we modify the algorithm to allow an attacker to monitor every epoch. Owing to this modification, the attackers can apply the early stopping technique, which is used to reduce overfitting and time complexity. Furthermore, we show the problems that arise when the algorithm is changed, and propose new neural network architectures to solve these problems.
2) *Introducing a novel neural network architecture in parallel to improve speed*
   The previous method had to set the neural network so that it corresponded to each key guess and trained the weights of each network separately. For example, when attacking the first subkey byte in the first AES round, the attacker has to train 256 neural networks, while the size of the key is guessed. Because the number of

backpropagations required for a single neural network is (number of epochs) × (training set size) / (batch size), the number of backpropagations for an attack is 256 × (number of epochs) × (training set size) / (batch size). When the size of the key being guessed is k, the total number of backpropagations is k × (number of epochs) × (training set size) / (batch size). In this paper, we propose a parallel strategy to reduce the time complexity of differential deep learning analysis. Our method performed training separately for each key estimated, while we integrated it into a single network.

3) *Proposing a novel method with shared layers to reduce memory and time costs*
When an attacker designs the algorithm, they can observe the intermediate training process, and this reduces speed. When an attacker uses a parallel architecture, the disadvantage is that the memory complexity increases because many weights are stored in memory at once. In this paper, we propose a new methodology that uses shared layers to reduce both memory and time complexity. In the parallel method, several training parameters are required because all the networks, according to all the key guesses, are independently composed. However, if the networks share layers or neurons, memory usage can be reduced. Our shared layers encode the data and decrease the number of weights for the networks by reducing the data dimensions during the encoding process, reducing memory complexity.
4) *Implementing the proposed methods as well as verifying and comparing these methods with the previous method*
We propose three methods to improve on the previous differential deep learning analysis. We expected our methods to recover the secret key and reduce time and memory usage costs, except in the method that was modified to use only early stopping techniques. To verify the strategies and compare them with the previous attack, we implemented the parallel method and shared layers method. Through implementation, we showed that the proposed methods can successfully perform side-channel attacks, similar to the conventional method. In our experiments, all the proposed methods were faster than Timon's work. In addition, the method with shared layers has less memory usage than the previous method. According to our experiments on the ASCAD database, an attack using the shared layers technique is 134 times faster than the execution time of a conventional attack with a similar memory usage level.

B. ORGANIZATION

We organised this paper as follows. Section II describes conventional side-channel attacks with deep learning and is most related to this study's work, differential deep learning analysis. In Section III, we present the proposed method using novel neural network architectures. Then, we verify this proposed method with previous work in Section IV. Finally, Section V includes concluding remarks and a discussion of future work.

II. PRELIMINARIES

In this section, we briefly review a previous attack, differential deep learning analysis, which is most closely related to our work. For a general introduction to deep learning, we refer interested readers to [17].

A. DEEP LEARNING-BASED PROFILED SIDE-CHANNEL ATTACKS

Deep learning has attracted much attention recently owing to developments in hardware and improvements in learning algorithms that have resulted in significantly improved results in various fields, such as computer vision, natural language processing, and speech recognition. Recently, deep learning has been actively studied in the field of side-channel analysis, and has been used as a classifier to classify side-channel measurements [18]–[21]. Research on classifying measurements using deep learning was conducted primarily in a profiling environment in which an attacker uses a profiling device. The attacker characterizes the leakage of the profiling device and then analyzes the measurements collected from a target device using the characterized information.

Maghrebi's research results show that deep learning-based profiling attacks can be analyzed regardless of whether masking response techniques are applied [11]. Results by Cagli *et al.* show that if an attacker uses convolutional neural networks, it is possible to recover the secret key through deep learning-based side-channel attacks without performing preprocessing steps such as alignment [12]. Open datasets are provided to compare and benchmark the analysis performance based on deep learning [22]. Recently, not only on block ciphers, but also deep learning based attack have been performed on public-key cryptosystems [23], [24].

B. DEEP LEARNING-BASED NON-PROFILED SIDE-CHANNEL ATTACK; DIFFERENTIAL DEEP LEARNING ANALYSIS

Differential deep learning analysis (DDLA) is the first type of deep learning-based side-channel analysis in use in non-profiled scenarios, where the attacker cannot obtain a template device [13]. Without a template device, it is not possible to obtain the label for side-channel measurements, so it becomes difficult to apply the profiling side-channel attacks based on deep learning, as described in the previous subsection. However, DDLA overcomes these limitations by providing a methodology for guessing labels. In a correlation power analysis, which is a type of conventional side-channel analysis, intermediate values guessed with the right key are most correlated with measurements. Similarly, in deep learning analysis, a neural network trained with the correct label performs better than the other neural networks trained with incorrect labels. Various metrics describe the performance

of deep learning, but in DDLA, these metrics are used as a distinguisher. The attacker can distinguish the right key from the wrong key using this type of distinguisher. Algorithm 1 is the algorithm for the DDLA in AES implementation.

Algorithm 1 Differential Deep Learning Analysis [13] (Case: AES)

Input: N traces $(t_i)_{0\le i<N}$ with corresponding plaintexts p_i. A neural network *Net*, number of epochs n_e, and substitution box $Sbox(\cdot)$.

Output: Metrics of training M.

1: Set training data as $X = (t_i)_{0\le i<N}$
2: **for** $k = 0$ to 255 **do**
3: Re-initialize trainable parameters of *Net*
4: Compute the series of hypothetical values $(H_{i,k})_{0\le i<N} = Sbox(p_i \oplus k)$
5: Set training labels as $Y_k = (H_{i,k})_{0\le i<N}$
6: // Perform Deep Learning training: $DL_k = DL(Net, X, Y_k, n_e)$
7: **for** $e = 1$ to n_e **do**
8: Perform Deep Learning training: $DL_k = DL(Net, X, Y_k, 1)$
9: **end for**
10: Calculate DL_k metrics m_k
11: **end for**
12: **return** Metrics $M = (m_0, m_1, \ldots, m_{255})$

The execution procedure of the algorithm is as follows. First, the attacker sets the collected traces as the training data X. Next, the following steps were repeated according to each key guess. First, initialize or re-initialize the trainable parameters of the neural network *Net*, such as the weights and biases. Second, we estimate the intermediate values through the guessed secret key value k and calculate the hypothetical values H_i using the hypothetical intermediate values and power consumption model. Third, we set the hypothetical power consumption value calculated above as the label Y_k of the training data according to the guessed key. Finally, we use the training data X and label Y_k to train *Net* for as many epochs as n_e. If the attacker has completed the deep learning training DL for all key guesses, the attacker can recover the right key, which leads to the best DL metrics M. Figure 1 shows the results of the training metrics, such as accuracy and loss, for the key guess. The loss and accuracy of training with the correct label differed significantly.

In the first method, sensitivity analysis based on multilayer perceptron (MLP) first layer weights uses the weights of the first hidden layer of the MLP, which is the same as the corresponding name. The first hidden layer has parameters, called weights, that are connected to the input and hidden layers. The neural network updates the weights of the hidden layer to minimize loss through learning. In this process, the updating increases the absolute value of the weights connected to the feature points of the input to transfer the features of the input to the next hidden layer. Therefore, when learning through a

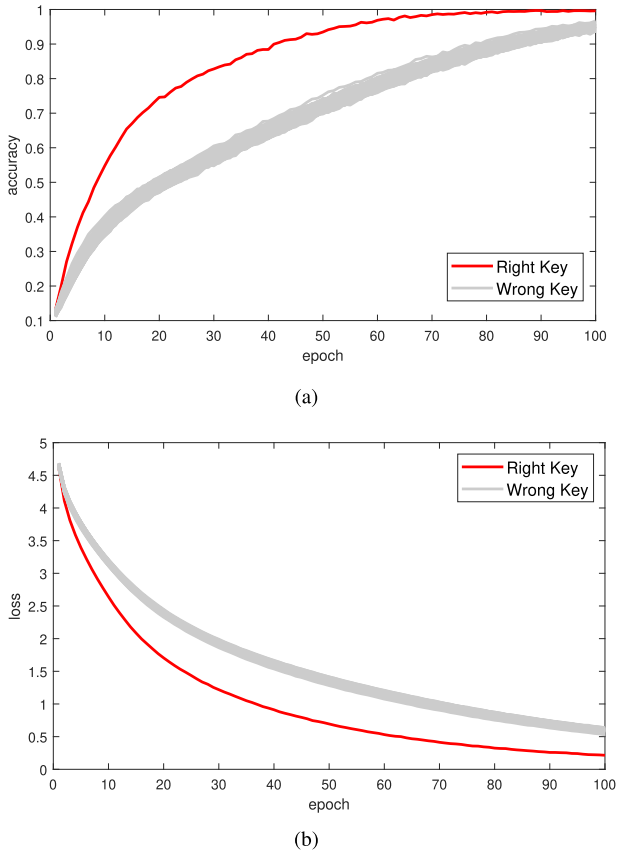


FIGURE 1. Examples of DDLA results: (a) accuracy and (b) loss.

label generated with the correct key, the weights connected to the feature point of the input have significant values and are updated considerably. This implies that the gradient of the weights was large. However, when learning with labels generated through the wrong key, the neural network cannot find the feature points that distinguish the label groups or the wrong feature points, so the weights connected to the feature point of the input are only updated slightly. Given these results, the attacker can recover the secret key using the gradient of the weight of the first hidden layer as the distinguisher. The attacker calculates $S_{weights}$ according to each key guess, and the $S_{weights}$ with the maximum value can be recovered for the secret key.

$$\nabla W_{i,j} = \frac{\partial \mathcal{L}}{\partial W_{i,j}}$$
$$S_{weights}[i] = \sum_{j=1} \left| \nabla W_{i,j} \right|$$

The second method is sensitivity analysis based on network inputs, which uses a backpropagation value, the gradient that updates the weight of the neural network. When attackers used a convolutional neural network to attack an implementation with hiding countermeasures, the first method became difficult to apply because there is no weight matched with the input dimensions one-to-one in the convolutional neural

network, making it dissimilar to MLP. Therefore, when an attacker uses CNN, the input-based sensitivity analysis technique should be applied. As in the first method, an input-based technique can also be applied by calculating the S_{inputs} according to the key guess, and the S_{inputs} with the maximum value can be recovered for the secret key.

$$S^1_{inputs}[j] = \sum_{i=1}^{N} \left| \nabla T_{i,j} \right|$$

$$S^2_{inputs}[j] = \sum_{i=1}^{N} (\nabla T_{i,j} \times T_{i,j})$$

In DDLA, loss and accuracy are primarily used as the metrics M to recover the secret key. Training with the correct labels tends to reduce loss and increase accuracy when learning is performed. However, when training with labels generated by wrong keys, the metrics do not change much or get worse, so the previous mentioned phenomenon is not observed. Therefore, the wrong key model is distinguishable from the right key model. Figure 1 shows the results for loss and accuracy for each key guess. In this paper, we used the most common metric, accuracy.

III. PROPOSED METHODS

In this section, we propose several methods to overcome the limitations of the previous DDLA proposed in Timon's work. First, we introduce a modified algorithm such that the metrics in the intermediate process can be monitored by the DDLA algorithm as well as the problems associated with this change. Second, we propose a parallel method to solve the problem of time complexity. We confirmed that the networks, according to each key guess, are independent of each other, and thus we know that each network can be used in parallel. Therefore, we propose a novel algorithm that can be performed in parallel by changing the neural network architecture: *networks in network*. Finally, we propose shared layers that solve the speed and memory problems. Our method using this parallel technique is fast, but the memory complexity is increased because the networks have to be stored in memory at the same time. We know all the networks used in the parallel method are the same network and solve the memory problem by sharing the hidden layer connected to the input layer.

A. MODIFIED DIFFERENTIAL DEEP LEARNING ANALYSIS FOR MONITORING METRICS

In Algorithm 1, the process of training *Net* is repeated with the labels according to each key guess. When the network is trained with the label computed by the guessed key k, the results or metrics for the other key (larger than k) cannot be checked until all the epochs the attacker selected have been performed. In other words, even if k has the best result among the metrics results from 0 to k, the attacker cannot stop the attack because the result of a value higher than k is unknown. Therefore, in order to monitor the results of all key guesses in the intermediate training process, key guesses should be the inner loop, and the epoch should be the outer loop. When modifying the algorithm, it is not reasonable to create a label for each epoch. Thus, it is efficient to calculate and store labels for all key guesses in advance. At this time, additional memory space is required to store the all key guessed labels.

When training for the n_e epochs, where the attacker presets for all key guesses, it is not guaranteed that the key can be selected at all times from the metrics M obtained through the attack. There are several reasons this might happen, but among them, if the number of training epochs is insufficient and then the attacker fails, the attacker must select a higher value n'_e than the previous number of epochs n_e and perform the algorithm again. In order to solve the failure problem, the trained networks for all key guesses should be stored, and the attacker must reload the networks when they fail to recover the key and perform additional training. At this time, the operation and memory resources for storing the trained networks are required. Algorithm 2 depicts a modified DDLA for observing metrics on a per-epoch basis.

Algorithm 2 Modified Differential Deep Learning Analysis for Early Stopping (Case: AES)

Input: N traces $(t_i)_{0\le i<N}$ with corresponding plaintexts p_i. A neural network *Net*, number of epochs n_e, substitution box $Sbox(\cdot)$, and early stop function $ES(\cdot)$.

Output: Metrics of training M.

```
Set training data as X = (t_i)_{0≤i<N}
for k = 0 to 255 do
    Compute the series of hypothetical values (H_{i,k})_{0≤i<N} = Sbox(p_i ⊕ k)
    Set training labels as Y_k = (H_{i,k})_{0≤i<N}
end for
Initialize trainable parameters of (Net_k)_{0≤i≤255}
for e = 1 to n_e do
    for k = 0 to 255 do
        Load trainable parameters of Net_k.
        Perform Deep Learning training: DL_k = DL(Net_k, X, Y_k, 1)
        Calculate DL_k metrics m_{k,e}
        Save trainable parameters of Net_k
    end for
    // Early stopping technique or monitoring
    if ES(M = (m_0, m_1, ..., m_255)) is true then
        Break
    end if
end for
return Metrics M
```

Algorithm 2 is a modified differential deep learning analysis that solves the problems of monitoring. Because the order for training and setting the labels is changed, it is possible to attack using the same network as the network used in Timon's algorithm. The label-setting process performed for each key guess was pulled out with a separate loop. The key guessing loop and the epoch loop are changed, and the loop

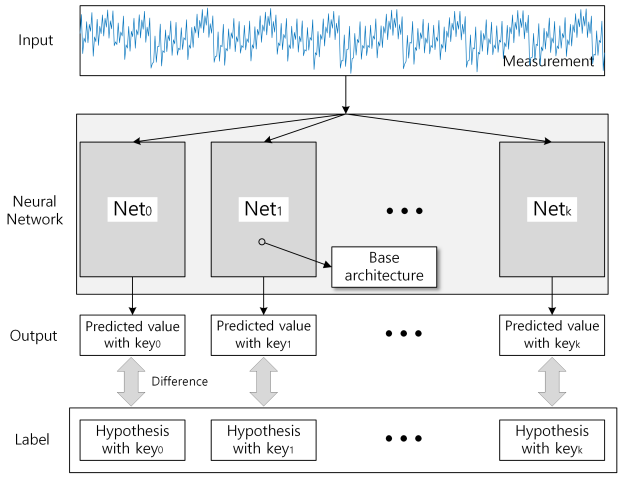


FIGURE 2. Proposed neural network architecture with parallel methodology.

change allows the network for each key guess to be stored. In addition, we added lines 15 to 17 to the algorithm to check whether the early stop condition is satisfied for each epoch with the metrics M so that the attacker can stop training early. With early stopping, if the attack is successful, the process is terminated earlier than expected to prevent overfitting, where the network memorizes the training data instead of learning the features of the data. Even in the case where the attack fails, it determines whether the proper hyperparameter is chosen early, so the algorithm can be stopped in the middle of the key recovery attack, and it reduces the unnecessary cost. However, additional storage is required for this modification.

B. PARALLEL METHODOLOGY FOR DIFFERENTIAL DEEP LEARNING ANALYSIS

In this subsection, we propose a new algorithm and a novel neural network architecture that can improve the performance of DDLA. Using the *networks in network* methodology, we design the architecture to work in parallel and replace the previous algorithm so that attacks can be performed by training only a single network. In this paper, a small network within a neural network, denoted Net_i, is called the base architecture for key i. The base architecture is the neural network used in the previous work. The proposed neural network architecture replicates the base architecture for the number of key guesses, sharing only the input layer and output layer. In addition, our architecture is locally-connected such that the other parameters are not connected. The proposed neural network architecture is illustrated in Figure 2.

In Timon's training parameters, MSB (most significant bit) or LSB (least significant bit) labeling was used, and we used the same labeling method. They used the dense output layer of two cells with the softmax function for one-hot encoding. If the output of the first cell is higher than the second, the MSB or LSB is predicted to be 0. If not, it is determined to be 1. However, we use the dense output layer of single cell with the sigmoid function for multi-hot encoding. If the output of the single cell is lower than 0.5, the bit is predicted to be 0. If not, it is determined to be 1. This technique reduces the dimensions of the output layer and cost because performing one-hot encoding is unnecessary and only MSB or LSB is used. Thus, the base architecture is the same as the hyperparameter used in Timon's paper, except for the output layer. Each output cell represents the predicted value for each key guess, as shown in Figure 2. Therefore, category loss functions are not available, and binary cross-entropy or the sum of squared functions should be used as loss functions. Table 1 compares the network used in Timon's work and the proposed network.

When using the proposed architecture, non-profiling attacks should be performed using Algorithm 3, not Algorithm 1 or 2. The main difference between the other algorithms and Algorithm 3 is that Algorithm 3 trains a single neural network, not several networks. Up to line 6, the label setting and initialization of the training parameters are performed, similar to Algorithm 2. After line 6, the algorithm trains the network, which can be seen in many other deep learning algorithms.

Algorithm 3 Improved Differential Deep Learning Analysis (Case: AES)

Input: N traces $(t_i)_{0\le i<N}$ with corresponding plaintexts p_i. A neural network Net, number of epochs n_e, AES substitution box $Sbox(\cdot)$, and early stop function $ES(\cdot)$.

Output: Metrics of training M.

1: Set training data as $X = (t_i)_{0\le i<N}$
2: **for** $k = 0$ to 255 **do**
3: Compute the series of hypothetical values $(H_{i,k})_{0\le i<N} = Sbox(p_i \oplus k)$
4: **end for**
5: Set training labels as $Y = (H_{i,k})_{(0\le i<N),(0\le k<256)}$
6: Initialize trainable parameters of Net
7: **for** $e = 1$ to n_e **do**
8: Perform Deep Learning training: $DL = DL(Net, X, Y, 1)$
9: Calculate DL metrics $(m_{k,e})_{0\le k<256}$
10: // Early stopping technique or monitoring
11: **if** $ES(M = (m_0, m_1, \ldots, m_{255}))$ is **true then**
12: Break
13: **end if**
14: **end for**
15: **return** Metrics M

However, the memory burden increases when the parallel method is used. This is because the networks, according to each key guess, should be put into memory all at once. For example, if the attacker recovers an AES 1-byte subkey, the attacker has to upload 256 networks to memory at once. The attacker can perform an attack with a DDLA method by choosing an algorithm that depends on the attack environment selectively.

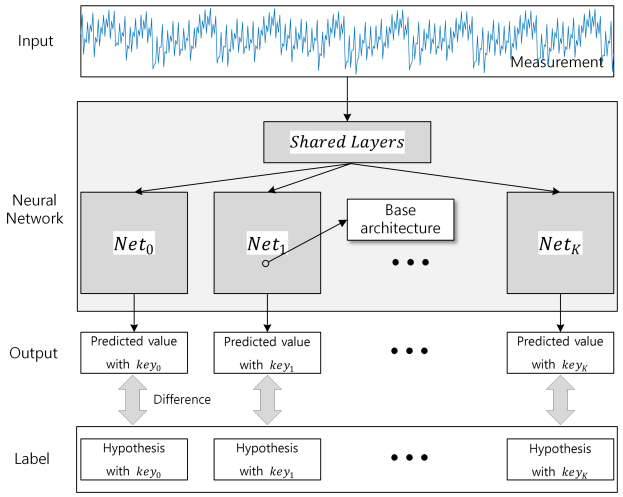


FIGURE 3. Proposed neural network architecture with shared layers.

C. IMPROVED DIFFERENTIAL DEEP LEARNING ANALYSIS WITH SHARED LAYERS

The DDLA with the parallel technique is faster than a conventional attack, but the memory burden increases with the size of the key being guessed. Memory burden comes from putting the weights of the networks for all key guesses into memory at once. The most memory-intensive part of the process is to store the weights of the first hidden layer. No matter how small the hyperparameter of the base architecture is designed to be, the number of weights increases significantly if the length of the trace is long. For example, when the length of the trace is 1, 000 and the dimensions of the first hidden layer of the base architecture are 200, $1000 \times 200 \times 256$ training parameters are required for the weights.

We propose an architecture that encodes measurements using shared layers to reduce the dimensions of the measurements before transferring them to each base architecture. A similar approach to this concept, which encodes side-channel measurements using deep learning was proposed by Robyns [25]. Also, in [26], they also separated a neural network for Points of Interest (PoI) detection and plaintext feature embedding. The part of the network that detects PoIs is similar to our shared layers. Considering that profiled deep learning-based side-channel attacks perform preprocessing and classification through the network; end-to-end attack, it is reasonable to have the network perform preprocessing. Therefore, the attacker was designed to preprocess the traces using the shared layers and deliver the preprocessed traces to each base architecture so that all key guesses can be performed through one network similar to the proposed parallel method. Because only the neural network architecture has been switched and the input and output are the same, the attack can use the same algorithm (Algorithm 3) as the parallel method. Figure 3 shows the neural network architecture with the proposed shared layers.

Comparing Figure 3 and Figure 2, it can be seen that when the number of layers in the shared layers is increased, many parts of the base architecture are shared, and the number of training parameters decreases. This approach clearly reduces the memory burden. Moreover, as the shared weight increases, the weights that need to be updated are reduced, and the training speed increases. Therefore, if the performance of the analysis is the same, it is advantageous to design the architecture to share as much as possible.

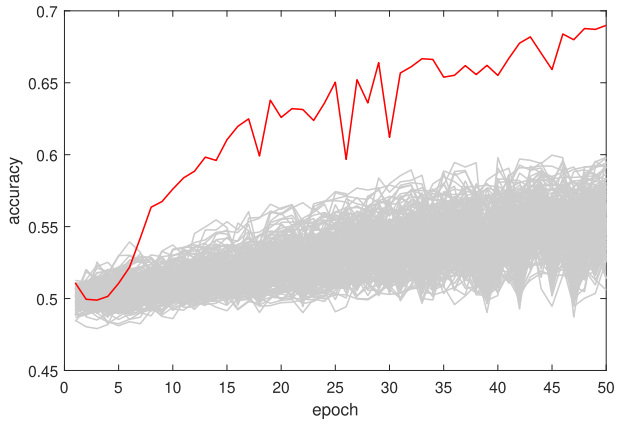
However, with a neural network, which is a black-box model, it is challenging to determine the extent to which the shared layers are affected. In other words, it is difficult to confirm in advance whether the performance of the attack will improve or worsen by sharing many weights, and the performance change according to the shared part is not verified. We show that it is possible to perform a non-profiled deep learning attack, and we experimentally validate the sharing technique in the following section. In order to verify the performance of the shared layers method, we set the architecture that maximized the shared part in all of our experiments. This means that we use the most different architecture from the parallel method.

IV. EXPERIMENTAL RESULTS

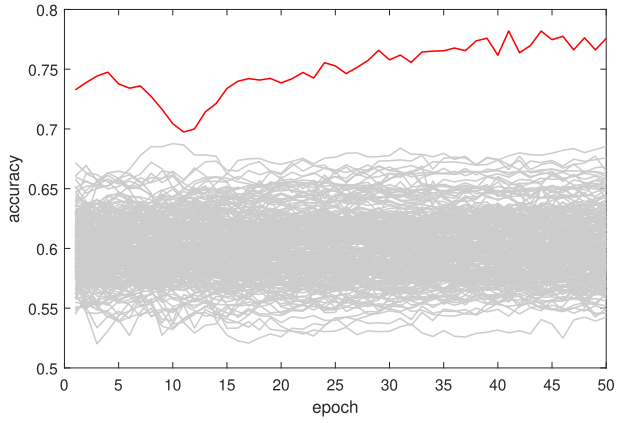
In this section, we validate the proposed methods introduced in Section III. We do not show the experimental results of the method proposed in Section III-A because it is simply a modification of the order of operation from Timon's work. In our experiments, we used same hyperparameters provided in Timon's work [13] as much as possible, except for a batch normalization layer [27], which makes training faster and reduces internal covariate shift through normalization for each training mini-batch, and fine-tuning only the initial learning rate. We did not consider that Timon's hyperparameters are optimal for our proposals because we do not claim that they are best. In this paper, we focus only on the success or failure of an attack and its complexity.

A. ATTACK ON ChipWhisperer-LITE

In order to verify the performance of the proposed methods, we use side-channel measurements collected by ChipWhisperer-lite [28]. We program the AES-128 1 Round SubBytes without countermeasures code on the ChipWhisperer-lite evaluation board with an Atmel XMEGA128 8-bit processor with a fixed clock frequency of 7.37 MHz. We captured the power trace sampled at four points per clock. The dataset includes 10,000 power consumptions of 800 samples for AES SubBytes operation with random plaintexts and a fixed key. The experimental results obtained using the proposed method with the collected power consumption dataset are shown in Figure 4. We target its 1-byte subkey; the red line represents the accuracy for the right key, and the gray lines are the results for the wrong keys. As shown in Figure 4, the difference between the two lines is clear. This results confirm that attackers can recover the secret key successfully.

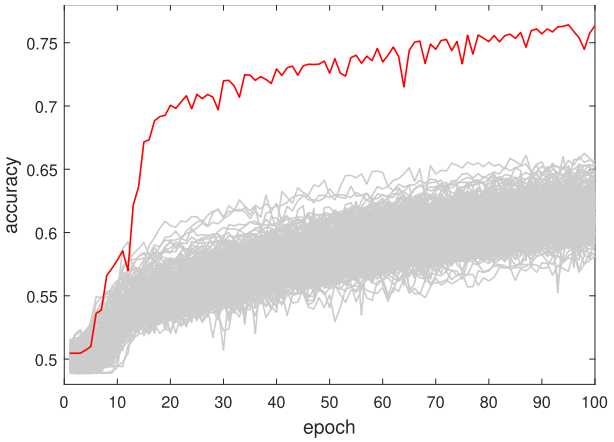


(a) PL

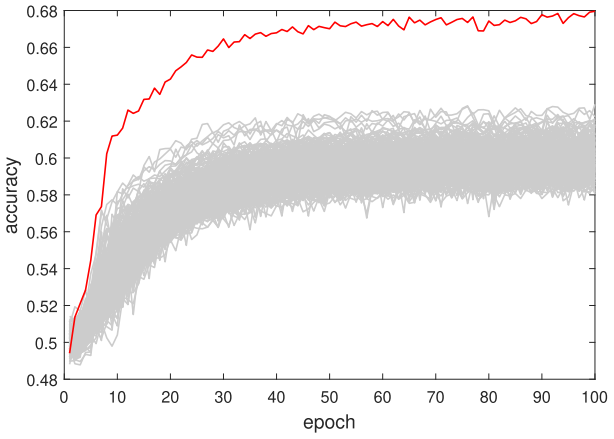


(b) SL

FIGURE 4. Comparing DDLA results against ChipWhisperer-Lite.



(a) PL



(b) SL

FIGURE 5. Comparing DDLA results against de-synchronized traces.

B. ATTACK ON DE-SYNCHRONIZED TRACES

Timon showed that if the attacker performs DDLA with convolutional neural networks (CNN), which is called CNN-DDLA, implementations with hiding countermeasures can be exploited without preprocessing because of the translation-invariance property of the CNN architecture [13]. In this paper, we show that the proposed method can also be applied to CNN-DDLA. We construct the base architecture with convolutional layers and attack de-synchronized traces.

The traces were created by arbitrarily shifting the measurements captured from the ChipWhisperer-Lite used in the first experiment. We set the maximum range for the shift to 20, which implies that we obtained a shifted trace of 780 points from the 800 point trace. The shifted traces were obtained from the traces in the ChipWhisperer-Lite database without increasing the number of traces. CNN_{exp} was used as the base architecture [13]. The experimental results obtained using CNN-DDLA with the proposed methods are shown in Figure 5. Similar to the previous results, the red line represents the right key, and the gray lines indicate the wrong keys. As can be seen in Figure 5, the difference between the two types of lines is clear. This shows that attackers can recover the secret key successfully, even if a hiding countermeasure is applied.

C. ATTACK ON ANSSI SCA DATABASE (ASCAD)

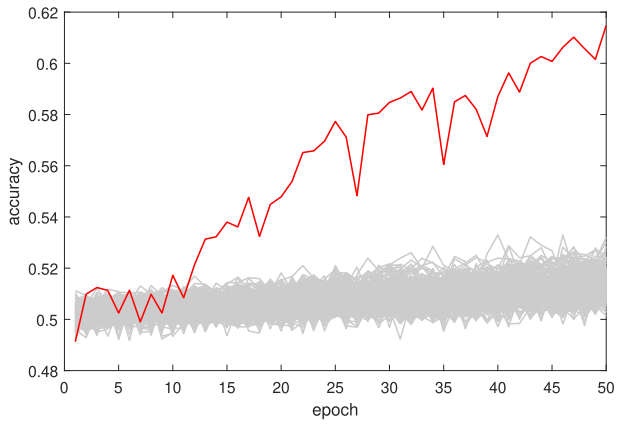
The third dataset to be analyzed is a set of databases, called ASCAD [22], which was released as a benchmark reference for side-channel analysis. The data were collected by a 2Gs/s sampling rate for the implementation of AES with Boolean masked countermeasure operated by an Atmel ATmega8515, and there were a total of 60,000 electromagnetic measurements. In our experiments, we used 20,000 electromagnetic traces captured during the AES SubBytes operation of the third byte, with 700 points in the ASCAD database. Figure 6 shows the experimental results of applying the proposed methods to the open dataset. The red line is the accuracy for the right key, and the gray lines are the results for the wrong keys. As can be seen in Figure 5, the difference between the two lines is clear. This indicates that attackers can successfully recover the secret key even if a masking countermeasure is applied.

D. COMPARISON OF TIME AND MEMORY COMPLEXITY

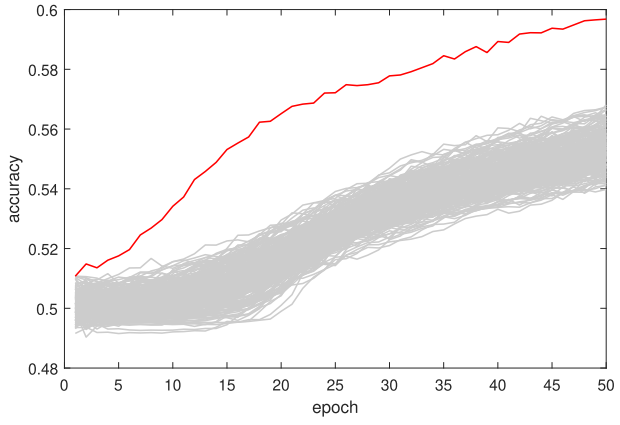
In Section IV-A, IV-B, and IV-C, we validate the performance of our proposed methods on the measurements in different situations, and our proposed methods can successfully recover

TABLE 1. Dimension and activation function of the neural networks with trace length *len*. (PL: ParalleL, SL: Shared Layer).

MLP_{exp}	Timon's work		This work(PL)		This work(SL)	
[13]	Dim.	Activation	Dim.	Activation	Dim.	Activation
Input layer	len	None	len	None	len	None
1^{st} hidden layer	20	ReLU	$5,120$	ReLU[1]	20	ReLU
2^{nd} hidden layer	10	ReLU	$2,560$	ReLU[1]	10	ReLU
Output layer	2	Softmax	256	Sigmoid[1]	256	Sigmoid



(a) PL



(b) SL

FIGURE 6. Comparing DDLA results against ASCAD.

the right key in all scenarios. In this subsection, we verify that the proposed method can improve the speed of the attack. In our experiments, we recovered the AES subkey. In other words, 8-bit key guessing is performed; the number of key guesses is 256.

Table 1 is a summary of the architecture used in our experiments. The conventional DDLA attacks were all performed with the same architecture, MLP_{exp}. Furthermore, we carried out all of the proposed methods with MLP_{exp} as the base architecture. Table 2 is a comparison of the time taken and the memory used for each method. All experiments were performed with the TensorFlow (Version 1.14.0) [29] and Keras (Version 2.2.4-tf) [30] library from Python on a personal computer with a single NVIDIA GeForce GTX 1080Ti GPU, a single Intel(R) Core(TM) i7-8700K CPU and 48 GB of RAM.

TABLE 2. Comparison of time and memory complexity for 1 byte subkey attacks. (MLP_{exp} with batch size 2000) (PL: ParalleL, SL: Shared Layer).

Method	Traces	Samples	Time	Memory[2]
Attack on ChipWhisperer-Lite				
DDLA [13]	10,000	800	704.8 s	1342.9
PL	10,000	800	318.0 s	5763.7
SL	10,000	800	5.0 s	1282.9
Attack on De-synchronized Traces				
DDLA [13]	10,000	780	7069.1 s	1969.5
PL	10,000	780	3983.3 s	6287.7
SL	10,000	780	31.9 s	1925.4
Attack on ASCAD				
DDLA[3] [13]	20,000	700	1950.9 s	1412.7
PL	20,000	700	693.8 s	5891.4
SL	20,000	700	14.5 s	1353.1

Table 2 summarizes the execution time and memory usage of the previous DDLA method and the proposed methods. For a fair comparison, 50 epochs were set to be performed equally by all the methods without applying early stopping techniques. Therefore, memory usage was compared using the results from measuring memory usage when each technique was performed for 50 epochs. As expected, all the proposed methods were faster than the conventional DDLA attack. Notably, the results of the shared layer method are significantly faster than those of the other attack methods. It is up to 140 times faster than the previous method on ChipWhisperer-Lite and 134 times faster on the ASCAD database. Memory usage is the largest in the parallel technique, and the other two techniques are smaller and are similar to each other. The results of the experiments show that the performance of the attack speed is faster when the proposed

[1]Locally-connected layer.

[2]MiB; Mebibyte.

[3]In [13], Timon recorded the performance of the DDLA attack on a computer with a single NVIDIA GeForce GTX 1080Ti GPU, two Intel Xeon E5-2620 v4 CPUs, and 64 GB of RAM. The attack against the ASCAD database consumed 852 s.

methods are used, and the method with shared layers is best in terms of speed and memory usage.

V. CONCLUSION

In this paper, we proposed three methods to efficiently perform a non-profiled side-channel attack based on deep learning. First, we modified DDLA to allow attackers to observe training metrics for each key guess at each epoch so that the attackers can monitor the model and apply early stopping techniques if appropriate. Second, we optimized the design of neural network architectures to work in parallel according to each key guess, so that attackers can perform all training at once and reduce the attack time. Finally, we reduced the number of training parameters by integrating parts of the parallel networks through shared layers, thereby reducing not only memory usage but also time complexity. Using the proposed method that reduces the time and memory costs, the attacker is able to apply DDLA with more epochs and more hyperparameters, which will, in turn, improve the success rate of deep learning attacks.

Timon's research showed that a non-profiled context in side-channel analysis is not the same as an unsupervised learning context in machine learning, proving that deep learning-based side-channel attacks can be applied in non-profiled attack scenarios. However, learning without correct labels has led to many problems. The first problem is that it is hard to search for network hyperparameters. It is challenging to distinguish whether wrong hyperparameters are selected or there is a lack of training, which makes it difficult to determine the cause of failures. In addition, when the attacker selects wrong hyperparameters, the wrong key may have a higher accuracy than the correct key due to overfitting, and the attack fails. The second problem is that there is no clear criterion for choosing a key in the DDLA. We recovered the guessed key with the highest accuracy value into the right key. Owing to the first problem, it cannot be claimed that the training result with the right key is always better than the other results. For an attacker who does not know the correct answer, it is difficult to select the correct key when the training metrics of several keys are similar. Further research will be undertaken to address these problems.

APPENDIX
OVERFITTING IN DIFFERENTIAL DEEP LEARNING ANALYSIS

In profiled deep learning analysis, overfitting is one of the main reasons for failure. It is also no exception in non-profiled attacks. Validation sets, which are separate data from training sets, were used to prevent overfitting in profiling attacks. Because attackers cannot know the real key for training data, they predict labels with guessed keys for training neural networks. So all collected measurements are used as training data without validation data. At first glance, it seems that it is unnecessary to separate a part of the training datasets into validation datasets because the attackers do not aim to increase the accuracy of the model, but are focused on

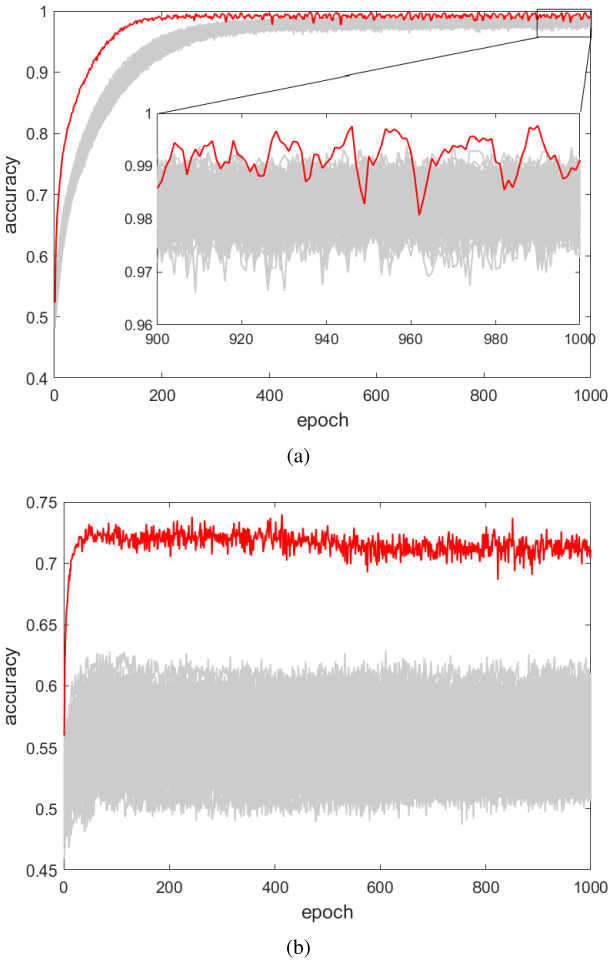


FIGURE 7. DDLA results for the (a) training set and (b) validation set.

distinguishing one of the models. However, the problem of overfitting can occur in DDLA, and we will display the experimental results. We also show that high accuracy on incorrect key-labeled data due to overfitting can be a major cause of attack failure in DDLA. As shown in Figure 7, the right key with the red line was well distinguished at the beginning of learning, but as the learning progressed, it was difficult to distinguish it from the wrong keys because of overfitting.

As in the case we presented, there are cases in which overfitting results in high accuracy not only for the right key, but also for the wrong keys. In this case, attackers who did not monitor intermediate processes were confused about the key to choose. In this paper, we recommend using a part of a training data as a validation set, like performing profiled attacks. If attackers check whether the features extracted from the training datasets also show meaningful results on the validation data, the attackers are able to distinguish the right key even if overfitting occurs on the training data, as shown in Figure 7. K-fold cross-validation is a powerful method for preventing overfitting [31]. Furthermore, applying early stopping techniques is useful to prevent the problem. These methods are not sufficient to solve the overfitting problem, but they can help mitigate it.

REFERENCES

[1] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "ECDSA key extraction from mobile devices via nonintrusive physical side channels," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1626–1638.

[2] T. W. Kim, T. H. Kim, and S. Hong, "Breaking Korea tansit card with side-channel analysis attack-unauthorized recharging," in *Proc. 18th Annu. BlackHat Asia Conf.*, 2017, pp. 1–13.

[3] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2002, pp. 13–28. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-36400-5_3

[4] W. Schindler, K. Lemke, and C. Paar, "A stochastic model for differential side channel cryptanalysis," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2005, pp. 30–46. [Online]. Available: https://link.springer.com/chapter/10.1007/11545262_3

[5] C. Paul Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. 19th Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA: Springer, Aug. 1999, pp. 388–397.

[6] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Proc. 6th Int. Workshop Cryptograph. Hardw. Embedded Syst.*, Cambridge, MA, USA: Springer, Aug. 2004, pp. 16–29.

[7] S. Yang, Y. Zhou, J. Liu, and D. Chen, "Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations," in *Proc. Int. Conf. Inf. Secur. Cryptol. (ICISC)*. Berlin, Germany: Springer, 2011, pp. 169–185. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-31912-9_12

[8] Z. Martinasek and V. Zeman, "Innovative method of the power analysis," *Radioengineering*, vol. 22, no. 2, pp. 586–594, 2013.

[9] Z. Martinasek, J. Hajny, and L. Malina, "Optimization of power analysis using neural network," in *Smart Card Research and Advanced Applications*, A. Francillon and P. Rohatgi, Eds. Cham, Switzerland: Springer, 2014, pp. 94–107.

[10] R. Gilmore, N. Hanley, and M. O'Neill, "Neural network based attack on a masked implementation of AES," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 106–111.

[11] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Proc. Int. Conf. Secur., Privacy, Appl. Cryptogr. Eng.* Cham, Switzerland: Springer, 2016, pp. 3–26. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-49445-6_1

[12] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.* Cham, Switzerland: Springer, 2017, pp. 45–68. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-66787-4_3

[13] B. Timon, "Non-profiled deep learning-based side-channel attacks with sensitivity analysis," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 2, pp. 107–131, Feb. 2019.

[14] L. Prechelt, *Early Stopping—But When*. Berlin, Germany: Springer, 2012, pp. 53–67.

[15] G. Perin, B. Ege, and J. V. Woudenberg, "Lowering the bar: Deep learning for side channel analysis," BlackHat USA, Las Vegas, NV, USA, Tech. Rep., 2018. [Online]. Available: https://www.blackhat.com/us-18/briefings/schedule/#lowering-the-bar-deep-learning-for-side-channel-analysis-11524

[16] E. Cagli, "Feature extraction for side-channel attacks," Ph.D. dissertation, Sorbonne Univ., Paris, France, 2018. [Online]. Available: https://hal.inria.fr/tel-02494260/

[17] I. Goodfellow, Y. Bengio, and A. Courville," *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[18] Z. Martinasek, L. Malina, and K. Trasy, *Profiling Power Analysis Attack Based on Multi-layer Perceptron Network*. Cham, Switzerland: Springer, 2015, pp. 317–339.

[19] Z. Martinasek, O. Zapletal, K. Vrba, and K. Trasy, "Power analysis attack based on the MLP in DPA contest v4," in *Proc. 38th Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2015, pp. 154–158.

[20] Z. Martinasek, O. Zapletal, K. Vrba, and K. Trasy, "Profiling power analysis attack based on MLP in DPA contest V4. 2," in *Proc. 38th Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2015, pp. 223–226.

[21] S. Picek, I. P. Samiotis, J. Kim, A. Heuser, S. Bhasin, and A. Legay, "On the performance of convolutional neural networks for side-channel analysis," in *Proc. Int. Conf. Secur., Privacy, Appl. Cryptogr. Eng. (SPACE)*. Cham, Switzerland: Springer, 2018, pp. 157–176.

[22] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ASCAD database," *J. Cryptograph. Eng.*, vol. 10, no. 2, pp. 163–188, 2019.

[23] L. Weissbart, S. Picek, and L. Batina, "One trace is all it takes: Machine learning-based side-channel attack on EdDDSA," in *Proc. Int. Conf. Secur., Privacy, Appl. Cryptogr. Eng.* Cham, Switzerland: Springer, 2019, pp. 86–105. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-030-35869-3_8

[24] M. Carbone, V. Conin, M.-A. Cornélie, F. Dassance, G. Dufresne, C. Dumas, E. Prouff, and A. Venelli, "Deep learning to evaluate secure RSA implementations," *Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 2, pp. 132–161, Feb. 2019.

[25] P. Robyns, P. Quax, and W. Lamotte, "Improving CEMA using correlation optimization," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 1, pp. 1–24, Nov. 2018.

[26] A.-T. Hoang, N. Hanley, and M. O'Neill, "Plaintext: A missing feature for enhancing the power of deep learning in side-channel analysis: Breaking multiple layers of side-channel countermeasures," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 4, pp. 49–85, Aug. 2020.

[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, vol. 37, Jul. 2015, pp. 448–456.

[28] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *Proc. Int. Workshop Constructive Side-Channel Anal. Secure Design*. Springer, 2014, pp. 243–260.

[29] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: tensorflow.org

[30] F. Chollet *et al.* (2015). *Keras*. [Online]. Available: https://keras.io and https://keras.io/getting_started/faq/#how-should-i-cite-keras

[31] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009. [Online]. Available: https://link.springer.com/book/10.1007/978-0-387-84858-7

DONGGEUN KWON received the B.S. degree in mathematics and the M.E. degree in information security from Korea University, South Korea, in 2014 and 2018, respectively, where he is currently pursuing the Ph.D. degree in information security with the Graduate School of Cyber Security. His research interests include cryptography, side-channel attacks, and machine learning-based cryptanalysis.

SEOKHIE HONG (Member, IEEE) received the M.S. and Ph.D. degrees in mathematics from Korea University, in 1997 and 2001, respectively. From 2000 to 2004, he was with SECURITY Technologies Inc. From 2004 to 2005, he was a Postdoctoral Researcher with COSIC, KU Leuven, Belgium. He joined the Graduate School of Cyber Security, Korea University. His research interests include cryptography, public and symmetric cryptosystems, hash functions, and MACs.

HEESEOK KIM received the B.S. degree in mathematics from Yonsei University, Seoul, South Korea, in 2006, and the M.S. and Ph.D. degrees in engineering and information security from Korea University, Seoul, in 2008 and 2011, respectively. He was a Postdoctoral Researcher with the University of Bristol, U.K., from 2011 to 2012. From 2013 to 2016, he was a Senior Researcher with the Korea Institute of Science and Technology Information (KISTI). Since 2016, he has been with Korea University. His research interests include side-channel attacks, cryptography, and network security.