Deep Learning-Based Side-Channel Attacks against Software-Implemented RSA using Binary Exponentiation with Dummy Multiplication

Seiya Shimada[1,*,†], Kunihiro Kuroda[1,†], Yuta Fukuda[1,†], Kota Yoshida[2,‡] and Takeshi Fujino[2,‡]

[1]*Graduate School of Science and Engineering, Ritsumeikan University 1–1–1 Nojihigashi, Kusatsu, Shiga, 525–8577 Japan*

[2]*Department of Science and Engineering, Ritsumeikan University 1–1–1 Nojihigashi, Kusatsu, Shiga, 525–8577 Japan*

Abstract

Several studies have recently reported on deep learning-based side-channel attacks (DL-SCA) against symmetric cryptography such as AES. However, there are relatively few studies on DL-SCA against public-key cryptography such as RSA, and threat assessment for such attacks is insufficient. The dummy multiplication countermeasure is a simple side-channel attack countermeasure for software-implemented RSA that adopts binary exponentiation. The countermeasure expects that typical side-channel attacks (e.g., timing attacks and simple power analysis) can-not distinguish the differences in operations depending on the exponential bit. In this paper, we report DL-SCA against software-implemented RSA with a simple dummy multiplication countermeasure. A deep neural network-based classifier distinguishes between true and dummy multiplication with high accuracy, and an adversary reveals secret keys.

Keywords

Side-Channel Attack, Deep Learning, RSA, Square-and-Multiply-Always

1. Introduction

Side-channel attacks (SCA) have been reported to reveal secret keys by analyzing side-channel information contained in data regarding power consumption and electromagnetic emission on cryptographic circuits. In the conventional SCA, an attacker designs leakage models to estimate secret keys in accordance with the architecture of the circuits.

Deep learning-based side-channel attacks (DL-SCA), applying deep lerning to SCAs, are proposed [1]. Several studies have recently reported on DL-SCA. In contrast to conventional SCA, DL-SCA learns models expressing the relationship between leakage and internal values through deep learning techniques. The introduction of DL-SCA is expected to reduce the expert knowledge (e.g., leakage modeling and details on circuit architectures) required for attackers. In addition, several studies have reported that DL-SCA can attack implementations of side-channel

The 4th *International Symposium on Advanced Technologies and Applications in the Internet of Things (ATAIT 2022), August 24–26, 2022, Ibaraki, Japan*

ri0091hh@ed.ritsumei.ac.jp (S. SHIMADA); ri0080rr@ed.ritsumei.ac.jp (K. KURODA); ri0073pi@ed.ritsumei.ac.jp (Y. FUKUDA); y0sh1d4@fc.ritsumei.ac.jp (K. YOSHIDA); fujino@se.ritsumei.ac.jp (T. FUJINO)

attack countermeasures. Cagli et al. reported profiling DL-SCA against software-implemented AES with random jitter countermeasures. They claim that the DL-SCA can skip the alignment process required for conventional SCA [2]. DL-SCA has been evaluated mainly on symmetric cryptography such as AES, while there are few studies on public-key cryptography such as RSA.

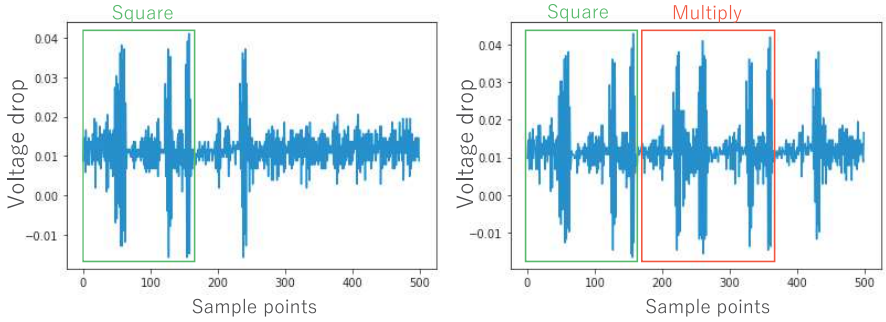
There are several studies on DL-SCA against RSA [3, 4, 5, 6]. Carbone et al. [3] evaluated DL-SCA against software-implemented RSA with countermeasures, that is the modulus randomization, input randomization, exponent randomization, and square-and-multiply-always algorithms. Lei et al. [4] focused on a decryption process for RSA with the left to the right binary method implemented on the Java Card. They trained a deep neural network (DNN) to classify power consumption waveforms into square or multiply operations. Saito et al. [5] trained a DNN to classify load and dummy load operations from power consumption waveforms that were acquired from CRT-RSA implemented in a 32-bit microcontroller. This implementation used the fixed window (FW) method for fast modular exponentiation and constant time operation. Barenghi et al. [6] reported DL-SCA against the decryption process for software-implemented RSA on 32-bit microcontrollers. This implementation used the square-and-multiply-always algorithm countermeasure. They applied a collision technique in which the correlation of consecutive instruction operands is used to identify dummy multiply operations.

In this paper, we evaluate DL-SCA against the decryption process for software-implemented RSA that uses the left to the right binary method for modular exponentiation and square-and-multiply-always algorithm for a side-channel attack countermeasure. An attacker collects power consumption waveforms from profiling devices and trains DNNs for classifying the waveforms into normal multiply operations and dummy multiply operations. We discuss which point of the waveforms the DNN focuses on when it makes a decision by using a simple 8-bit microcontroller as a fundamental study. For the same reason, side-channel attack countermeasures and faster arithmetic methods other than square-and-multiply-always are not used in our implementation. Our experimental results show that DL-SCA successfully classifies normal and dummy operations, and an attacker can then estimate all the bits of the secret key. Conventional SCA methods, such as simple timing attacks [7] and simple power analysis [8], have difficulty identifying dummy operations. We also analyze a trained DNN model by using gradient visualization, and we discuss where the DNN finds leakage points from our implementation with the countermeasure.

2. Conventional side-channel attacks against RSA with square-and-multiply-always algorithm

In public-key cryptography such as RSA, the binary exponentiation method is often used because it reduces the operation time. Binary representation of the exponent bit is performed bit by bit from the most to the least significant bit using the left to right binary method. A square operation is performed when the exponent bit is “0,” and square and multiply operations are performed when the exponent bit is “1.”

The timing attack [7] and simple power analysis (SPA) [8] are reported as conventional SCA methods against RSA. They mainly exploit the differences in operation between exponent bit



(a) Waveform of "0" operation (b) Waveform of "1" operation

Figure 1: Part of acquired power waveform from software-implemented RSA without countermeasures

values. Figure 1 shows the power consumption waveform from software-implemented RSA without countermeasures when the exponent bits are "0" and "1." As shown in the figure, the attacker easily identifies the presence or absence of the multiply operation from the characteristic differences in the waveform and can estimate whether the exponent bit is "0" or "1."

The square-and-multiply-always algorithm is a simple countermeasure against these conventional SCAs. The algorithm performs a dummy multiply operation when the exponent bit is "0." That is, square and multiply operations are performed on every bit whether the exponent bit is "0" or "1." This makes it difficult to distinguish between them in terms of operation time or power waveform. The dummy multiply operation stores the result into a dummy register so as not to contaminate the correct results. Algorithm 1 shows the square-and-multiply-always algorithm. The A and D are registers that store correct and dummy results, respectively, and "$\leftarrow$" represents a store operation. The $A \cdot A$ in line 3 represents a square operation. The d_i is the i-th bit of the secret key d. The result is assigned to either register A or register D depending on the bit. Note that the dummy register D does not affect the result of RSA.

Algorithm 1 Square-and-Multiply-Always

Input: $m, d = (d_{l-1}, \cdots, d_1, d_0), N$
Output: $m^d \bmod N$

1: $A \leftarrow 1$
2: **for** i=l-1 to 1 **do**
3: $\quad A \leftarrow (A \cdot A) \bmod N$
4: $\quad$ **if** $d_i = 1$ **then**
5: $\quad\quad A \leftarrow (A \cdot m) \bmod N$
6: $\quad$ **else**
7: $\quad\quad D \leftarrow (A \cdot m) \bmod N$
8: $\quad$ **end if**
9: **end for**
10: **return** A

3. Identification of dummy multiply operations by waveform classification using deep learning techniques

We apply DL-SCA to RSA in order to break the square-and-multiply-always countermeasure. Our DL-SCA is for a profiling attack scenario. We expect an attacker to have a profiling device that can be freely accessed by them. A DL-SCA in the profiling scenario consists of a profiling phase and an attack phase.

First is the profiling phase. The attacker performs the RSA decryption process on the profiling device with a known secret key and collects power consumption waveforms. Then, the attacker divides the waveforms into square and (dummy) multiply operation pairs (i.e., operations per exponent bit) and labels them with "0" and "1" in accordance with the secret key. The start time of each operation is specified by the template matching algorithm with the sum of absolute difference (SAD). The power consumption waveform from the square operation is used for the template. SAD calculates the similarity between the target waveform (T) and the template waveform ($T^{(r)}$) as follows:

$$SAD_j = \sum_{i=0}^{I-1} |T_{j+i} - T_i^{(r)}|, \tag{1}$$

where i and j represent sample points. Sample points where the SAD score becomes lower than the threshold level are expected to indicate the start time of the square operation. The waveform is cut out so that it includes the start of the square operation to (dummy) multiply operation as points of interest (POI). The attacker trains a binary classifier using a deep neural network (DNN) for distinguishing the square and multiply operation and square and dummy multiply operation by inputting the POIs of the power consumption waveforms.

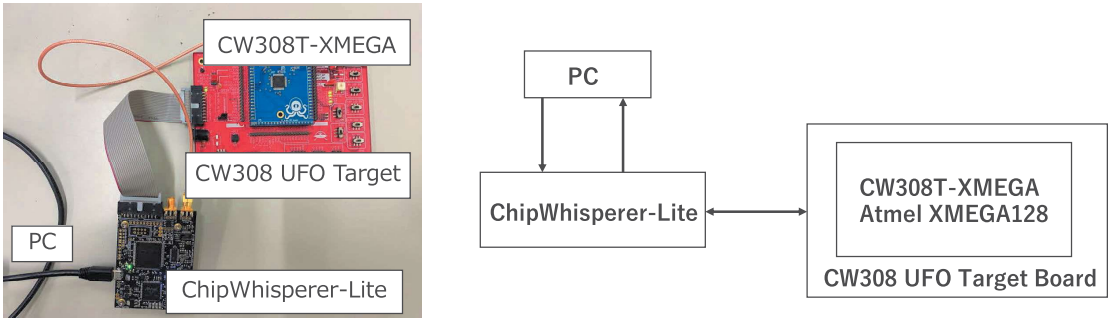
The second is the attack phase. The attacker acquires power consumption waveforms from the RSA decryption process that is performed on the target device with an unknown secret key. The attacker divides the acquired waveforms into POIs with the same procedures as the profiling phase. The POIs are input to the trained DNN, and it classifies the POIs into square and multiply or square and dummy multiply operations. The attacker identifies whether the exponent bit is "0" or "1" for the operation from the classification result, and the attacker recovers the secret key from a series of the results.

4. Experiments and results

4.1. Experimental setup

ChipWhisperer[1], developed by New AE Technology for evaluating embedded device security, was used in our experiment. Figure 2 shows a photograph of the experimental environment and a block diagram. Software-implemented RSA with the square-and-multiply-always countermeasure was executed on a CW308T-XMEGA (operating frequency: 7.38 MHz) mounted on a CW308-UFO-Target board. The RSA decryption process was written in C language and compiled by AVR-GCC version 5.4.0. The secret key length for RSA was set to 1,024 bits. We acquired

[1]https://www.newae.com/chipwhisperer



(a) Photograph (b) Block diagram

Figure 2: Experimental environment for acquiring power waveforms

Table 1
Structure of DNN for DL-SCA

Input layer	Convolution layer × 6	Fully connected layer × 2	Output layer
500	Size × filters = 3 × 10 Stride = 3, ReLU	32 neurons Tanh	2 neurons Softmax

power consumption waveforms during the RSA decryption process by using ChipWhisperer-Lite with a 29-MS/s sampling rate.

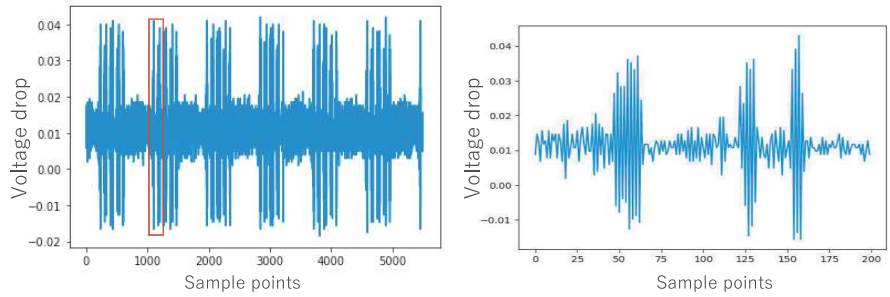
The network structure of the DNN is shown in Table 1. The number of nodes on the input layer was 500, which is the same number of sample points on POIs (divided waveform).

4.2. Profiling phase

We acquired 500 waveforms for each different known secret key. The waveforms of the square and multiply operations corresponding to each of the 1,023 exponent bits of the secret key, excluding the most significant bits (MSB), were cut from each waveform and labeled.

We divided the waveforms into 1,023 POIs corresponding to the exponent bits, that is the secret key without MSB, and we labeled them. The POI during the operation corresponding to the MSB of the secret key was significantly different from the others so it was excluded from the training data. Figure 3 shows the average power consumption waveform of the acquired waveforms and the template for identifying the start time of the square operation. The number of sample points of the template was 200, and the template covered from the start of the square operation to the end, as shown in the figure. Figure 4 shows an SAD score when SAD template matching was performed on one of the power consumption waveforms. The SAD threshold level was set at 0.5, and the point when the SAD value becomes lower than the level was used for the start time of the square operation. POIs consisted of 500 points from the start of the square operation specified by SAD to the end of the multiply operation.

As a result of these operations, 511,500 POIs were prepared as DNN training datasets. Figure 5 shows the POIs from the waveforms when the labels “0” (square and dummy multiply operation) and “1” (square and multiply operation) from the data set. We set the batch size to 1,000, the



(a) Part of obtained power consumption waveform (b) Template waveform

Figure 3: (a) Part of mean power consumption waveform acquired from decryption process of software-implemented RSA with square-and-multiply-always countermeasure. (b) Template for SAD. It corresponds to part of (a) covered by red square.

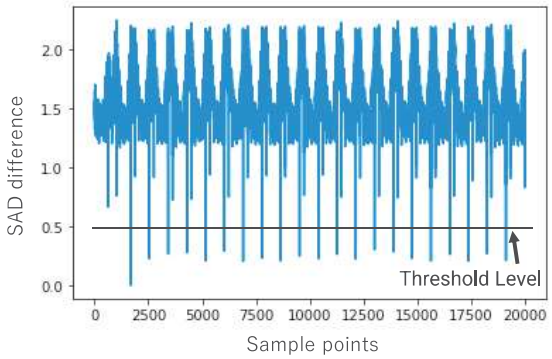


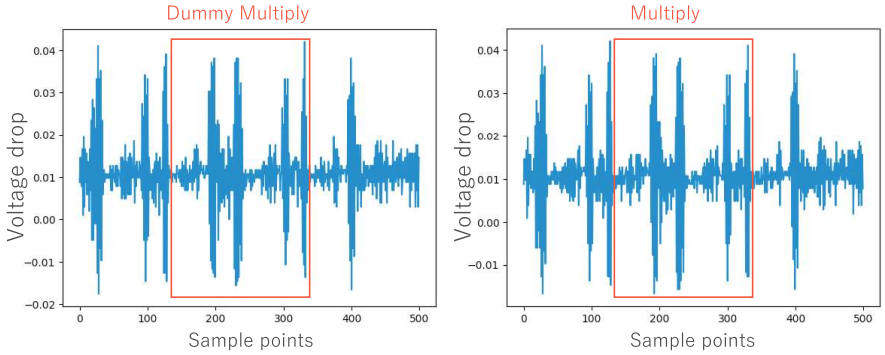
Figure 4: SAD score when SAD template matching was performed on one of power consumption waveforms. Threshold is set to 0.5.

number of training epochs to 100, and the loss function to cross-entropy loss, and we used the Adam optimizer to train the DNN with a 0.001 learning rate. The test accuracy achieved 99.51% after the training process.

4.3. Attack phase

We acquired 50 waveforms with different secret keys. We extracted POIs from the waveforms by using SAD template matching with the same template as the profiling phase. The trained DNN classified the POIs into "0" and "1," which corresponded to the exponent bit's value.

We confirmed that the DNN correctly classified all of the POIs (100% accuracy for 50 different keys); that is, the DNN could identify the dummy and normal multiply operations. We mentioned that the waveform of the operation corresponding to the MSB of the secret key was significantly different but noted that the RSA decryption process always started with exponent bit "1." An attacker could easily identify the exponent bit of the first operation as "1."



(a) Waveform of "0" operation (b) Waveform of "1" operation

Figure 5: Enlarged waveform of (a) dummy multiply and (b) multiply operation.

4.4. Discussion

For DL-SCAs, gradient visualization (GV) has been used for analyzing SCA leakage points learned by a model [9]. We use gradient visualization (GV) to reveal how the trained DNN identifies the dummy multiply operation (i.e., the waveform positions from which the information on exponent bits leaks). The GV is represented by Eq. 2.

$$\nabla T_i = \frac{\partial L_T}{\partial T_i}, \tag{2}$$

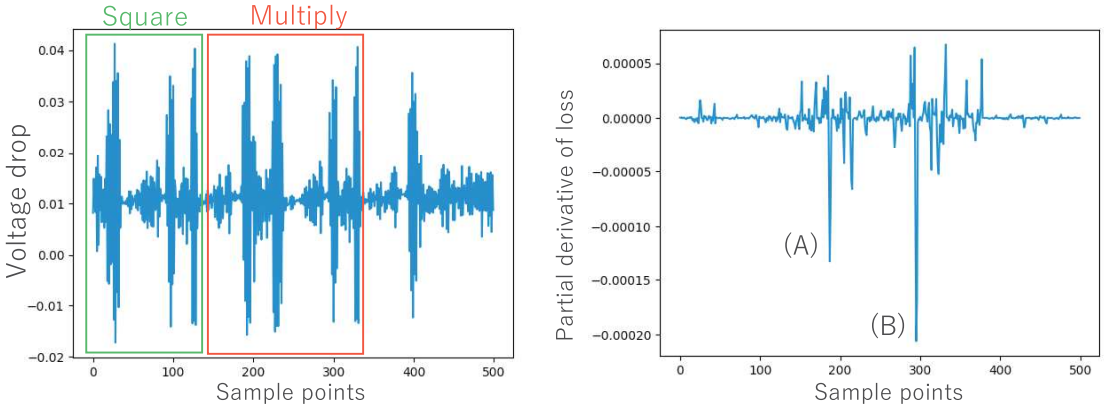
where ∇T_i is the partial differentiation of the loss at the i-th sample point of the input waveform T, and it indicates the contribution of the sample point to the DNN inference result. L_T is the loss between the DNN's inference and the correct label when the waveform T is input.

Figure 6 (a) shows a power consumption waveform around the operation corresponding to the exponent bit of "1." Figure 6 (b) shows the gradient ∇T at each sample point based on the correct label when waveform (a) was input to the DNN model. We found two large peaks of ∇T. We denote peak position (A) around time 200 and the peak position (B) around time 300. The two large peaks of ∇T appeared during the multiply operation. The DNN was assumed to have used these two points for identifying the dummy multiply and true multiply operations from the waveform.

We found that peak position (A) was at the timing of the instruction "in" in additional experiments in which NOP instructions were inserted into the compiled assembly code. Table 2 shows the assembly code of the multiply operation executed in the normal and the dummy cases, near peak position (A). Table 3 shows the details on the instruction sets used above. Table 2 indicates that there was a difference between the normal and the dummy multiply operations where the "adiw" instruction was executed only for dummy multiply as shown in row 4.

Figure 7 (a) shows overlapped and enlarged waveforms around peak (A) corresponding to "0" and "1". Figure 7 (b) shows the gradient of the range corresponding to (a).

According to the gradient in Fig. 7 (b), the DNN is expected to focus on the shift in the execution timing of instruction "in" caused by the appearance of "adiw." The sign of the gradient at peak point (A) is + for the exponent bit of "0" and − for "1." That is, the DNN focused on the



(a) Power consumption waveform around operation corresponding to exponent bit "1"

(b) Gradient when waveform is input to DNN

Figure 6: Power consumption waveform and the result of Gradient visualization.

Table 2
Comparison of instructions between normal and dummy multiply operation

"0" operation (dummy multiply)	"1" operation (multiply)	peak of ∇T
subi	subi	
sbci	sbci	
movw	movw	
adiw		
call	call	
push ×18	push ×18	
in ×2	in ×2	(A)
sbiw	sbiw	
out ×2	out ×2	
movw ×3	movw ×3	

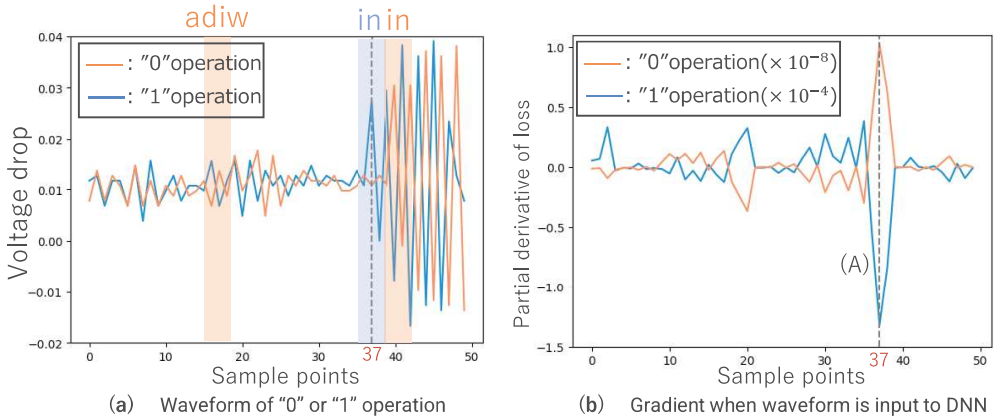
amount of power consumption on the 37-th sample point and inferred "0" when it was lower and "1" when it was higher.

Peak (B) was examined with the same process as that used for peak (A) mentioned above, and it was found that the same leakage as peak (A) occurred. The position of the peak was at the timing of the instruction "in." The instruction "adiw" the peak (A) caused a delay of the following instruction. As a result, the normal and dummy operations were distinguished by the difference in the peak position for the "in" command.

The reason for the appearance of "adiw" is to specify the pointer to the dummy register that stores the calculation. Correct and dummy results are stored in different registers. The instruction "adiw" specifies the pointer to the register that the dummy result is stored.

Table 3
AVR Instruction Set

Instruction	Description	Cycles
movw	Copy Register Word	1
subi	Subtract Immediate	1
sbci	Subtract Immediate with Carry SBI	1
call	Long Call to a Subroutine	3
push	Push Register on Stack	1
in	Load an I/O Location to Register	1
out	Store Register to I/O Location	1
sbiw	Subtract Immediate from Word	2
adiw	Add Immediate to Word	2
nop	No Operation	1



(a) Waveform of "0" or "1" operation (b) Gradient when waveform is input to DNN

Figure 7: (a) Overlapped waveforms around peak (A) corresponding to "0" and "1." (b) Gradient of range corresponding to (a).

5. Conclusion

In this paper, we reported about DL-SCA against RSA. We performed DL-SCA against software-implemented RSA with the square-and-multiply-always algorithm running on a CW308T-XMEGA. In this algorithm, a dummy multiply operation is performed on the operation when the processing bit of the secret key is "0." Thus, it is difficult to distinguish between operations when the exponent bit is "0" and when it is "1."

Our experimental results showed that a DNN can identify a secret key accurately by learning the power consumption waveforms corresponding to the operation of exponent bits "0" and "1." We discussed how the DNN classifies these waveforms by using gradient visualization. We found that the execution timing of the instruction "in" was shifted by the instruction "adiw," which is only present for the dummy multiply operation. This result indicates that the DNN extracted small differences in the assembly instructions between the normal and dummy multiply operations from the power consumption waveforms.

In future work, we will implement the square-and-multiply-always algorithm so that the

number of instructions is identical between the normal and dummy multiply operations and evaluate whether the DNN can classify them. In this paper, we report DL-SCA against RSA using the left to right binary method. There are RSA implementations using the right to left binary method. We will evaluate DL-SCA against RSA using the right to left binary method.

References

[1] H. Maghrebi, T. Portigliatti, E. Prouff, Breaking cryptographic implementations using deep learning techniques, in: C. Carlet, M. A. Hasan, V. Saraswat (Eds.), Security, Privacy, and Applied Cryptography Engineering, Springer International Publishing, Cham, 2016, pp. 3–26.

[2] E. Cagli, C. Dumas, E. Prouff, Convolutional neural networks with data augmentation against jitter-based countermeasures, in: W. Fischer, N. Homma (Eds.), Cryptographic Hardware and Embedded Systems – CHES 2017, Springer International Publishing, Cham, 2017, pp. 45–68.

[3] M. Carbone, V. Conin, M.-A. Cornélie, F. Dassance, G. Dufresne, C. Dumas, E. Prouff, A. Venelli, Deep learning to evaluate secure rsa implementations, IACR Transactions on Cryptographic Hardware and Embedded Systems 2019, Issue 2 (2019) 132–161.

[4] Q. Lei, C. Li, K. Qiao, Z. Ma, B. Yang, Vgg-based side channel attack on rsa implementation, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 1157–1161.

[5] K. Saito, A. Ito, R. Ueno, N. Homma, A deep-learning based single-trace side-channel attack on tamper-resistant CRT-RSA software, IEICE Tech. Rep., vol. 121, no. 206, HWS2021-42, pp. 7-12,, 2021. (in Japanese).

[6] A. Barenghi, D. Carrera, S. Mella, A. Pace, G. Pelosi, R. Susella, Profiled side channel attacks against the rsa cryptosystem using neural networks, Journal of Information Security and Applications 66 (2022) 103122.

[7] P. C. Kocher, Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems, in: N. Koblitz (Ed.), Advances in Cryptology — CRYPTO '96, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 104–113.

[8] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: M. Wiener (Ed.), Advances in Cryptology — CRYPTO' 99, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 388–397.

[9] L. Masure, C. Dumas, E. Prouff, Gradient visualization for general characterization in profiling attacks, in: I. Polian, M. Stöttinger (Eds.), Constructive Side-Channel Analysis and Secure Design, Springer International Publishing, Cham, 2019, pp. 145–167.