

## Article

# Key Bit-Dependent Side-Channel Attacks on Protected Binary Scalar Multiplication <sup>†</sup>

Bo-Yeon Sim <sup>1,‡</sup>, Junki Kang <sup>2,‡</sup> and Dong-Guk Han <sup>1,\*</sup><sup>1</sup> Department of Mathematics, Kookmin University, 77 Jeongneung-ro, Seongbuk-gu, Seoul 02707, Korea; qjdusls@kookmin.ac.kr<sup>2</sup> The Affiliated Institute of ETRI, 1559 Yuseong-daero, Yuseong-gu, Daejeon 34044, Korea; kang.junki@gmail.com

\* Correspondence: christa@kookmin.ac.kr

<sup>†</sup> This paper is an extended version of our paper published in ISPEC 2017, Melbourne, Australia, 13–15 December 2017.<sup>‡</sup> These authors contributed equally to this work.

Received: 15 September 2018; Accepted: 22 October 2018; Published: 6 November 2018



**Abstract:** Binary scalar multiplication, which is the main operation of elliptic curve cryptography, is vulnerable to side-channel analysis. It is especially vulnerable to side-channel analysis using power consumption and electromagnetic emission patterns. Thus, various countermeasures have been reported. However, they focused on eliminating patterns of conditional branches, statistical characteristics according to intermediate values, or data inter-relationships. Even though secret scalar bits are directly loaded during the check phase, countermeasures for this phase have not been considered. Therefore, in this paper, we show that there is side-channel leakage associated with secret scalar bit values. We experimented with hardware and software implementations, and experiments were focused on the Montgomery–López–Dahab ladder algorithm protected by scalar randomization in hardware implementations. We show that we could extract secret key bits with a 100% success rate using a single trace. Moreover, our attack did not require sophisticated preprocessing and could defeat existing countermeasures using a single trace. We focused on the key bit identification functions of mbedTLS and OpenSSL in software implementations. The success rate was over 94%, so brute-force attacks could still be able to recover the whole secret scalar bits. We propose a countermeasure and demonstrate experimentally that it can be effectively applied.

**Keywords:** side-channel analysis; elliptic curve cryptography; single-trace attack; key bit-dependent attack; countermeasure

## 1. Introduction

The blockchain and fast identity online (FIDO), which are emerging as key technologies to lead the Fourth Industrial Revolution, authenticate users by using an elliptic-curve digital signature algorithm (ECDSA). However, scalar multiplication, which is the core operation of ECDSA, is vulnerable to side-channel analysis (SCA). SCAs were first proposed by Paul Kocher in 1996 [1]; they use the leakage consumed while cryptographic algorithms are performed on embedded systems. Various side-channel attacks against elliptic-curve cryptography (ECC) have been researched [2–16]. Among them, power analysis using power patterns consumed during algorithm operations is known as the most powerful. Electromagnetic analysis using emitted electromagnetic patterns is similar to power analysis, but there is a difference in useable side-channel information. Therefore, in this paper, we focus on power analysis.

As SCAs become more powerful, various countermeasures to resist them have been studied [17–22]. However, only countermeasures to eliminate patterns of data-dependent conditional branches, statistical characteristic according to intermediate values, or data inter-relationships have been studied. No countermeasure has been taken into account for the secure design of the key bit identification phase even though secret scalar bits are directly loaded during that phase. Since the secret scalar bit value is extracted and stored in the variable, the secret scalar can be exposed if the vulnerability is discovered.

**Our Contributions.** In this paper, we analyzed the power consumption (we also considered information leakage via electromagnetic emanation throughout this paper.) properties of the key bit identification phase and experimentally showed that attacks based on these properties can recover secret scalar bits. Our proposed attacks require only a single power consumption or electromagnetic trace. They also do not require any knowledge of in–out values; thus, they can defeat any combination of existing countermeasures. Two implementations (i.e., hardware and software) were targeted, and we could recover secret scalar bits by applying SPA-VI (SPA based on visual inspection) and a k-means clustering algorithm. Among various scalar multiplication algorithms, we focused on binary scalar multiplication algorithms. The first set of experiments is based on hardware implementation of the Montgomery–López–Dahab ladder algorithm protected by scalar randomization. Experimental results show that the secret scalar bits can be recovered with a 100% success rate using only single power consumption or electromagnetic trace. In the second set of experiments, on software implementation, we targeted algorithms composed using the key bit identification functions of mbedTLS and OpenSSL. Here, secret scalar bits could be recovered with over 94% success rate. If we attacked the power consumption trace using the leakage associated with referenced register addresses, the success rate was 100%. We propose two kinds of countermeasures, one each for hardware and software implementations. Their effectiveness is experimentally demonstrated.

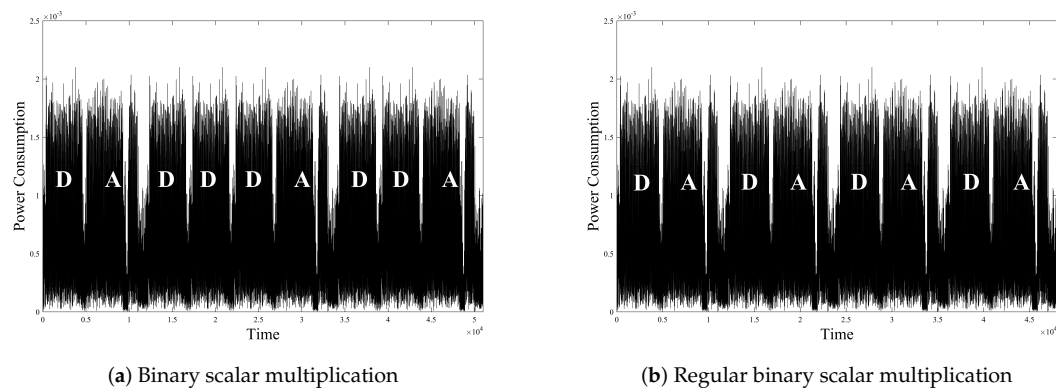
**Extension.** This paper is an extended version of our paper published in ISPEC 2017 [23]. In that paper, we showed key bit-dependent attack results using only a single power consumption trace. However, in this paper, we show new key bit-dependent attack results using a single electromagnetic trace and a low-pass filter. Thus, we show four experimental results using a power-consumption trace, a power-consumption trace passed through a low-pass filter, electromagnetic trace, and electromagnetic trace passed through a low-pass filter. Measuring electromagnetic traces is not an easy task because it very much depends on the angle and position of the probe. Moreover, in the case of hardware implementation, our latest results using electromagnetic traces have a higher success rate than previous results.

**Organization.** The rest of this paper is organized as follows. In Section 2, we describe SCAs in scalar multiplication algorithms. In Section 3, we regulate the leakage properties of the attack targets; in Section 4, we establish the attack framework. Experimental results are described in Section 5. We discuss countermeasures in Section 6, and conclusions are presented in Section 7.

## 2. Conventional SCAs on Scalar Multiplication

### 2.1. Simple-Power Analysis

Simple-power analysis (SPA) is a method of directly analyzing a secret scalar using only one trace or a few traces collected during cryptographic operations [9]. Because cryptographic algorithms have different power-consumption patterns according to the instructions of the processor, the secret scalar or instantaneous command could be analyzed from these patterns. For instance, in the case of a binary scalar multiplication algorithm that performs a point-doubling operation at all times, and performs a point addition operation only when the secret key bit value is 1, the secret key can be found if the point-doubling and point-addition operations have different power-consumption patterns. That is, as per Figure 1a, this irregular sequence of instructions according to the secret scalar bit (i.e., the data-dependent conditional branch) leads to a serious security problem.



**Figure 1.** Power-consumption trace of binary scalar multiplication. (a) Binary scalar multiplication; (b) regular binary scalar multiplication.

## 2.2. Differential Power Analysis (DPA)

DPA is a statistical analysis method that analyzes multiple power-consumption traces to find the secret scalar [9]. Typically, DPA is based on the fact that power consumption depends on data values being manipulated. To perform DPA, input or output values of cryptographic algorithms have to be known. Similarly, there is an address-bit DPA based on the fact that power consumption depends on the address value of the register that loads or stores data during the operation. Thus, even if an SPA countermeasure [19,21,22], which has a regular power-consumption sequence, as shown in Figure 1b, is applied, it is vulnerable to DPA. To cope with this, randomization techniques that eliminate association between all possible intermediate values and power consumption are generally used [17,18,20].

## 2.3. Sophisticated Power Analysis

SPA-and DPA-resistant countermeasures can be defeated by sophisticated attacks, such as a template attack (TA) [6,10,12] or collision attack (CA) [7,11,13]. A TA characterizes power-consumption traces by a multivariate normal distribution to build templates, and matches power-consumption leakage to the templates to find a secret scalar value. A CA is a kind of higher-order DPA and is an attack based on the inter-relationships among intermediate data (i.e., collisions of two intermediate values). So far, no theoretically perfect countermeasures against TAs and CAs have been presented. However, there is a disadvantage, in that they require precise preprocessing, such as decapsulation, localization, and a multiprobe to obtain a power-consumption trace having a high signal-to-noise ratio [6,7,11,13]. Decapsulation in particular requires to physically modify the target devices, and numerous traces are required to build templates.

To thwart previous attacks, various countermeasures to eliminate patterns of data-dependent conditional branches, statistical characteristic according to intermediate values, or data inter-relationships have been studied. However, no countermeasure has been taken into account for the secure design of the key bit identification phase, although secret scalar bits are directly loaded during that phase. Since the secret scalar bit value is extracted and stored in the variable, the secret scalar can be exposed if the vulnerability is discovered. Thus, in this paper, we verify that this vulnerability is sufficient to find a secret scalar.

## 3. Materials

### 3.1. Key Bit Identification Phase

Elliptic-curve scalar multiplication is a method for computing  $dP$ , where  $d$  is a secret scalar and  $P$  is a point on an elliptic curve. It is an elementary operation of ECC, so it has been used in numerous

PKCs. It basically consists of iterative operations determined according to the  $i$ -th bit  $d_i$  value of the secret scalar  $d$ , where  $d$  is a  $\lambda$ -bit scalar, so  $d = (d_{\lambda-1}, d_{\lambda-2}, \dots, d_1, d_0)_2$  and  $0 \leq i < \lambda$  [19,21,22,24]. For instance, in the algorithms shown in Figure 2, while performing Steps 2 to 5, addresses of registers  $R_x$  ( $x = 0$  or  $1$ ) to be referenced are determined by the  $d_i$  value.

Thus, at the beginning of the  $i$ -th iterative operation, the  $i$ -th secret scalar bit value  $d_i$  is extracted from a  $\lambda$ -bit scalar string and stored in a variable. This phase exists in almost all elliptic-curve scalar multiplication algorithms because they are composed of iterative operations based on the value of  $d_i$ . At this phase, secret scalar bits,  $d_i$ , are extracted at the beginning of each iterative operations. We define this step as the key bit identification phase.

### 3.1.1. Key Bit-Dependent Properties

Binary scalar multiplication consists of iterative operations determined according to the  $i$ -th bit  $d_i$  value of secret scalar  $d$  (Figure 2). Therefore, there exists a key bit identification phase in which the  $i$ -th scalar bit value is extracted from a  $\lambda$ -bit scalar string  $d = (d_{\lambda-1}, d_{\lambda-2}, \dots, d_1, d_0)_2$  and stored in a  $d_i$  variable at the beginning of each  $i$ -th iteration. Thus, power consumption associated with the  $d_i$  value occurs. We can categorize these properties according to hamming distance (HD) and hamming weight (HW), mainly used as power-consumption models as follows.

Left to Right	Right to Left
<b>Input</b> : $P$ is a point on an elliptic curve, a $\lambda$ -bit scalar $d = (d_{\lambda-1}, \dots, d_0)_2$ <b>Output</b> : $Q = dP$ 1: $R_0 \leftarrow \infty, R_1 \leftarrow P$ 2: <b>for</b> $i = \lambda - 1$ down to $0$ <b>do</b> 3: $R_{1-d_i} \leftarrow R_{d_i} + R_{1-d_i}$ 4: $R_{d_i} \leftarrow 2R_{d_i}$ 5: <b>end for</b> 6: <b>Return</b> $R_0$	<b>Input</b> : $P$ is a point on an elliptic curve, a $\lambda$ -bit scalar $d = (d_{\lambda-1}, \dots, d_0)_2$ <b>Output</b> : $Q = dP$ 1: $R_0 \leftarrow \infty, R_1 \leftarrow P, R_2 \leftarrow P$ 2: <b>for</b> $i = 0$ up to $\lambda - 1$ <b>do</b> 3: $R_{1-d_i} \leftarrow R_{1-d_i} + R_2$ 4: $R_2 \leftarrow R_0 + R_1$ 5: <b>end for</b> 6: <b>Return</b> $R_0$

**Figure 2.** Examples of simple-power analysis (SPA)-resistant regular algorithms for binary scalar multiplication.

**Property 1.** In hardware implementations, power consumption in the key bit identification phase is simultaneously affected by the hamming distance between two consecutive bits  $d_{i+1}$  and  $d_i$ , i.e.,  $d_{i+1} \oplus d_i$  ( $0 \leq i < \lambda - 1$ ). Thus, if two consecutive bits are the same, i.e.,  $d_{i+1} = d_i$ , power consumption related to  $d_{i+1} \oplus d_i = 0$  occurs. Otherwise, power consumption related to  $d_{i+1} \oplus d_i = 1$  occurs.

**Property 2.** In software implementations, power consumption in the key bit identification phase is affected by the hamming weight of  $d_i$  ( $0 \leq i \leq \lambda - 1$ ). Thus, if the value of  $i$ -th secret bit is 0, i.e.,  $d_i = 0$ , then power consumption is related to 0. Otherwise, power consumption related to 1 occurs.

### 3.1.2. Key Bit-Dependent Properties of SPA-Resistant Regular Algorithms

The binary scalar multiplication algorithm (Reference [25], Algorithm 3.26 and 3.27) can be easily broken by SPA. Therefore, various SPA-resistant regular algorithms, as shown in Figure 2, have been used. In regular algorithms, the referred register addresses  $RegAddr_{d_i}$  differ depending on the  $d_i$  value, and these influence power consumption. Since hardware and software operating structures are different from each other, the effect on power consumption especially differs then.



In hardware implementations, operations are executed in parallel. Thus, at the same time as the secret scalar bits  $d_i$  are extracted at the beginning of each iterative operation, register addresses  $RegAddr_{d_i}$  to be referenced are also determined. In accordance with this characteristic, power consumption when the secret scalar bit  $d_i$  is determined is also influenced by the HD between the register addresses used in two successive loops. In software implementations, differing from hardware implementations, operations are executed sequentially. Hence, register addresses  $RegAddr_{d_i}$  to be referenced do not affect power consumption at the same time as the secret scalar value  $d_i$ . In the following, we describe additional power-consumption properties of SPA-resistant regular algorithms. Note that  $RegAddr_0$  is different from  $RegAddr_1$ .

**Property 3.** In hardware implementations, power consumption in the key bit identification phase is simultaneously affected by:

- the hamming distance between two consecutive bits  $d_{i+1}$  and  $d_i$ , i.e.,  $d_{i+1} \oplus d_i$
- the hamming distance between referred register addresses  $RegAddr_{d_{i+1}}$  and  $RegAddr_{d_i}$  determined by  $d_{i+1}$  and  $d_i$ , i.e.,  $RegAddr_{d_{i+1}} \oplus RegAddr_{d_i}$

Thus, if two consecutive bits are the same, i.e.,  $d_{i+1} = d_i$ ; power consumption related to  $d_{i+1} \oplus d_i = 0$  and  $RegAddr_{d_{i+1}} \oplus RegAddr_{d_i} = 0$  occurs at the same time. Otherwise, power consumption related to  $d_{i+1} \oplus d_i = 1$  and  $RegAddr_{d_{i+1}} \oplus RegAddr_{d_i} \neq 0$  occurs at the same time ( $0 \leq i < \lambda - 1$ ).

**Property 4.** In software implementations, power consumption is affected by:

- the hamming weight of  $i$ -th secret bit value  $d_i$
- the hamming weight of referred register address  $RegAddr_{d_i}$  determined by value of  $i$ -th secret bit  $d_i$

Thus, if the  $i$ -th secret bit value is 0, i.e.,  $d_i = 0$ , then power consumption related to 0 and  $RegAddr_0$  occurs. Otherwise, power consumption related to 1 and  $RegAddr_1$  occurs ( $0 \leq i \leq \lambda - 1$ ).

We can classify power-consumption traces into two groups,  $G_0$  and  $G_1$ , using the properties.  $G_0$  includes power-consumption traces when leakage is zero, and  $G_1$  includes traces when leakage is nonzero. Once the traces are classified into two groups, we can recover the respective bit  $d_i$ , since the most significant bit is always 1. We define a study exploiting Property 1 and 2 as *Case Study 1* (Figures 3a,c, 4a,c and 5a,c,e). Then, we define a study exploiting Property 3 and 4 as *Case Study 2* (Figures 3b,d, 4b,d and 5b,d,f).

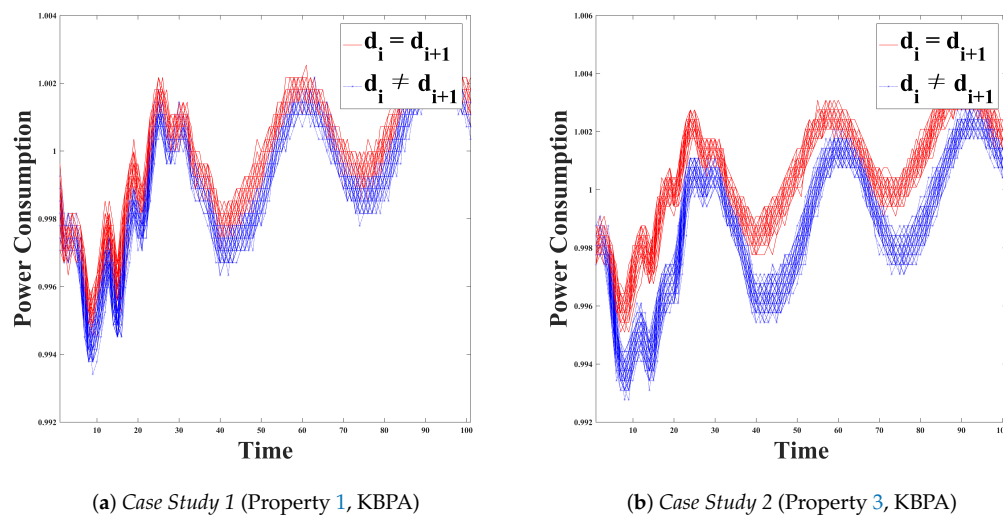
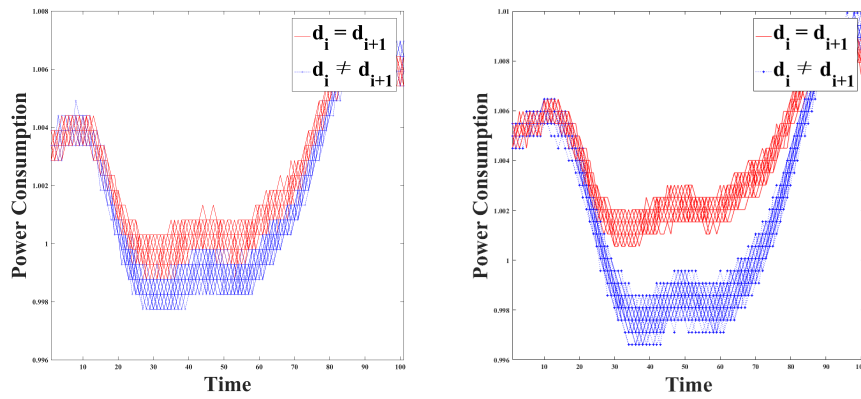
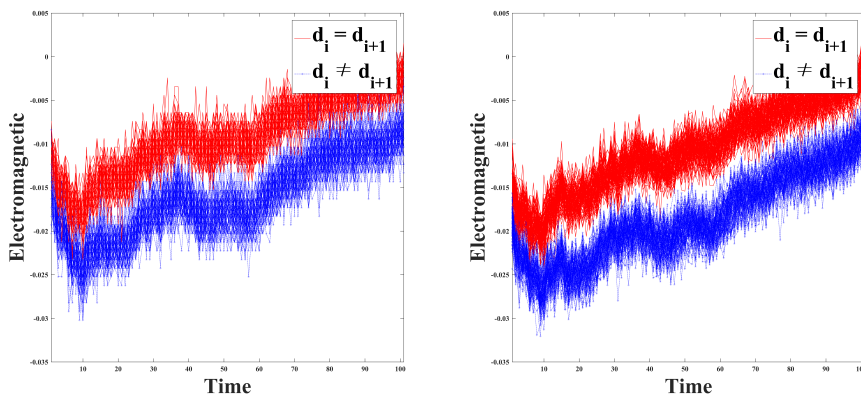


Figure 3. Cont.

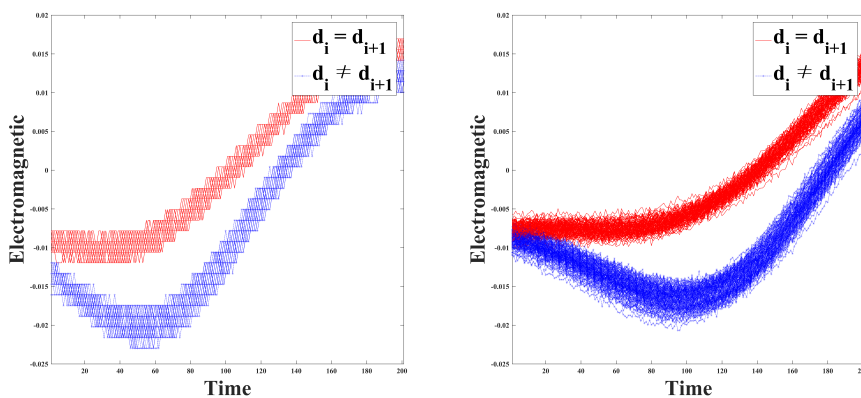


(c) Case Study 1 (Property 1, KBPA, BLP 100-75) (d) Case Study 2 (Property 3, KBPA, BLP-100-75)

**Figure 3.** Classification result of Points of Interest (PoIs) (Hardware Implementation, Power Consumption). (a) Case Study 1 (Property 1, KBPA); (b) Case Study 2 (Property 3, KBPA); (c) Case Study 1 (Property 1, KBPA, BLP 100-75); (d) Case Study 2 (Property 3, KBPA, BLP-100-75).

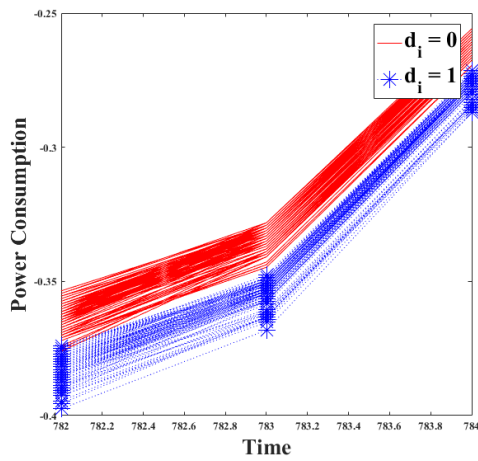


(a) Case Study 1 (Property 1, KBEA, LF-R 400) (b) Case Study 2 (Property 3, KBEA, LF-R 400)

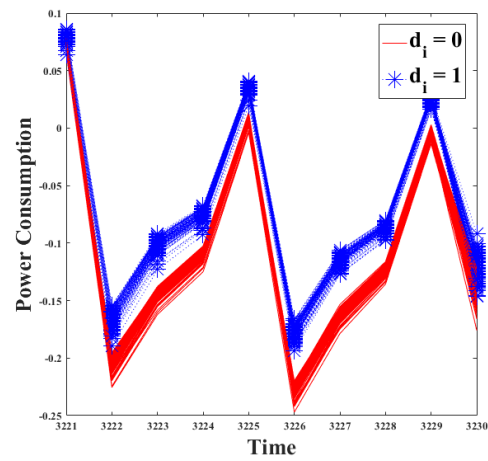


(c) Case Study 1 (Property 1, KBEA, LF-R 400, BLP 15-75+) (d) Case Study 2 (Property 3, KBEA, LF-R 400, BLP 10.7-75+)

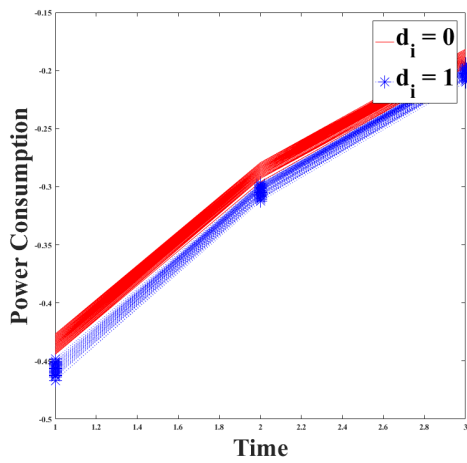
**Figure 4.** Classification result of Points of Interest (PoIs) (Hardware Implementation, Electromagnetic). (a) Case Study 1 (Property 1, KBEA, LF-R 400); (b) Case Study 2 (Property 3, KBEA, LF-R 400); (c) Case Study 1 (Property 1, KBEA, LF-R 400, BLP 15-75+); (d) Case Study 2 (Property 3, KBEA, LF-R 400, BLP 10.7-75+).



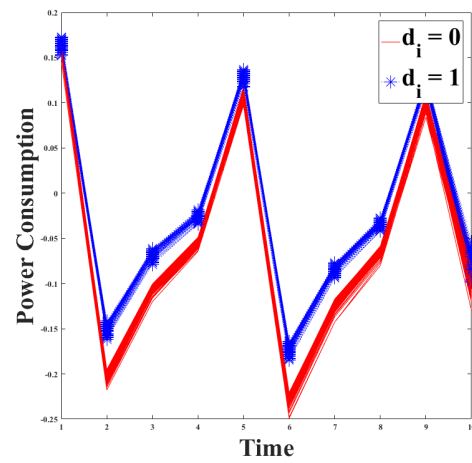
(a) Case Study 1 (Property 2, KBPA)



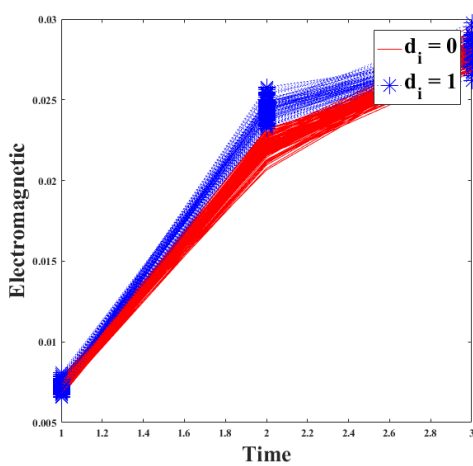
(b) Case Study 2 (Property 4, KBPA)



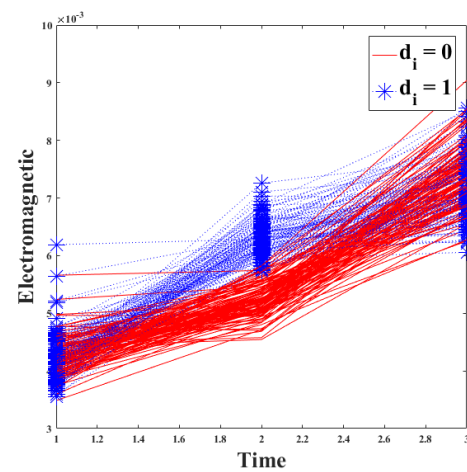
(c) Case Study 1 (Property 2, KBPA, BLP 70+)



(d) Case Study 2 (Property 4, KBPA, BLP 70+)



(e) Case Study 1 (Property 2, KBEA, LF-U 5, BLP 10.7+)



(f) Case Study 2 (Property 4, KBEA, LF-U 5, BLP 10.7+)

**Figure 5.** Classification result of PoIs (Software Implementation, mbedTLS). (a) Case Study 1 (Property 2, KBPA); (b) Case Study 2 (Property 4, KBPA); (c) Case Study 1 (Property 2, KBPA, BLP 70+); (d) Case Study 2 (Property 4, KBPA, BLP 70+); (e) Case Study 1 (Property 2, KBEA, LF-U 5, BLP 10.7+); (f) Case Study 2 (Property 4, KBEA, LF-U 5, BLP 10.7+).

## 4. Methods

### 4.1. Key Bit-Dependent Attack Framework

In this paper, we consider binary scalar multiplication algorithms that are resistant against SPA and DPA. In particular, we targeted algorithms based on regular algorithms protected by intermediate data randomization. Therefore, we suppose that an attacker is obliged to use a single trace rather than numerous traces. In addition, we assumed that the attacker could distinguish the iterative structure in the traces of regular algorithms. We categorized the attack framework in four steps as follows. Note that we did not consider side-channel atomicity algorithms that are SPA-resistant since it is impossible to distinguish the starting point of iterative loop operations.

- Preprocessing

The attacker can divide trace  $T$  into  $\lambda$  substraces,  $O_i$ , corresponding to each iteration ( $0 \leq i \leq \lambda - 1$ ). As shown in Figure 6, trace  $T$  is described as a series of  $\lambda$  sub-races as

$$T = \{O_{\lambda-1} || O_{\lambda-2} || \cdots || O_0\}$$

since  $\lambda$  iterative operations are performed when the secret scalar is  $\lambda$ -bit, we divide trace  $T$  into  $\lambda$  substraces and align them.

- Select Points of Interest (PoIs)

If the attacker can use the same device as the target and acquire a trace with a known key, it is easy to find PoIs. The attacker can calculate the sum of squared pairwise t-differences (SOST) [26] of the substraces classified based on the properties described in Sections 3.1.1 and 3.1.2. Then, the PoIs are the points that have high SOST values. SOST is calculated as follows:

$$SOST = \left( \frac{m_{G_1} - m_{G_2}}{\sqrt{\frac{\sigma_{G_1}^2}{n_{G_1}} + \frac{\sigma_{G_2}^2}{n_{G_2}}}} \right)^2 \quad (1)$$

where  $m$  denotes the mean,  $\sigma$  is standard deviation, and  $n$  is the number of elements. If it is not possible to use the same device, the attacker must know how the target algorithm is implemented to find PoIs. Moreover, the key bit identification phase section should be recognized in the trace. In general, since the  $d_i$  value must be decided in advance before each loop operation, the target phase is positioned near the beginning of each subtrace  $O_i$ . We represent  $p_i$  as PoIs of each subtrace  $O_i$  ( $0 \leq i \leq \lambda - 1$ ).

- Classify into Two Groups and Extract Secret Scalar Bits

The attacker can separate  $p_i$  into two groups,  $G_0$  and  $G_1$ , applying SPA-VI or a clustering algorithm (e.g., k-means, fuzzy k-means, or EM algorithm [27,28]). Because the most significant bit is always 1, the attacker can configure  $d_{\lambda-1}$  as 1 and find the respective scalar bit  $d_i$  based on the power model and properties described in Sections 3.1.1 and 3.1.2. For instance, when power consumption complies with the HD model, the attacker can recover secret scalar bits  $d_i$  as follows. It is possible to assume that the group that contains  $p_{\lambda-1}$  indicates that leakage is nonzero, if  $d_i$  is at first initialized as zero. Consequently, if  $p_i$  is contained in the same group that contains  $p_{\lambda-1}$ ,  $d_i$  is one; otherwise,  $d_i$  is zero ( $0 \leq i < \lambda - 1$ ). Similarly, when power consumption complies with the HW model, the group that includes  $p_{\lambda-1}$  indicates that leakage is non-zero, and the other group indicate that leakage is zero. Consequently, if  $p_i$  is contained in the same group that contains  $p_{\lambda-1}$ , then  $d_i$  is one; otherwise,  $d_i$  is zero ( $0 \leq i < \lambda - 1$ ).

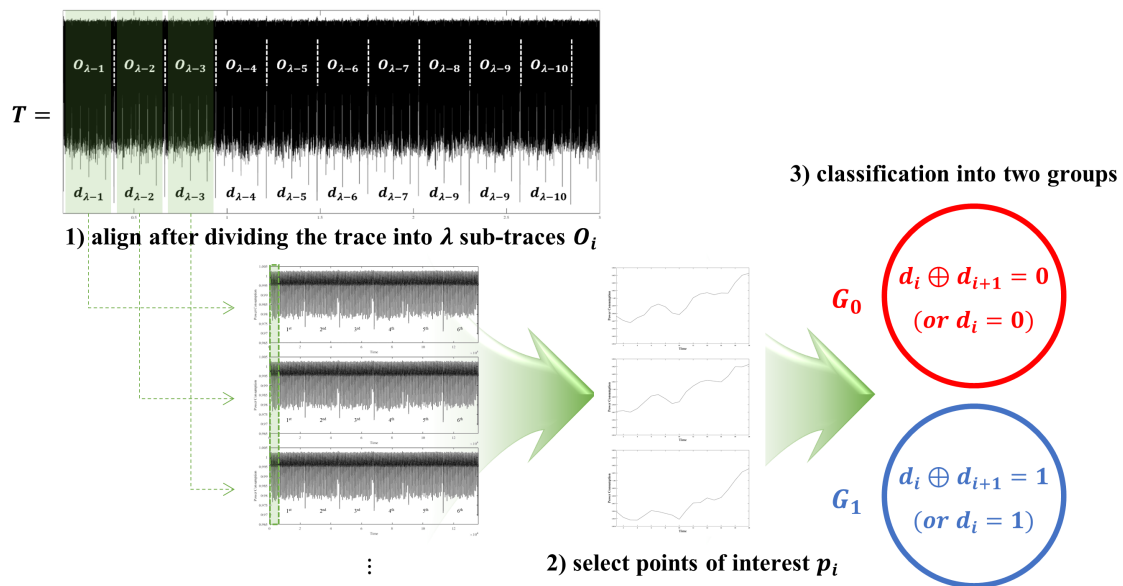


Figure 6. Key bit-dependent attack framework.

#### 4.2. Experiment Environments

The first experimental platform is VHDL implementation on a SASEBO-GII FPGA board, as shown in Figure 7. We measured traces using a Teledyne Lecroy HDO6104A oscilloscope at a sampling rate of 2.5 GS/s. Electromagnetic traces were recorded using a Langer LF-R 400. Additionally, we used Mini Circuit BLP (low-pass filter) to increase the signal-to-noise ratio. The second experimental platform was software implementation on an Atmel AVR XMEGA 128D4 microcontroller equipped with a CW-Lite XMEGA target board, as shown in Figure 7. We measured power-consumption traces using the CW-Lite main board at a sampling rate of 29.5 MS/s. Electromagnetic traces were recorded using a Teledyne Lecroy HDO6104A oscilloscope at a sampling rate of 2.5 GS/s, using a Langer LF-U 5 and Mini Circuit BLP(low-pass filter) to increase the signal-to-noise ratio.

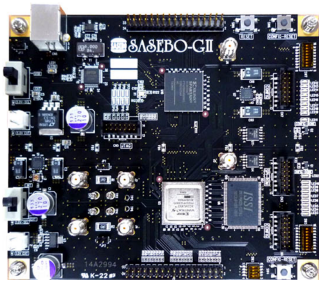
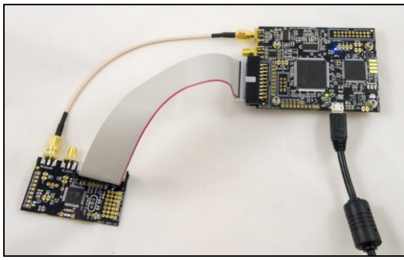
Hardware Implementation	Software Implementation
	
Power Consumption	Power Consumption
Electromagnetic	Electromagnetic

Figure 7. Key bit-dependent attack experiment environments.

## 5. Experimental Results

In this section, we demonstrate that a key bit-dependent attack could extract secret scalar bit using a single trace.

### 5.1. Key Bit-Dependent Power/Electromagnetic Attack on Hardware Implementation

Our target binary scalar multiplication algorithm was the Montgomery–López–Dahab ladder algorithm [24] protected by scalar randomization [18]. Therefore, the attacker is restricted to using a single trace. To attack algorithms operating on the first experimental platform, we focused on Properties 1 and 3, described in Sections 3.1.1 and 3.1.2, respectively. However, in hardware implementations, operations are executed in parallel. Thus, at the same time as secret scalar bits  $d_i$  are extracted at the beginning of each iterative operation, the addresses of registers  $R_x$  ( $x = 0$  or  $1$ ) to be referenced are also determined. Thus, there is no SPA-resistant regular algorithm that only satisfies Property 1. Our target was an SPA-resistant regular algorithm. Hence, we modified the code as shown in Figure 8a to identify how much information was present according to Property 1. The code as shown in Figure 8b is a general implementation that satisfies Property 3.

<pre> 1: <b>assign</b> <math>d_i</math>      = <math>regD[i]</math>; 2: <b>assign</b> <math>R_{1-d_i}</math> = <math>R_0</math>; 3: <b>assign</b> <math>R_{d_i}</math>    = <math>R_1</math>; </pre>	<pre> 1: <b>assign</b> <math>d_i</math>      = <math>regD[i]</math>; 2: <b>assign</b> <math>R_{1-d_i}</math> = <math>(d_i) ? R_0 : R_1</math>; 3: <b>assign</b> <math>R_{d_i}</math>    = <math>(d_i) ? R_1 : R_0</math>; </pre>
(a) Case Study 1	(b) Case Study 2

Figure 8. Hardware implementation: Case Study 1 (a) and Case Study 2 (b).

- **Preprocessing**  
Operations for the most significant bit  $d_{\lambda-1}$  do not exist in the Montgomery–López–Dahab ladder algorithm, as shown in Algorithm A1 and Appendix A. In accordance, trace  $T$  is composed of  $\lambda-1$  subtraces for a  $\lambda$ -bit scalar, so we divided trace  $T$  into  $\lambda-1$  subtraces  $O_i$ , and aligned them ( $0 \leq i \leq \lambda-2$ ). Figure 9 (top) shows one of the subtraces, consisting of six finite-field multiplications, captured from the first experimental platform.
- **Select Points of Interest**  
The key bit identification phase is operated on the second clock cycle of each subtrace of the target algorithm. We also confirmed that points of the second clock cycle of each subtrace are PoIs  $p_i$ , since the SOST value is the greatest on the points of the second clock cycle, as shown in Figure 9 (bottom) ( $0 \leq i \leq \lambda-2$ ). When we calculated the SOST value, we classified PoIs of subtraces  $p_i$  into two groups according to Property 1 (or 3).
- **Classify into Two Groups and Extract Secret Scalar Bits**  
(1) When we targeted Case Study 1 and exploited the power-consumption trace, it was impossible to clearly split them into two groups through SPA-VI. Since two distributions are overlapped as shown in Figure 3a, we could extract secret scalar bits  $d_i$  with a 96.75% success rate when we classified  $p_i$  into two groups based on the differences from an average trace. We could also extract secret scalar bits  $d_i$  with a 96.74% success rate when we applied the k-means clustering algorithm to classify  $p_i$  into two groups (i.e., 8 errors). Consequently, a brute-force attack to recover the entire secret scalar could be viable, because the error rate is sufficiently small. Therefore, it was confirmed that the key bit-dependent leakage based on Property 1 was sufficiently large to recover the secret scalar bits.  
(2) By using the low-pass filter to increase the signal-to-noise ratio, the success rate slightly improved to 97.17% when we applied the k-means clustering algorithm. It could not be classified into two groups through SPA-VI because distribution overlapped, as shown in Figure 3c.  
(3, 4) When we targeted Case Study 2 and exploited the power-consumption trace, it was possible



to perfectly split it into two groups via SPA-VI, as shown in Figure 3b,d. The success rate of classification was 100%, as shown in Table 1; thus, we could acquire all the secret scalar bits  $d_i$ . From this result, we noticed that changing a referring register leaks more significant information than changing the secret scalar bits. Moreover, we demonstrated that we could recover whole secret scalar bits based on Property 3 using only one power-consumption trace. We define attacks such as in Steps (1) to (4) as key bit-dependent power attacks (KBPA).

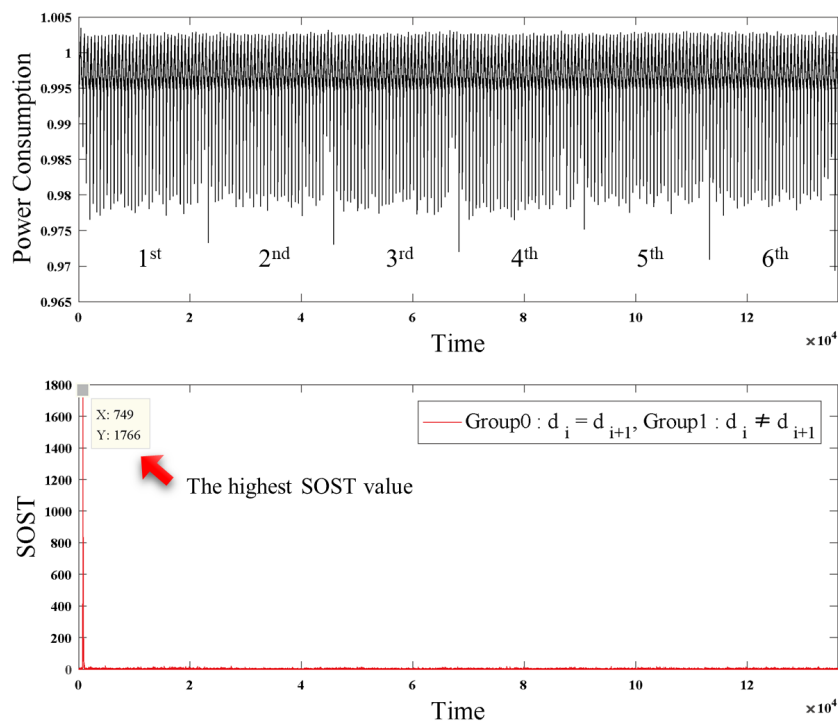
(5) Figure 4a shows the PoIs chosen from electromagnetic subtraces when we targeted *Case Study 1*. Although it was not easy to clearly divide them into two groups via SPA-VI, we could classify  $p_i$  into two groups with a 100% success rate using the differences from an average trace. Accordingly, the classification success rate based on k-means clustering algorithm was also 100%; thus, we could find the entire secret scalar bits  $d_i$  based on Property 1.

(6) Moreover, if we could use the low-pass filter to increase the signal-to-noise ratio, we could extract whole secret scalar bits through SPA-VI as shown in Figure 4c.

(7) Figure 4b shows the PoIs chosen from electromagnetic subtraces when we targeted *Case Study 2*. Unlike the result of (3, 4), it was not easy to clearly divide them into two groups via SPA-VI. However, it was possible to divide  $p_i$  into two groups based on the differences from an average trace. Thus, the classification success rate based on k-means clustering algorithm was also 100%; therefore, we could find all the secret scalar bits  $d_i$  based on Property 3 ( $0 \leq i \leq \lambda - 1$ ).

(8) Moreover, if we could use the low-pass filter to increase the signal-to-noise ratio, we could extract whole secret scalar bits through SPA-VI as shown in Figure 4d.

To sum up, we also showed that we could recover all secret scalar bits using only one electromagnetic trace. Compared to the key bit-dependent power attack, the secret scalar bits could be recovered with a 100% success rate based on Property 1. We define attacks such as in Steps (5) to (8) as key bit-dependent electromagnetic attacks (KBEA).



**Figure 9.** Hardware implementation: one of the subtraces (top) and the SOST value between two subgroups (bottom).

**Table 1.** Experimental results: hardware implementations.

Hardware	Power Consumption (Figure 3)				Electromagnetic (Figure 4)			
	None		Low-Pass Filter		None		Low-Pass Filter	
	diff	k-means	diff	k-means	diff	k-means	diff	k-means
Property 1	96.75%	96.74%	96.75%	97.71%	100%	100%	100%	100%
Property 3	100%	100%	100%	100%	100%	100%	100%	100%

diff: clustering based on the difference from an average trace; k-means: clustering using the k-means clustering algorithm.

## 5.2. Key Bit-Dependent Power/Electromagnetic Attack on Software Implementation

In this section, we focus on the key bit identification function of mbedTLS (polarSSL) as shown in Figure 10, which is an extensively used embedded transmission security TLS/SSL public encryption library. It should be noted that to capture an entire binary scalar multiplication trace using the CW-Lite main board is impossible; thus, we used the modified algorithm shown in Figure 11 based on the function in Figure 10 to identify how much information exists. In Appendix B, we describe the key bit identification function of OpenSSL as shown in Figure A1. We also show the experimental results of when we used it. For attack algorithms operating on the second experimental platform, we focused on Properties 2 and 4 described in Sections 3.1.1 and 3.1.2, respectively.

```

1: int mbedtls_mpi_get_bit (const mbedtls_mpi *X, size_t pos)
2: {
3:     if (X->n * biL <= pos)
4:         return(0);
5:
6:     return ((X->p[pos / biL] >> (pos % biL)) & 0x01);
7: }

```

**Figure 10.** Key bit identification function of mbedTLS.

```

1:  $d_i = \text{mbedtls\_mpi\_get\_bit}(\&d, i);$ 
2: BN_MUL(&R2, &R $_{d_i}$ , &R2);

```

**Figure 11.** Software implementation (mbedTLS): acquisition range.

- **Preprocessing**  
We uniformly divided trace  $T$  into  $\lambda$  subtraces,  $O_i$ , and aligned them ( $0 \leq i \leq \lambda - 1$ ). Figure 12 (top) shows one of the subtraces.
- **Select Points of Interest**  
In software implementation, operations are sequentially executed. Hence, differing from hardware implementations, we targeted two positions. The first came immediately after the  $\&0x01$  operation was performed, as shown in Figure 10. The second was where the register was referred to. The register addresses to be referenced were determined according to secret scalar bit  $d_i$ , so there was information associated with  $d_i$ . Thus, we targeted where the register LOAD operation was performed for a long integer operation. Points with high SOST values are located where the key bit identification function is performed (see Figure 12). The second target points were located behind the key bit identification function. Here, we chose points with high SOST values as PoIs. When we calculated SOST values, we classified PoIs of subtraces  $p_i$  into two groups according to Property 2 (or 4).

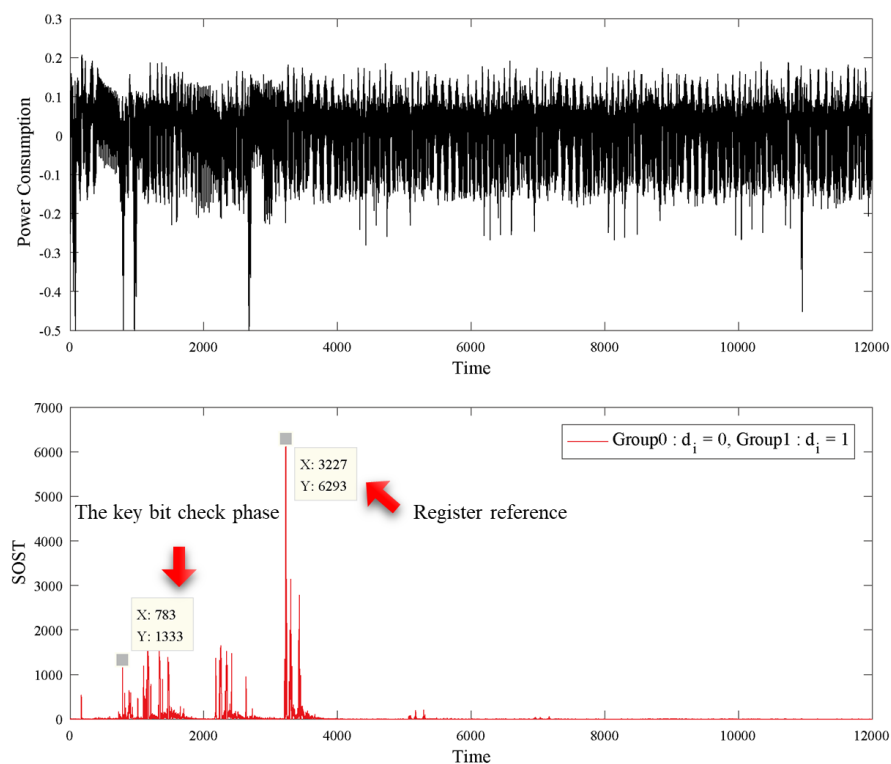
- Classify into Two Groups and Extract Secret Scalar Bits

(1) When we targeted *Case Study 1* and exploited the power-consumption trace, we could not clearly split it into two groups via SPA-VI, because the two distributions overlapped as shown in Figure 5a, so we applied the k-means clustering algorithm to classify  $p_i$  into two groups ( $0 \leq i \leq \lambda - 1$ ). Approximately 97.60% of the secret scalar bits  $d_i$  could be extracted, as shown in Table 2. There are misclassified bits, but the number of error bits is sufficiently small. Hence, it is possible to recover whole secret scalar bits with a brute-force attack. Consequently, we confirmed that the key bit-dependent leakage based on Property 2 was sufficiently large to recover the secret scalar bits.

(2) By using the low-pass filter to increase the signal-to-noise ratio, success rate was slightly improved to 98.24% when we applied the k-means clustering algorithm. It could not be classified into two groups through SPA-VI because the distribution overlapped, as shown in Figure 5c.

(3, 4) We investigated leakage associated with referred register addresses determined according to  $d_i$  in *Case Study 2*. When we exploited the power-consumption trace, subtraces  $p_i$  could be divided into two groups through SPA-VI with a 100% success rate, see Figure 5b,d.

(5) Figure 5e shows the PoIs chosen from electromagnetic subtraces when we targeted *Case Study 1*. They could not be clearly divided into two groups via SPA-VI; thus, the k-means clustering algorithm was needed. Secret scalar bits recovery rate was 94.17%, as shown in Table 2. This was slightly higher (0.17%) than the success rate when we divided  $p_i$  into two groups based on the differences from an average trace. (6) Unlike the result of (2), PoIs could not be perfectly split into two groups by SPA-VI, as shown in Figure 5f. Thus, we applied the k-means clustering algorithm and we could find approximately 95.96% of the secret scalar bits. This was slightly better than the 93.72% success rate when we divided  $p_i$  into two groups based on differences from an average trace. Here, we demonstrated that single-trace KBPA and KBEA can also defeat binary scalar multiplication algorithms that are resistant against SPA and DPA in software implementations.



**Figure 12.** Software implementation (mbedtls): one of the subtraces (top) and the SOST value between two subgroups (bottom).

**Table 2.** Experimental results: software implementations.

Hardware	Power Consumption				Electromagnetic	
	None		Low-Pass Filter		Low-Pass Filter	
	diff	k-means	diff	k-means	diff	k-means
Property 2	97.60%	97.60%	97.31%	98.24%	93.72%	94.17%
Property 4	100%	100%	100%	100%	94.17%	95.96%

diff: clustering based on the difference from average trace; k-means: clustering using the k-means clustering algorithm.

## 6. Countermeasures

We have shown that single-trace KBPA and KBEA could recover whole secret scalar bits. Here, we discuss countermeasures against KBPA and KBEA. We propose two kinds of countermeasures, one each for hardware and software implementations.

### 6.1. Countermeasure for Hardware Implementations

For hardware implementations, we suggest random initialization that initializes the  $d_i$  variable with random bit before each key bit identification phase, as per Algorithm 1. We verified that the leakage based on Properties 1 and 3 could be efficiently eliminated. The result of the classification of  $p_i$  is shown in Figures 13a,b, and 14 (top). The success rate of the attack was approximately 50%, and it was similar to randomly guessing the secret scalar bits with a probability of 1/2.

---

#### Algorithm 1: ECC Scalar Multiplication (initialized by random bit)

---

**Input** :  $P$  is a point on an elliptic curve, a  $\lambda$ -bit scalar  $d = (d_{\lambda-1}, d_{\lambda-2}, \dots, d_0)_2$   
**Output** :  $Q = dP$

- 1:  $regD[\lambda - 1 : 0] \leftarrow \{d_{\lambda-1}, d_{\lambda-2}, \dots, d_0\}$
- 2:  $R_0 \leftarrow \infty, R_1 \leftarrow P$
- 3:  $d_i \leftarrow \text{random bit}$
- 4: **for**  $i = n - 1$  down to 0 **do**
- 5:    $d_i \leftarrow regD[i]$
- 6:    $R_{1-d_i} \leftarrow R_{d_i} + R_{1-d_i}$
- 7:    $R_{d_i} \leftarrow 2R_{d_i}$
- 8:    $d_i \leftarrow \text{random bit}$
- 9: **end for**
- 10: **Return**  $R_0$

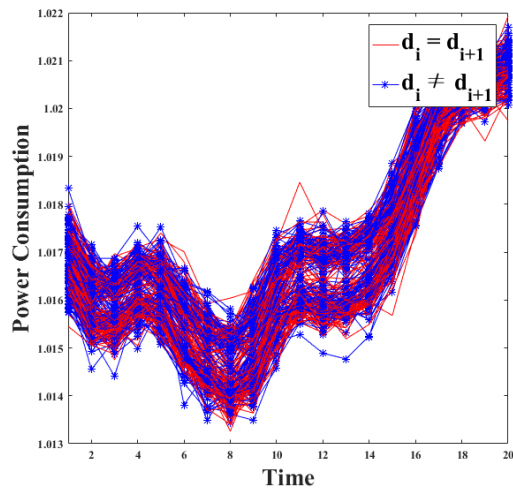
---

### 6.2. Countermeasure for Software Implementations

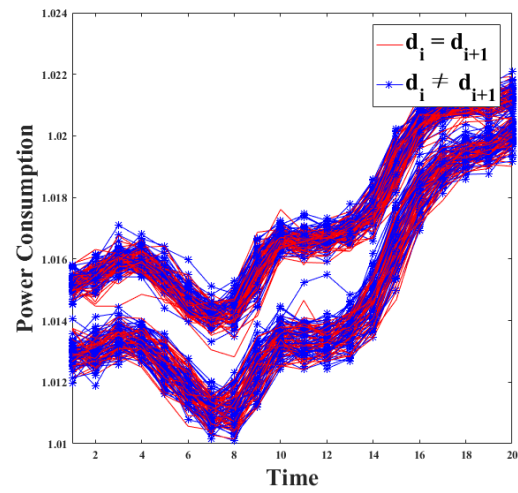
As a countermeasure for software implementations, we propose bit masking to remove the leakage of Properties 2 and 4, as Algorithm 2. This method is a type of address-bit randomization [29,30]. However, there is an important difference, in that bit masking must be performed before loop operation begins, which is shown in Step 2 of Algorithm 2. The result of classification of  $p_i$  is shown in Figures 13c,d and 14 (bottom). The success rate of the attack is also approximately 50%, and it is similar to randomly guessing the secret scalar bits with a probability of 1/2.

**Algorithm 2:** ECC Scalar Multiplication (Masking with random bit)**Input** :  $P$  is a point on an elliptic curve, a  $\lambda$ -bit scalar  $d = (d_{\lambda-1}, d_{\lambda-2}, \dots, d_0)_2$ **Output** :  $Q = dP$ 

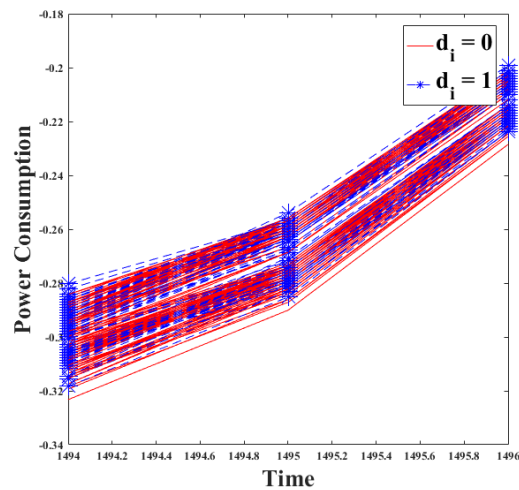
- 1: Generate  $\lambda$ -bit random number  $r = (r_{\lambda-1}, r_{\lambda-2}, \dots, r_0)_2$
- 2:  $md \leftarrow d \oplus (d \ll 1) \oplus r$
- 3:  $R_{r_{\lambda-1}} \leftarrow 2P, R_{1-r_{\lambda-1}} \leftarrow P$
- 4: **for**  $i = n - 1$  **down to** 1 **do**
- 5:    $R_2 \leftarrow 2R_{md_i}$
- 6:    $R_{1-r_{i-1}} \leftarrow R_0 + R_1$
- 7:    $R_{r_{i-1}} \leftarrow R_2$
- 8: **end for**
- 9: **Return**  $R_{r_0}$



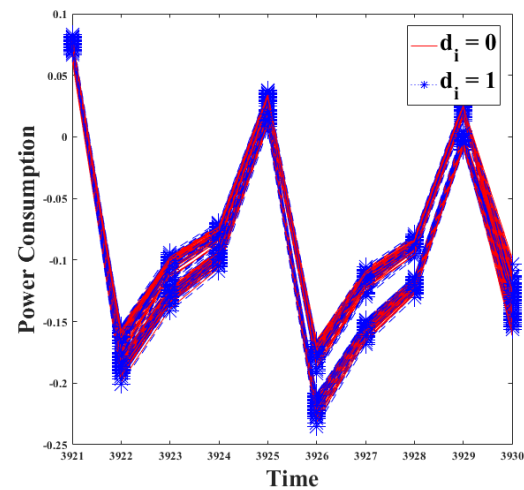
(a) Property 1 (hardware implementation)



(b) Property 3 (hardware implementation)

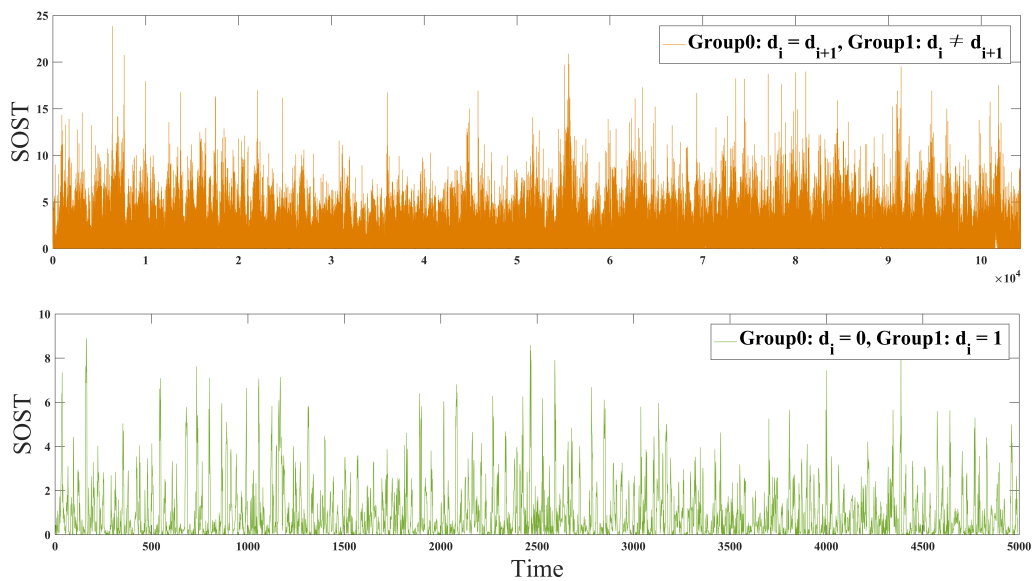


(c) Property 2 (software implementation)



(d) Property 4 (software implementation)

**Figure 13.** Countermeasure (Power). (a) Property 1 (hardware implementation); (b) Property 3 (hardware implementation); (c) Property 2 (software implementation); (d) Property 4 (software implementation).



**Figure 14.** SOST value of countermeasure: hardware implementations (**top**) and software implementations (**bottom**).

## 7. Conclusions

In this paper, we suggested attacks using the leakage that occurs on the key bit identification phase and demonstrated that such attacks could extract secret scalar bits using a single trace without profiling. The attacks could be done not only by power consumption, but also by electromagnetic trace. Compared with previous attacks that required sophisticated preprocessing and multitraces, this represents a significant advantage. There is no need to apply preprocessing, and we could recover the entire secret scalar bits through SPA-VI. Since the proposed KBPA and KBEA attacks could defeat existing countermeasures, this leads to a very robust attack model. Although we focused on ECC binary scalar multiplication algorithms, our proposed attacks are also applicable to RSA binary modular exponentiation algorithms. We proposed countermeasures and experimentally verified that the leakage was removed.

## 8. Patents

This section is not mandatory, but may be added if there are patents resulting from the work reported in this manuscript.

**Author Contributions:** These authors contributed equally to this work.

**Funding:** This research received no external funding.

**Acknowledgments:** This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00520, Development of SCR-Friendly Symmetric Key Cryptosystem and Its Application Modes).

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A. Montgomery–López–Dahab Ladder Algorithm

Algorithm A1 performs except operations for the most significant bit,  $d_{\lambda-1}$ . Each iterative operation consists of six patterns, because Steps 7 to 9 (Steps 10 to 11) consist of six finite-field multiplications.



**Algorithm A1:** Binary scalar multiplication: Montgomery–López–Dahab Ladder**Input** :  $P = (x, y)$  is a point on elliptic curve, an  $\lambda$ -bit scalar  $d = (d_{\lambda-1}, d_{\lambda-2}, \dots, d_0)_2$ **Output** :  $Q = dP$ 

```

1: if  $d = 0$  or  $x = 0$  then
2:   output  $(0, 0)$  and stop
3: end if
4:  $X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^2 + b, Z_2 \leftarrow x^2$ 
5: for  $i := \lambda - 2$  down to 0 do
6:    $Z_3 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$ 
7:   if  $d_i = 1$  then
8:      $X_1 \leftarrow x Z_3 + (X_1 Z_2)(X_2 Z_1), Z_1 \leftarrow Z_3$ 
9:      $X_2 \leftarrow X_2^4 + b Z_2^4, Z_2 \leftarrow X_2^2 Z_2^2$ 
10:  else
11:     $X_2 \leftarrow x Z_3 + (X_1 Z_2)(X_2 Z_1), Z_2 \leftarrow Z_3$ 
12:     $X_1 \leftarrow X_1^4 + b Z_1^4, Z_1 \leftarrow X_1^2 Z_1^2$ 
13:  end if
14: end for
15:  $A \leftarrow Z_1 Z_2, B \leftarrow x Z_2, C \leftarrow (xA)^{-1}$ 
16:  $D \leftarrow ((x^2 + y)A + (B + X_2)(xZ_1 + X_1))C$ 
17:  $x_0 \leftarrow BX_1 C, y_0 \leftarrow (x + x_0) + y$ 
18: Return  $dP = (x_0, y_0)$ 

```

**Appendix B. openssl**

The first target points come immediately after the  $\&()(\text{BN\_ULONG})1$  operation is performed, as shown in Figure A3. The second is where the register is referred according to the  $d_i$  value, and it is located after the key bit identification function is performed. We chose points with a high SOST value as PoIs. (1) When we targeted *Case Study 1* and exploited the power-consumption trace, it was impossible to clearly split it into two groups through SPA-VI. Thus, we used the k-means clustering algorithm and recovered secret scalar bits  $d_i$  with a 96.25% success rate. (2) Figure 4b shows the PoIs chosen from power-consumption subtraces when we targeted *Case Study 2*. They could be clearly divided into two groups through SPA-VI. The success rate was 100%, i.e., we could recover the whole secret scalar bits.

```

1: int BN_is_bit_set (const BIGNUM *a, int n)
2: {
3:     int i, j;
4:
5:     bn_check_top(a);
6:     if (n < 0)
7:         return(0);
8:
9:     i = n / BN_BITS2;
10:    j = n % BN_BITS2;
11:    if (a->top <= i)
12:        return(0);
13:
14:    return (int)((((a->d[i] >> j) & ((BN_ULONG)1)));
15: }

```

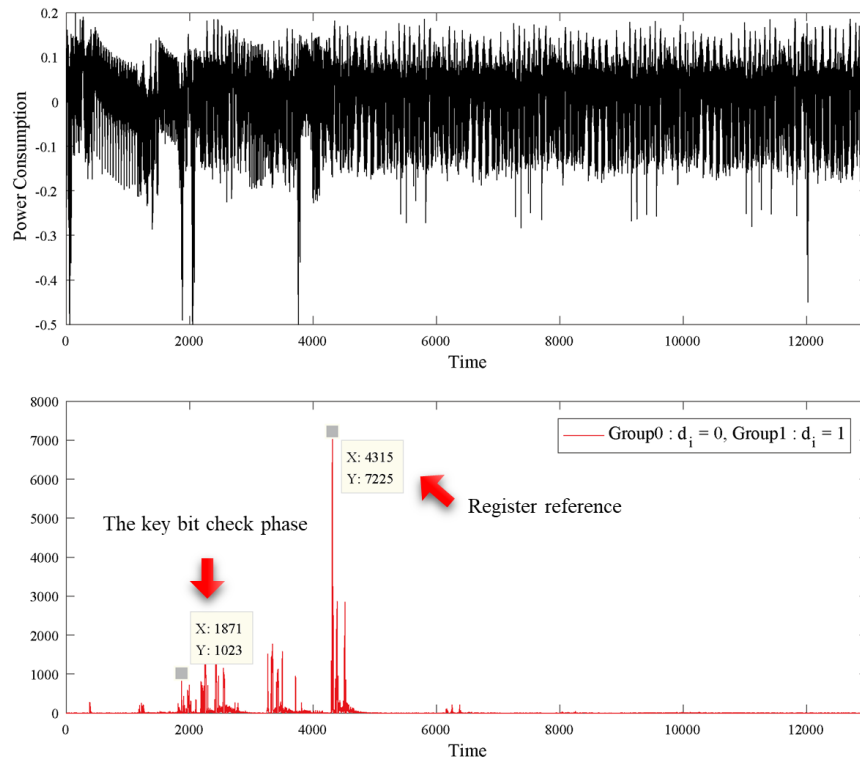
**Figure A1.** Key bit identification function of OpenSSL.

```

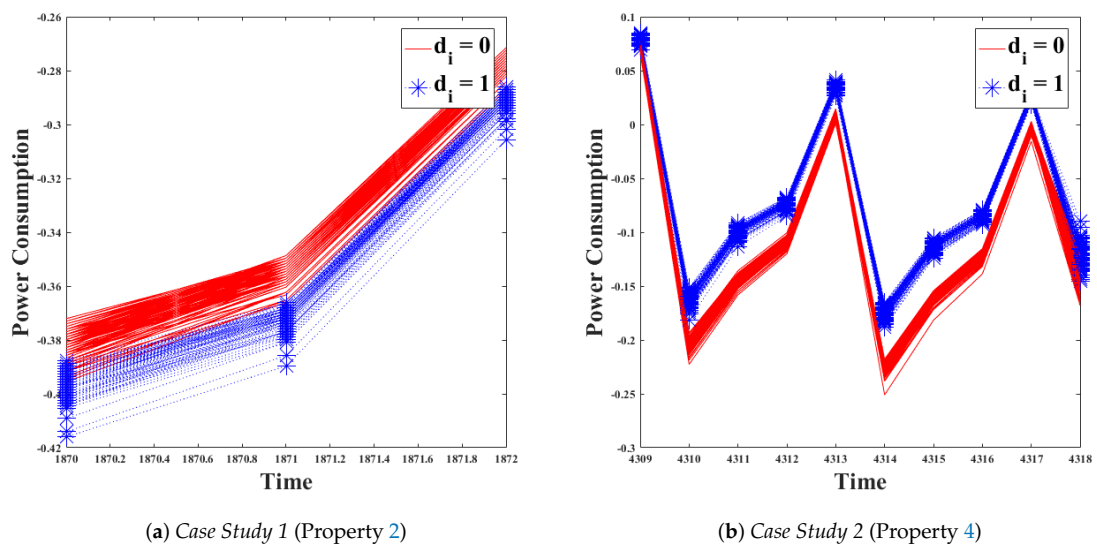
1:  $d_i = \text{BN\_is\_bit\_set}(\&d, i);$ 
2:  $\text{BN\_MUL}(\&R_2, \&R_{d_i}, \&R_2);$ 

```

**Figure A2.** Software implementation (OpenSSL) acquisition range.



**Figure A3.** Software implementation (OpenSSL): one of the subtraces (**top**), and the SOST value between two subgroups (**bottom**).



**Figure A4.** Points of Interest in the power-consumption trace (Software Implementation, OpenSSL). (a) Case Study 1 (Property 2); (b) Case Study 2 (Property 4).

## References

1. Kocher, P. Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS, and Other Systems. *CRYPTO* **1996**, *9*, 104–113. [\[CrossRef\]](#)
2. Diop, I.; Liardet, P.Y.; Maurine, P. Collision Based Attacks in Practice. *DSD* **2015**, *24*, 367–374. [\[CrossRef\]](#)
3. Clavier, C.; Feix, B.; Gagnerot, G.; Roussellet, M.; Verneuil, V. Horizontal Correlation Analysis on Exponentiation. *ICISC* **2010**, *5*, 46–61. [\[CrossRef\]](#)
4. Diop, I.; Carbone, M.; Ordas, S.; Linge, Y.; Liardet, P.Y.; Maurine, P. Collision for estimating SCA Measurement Quality and Related Applications. *CARDIS* **2015**, *9*, 143–157. [\[CrossRef\]](#)
5. Hanley, N.; Kim, H.S.; Tunstall, M. Exploiting Collisions in Addition Chain-based Exponentiation Algorithms Using a Single Trace. *CT-RSA* **2015**, *23*, 431–448. [\[CrossRef\]](#)
6. Heyszl, J.; Mangard, S.; Heinz, B.; Stumpf, F.; Sigl, G. Localized Electromagnetic Analysis of Cryptographic Implementations. *CT-RSA* **2012**, *15*, 231–244. [\[CrossRef\]](#)
7. Heyszl, J.; Ibing, A.; Mangard, S.; Saints, F.; Sigl, G. Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations. *CARDIS* **2013**, *6*, 79–93. [\[CrossRef\]](#)
8. Homma, N.; Miyamoto, A.; Aoki, T.; Satoh, A. Comparative Power Analysis of Modular Exponentiation Algorithms. *IEEE* **2010**, *176*, 795–807. [\[CrossRef\]](#)
9. Kocher, P.; Jaffe, J.; Jun, B. Differential Power Analysis. *CRYPTO* **1999**, *6*, 388–397. [\[CrossRef\]](#)
10. Nascimento, E.; Chmielewski, L.; Oswald, D.; Schwabe, P. Attacking embedded ECC implementations through cmov side channels. *SAC* **2016**, *6*, 99–119. [\[CrossRef\]](#)
11. Perin, G.; Imbert, L.; Torres, L.; Maurine, P. Attacking Randomized Exponentiations Using Unsupervised Learning. *COSADE* **2014**, *11*, 144–160. [\[CrossRef\]](#)
12. Perin, G.; Chmielewski, L. A Semi-Parametric Approach for Side-Channel Attacks on Protected RSA Implementations. *CARDIS* **2015**, *3*, 34–53. [\[CrossRef\]](#)
13. Specht, R.; Heyszl, J.; Kleinstüber, M.; Sigl, G. Improving Non-profiled Attacks on Exponentiations Based on Clustering and Extracting Leakage from Multichannel High-Resolution EM Measurements. *COSADE* **2015**, *1*, 3–19. [\[CrossRef\]](#)
14. Sugawara, T.; Suzuki, D.; Saeki, M. Internal collision attack on RSA under closed EM measurement. *SCIS* **2014**, *1*, 1–8.
15. Sugawara, T.; Suzuki, D.; Saeki, M. Two Operands of Multipliers in Side-Channel Attack. *COSADE* **2015**, *5*, 64–78. [\[CrossRef\]](#)
16. Walter, C.D. Sliding Windows Succumbs to Big Mac Attack. *CHES* **2001**, *24*, 286–299. [\[CrossRef\]](#)
17. Ciet, M.; Joye, M. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. *ICISC* **2003**, *32*, 348–359. [\[CrossRef\]](#)
18. Coron, J. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. *CHES* **1999**, *25*, 292–302. [\[CrossRef\]](#)
19. Joye, M.; Yen, S.M. The Montgomery Powering Ladder. *CHES* **2002**, *22*, 291–302. [\[CrossRef\]](#)
20. May, D.; Muller, H.; Smart, N. Random Register Renaming to Foil DPA. *CHES* **2001**, *4*, 28–38. [\[CrossRef\]](#)
21. Montgomery, P. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Math. Comput.* **1987**, *48*, 243–264. [\[CrossRef\]](#)
22. Joye, M. Highly regular right-to-left algorithms for scalar multiplications. *CHES* **2007**, *10*, 135–147. [\[CrossRef\]](#)
23. Sim, B.-Y.; Han, D.-G. Key Bit-Dependent Attack on Protected PKC Using a Single Trace. *ISPEC* **2017**, 168–185. [\[CrossRef\]](#)
24. Lopez, J.; Dahab, R. Fast Multiplication on Elliptic Curves over  $GF(2^m)$  without Precomputation. *CHES* **1999**, *27*, 316–327. [\[CrossRef\]](#)
25. Hankerson, D.; Menezes, A.; Vanstone, S. Elliptic Curve Arithmetic. In *Guide to Elliptic Curve Cryptography*; Springer: New York, NY, USA, 2004; pp. 75–152, ISBN 0-387-95273-X.
26. Gierlichs, B.; Lemke-Rust, K.; Paar, C. Templates vs. Stochastic Methods. A Performance Analysis for Side Channel Cryptanalysis. *CHES* **2006**, *2*, 15–29. [\[CrossRef\]](#)
27. Duda, R.O.; Hart, P.E.; Stork, D.G. *Pattern Classification*, 2nd ed.; Wiley Interscience: Hoboken, NJ, USA, 2001; ISBN 978-0-471-05669-0.
28. Bishop, C. Pattern Recognition and Machine Learning. In *Information Science and Statistics*; Springer: New York, NY, USA, 2007; ISBN 978-1-4939-3843-8.

29. Izumi, M.; Ikegami, J.; Sakiyama, K.; Ohta, K. Improved Countermeasure against Address-bit DPA for ECC Scalar Multiplication. *IEEE* **2010**, 981–984. [[CrossRef](#)]
30. Itoh, K.; Izu, T.; Takenaka, M. A Practical Countermeasure against Address-Bit Differential Power Analysis. *CHES* **2003**, *30*, 382–396. [[CrossRef](#)]



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).