Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 6495546, 12 pages
https://doi.org/10.1155/2022/6495546

Research Article

Novel Shuffling Countermeasure for Advanced Encryption Standard (AES) against Profiled Attack in Mobile Multimedia Services

JongHyeok Lee,[1] **Jiyoon Kim**,[1] **and Dong-Guk Han**[1,2]

[1]*Department of Financial Information Security, Kookmin University, 02707 Seoul, Republic of Korea*
[2]*Department of Information Security, Cryptology and Mathematics, Kookmin University, 02707 Seoul, Republic of Korea*

Correspondence should be addressed to Dong-Guk Han; christa@kookmin.ac.kr

Received 12 May 2022; Accepted 21 June 2022; Published 13 July 2022

Academic Editor: Yuanlong Cao

Mobile multimedia services are gaining popularity among many users by developing wireless communication and mobile devices. Mobile multimedia has alleviated conventional multimedia's time and space limits, making it easier for consumers to access services and meet content demands. However, cyber risks lie in the shadows of the expansion of mobile multimedia services, threatening to continue wreaking havoc. Although various methods exist to defend against these cyber threats, side-channel analysis has remained a critical challenge in the current approaches that rely on cryptographic algorithms. Nowadays, research on deep learning-based side-channel analysis is receiving much attention. Attacks are constantly performed against implementations, to which existing countermeasures against traditional side-channel analysis are applied, using various artificial neural network structures. However, while studies on the implementations to which masking and simple hiding schemes using jitter are active, studies on the implementations to which the shuffling scheme or the random insertion of dummy operations scheme are applied have been relatively less attention. In a previous study, Lee and Han has used deep learning to distinguish between real and dummy operations in an implementation that combined shuffling scheme and random insertion of dummy operations scheme. They also proposed countermeasures against their attacks. However, they did not choose an appropriate environment that is as close to noise-free as possible, and their countermeasure still has flaws. Therefore, in this study, we analyze the causes of vulnerability of the previous countermeasure and propose a novel countermeasure that can completely solve them. The novel countermeasure is a method of uniformly applying shuffling schemes and random insertion of dummy operation schemes to byte-independent and byte-dependent operations of an advanced encryption standard, respectively. It was confirmed that our countermeasure is safe from attackers who perform profiled attacks even in an experimental environment with almost no noise.

1. Introduction

Mobile multimedia services have risen in response to the advancement of mobile communication technologies and increasing demand for content. Especially with the introduction of 5G, multimedia content can now be quickly delivered to users with high communication capacity, transmission speed, and low latency. Accordingly, the global mobile subscriber base is predicted to grow from 5.1 billion in 2018 to 5.7 billion in 2023, representing an increase from 66% of the worldwide population in 2018 to 71% in 2023 [1]. However, as the mobile multimedia industry grows, cyber threats against it become more diverse and complex. These threats can cause various damages to both customers and service providers. In this paper, we focus on side-channel analysis among the cyber threats of mobile multimedia services.

Side-channel analysis reveals secret information based on the fact that the power consumption of cryptographic devices depends on intermediate values of the cryptographic algorithms. These dependencies are of two types, data and operation dependency [2]. Typical attack methods utilizing

data dependency are differential and correlation power analysis [3, 4]. On the other hand, other methods use operation dependency is simple power analysis [3]. Simple power analysis recovers secret information using the difference in operation according to the secret information for asymmetric cryptographic algorithms or is mainly used to distinguish the operation of symmetric cryptographic algorithms to enable intensive side-channel trace collection. Therefore, simple power analysis uses a single trace, whereas differential and correlation power analyses use many traces because these are attacks that recover secret information using statistical techniques.

Countermeasures against side-channel analysis break data dependency or operation dependency of side-channel information. Software countermeasures usually break data dependency which is divided into masking [5, 6] and hiding schemes [2, 7]. Masking schemes overlay random values on an intermediate value to make it seem to be random. This makes it impossible for an attacker to infer the intermediate value. On the other hand, hiding schemes randomize the execution time of operations to prevent an attacker from estimating the operation time. Masking schemes logically block specific attacks, and for an attacker to attack a target that the masking schemes block, high-level attacks must be used to neutralize the masking schemes, compared with hiding schemes where it only increases an attack complexity by increasing the number of traces required by the attacker.

There are several methods to reduce the attack complexity increased by hiding schemes. One is the alignment method, which is commonly used. This type of method comprises static alignment, elastic alignment, and alignment schemes using pattern recognition or hidden Markov models [2, 8–10]. Recently, as research on deep learning-based side-channel analysis (DLSCA) progresses, studies have been published where some deep learning-based side-channel analyses neutralize hiding schemes [11–13]. In addition, studies on DLSCA against shuffling and dummy operations are being actively conducted [14–16]. However, they mainly deal with desynchronization due to jitter, and so on, rather than shuffling scheme or the random insertion of dummy operations scheme. Lee and Han performed machine learning-based side-channel analysis for the first time on a target in which the shuffling scheme and the random insertion of dummy operation schemes were used [17]. They showed that an attack was possible and at the same time suggested a countermeasure.

1.1. Our Contributions. In this study, we question the safety of the previous countermeasure. In a previous study, the authors experimentally demonstrated the safety of the proposed countermeasure but their experimental environment was noisy when compared to this present study [17]. Moreover, the previous countermeasure directly refers to the shuffled order array exposing the vulnerability. Because of the noise in the experimental environment, this vulnerability did not appear in the experimental results.

The current study shows that the previous countermeasure is insecure by performing profiled attacks on the ChipWhisperer-Lite board [18], which is considered an ideal environment with particle noise. Furthermore, this shows that the previous countermeasure is ineffective and reveals the cause of its weakness.

Input: IN[32], ORD[32]
Output: OUT[32]
1: for $i \longleftarrow$ 0 to 31do
2: OUT[ORD[i]]$\longleftarrow$Sbox[IN[ORD[i]]]
3: end for

Algorithm 1: Pseudocode for the previous countermeasure [19]

```
1; OUT[ORD[i]] = Sbox[IN[ORD[i]]]
2 movw   r28, r24
3 ldi    r26, 0x13
4 ldi    r27, 0x22
5 ldi    r20, 0x33
6 ldi    r21, 0x22
7 ld     r18, X+
8 ldi    r19, 0x00
9 movw   r30, r28
10 add    r30, r18
11 adc    r31, r19
12 ld     r30, Z
13 ldi    r31, 0x00
14 subi   r30, 0xF6
15 sbci   r31, 0xDF
16 ld     r25, Z
17 movw   r30, r22
18 add    r30, r18
19 adc    r31, r19
20 st     Z, r25
```

Listing 1: Assembly code for the previous countermeasure [19].

A novel countermeasure, which is presently used in the study, was designed based on the vulnerability causes that were analyzed. The design concept of the novel countermeasure is to apply the shuffled operation order to the confusion and diffusion layers. Therefore, a uniform hiding scheme can be applied to the entire encryption algorithm. The novel countermeasure has been experimentally proven to be safe from profiled attacks with strong attacker assumptions in an ideal environment.

1.2. Organization. The remainder of this study is structured as follows: Section 2 introduces hiding schemes, related works, and previous countermeasures as preliminaries. Section 3 describes the profiled attacks targeting the previous countermeasure and their results. Section 4 shows the novel shuffling countermeasure, which tolerates profiled attacks. Section 5 demonstrates the safety of the proposed countermeasure. Section 6 reveals the conclusion.

2. Preliminaries

This section briefly describes hiding schemes, related works, and the previous countermeasures as preliminaries.

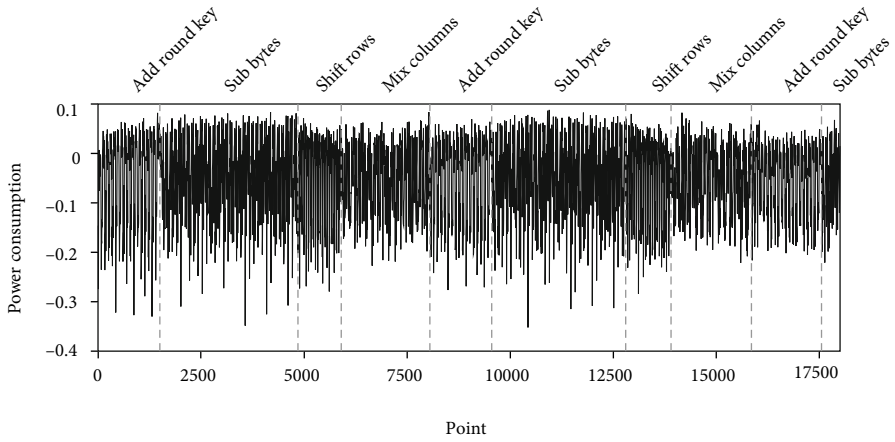


Figure 1: A power consumption trace of the previous countermeasure at optimization level -Os.

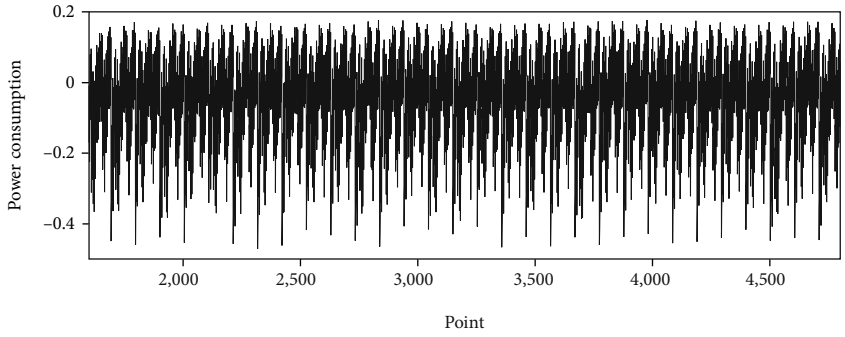


Figure 2: A power consumption trace of the first SubBytes of Figure 1.

2.1. Hiding Schemes. Hiding schemes make the power consumption of cryptographic devices independent of the intermediate values and the operations that are performed [2]. There are two approaches to achieving this purpose. The first approach is to make devices consume power randomly, and the second approach is to make devices consume the same amount of power for every operation and data value. Unfortunately, the ideal goal of randomizing or equalizing power consumption is not realistically achievable. However, there are several proposals to help get closer to this goal. These proposals are divided into two groups. The first group randomizes power consumption by performing operations at different moments. The second group touches on the amplitude dimension of power consumption. Because this study is about the first group, we would explain the first group in more detail.

The most common techniques for randomizing the execution of operations are the random insertion of dummy operations and shuffling. The random insertion of dummy operations is to randomly insert dummy operations during the execution of the operations. In this technique, randomly generated numbers are used to determine how many dummy operations to insert at different positions. As these random numbers are larger, it becomes difficult for an attacker to successfully perform an attack, but there is a disadvantage in that the implementation throughput is lowered. The shuffling randomly changes the sequence of operations that can be performed in an arbitrary order. The shuffling similarly randomizes power consumption as the random insertion of dummy operations, but the operations that can be shuffled depend on the cryptographic algorithm and is limited. Therefore, in practice, the shuffling and the random insertion of dummy operations are often combined and used.

2.2. Related Works. Assuming that shuffling and random insertion of dummy operations are combined and implemented and that up to d dummy operations can be added to n real operations, the attack complexity for recovering one key byte is $1/(n+d)$. That is, when the number of side-channel traces required to recover one key byte is α in the implementation without countermeasure, the number of required side-channel traces increases to $\alpha \times (n+d)^2$ when the countermeasures are applied [2]. However, if the attacker can distinguish dummy operations from real operations, the random insertion of dummy operations is neutralized and the number of required side-channel traces is reduced to $\alpha \times n^2$. For Sbox of Advanced Encryption Standard (AES) is fatal with a reduction of 75% when $n = d = 16$. This is a lethal number and can lower the attack complexity intended by the designer. Therefore, it is critical to make the dummy operation indistinguishable from the real operation.

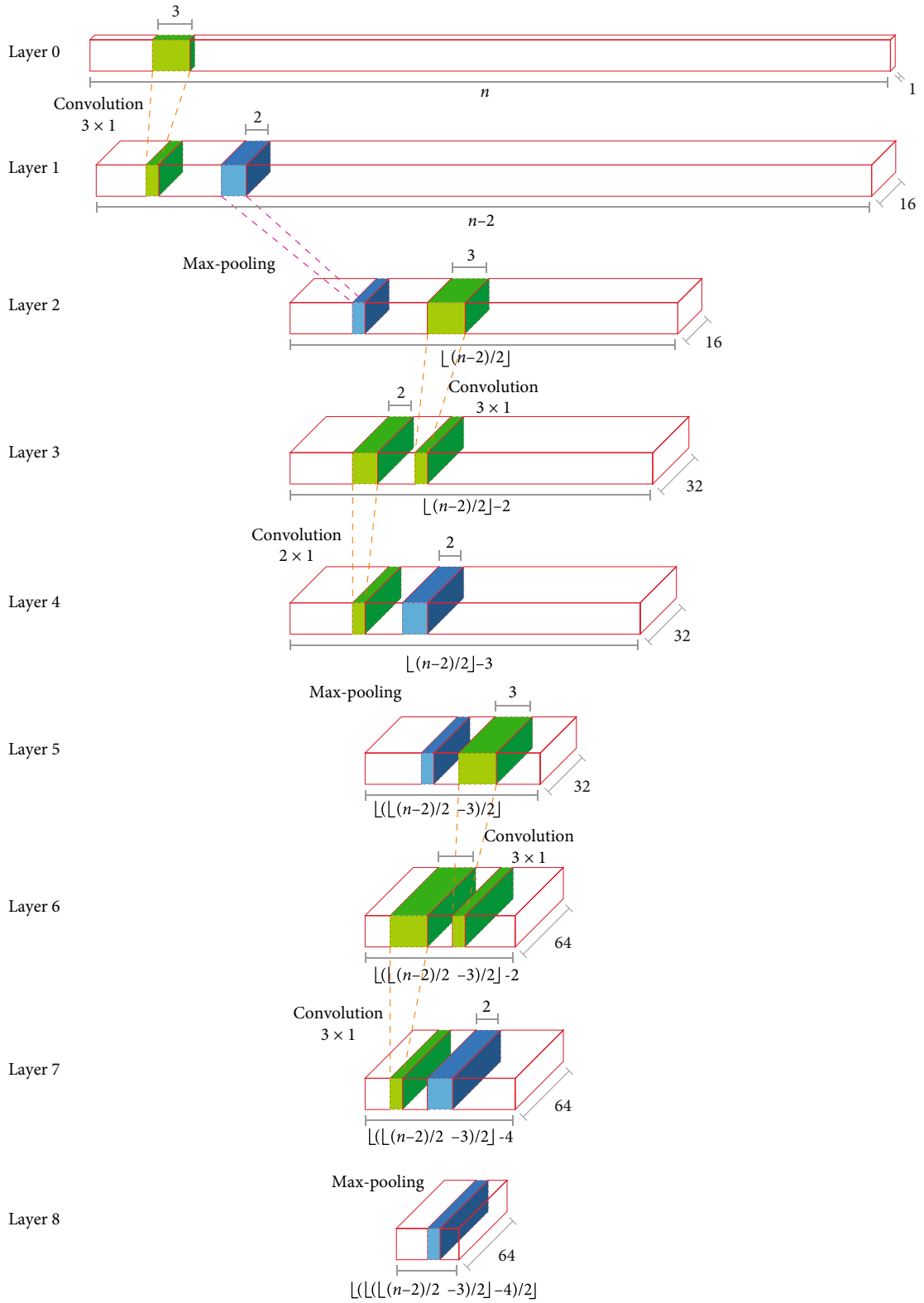


Figure 3: Convolutional neural network architecture.

Lee and Han showed for the first time the possibility of distinguishing dummy operations according to the declaration form of variables used to implement dummy operations [19]. The dummy operations were distinguished using the bounded collision detection criterion (BCDC) [20], which is a simple criterion of signal similarity. To briefly explain the method, a part of the first Sbox operation section is set as a reference area, and the BCDC value is obtained for each

area shifted by 1 point. If the first Sbox is a dummy (real) operation, the area having a low BCDC value is a part in which the dummy (real) Sbox operation is performed. As a result, the attacker can filter out dummy operations with just two trials even in a situation where the attacker does not know whether the first Sbox operation is a dummy or a real operation. However, the attack using BCDC has a disadvantage where the reference region must be selected heuristically.

The same authors later performed an attack using a neural network on the side-channel traces collected in a more noisy environment [17]. A convolutional neural network (CNN) for multilabel classification problems were used. The attacks were successful for all other declaration types except for the countermeasure proposed in their previous study [19]. However, vulnerability still exists in the countermeasure that was proposed. There is noise in the experimental environment used; thus, no vulnerability was found.

2.3. Previous Countermeasure. Lee and Han also presented a countermeasure against the proposed attack in their study [19], which proposes the attack that classifies dummy operations using BCDC. Algorithm 1 is a pseudocode of their countermeasures. It was initially thought that the vulnerabilities occur because the assembly codes are generated differently. After all, the arrays used by the dummy operations and the real operations are different, or the memory addresses referenced are different even when the same array is used. Therefore, the switch-case statements were not used to circumvent the vulnerabilities analyzed. Furthermore, the countermeasure was designed so that the dummy and the real operations can use the same array and refer directly to the array in which the shuffled order is stored.

Listing 1 is an assembly code compiled with WinAVR 20100110 (GCC-4.3.3) by implementing Algorithm 1 in the C language.

3. Profiled Attacks on Previous Countermeasure

We implemented AES with hiding schemes of Algorithm 1 on an XMEGA128D4 microprocessor [21] using C language. The power consumption was measured with a ChipWhisperer-Pro (CW1200) [18]. WinAVR 20100110 (GCC-4.3.3) is used in the compiling process and it provides -O0, -O1, -O2, -O3, and -Os as optimization levels. Detailed descriptions of each compiler optimization levels are as follows:

(i) -O0: this option does not attempt to optimize the execution time and code size. It reduces the compilation time and makes debugging generate the expected results

(ii) -O1: this compiler reduces the code size and execution time. This option only performs basic optimizations

Table 1: Test accuracies of the previous countermeasure according to optimization levels.

Optimization levels	Test accuracy
-O0	98.01875%
-O1	99.88125%
-O2	99.65000%
-O3	98.45625%
-Os	99.68125%

(iii) -O2: this compiler performs nearly all supported optimizations that do not involve a space-speed trade-off

(iv) -O3: this compiler turns on all optimizations

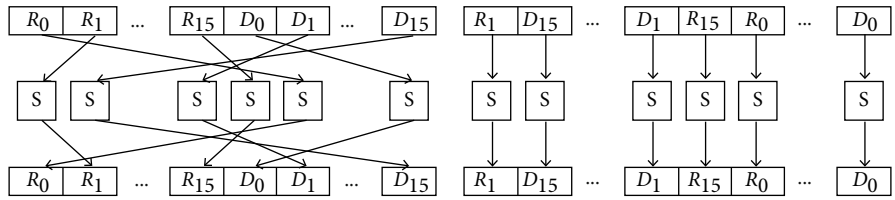
(v) -Os: this enables all -O2 optimizations, except those that often increase code size

Figure 1 shows the trace collected at optimization level -Os. It was collected to include the first two rounds and part of the third round because we unified the number of points in the trace to use the same neural network as the experiments (see Section 5). The trace for the SubBytes function of the first round is shown in Figure 2. Sixteen real and dummy operations were shuffled and performed, and it is impossible to visually distinguish whether each operation is real or the dummy.

Figure 3 shows the neural network structure to be used in the experiments in this study. The data length of the 0th layer in Figure 3 is indicated by n, which is an expression for generalization because the number of trace points varies according to the optimisation level. The number of points collected in the trace at optimisation level -O0 is 62,000, whereas the number of points collected at the rest is 18,000. This neural network uses a one-dimensional CNN, five convolution layers, and three pooling layers. ReLU is used as the activation function for each convolution layer, and batch normalization is performed after the convolution layer. The pooling layer is of max-pooling type, and a dropout ratio of 0.25 is applied after each pooling layer. After layer 8, it passes through a dense layer with 32 output nodes. The kernels of the dense layer are initialized using He normal initialization [22], and the activation function is Sigmoid. The neural network is compiled using the Adam optimizer with a learning rate of 0.001 and decay of 0.0001 and binary cross entropy as the loss function.

The attacker assumption which was set is that the attacker can collect data to train the neural network and obtain the shuffled order of operations from the profiling device. Similar to the previous study [17], the labels are composed of binary that only indicates whether the 32 operations were real or dummy. For example, if the following index of the real operations were performed:

$$[2, 3, 5, 6, 8, 10, 14, 16, 17, 18, 19, 24, 26, 27, 30, 31], \quad (1)$$



(a) In the case of the previous countermeasure (b) In the case of a novel countermeasure

Figure 4: Methods of applying the shuffling technique to the SubBytes function.

Input: Plaintext P[32] with dummy
Output: Plaintext P with dummy, shuffled orders K[32], L[32], and M[32][4], and inverse order K^{-1}

1: for $i \longleftarrow 0$ to 31 do ▷ Initialize arrays as non-shuffled orders
2: $K[i] \longleftarrow i$ and $K^{-1}[i] \longleftarrow i$ ▷ K for AddRoundKey and SubBytes
3: $S \longleftarrow (5 \times (i \bmod 16) \bmod 16) + 16 \times \lfloor i/16 \rfloor$
4: $L[i] \longleftarrow S$ and $L^{-1}[S] \longleftarrow i$ ▷ L for ShiftRows
5: for $j \longleftarrow 0$ to 3 do
6: $M[i][j] \longleftarrow \lfloor i/4 \rfloor + (i + j \bmod 4)$ ▷ M for MixColumns
7: end for
8: end for
9: for $i \longleftarrow 31$ to 1 do ▷ Shuffling
10: $R \overset{\$}{\longleftarrow} \{0, \cdots, i\}$
11: Swap P[i] and P[R]
12: Swap K[i] and K[R]
13: Swap $K^{-1}[K[i]]$ and $K^{-1}[K[R]]$
14: Swap L[i] and L[R]
15: Swap $L^{-1}[L[i]]$ and $L^{-1}[L[R]]$
16: $L[L^{-1}[i]] \longleftarrow R$, $L[L^{-1}[R]] \longleftarrow i$
17: Swap $L^{-1}[i]$ and $L^{-1}[R]$
18: Swap M[i][1] and M[R][1]
19: Swap M[i][2] and M[R][2]
20: Swap M[i][3] and M[R][3]
21: if M[i][1] = i and M[R][3] = R
22: Swap M[M[i][1]][1] and M[M[R][3]][3]
23: else
24: Swap M[M[i][1]][3] and M[M[R][3]][1]
25: end if
26: Swap M[M[i][2]][2] and M[M[R][2]][2]
27: if M[i][3] = i and M[R][1] = R
28: Swap M[M[i][3]][3] and M[M[R][1]][1]
29: else
30: Swap M[M[i][3]][1] and M[M[R][1]][3]
31: end if
32: end for
33: return P, K, L, M, K^{-1}

Algorithm 2:Generate orders for full shuffling

we can construct the following thirty-two labels:

$$[0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0]. \quad (2)$$

Here, 0 represents the dummy operation and 1 represents the real operation.

For each optimization level, 10,000 traces were collected in the variable key environment, 7,500 were used for training and 2,500 were used for validation. With a fixed key, 1,000 traces were collected for testing. The batch size was set to 10 and 100 epochs were performed; if the validation accuracy did not improve during the 10 epochs, the training was terminated early.

As a result of the training, the training was terminated early before 80 epochs in all five optimization levels and the

Input: State S[32], round key rk[32], and shuffled orders L[32] and M[4, 32]
Output: State S

1: for $i \longleftarrow 0$ to 31 do ▷ AddRoundKey
2: $S[i] \longleftarrow S[i] \oplus rk[i]$
3: end for
4: for $i \longleftarrow 0$ to 31 do ▷ SubBytes
5: $S[i] \longleftarrow Sbox[S[i]]$
6: end for
7: for $i \longleftarrow 0$ to 31 do ▷ ShiftRows
8: $T[i] \longleftarrow S[L[i]]$
9: end for
10: for $i \longleftarrow 0$ to 31 do ▷ MixColumns
11: $temp1 \longleftarrow T[M[i][1]] \oplus T[M[i][2]] \oplus T[M[i][3]]$
12: $temp2 \longleftarrow xtime(T[i] \oplus T[M[i][1]])$
13: $S[i] \longleftarrow temp1 \oplus temp2$
14: end for
15: return S

Algorithm 3: Shuffled round function of AES

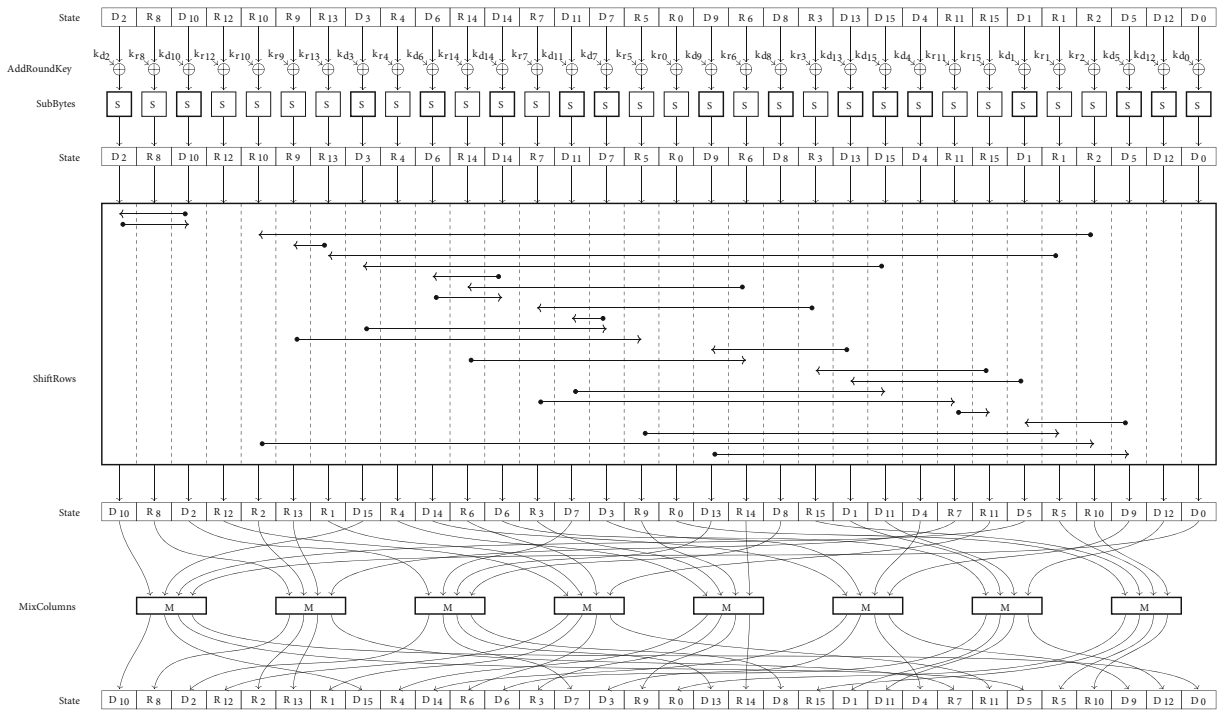


Figure 5: Structure of the novel shuffling countermeasure.

```
1;   OUT[i] = Sbox[IN[i]]
2    movw    r26, r22
3    movw    r28, r24
4    ld      r30, Y+
5    movw    r24, r28
6    ldi     r31, 0x00
7    subi    r30, 0xF6
8    sbci    r31, 0xDF
9    ld      r20, Z
```

Listing 2: Assembly code for the novel countermeasure.

validation accuracies were over 98%. There was no overfitting, and test accuracy at the optimization levels is shown in Table 1.

4. Novel Shuffling Countermeasure

In this section, we highlight the problem of the previous countermeasure and proposed a novel shuffling countermeasure that is safe from profiled attacker's assumption.

4.1. Motivation. Previous research has shown that the compiled assembly codes are different when the variables used by the real and dummy operations are declared separately, and the assembly codes are different when the switch-case

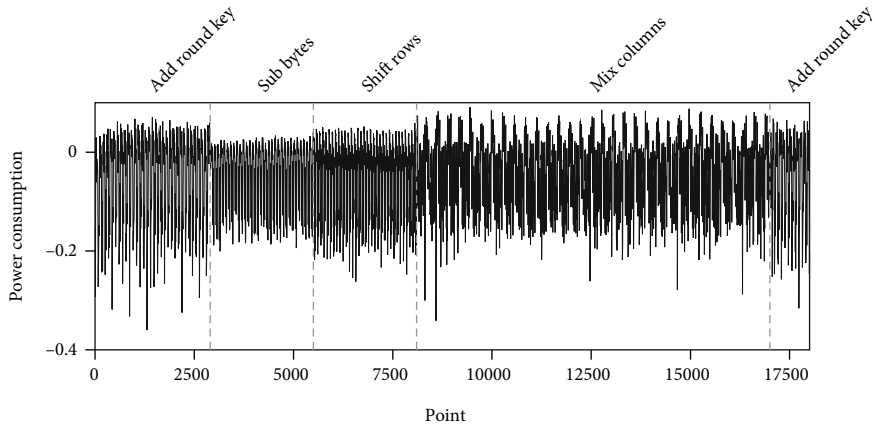


Figure 6: A power consumption trace of the novel countermeasure at optimization level -Os.

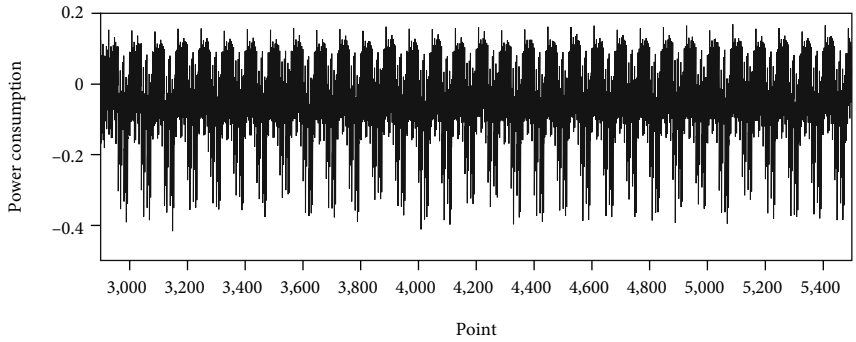


Figure 7: A power consumption trace of the first SubBytes of Figure 6.

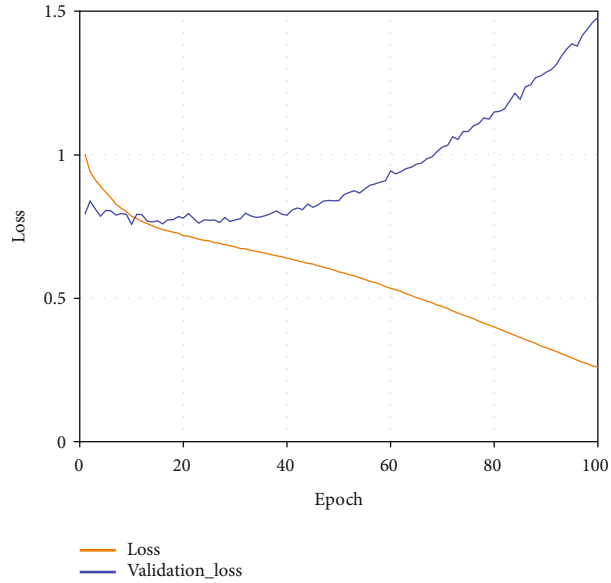
statement is used even when the same variable is used [19]. Therefore, similar to Algorithm 1, the authors made the real and dummy operations use the same array and operated by directly referencing the array in which the shuffled order is stored without using a switch-case statement. Therefore, the assembly codes for the real and dummy operations are generated identically as shown in Listing 1. Based on this, the authors assumed that their countermeasure was safe.

Their experiment has a limitation in that the experimental verification was conducted on a white card with many noises compared to the ideal environment. Because of the low signal-to-noise ratio in noisy environments, leakage is obscured by noise, making accurate experimental verification difficult. Therefore, because of the experimental safety verification of the previous countermeasure in a low-noise environment, it was not safe at all optimization levels (see Table 1).

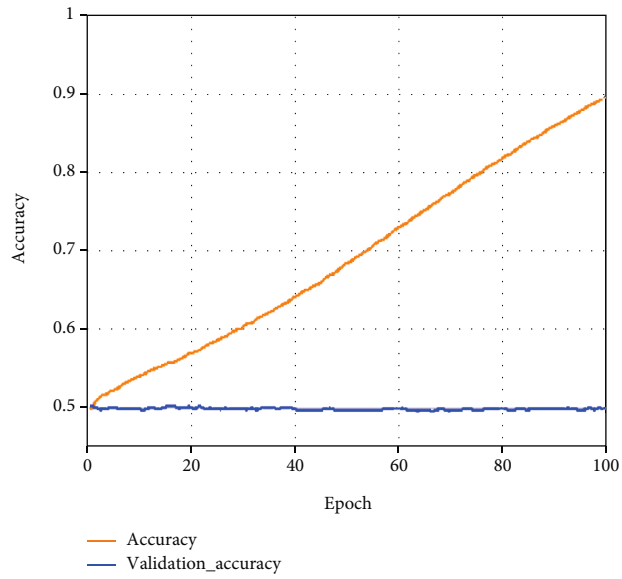
To analyze the cause of this vulnerability, the assembly code of Listing 1 was noted. Lines 3 and 4 of Listing 1 store the address of the ORD array to the r26 and r27 registers. Then, in line 7, the value of the element of the currently pointed ORD array is loaded into the r18 register, and the address of the following element is pointed. It was assumed that the attacker could see the shuffled order (see Section 3). Therefore, line 7, in which the value of shuffled order is stored in the register, is a vulnerable cause. The traces were collected by deleting line 7 and attempted to attack the collected traces, which yielded a test accuracy of nearly 50%. Therefore, this demonstrated that line 7 is a vulnerable cause.

This study concluded that the method of applying shuffling should be modified to eliminate the abovementioned vulnerability. A schematic of the method for applying the shuffling scheme to the SubBytes function of the previous countermeasure is shown in Figure 4(a). The values for real and dummy operations are sequentially stored in the array, and the operations are performed regardless of the order stored in the array by referring to the shuffled order. Because the vulnerability exists in the section that refers to the shuffled order, it is necessary to configure the plaintext to be stored in the shuffled order from the start (see Figure 4(b)). However, it is difficult to perform encryption while storing the values in the array in the shuffled order. This is because, among the internal operations of the AES encryption algorithm, ShiftRows and MixColumns are byte-dependent operations. AddRoundKey and SubBytes, which are byte-independent operations, can sequentially operate on the shuffled and stored state array, but byte-dependent operations must calculate the indices of the values to be operated together. This will be covered in more detail in the following section.

4.2. The Expansion of Shuffling Scheme to Other Operations. In this section, a method was proposed for expanding the shuffling scheme used in SubBytes to ShiftRows and MixColumns, which are byte-dependent operations of AES.



(a) Loss



(b) Accuracy

Figure 8: Training result of the novel countermeasure with optimization level -Os.

Table 2: Test accuracies of the novel countermeasure according to optimization levels.

Optimization levels	Test accuracy
-O0	49.56250%
-O1	50.25625%
-O2	50.23750%
-O3	50.60625%
-Os	49.93125%

Algorithm 2 is a pseudocode that shuffles plaintext array and order arrays for AddRoundKey, SubBytes, ShiftRows, and MixColumn. In this algorithm, 16 dummy operations are used. P is the plaintext array with dummy added, and K, L, and M are the order arrays of AddRoundKey and SubBytes, ShiftRows, and MixColumns, respectively. Additionally, a reverse-order array K^{-1} of K is generated for the recovery of the ciphertext. Only the order array M for MixColumn, where four bytes are used for operation at once, is a two-dimensional array with a size of 32×4, and the rest are all one-dimensional arrays with a length of 32.

First, lines 1 to 8 initialize the arrays in order before applying the shuffling scheme. After that, lines 9 to 32 apply the shuffling scheme using Fisher-Yates shuffle [23]. This loop from the highest index to the lowest, and lines 11 to 13 shuffle the plaintext array and the order array for AddRoundKey and SubBytes, respectively. Lines 14 and 17 shuffle the order array for ShiftRows, which is different from

Table 3: Cycles per byte of AES implementations with and without a countermeasure.

Implementation	Cycles per byte
Unprotected AES	127
AES with the previous countermeasure	572
AES with the novel countermeasure	1028

order K for byte-independent operation, since values referring to the swapped value must also be swapped, so line 17 is required. Lines 18 to 31 should be shuffled in the order of MixColumns. In the case of M, because it is a two-dimensional array and four bytes are used in the operation immediately, a maximum of 12 values must be swapped.

Algorithm 3 is the pseudocode for the round function that uses the shuffled orders generated in Algorithm 2. AddRoundKey and SubBytes are byte-independent operations, and because the state array has already been shuffled and stored, sequential operations are performed in the order in which they are stored. ShiftRows is a one-byte dependent operation that operates regarding the order array L (line 8). In MixColumns, the operations from lines 11 to 13 are conducted. The operation on one column of MixColumns is a polynomial product with a fixed polynomial $m(x)$ using $x^4 + 1$ as the multiplied modulo over GF(2^8), which is given by

$$m(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}. \tag{3}$$

Formulating this in terms of the coefficient is as follows:

$$s'_i = \{02\} \cdot s_i \oplus \{03\} \cdot s_{(i+1) \bmod 4} \oplus s_{(i+2) \bmod 4} \oplus s_{(i+3) \bmod 4}. \tag{4}$$

This can be written as follows:

$$s'_i = \left(s_{(i+1) \bmod 4} \oplus s_{(i+2) \bmod 4} \oplus s_{(i+3) \bmod 4}\right) \oplus \{02\} \cdot \left(s_i \oplus s_{(i+1) \bmod 4}\right). \tag{5}$$

The codes from lines 11 to 13 of Algorithm 3 are the same as Equation (5). Here, $x$time means $\{02\} \cdot x$.

Figure 5 schematically shows Algorithm 3. The index of the state array is an example, and arrows in the schematic of ShiftRows and MixColumns change each time according to shuffling.

5. Demonstration

In this section, the safety of the novel shuffling countermeasure proposed in Section 4 is experimentally confirmed. First, the assembly code generated by the compiler was observed. Listing 2 is the assembly code of the SubBytes part of the novel countermeasure. Sbox operations are performed in the order stored in the state array without referring to the order array.

Figure 6 shows the power consumption trace of the novel countermeasure. Compared with Figure 1, the length of one round is approximately 2.1 times longer. In detail, the length is increased by applying the dummy operation and shuffling scheme to AddRoundKey, ShiftRows, and MixColumns, whereas the length of SubBytes is shortened by reducing the array reference once. The power consumption trace of the SubBytes part is shown in Figure 7. It is impossible to distinguish between the dummy and the real operation.

The traces of the novel countermeasure were also collected in the same environment as the traces of the previous countermeasure. The power consumption trace of the XMEGA128D4 chip was collected with ChipWhisperer-Pro (see Section 3). At each of the four optimization levels, 10,000 traces were collected with the variable key and 1,000 traces were collected with the fixed key. Additionally, the artificial neural network also used the same model used in the previous countermeasure (see Figure 3). The experiments were conducted using the same learning environment in Section 3. The learning graph at the optimization level -Os is shown in Figure 8. The learning graphs at the remaining optimization levels also have a similar shape to Figure 8. While the training loss decreases and the training accuracy rises, the validation loss increases again and the validation accuracy stops around 0.5. The test accuracy of the novel countermeasure at all optimization levels is shown in Table 2. The accuracy of 0.5 indicates that the neural network does not properly classify because it is a binary classification problem. Therefore, our novel countermeasure is safe at all optimization levels because attackers cannot distinguish between real and dummy operations.

The countermeasure of the study is not only safe but also effective. The cycles per byte of AES implementations are shown in Table 3. The implementation with the novel countermeasure takes approximately eight times more cycles per byte than the unprotected implementation, and approximately 1.8 times as long as the implementation with the previous countermeasure applied. The previous countermeasure used dummy operations for only the SubBytes function, but the novel countermeasure used dummy operations for all functions, so the overhead is inevitable. According to H. Kim et al. [24], when a masking countermeasure is applied to AES, the first and second masking takes approximately 1.7 times and 23.7 times more cycles per byte, respectively, compared to the nonprotected implementation. Considering this, the cost of adding dummy operations and applying shuffling to the entire encryption process, which is eight times more cycles, is tolerable.

6. Conclusions

In this study, the authors questioned the safety of the previous shuffling countermeasure. In a previous study [17], the authors designed the countermeasure to be safe against attackers using machine learning and performed experimental verification but were unable to fully confirm the countermeasure because the experimental environment was set to a noisy environment. As a result of reverification in the appropriate environment, it was confirmed that the previous countermeasure was not safe.

Previous countermeasures applied the hiding schemes only to the byte-independent operations of the cryptographic algorithm. The cause of the weakness of the previous countermeasure was analyzed, and a novel countermeasure was designed using the shuffling and random insertion of dummy operations schemes up to the byte-dependent operation of AES to avoid the weakness. Moreover, to confirm the safety of the proposed countermeasure, an experimental verification was conducted in an environment with as little noise as possible. As a result, even assuming a strong attacker who knows the indices of the dummy operations, the neural network has not learned to distinguish the dummy operations. Therefore, the proposed countermeasure is safe.

In future works, countermeasures for other block ciphers can be designed similarly to those designed for AES in this paper. Since the types of components used for each block cipher are different, a dedicated design for each is required. In this paper, an optimized design of the proposed countermeasure was not considered. Therefore, an optimized design for the full-hiding scheme is also required.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00903, Development of physical channel vulnerability-based attacks and its countermeasures for reliable on-device deep learning accelerator design).

References

[1] Cisco, *Cisco annual internet report (2018-2023)*, 2020, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internetreport/white-paper-c11-741490.html.

[2] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, vol. 31, Springer Science & Business Media, 2008.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*, pp. 388–397, Berlin, Heidelberg, 1999.

[4] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 16–29, Springer, Berlin, Heidelberg, 2004.

[5] C. Herbst, E. Oswald, and S. Mangard, "An AES smart card implementation resistant to power analysis attacks," in *International conference on applied cryptography and network security*, pp. 239–252, Berlin, Heidelberg, 2006.

[6] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 413–427, Springer, Berlin, Heidelberg, 2010.

[7] M. Rivain, E. Prouff, and J. Doget, "Higher-order masking and shuffling for software implementations of block ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 171–188, Springer, Berlin, Heidelberg, 2009.

[8] J. G. van Woudenberg, M. F. Witteman, and B. Bakker, "Improving differential power analysis by elastic alignment," in *Cryptographers' Track at the RSA Conference*, pp. 104–119, Berlin, Heidelberg, 2011.

[9] D. Strobel and C. Paar, "An efficient method for eliminating random delays in power traces of embedded software," in *International Conference on Information Security and Cryptology*, pp. 48–60, Berlin, Heidelberg, 2011.

[10] F. Durvaux, M. Renauld, F. X. Standaert, L. V. Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon, "Efficient removal of random delays from embedded software implementations using hidden Markov models," in *International Conference on Smart Card Research and Advanced Applications*, pp. 123–140, Berlin, Heidelberg, 2012.

[11] L. Wu and S. Picek, "Remove some noise: on pre-processing of side-channel measurements with autoencoders," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 4, pp. 389–415, 2020.

[12] Y.-S. Won, X. Hou, D. Jap, J. Breier, and S. Bhasin, "Back to the basics: seamless integration of side-channel pre-processing in deep neural networks," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3215–3227, 2021.

[13] D. Kwon, H. Kim, and S. Hong, "Non-profiled deep learning-based side-channel preprocessing with autoencoders," *IEEE Access*, vol. 9, pp. 57692–57703, 2021.

[14] H. Maghrebi, "Assessment of common side channel countermeasures with respect to deep learning based profiled attacks," in *2019 31st International Conference on Microelectronics (ICM)*, pp. 126–129, Cairo, Egypt, 2019.

[15] H. Maghrebi, *Deep Learning Based Side Channel Attacks in Practice*, Cryptology ePrint Archive, 2019.

[16] L. Masure, C. Dumas, and E. Prouff, "A comprehensive study of deep learning for side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 1, pp. 348–375, 2020.

[17] J. Lee and D.-G. Han, "DLDDO: deep learning to detect dummy operations," in *International Conference on Information Security Applications*, pp. 73–85, Cham, 2020.

[18] NewAE, *CW 1200: Chipwhisperer-pro*, NewAE Technology, 2021, https://media.newae.com/datasheets/NAE-CW1200_datasheet.pdf.

[19] J. Lee and D.-G. Han, "Security analysis on dummy based side-channel countermeasures–case study: AES with dummy and shuffling," *Applied Soft Computing*, vol. 93, p. 106352, 2020.

[20] I. Diop, P.-Y. Liardet, Y. Linge, and P. Maurine, "Collision based attacks in practice," in *2015 Euromicro Conference on Digital System Design*, pp. 367–374, Madeira, Portugal, 2015.

[21] Atmel, *AVR XMEGA D4 Devices Datasheet*, Atmel Corporation, 2021, http://ww1.microchip.com/downloads/en/

DeviceDoc/Atmel-8135-8-and-16-bit-AVRmicrocontroller-ATxmega16D4-32D4-64D4-128D4_datasheet.pdf.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on ImageNet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, Santiago, Chile, 2015.

[23] R. Durstenfeld, "Algorithm 235: random permutation," *Communications of the ACM*, vol. 7, no. 7, p. 420, 1964.

[24] H. Kim, S. Hong, and J. Lim, "A fast and provably secure higher order masking of AES S-Box," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 95–107, Springer, Berlin, Heidelberg, 2011.