A Masked White-box Cryptographic Implementation for Protecting against Differential Computation Analysis

Seungkwang Lee

Information Security Research Division, ETRI
skwang@etri.re.kr

Abstract. Recently, gray-box attacks on white-box cryptographic implementations have succeeded. These attacks are more efficient than white-box attacks because they can be performed without detailed knowledge of the target implementation. The success of the gray-box attack is reportedly due to the unbalanced encodings used to generate the white-box lookup table. In this paper, we propose a method to protect the gray-box attack against white-box implementations. The basic idea is to apply the masking technique before encoding intermediate values during the white-box lookup table generation. Because we do not require any random source in runtime, it is possible to perform efficient encryption and decryption using our method. The security and performance analysis shows that the proposed method can be a reliable and efficient countermeasure.

Keywords: White-box cryptography, power analysis, differential computation analysis, countermeasure.

1 Introduction

As personal devices become more diverse, the amount of data that needs to be protected has also increased. To protect this broad category of personal information, we use various encryption algorithms which are publicly known. For this reason, we should securely protect the secret key. The attack models that malicious attackers use to recover the secret key can be divided into three layers: the black-box, the gray-box, and the white-box models. As the color of the layer becomes brighter, the amount of information that the attacker can access increases. Attackers in the black-box model are given the in- and output for cryptographic primitives, but in the gray-box model they also utilize additional information leakage, i.e., side-channel information, such as timing or power consumption. As a representative example, Kocher *et al.* presented Differential Power Analysis (DPA) [18], a statistical analysis of power traces acquired during the execution of a target cryptographic primitive. In addition to all of these, attackers in the white-box model can access and modify all resources in the execution environment. Therefore, if the secret key used for the cryptographic primitive resides in memory without any protection, it may leak directly to the white-box attacker.

The white-box cryptographic implementation is intended to counter this white-box attack: the key idea behind is to embed the secret key in the implementation using precomputed lookup tables and apply linear and non-linear encodings so that it becomes difficult for a white-box attacker to extract the secret key [11] [12]. Although it is a strong point to hide the key in the software implementation, there are three main disadvantages that have been known so far. Since the table itself acts as a secret key, taking the table has the same meaning as taking the secret key. It is often called a code-lifting attack [38]. In this regard, many researchers have attempted to mitigate the code-lifting attack by significantly increasing the size of the lookup table [4] [6]. The serious problem is that spending up to 20-50GB of storage to cope with code-lifting attacks is too costly and at the same time impractical. Second, the use of lookup tables increases the memory requirement and slows down the execution speed compared to a non-white-box implementation of the same algorithm. Moreover, the size of the lookup table has increased considerably with the aforementioned anti-code-lifting technique. Finally, many white-box implementations have been practically broken by various attacks including key extraction, table-decomposition, and fault injection attacks [35]. The first two white-box implementations for DES [12] and AES [11] were shown to be vulnerable to differential cryptanalysis [15] [39] as well as algebraic cryptanalytic attacks [3] [23] [27]. Although several further variants of white-box implementations for DES and AES have been proposed [10] [40] [17] [21], many of them were broken [31] [32]. In addition to standard ciphers, research has also been conducted on various non-standard ciphers, commonly named dedicated white-box ciphers [4] [6] [28]. It is worth noting that these attacks have been performed in the white-box model requiring the details of the target implementation.

However, the white-box cryptography currently faces the most serious problem: the gray-box model attack on white-box implementations has succeeded. In other words, it is possible to reveal the secret key embedded in a white-box implementation using side-channel information without any detailed knowledge about it. In general, side-channel analysis, more specifically power analysis, is successful if the key hypothesis of the attacker is correct, since the intermediate value calculated from the correct hypothesis correlates to the power consumption value at a particular point in the power trace. The authors of [7] have developed plugins for dynamic binary instrumentation (DBI) tools including Pin [24] and Valgrind [33] to obtain software execution traces that contain information about the memory addresses being accessed. Their aptly named Differential Computation Analysis (DCA) is more effective because there is no measurement noise in software traces unlike power traces obtained using the oscilloscope in classical DPA. The main reason behind the success of DCA is due to the imbalances in linear and non-linear encodings used in the white-box implementation [36]. The authors of [7] have suggested several methods to counteract DCA including variable encodings [30], threshold implementations [34], splitting the input in multiple shares to different affine equivalence, and a masking scheme using the input data as a random source. Since DCA uses the memory address accesses available in the software traces, some obfuscation techniques including control flow obfuscation and table location randomization have been discussed.

Our contribution. After producing the software traces based on accessed addresses and data, DCA uses them to perform statistical analysis like DPA. Therefore, DCA protection is in line with defense against power analysis. This study is to present a masked white-box implementation for protecting against DCA as well as power analysis. Particularly, Boolean masking is applied during the lookup table generation unlike the existing masking techniques that are applied in runtime. In other words, we do not need any random source at runtime. As a result, the runtime overhead does not increase significantly. We begin by going over the initial white-box AES (WB-AES) [11] to demonstrate its vulnerability to DCA. We apply a masking technique to this vulnerable implementation, and present three variants of the implementation according to the level of security requirements. To evaluate the security of our proposed method, we perform DCA on the masked WB-AES implementation with 128-bit key, and use the Walsh transforms to analyze its security in more detail. The experimental results show that our proposed method effectively defends the attacks. Compared to the existing WB-AES implementation, the lookup table size increases approximately 1.56 to 9.59 times depending on the choice of the implementation variants and the number of lookups increases approximately 1.6 times.

Organization of the paper. The remainder of this paper is organized as follows: Section 2 provides an overview of white-box cryptography and its vulnerabilities to the gray-box attack. We propose a white-box implementation for protecting against DCA in Section 3. We introduce a masked WB-AES implementation and analyze its performance including the lookup table size. In Section 4, we demonstrate the security of our proposed method through DCA and the Walsh transforms. Section 5 concludes this paper.

2 Preliminaries

In this section, we introduce the basic concept of white-box cryptography and provide experimental results about its vulnerability to gray-box attacks.

2.1 Overview of White-box Cryptography

In most cases, a white-box implementation is simply a series of encoded lookup tables which replace individual computational steps of a cryptographic algorithm. Let us give a simple example. For a computational step $y = E_k(p)$, where y, p, $k \in \mathrm{GF}(2^8)$ and k is a small portion of the secret key, let $\mathcal{E}_k$ be an $8{\times}8$ lookup table to map p to y. The secret and invertible encodings are then applied to $\mathcal{E}$ in order to prevent a white-box attacker from recovering the secret key using the input and output values. Let us denote the encodings by G and F, for

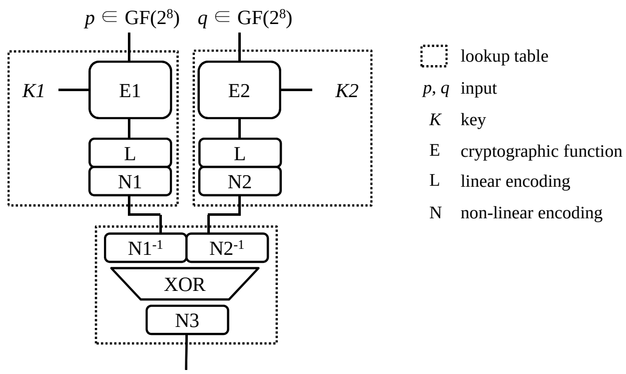


Fig. 1: Basic principle of existing white-box cryptographic implementations.

example. Then we have: $\mathcal{E}_k = G \circ E_k \circ F^{-1}$. It is important to remember that each encoding consists of linear and non-linear encodings.

Figure 1 shows a basic principle of existing white-box implementations for a simple cryptographic operation, $E1(p, K1) \oplus E2(q, K2)$. With the same linear encoding applied, the XOR lookup table can be simply generated without decoding the linear encoding. This is because the distributive property of multiplication over addition is satisfied if the same linear encoding is applied to the two lookup values.

2.2 Gray-box Attacks on White-box Cryptography

For a gray-box attacker, suppose the followings:

- The underlying cryptographic algorithm is known, for example AES.
- The details about the type of the implementation and its structure are unknown.
- There is no external encoding in the target implementation; the cryptographic operation seen by the attacker is standard AES encryption (or decryption).
- The attacker can collect power traces (software traces in the case of DCA) while it is operated.

We examine DCA on 20 instances of an unprotected WB-AES-128 implementation [11] under this gray-box attack model. To collect the software execution traces we have followed the approach presented in [7]. We have used Valgrind, a DBI framework, to trace each execution of the target implementation with a random plaintext and recorded all accessed addresses and data over time. Then, those values have been serialized into vectors of ones and zeros for a classical representation of power traces. For each target instance, we have collected 200 software traces with random plaintexts and performed mono-bit Correlation Power Analysis (CPA) [9] attacks, which is known to be more effective than

DPA, on the SubBytes output in the first round using Daredevil [8]. The result reports two top 10 lists:

- the **sum** of the correlation coefficients for 8 mono-bit CPA attacks for each subkey candidate
- the **highest** correlation coefficient among the mono-bit CPA results for all subkey candidates

If the subkey is ranked in the top at least one of the two lists, we assumed that it is revealed.

Table 1 shows one of the best cases for the attacker where all the subkeys are revealed, but this is not always happening. For DCA attacks with only 200 software traces on each of 20 instances of the unprotected WB-AES implementation, DCA recovered an average of 14.3 out of 16 subkeys and the standard deviation (S.D) was 2.17. Recovering the small number of missing subkeys is trivial using brute-force attacks. The attack success rate was about 89% (286/320), and the highest value average of the mono-bit CPA correlation coefficient for the correct subkey was 0.557 (S.D = 0.173). In the presence of such correlation to the key, both attack success rate and correlation coefficients can become higher if the number of traces provided to DCA is more than 200.

CPA attacks with the Hamming Weight (HW) model are based on the fact that the power consumption of the target device at any given point in time is proportional or inversely proportional to the HW of the intermediate value. But as shown in Table 1, even in this best case, nearly half of the target bits for each subkey do not show a correlation. For this reason, CPA with the HW model is not used to attack the white-box cryptographic implementation. The multi-bit based CPA also depends on the value of a particular bit set to predict the power consumption, and thus is hardly successful for the same reason. The mono-bit DPA divides the traces based on the target intermediate bit and calculates the difference between the two sets. If the two sets are divided based on the correct hypothetical key, a noticeable spike appears at the target operation point in the differential trace. In the same way, multi-bit DPA divides the two sets based on the HW of the target intermediate value. The important point over here is that there is no fixed set of intermediate bits that always shows the correlation to the key due to the linear and non-linear encodings of the white-box implementation. For this reason, the white-box implementation is being attacked by mono-bit analysis.

For an in-depth understanding where and how key leaks occur, we conducted additional experiments using SCARF [37] [20] [22]. Instead of collecting power traces using an osilloscope, we also collected 200 software traces which serialize the target intermediate value into vectors of ones and zeros, and mounted mono-bit CPA using the SubBytes output in the first round. The highest peak in the correlation plot shown in Figure 2 was found at the point where the SubBytes output multiplied by 01 was looked up. As will be discussed later, this white-box implementation contains table lookups for MixColumns, where the SubBytes outputs multiplied by 01 are frequently looked up. Those are the point of interest for this attack.

Table 1: DCA ranking for the target WB-AES implementation [11] when conducting mono-bit CPA on the SubBytes output in the first round with 200 software traces.

TargetBit \ SubKey	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	183	219	1	1	213	1	1	1	213	186	229	1	81	1	1
2	1	1	1	1	87	1	1	1	209	1	1	1	1	1	1	1
3	17	66	83	46	41	146	151	172	159	34	203	1	1	252	242	205
4	1	1	99	225	1	1	249	131	1	1	118	193	1	199	174	223
5	141	1	1	174	106	1	1	144	205	1	1	68	171	1	1	25
6	256	9	177	194	140	1	182	13	201	1	222	54	155	1	69	150
7	83	212	1	184	78	246	25	181	60	195	196	117	63	65	134	155
8	1	232	204	1	1	249	183	27	1	211	103	95	1	176	230	17
sum	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
highest	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

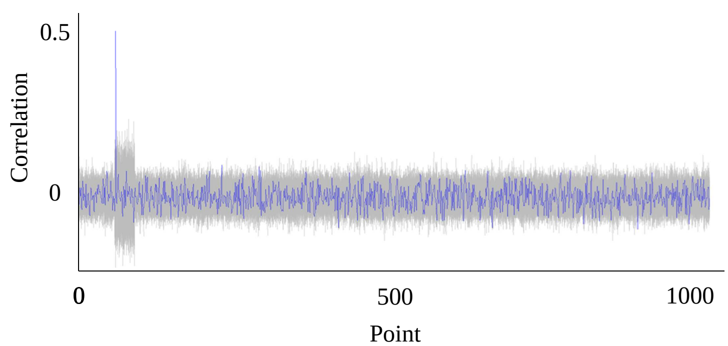


Fig. 2: A peak in the CPA result when attacking the SubBytes output in the first round. Blue line: correct key hypothesis, gray line: wrong key hypothesis.

Sasdrich *et al.* [36] have indicated that the main reason behind successful DCA and CPA attacks is largely due to the high imbalance in encoding used to generate white-box lookup tables. Based on their definitions below, we demonstrate the imbalance in the encoding used for the same lookup table that was attacked above.

Definition 1. *Let* $x = \langle x_1, \ldots, x_n \rangle$, $\omega = \langle \omega_1, \ldots, \omega_n \rangle$ *be elements of* $\{0,1\}^n$ *and* $x \cdot \omega = x_1\omega_1 \oplus \ldots \oplus x_n\omega_n$. *Let* $f(x)$ *be a Boolean function of* n *variables. Then the Walsh transform of the function* $f(x)$ *is a real valued function over* $\{0,1\}^n$ *that can be defined as* $W_f(\omega) = \Sigma_{x \in \{0,1\}^n} (-1)^{f(x) \oplus x \cdot \omega}$.

Definition 2. *Iff the Walsh transform* W_f *of a Boolean function* $f(x_1, \ldots, x_n)$ *satisfies* $W_f(\omega) = 0$, *for* $0 \leq HW(\omega) \leq m$, *it is called a balanced* m^{th} *order* *correlation immune function or an m-resilient function.*

By *Definition 1*, the larger the absolute value of $W_f(\omega)$ is, the stronger the correlation between $f(x)$ and $x \cdot \omega$ is. Let us denote the output of SubBytes by x, and the combination with MixColumns, linear and non-linear encodings by 32 Boolean functions $f_{i\in\{1,\ldots,32\}}(x)$: $\{0,1\}^8 \rightarrow \{0,1\}$. Here we expect that $f(x)$ will have the greatest correlation when x is derived from the correct key candidate. For all key candidates k^* and for all ω we calculated the Walsh transforms W_{f_i} and summed up all the imbalances for each key candidate as follows:

$$\Delta^f_{k\in\{0,1\}^8} = \sum_{\forall\omega\in\{0,1\}^8} \sum_{i=1,\ldots,32} |W_{f_i}(\omega)|; k^* = k.$$

Then this gives us as shown in Figure 3 that Δ^f_k of the correct key candidate (*0x*88, 136) is obviously distinguishable from that of other key candidates. This indicates that $f_i(x)$ and $x \cdot \omega$ are best correlated with the correct key *0x*88. In this way the Walsh transforms can be used to calculate the correlation between key-dependent lookup values and hypothetical values.

3 Proposed Method

As aforementioned, the vulnerability to DCA of the previous white-box implementation is due to the imbalanced encoding. Our goal is to reduce the correlation to the key at the intermediate values before encoding them in the process of generating the white-box lookup table. To achieve this, we apply masking with a balanced distribution at the key-sensitive intermediate value. Originally, the masking techniques [1] [5] [13] [26] have been used to force the power consumption signals to be uncorrelated with the secret key and the input and output. We apply this technique, in particular Boolean masking, during the lookup table generation. Before going into more depth, we provide a key idea.

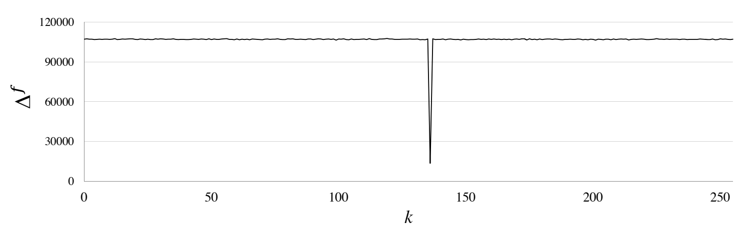


Fig. 3: Sum of all imbalances Δ^f_k for all key candidates of the previous WB-AES implementation.

3.1 Key Idea Behind

Figure 4 shows an example of the proposed method applied to $E1(p, K1) \oplus E2(q, K2)$. The key idea behind is to apply masking before encoding the outputs of $E1$ and $E2$ while generating lookup tables. Let us denote the lookup tables for $E1$ and $E2$ by $\mathcal{E}1$ and $\mathcal{E}2$, respectively. An example of $\mathcal{E}1$-generating code might look like this:

for $p = 0$ to 255 do
    pick random $m \in \{0, 1\}^8$
    $y \leftarrow E1(p,\ K1) \oplus m$
    $\mathcal{E}1[0][p] \leftarrow \mathrm{N1}(\mathrm{L}(m))$
    $\mathcal{E}1[1][p] \leftarrow \mathrm{N2}(\mathrm{L}(y))$,

where the input p is not assumed to be encoded. The most important point over here is that the mask should be selected uniformly at random, so 256 different masks are used to generate $\mathcal{E}1$ (or $\mathcal{E}2$). Encoding the used masks, in particular with the same linear transformation, not only protects them but also makes it easy to unmask through the XOR operations by the distributive property of multiplication over addition. The lookup values for an input p (resp. q) to $\mathcal{E}1$ (resp. $\mathcal{E}2$) are the following two values: an encoded key-sensitive intermediate value which is masked, and an encoded mask. To cancel out the masks, they are XORed by the following XOR lookup tables as shown in Figure 4. The order of the XOR table lookups has to be kept for the complete unmasking. We implement a WB-AES implementation with 128-bit key using this principle.

3.2 White-box AES Implementation

Since we protect a particular part of the implementation presented in [11] [29] we focus on the protected part and briefly describe the rest. With AES-128 written below, AddRoundKey, SubBytes, and part of MixColumns are combined into a series of lookup tables. In the following, we use k^r for the 4×4 round key matrix at round r, lowering indices $_{i,j}$ for the current byte position in the round key matrix, and use $\boldsymbol{k}^r_{i,j}$ to indicate that the ShiftRows is applied to $k^r_{i,j}$, where i denotes the row index and j the column index.

state $\leftarrow$ *plaintext*
for $r = 1$ to 9 do
    ShiftRows(state)
    AddRoundKey(state, $\boldsymbol{k}^{r-1}$)
    SubBytes(state)
    MixColumns(state)
ShiftRows(state)
AddRoundKey (state, $\boldsymbol{k}^9$)
SubBytes(state)
AddRoundKey(state, k^{10})

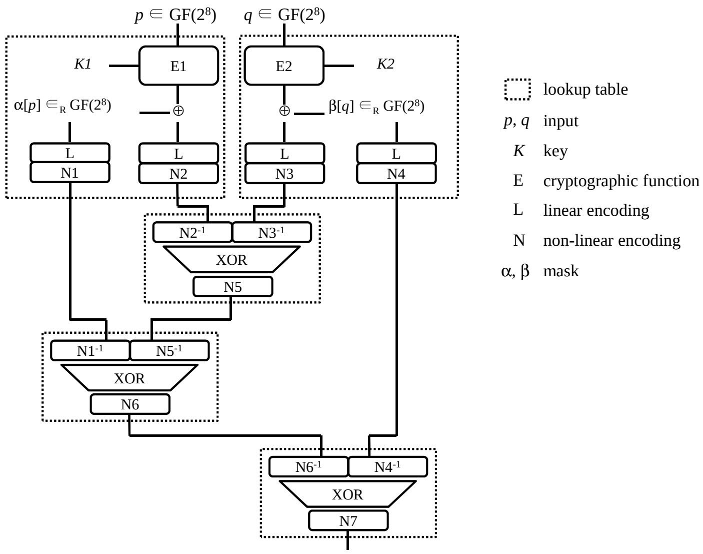


Fig. 4: Basic principle of the proposed white-box cryptographic implementation.

$\quad$ *ciphertext* $\leftarrow$ state

At first, *T-boxes* which is a set of 160 8×8 lookup tables combines AddRoundKey and SubBytes as follows:

$$\begin{aligned} T^{r}_{i,j}(p) &= S(p \oplus \boldsymbol{k}^{r-1}_{i,j}), \qquad \text{for } 0 \le i, j \le 3, \text{ and } 1 \le r \le 9, \\ T^{10}_{i,j}(p) &= S(p \oplus \boldsymbol{k}^{9}_{i,j}) \oplus k^{10}_{i,j}, \text{ for } 0 \le i, j \le 3. \end{aligned}$$

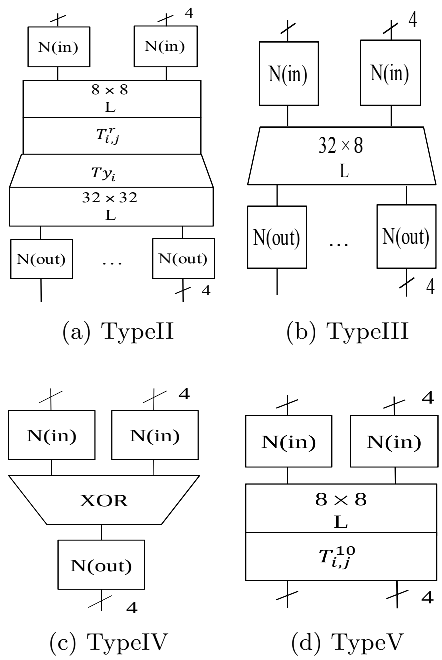
Let us denote (x_0, x_1, x_2, x_3) a column of four bytes to be multiplied with the MixColumns matrix. That multiplication is then decomposed as follows:

$$\begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} =$$

$$x_0 \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \oplus x_1 \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \oplus x_2 \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \oplus x_3 \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix}.$$

For the right-hand side (say y_0, y_1, y_2, y_3), the commonly named Ty_i tables are defined as follows:

$$\begin{aligned} Ty_0(x) &= x \cdot [02\ 01\ 01\ 03]^T \\ Ty_1(x) &= x \cdot [03\ 02\ 01\ 01]^T \\ Ty_2(x) &= x \cdot [01\ 03\ 02\ 01]^T \\ Ty_3(x) &= x \cdot [01\ 01\ 03\ 02]^T. \end{aligned}$$



(a) TypeII (b) TypeIII

(c) TypeIV (d) TypeV

Fig. 5: TypeII, III, IV and TypeV tables in the unprotected WB-AES implementation [11]. In/outputs at the same level have the same length.

The 32-bit result of $y_0 \oplus y_1 \oplus y_2 \oplus y_3$ can be computed via the XOR table lookups. An XOR lookup table takes two 4-bit inputs and maps them to their XORed value, so the XOR operation of two 32-bit values is performed using 8 copies of XOR lookup tables. Because the MixColumns result requires twelve 32-bit XORs for each round, the previous WB-AES implementation includes 96 copies of the XOR lookup tables per round, a total of 864 copies.

Figure 5 simply illustrates the commonly named TypeII, TypeIII, TypeIV and TypeV tables. TypeII is generated from the composition of *T-boxes* and Ty_i, and TypeIII cancels the effect of linear transformation applied to TypeII outputs and takes care of the inversion of linear transformation applied to TypeII inputs of the next round. To avoid the huge size of TypeIII tables, the 32×32 decoding matrix for the inversion is divided into four submatrices. In addition, TypeIV which is a set of the XOR lookup tables is looked up to combine intermediate values of TypeII and TypeIII. The tables for combining the lookup values of TypeII and TypeIII are named TypeIV_II and TypeIV_III, respectively [11]. Finally, the lookup table for the final round, say TypeV, is generated from T^{10} without Ty_i because MixColumns is not included in the final round (TypeI which is related to external encoding is not discussed in this paper).

3.3 Masked White-box AES Implementation

In the proposed method, we mainly protect three key-dependent values. First, the MixColumns output must be protected because it can not be secured solely by the problematic encodings. As demonstrated previously, each subkey can be easily revealed by performing DCA with 2^8 guesses. Second, the last round input, that is each subbyte input to the last round, can be a target intermediate value if an attacker knows $\boldsymbol{k}^9_{i,j}$ and $k^{10}_{i,j}$, which require 2^{16} guesses, because the inverse S-box is known. Third, each subbyte of the second round input can also be a target intermediate value if an attacker is able to guess 2^{32} subkey candidates. Therefore, we basically apply random masks to the MixColumns outputs before encoding them, and replace the 4-bit random bijections used in the non-linear encodings with the 8-bit random bijections for the second and the final round inputs, depending on the security requirement. We note that the non-linear encodings for each subbyte of the round output in the previous unprotected white-box implementation consist of two concatenated 4-bit random bijections, but they could not hide the correlation to the key. This is because when a non-linear encoding is performed on a given subbyte, the upper 4-bit bijections can not influence the lower 4-bit bijections at all. In other words, if one of the upper 4 bits is changed but the lower 4 bits are the same, the lower 4 bits after the two concatenated 4-bit bijections are not affected by the upper 4 bits. Because of this fact, the two concatenated 4-bit random bijections could not be 8-bit random bijections. Therefore, it is decided to perform non-linear encodings by 8-bit random bijections at the attack point although the size of the XOR lookup table increases. This gives us the following three cases of the proposed implementations depending on the security requirements.

- CASE 1: Applying the masking technique to the intermediate values before encoding them
- CASE 2: And applying the non-linear encoding of the 8-bit random bijections at the 9^{th} round output
- CASE 3: And applying the non-linear encoding of the 8-bit random bijections at the 1^{st} round output.

CASE 1. In CASE 1 [25], we mainly protect the output of Ty_i; recall that the linear and non-linear encodings were directly applied to it. Let (z_0, z_1, z_2, z_3) denote the four-byte output of Ty_i. Each byte of them is to be masked using M defined in Algorithm 1. The used masks are also encoded and stored in our protected lookup table named TypeII-M (Masked) as illustrated in Figure 6. As pointed out earlier, the linear encoding applied to ($\hat{z}_0$, $\hat{z}_1$, $\hat{z}_2$, $\hat{z}_3$) and the masks has to be the same, so that the unmasking can be performed by the XOR table lookups. We generate TypeIV_IIA and TypeIV_IIB tables to perform the XOR operations on the masked values and unmask them, respectively, as shown in Figure 7. To be specific, TypeIV_IIA consists of 864 (=9×96) copies of TypeIV, but TypeIV_IIB contains 1152 (=9×128) copies.
As we know that

$$T^{10}_{i,j}(p) = S(p \oplus \boldsymbol{k}^9_{i,j}) \oplus k^{10}_{i,j}, \text{for } 0 \leq i, j \leq 3,$$

Algorithm 1 Masking function M

procedure M(z) ▷ Choose a random mask and apply it to z
  $m \in_R \{0,1\}^8$
  $\hat{z} \leftarrow z \oplus m$
  return $(\hat{z}, m)$ ▷ masked z and the mask used

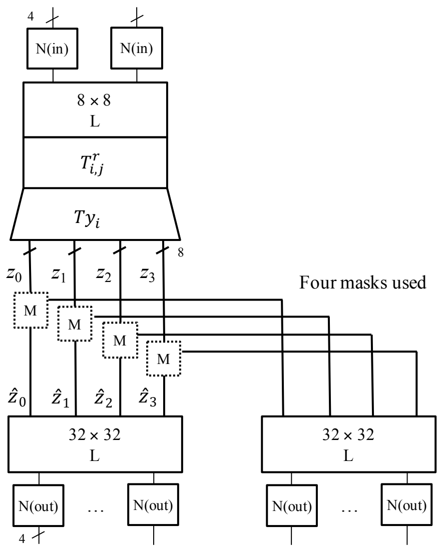


Fig. 6: TypeII-M tables in our WB-AES implementation.

each subbyte of the 9^{th} round output can be attacked if an attacker can guess two subkeys ($\boldsymbol{k}^9_{i,j}$ and $k^{10}_{i,j}$) of the final round. This is because there is no MixColumns in the final round, and is not impossible due to the fact that the encoding to protect the round output is imbalanced. Let S^{-1} be the inverse S-box. Then we know

$$S^{-1}(T^{10}_{i,j}(p) \oplus k^{10}_{i,j}) \oplus \boldsymbol{k}^9_{i,j} = p, \text{for } 0 \leq i, j \leq 3.$$

There are two points to keep in mind. First, the $T^{10}_{i,j}(p)$ output is not encoded because we assumed that there is no external encoding, and thus this becomes a subbyte of the ciphertext. Second, p of $T^{10}_{i,j}(p)$ is equal to a decoded subbyte of the 9^{th} round output, which is a decoded input to TypeV. The crucial observation over here is that there will be a correlation between p and the corresponding subbyte of the 9^{th} round output owing to the encoding imbalance. To demonstrate an attack based on this fact, we did a simple experiment using the Walsh transform. Suppose that $\boldsymbol{k}^9_{0,0}$ is known and we want to see if the sum of all imbalances for $k^{10}_{0,0}$ will produce a noticeable peak like in the case of Figure 3. So given $T^{10}_{0,0}(p)$, the first subbyte of the ciphertext, we define Δ^g_k in a similar way

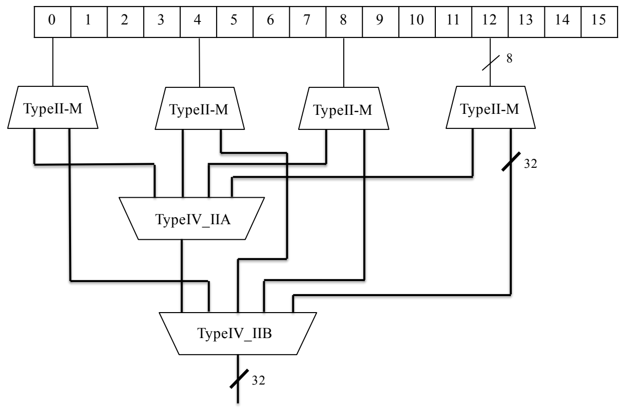


Fig. 7: TypeII-M and TypeIV_II tables (ShiftRows omitted).

to Section 2.2:

$$\Delta^g_{k\in\{0,1\}^8} = \sum_{\forall \omega \in \{0,1\}^8} \sum_{i=1,\ldots,8} |W_{g_i}(\omega)|,$$

where 8 Boolean functions $g_{i\in\{1,\ldots,8\}}(p)$: $\{0,1\}^8 \to \{0,1\}$ provide each bit of the first subbyte of the 9^{th} round output (TypeV input). In other words, $g(p)$ is the encoded p to the last round lookup table, TypeV. As a result, Figure 8 shows Δ^g_k that distinguishes the correct subkey ($k^{10}_{0,0} = \mathit{0x}36, 54$) from other candidates.

CASE 2 & CASE 3. To increase the security level in relation to this vulnerability, the CASE 2 and CASE 3 implementations require that the non-linear encoding be 8-bit random bijections, instead of 4-bit bijections, at the boundary between the 9^{th} and the final rounds. By doing so, there will be the similar effect as if masking is applied to each subbyte of the 9^{th} round output. TypeIV_IIC is defined for this purpose and shown in Figure 9a. This taskes two bytes as input: one comes from TypeIV_IIB and the other comes from TypeII-M as shown in Figure 10. In a nutshell, TypeIV_IIA combines the masked Ty_i intermediate

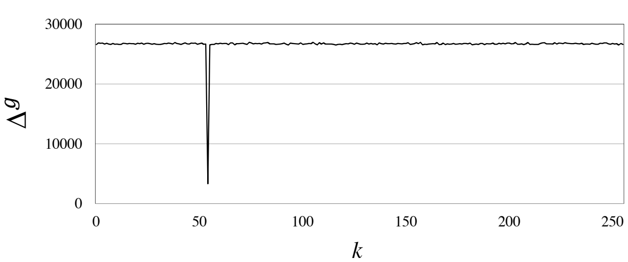
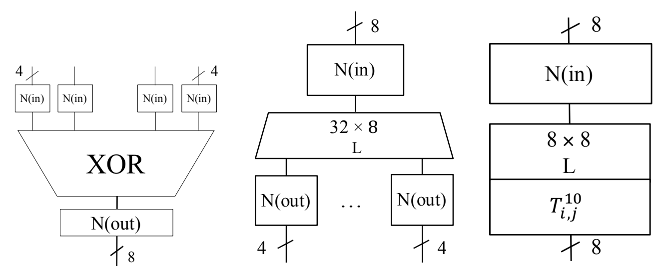


Fig. 8: Sum of all imbalances Δ^g_k at the TypeV input for all key candidates.

values, and TypeIV_IIB combines the TypeIV_IIA lookup value and the masks. Then, TypeIV_IIC combines the TypeIV_IIB lookup value and the remaining mask, and its lookup values are protected particularly by using the 8-bit random bijections. Thus, TypeIII in the 9^{th} round, named TypeIII-N (8-bit Non-linear encoding) shown in Figure 9b must be generated with the corresponding 8-bit bijections for the input decoding. While TypeIV_IIIA is the same type with TypeIV, TypeIV_IIIB is generated with the 8-bit random bijections for the round output like in the case of TypeIV_IIC. TypeIII-N and TypeIV_III are illustrated in Figure 11. Since the 8-bit bijections are applied to each subbyte of the 9^{th} round output, the decoding for this must also be 8-bit bijections. The lookup table for the final round in the CASE 2 and CASE 3 implementations is then defined as TypeV-N (8-bit Non-linear encoding) as illustrated in Figure 9c.



(a) TypeIV_IIC & IV_IIIB (b) TypeIII-N (c) TypeV-N

Fig. 9: Added tables in the CASE 2 and CASE 3 implementations.

CASE 3. The last vulnerability we want to deal with is that the CPA attack on the second round input using the 32-bit key hypothesis is computationally expensive but theoretically feasible because the proposed masking method does not provide protection at this point. To solve this problem, the first round in CASE 3 is also implemented like in the case of the protected 9^{th} round. Therefore, the decoding of the input to the 2^{nd} round lookup table has to use the 8-bit bijections. This second round lookup table is named TypeII-MN (Masked and 8-bit Non-linear encoding) and illustrated in Figure 12. Table 2 summarizes the bijection length for each table added in our implementation and Figure 13 provides table lookup sequences for each case.

3.4 Size and Performance

Lookup table size. We now have a masked white-box implementation of AES-128. With the external encoding excluded, the total table size of the unprotected implementation [11] is computed as follows:

- TypeII : $9 \times 4 \times 4 \times 256 \times 4 = 147{,}456$ bytes.

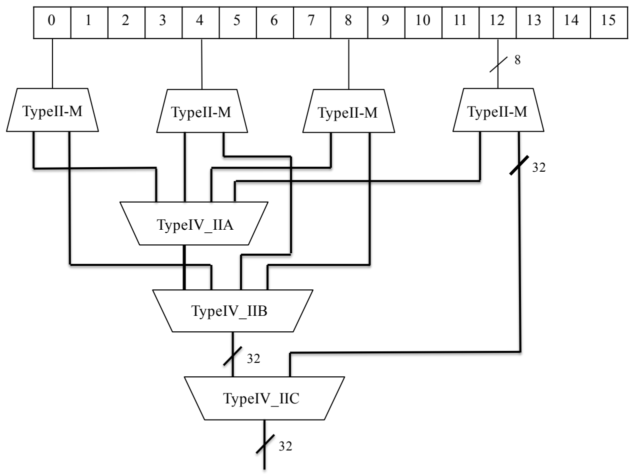


Fig. 10: TypeII-M and TypeIV_II tables in the 9^{th} round of the CASE 2 and CASE 3 implementations.

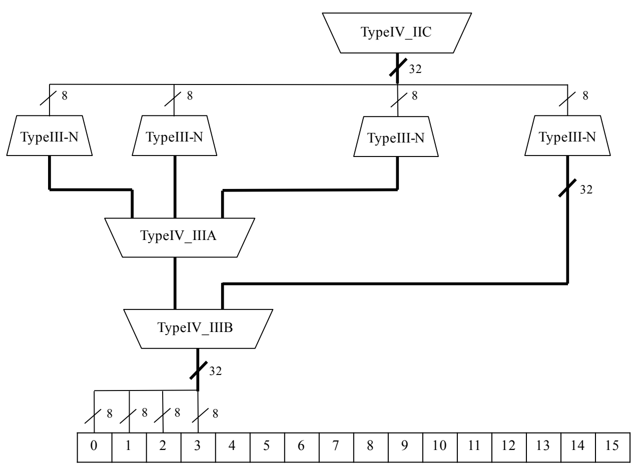


Fig. 11: TypeIII-N and TypeIV_III in the 9^{th} round of the CASE 2 and CASE 3 implementations.

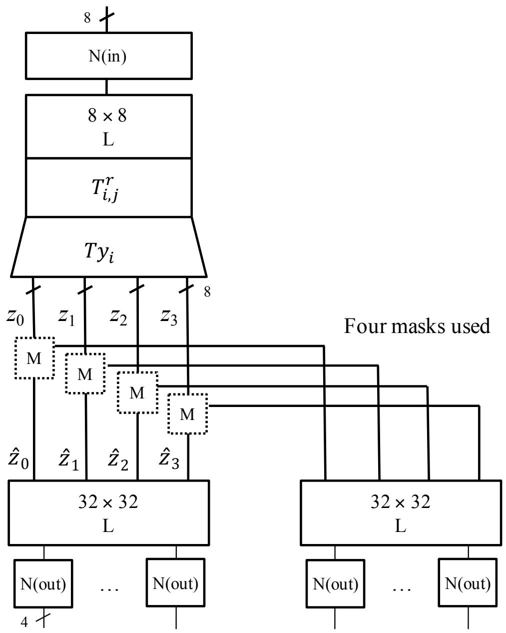


Fig. 12: TypeII-MN in the second round of the CASE 3 implementation.

- TypeIV_II : 9×4×4×3×2×128 = 110,592 bytes.
- TypeIII : 147,456 bytes.
- TypeIV_III : 110,592 bytes.
- TypeV : 4×4×256 = 4,096 bytes.

Thus their total size is 520,192 bytes.
In CASE 1, TypeIII, TypeIV_III, TypeV are the same with the unprotected implementation, but the sizes of TypeII-M and TypeIV_II (TypeIV_IIA + TypeIV_IIB) are

- TypeII-M : 9×4×4×256×2×4 = 294,912 bytes.
- TypeIV_II : 9×4×4×(3×2×128 + 4×2×128) = 258,048 bytes.

Then, the total size of the lookup tables is 815,104 bytes. In comparison, the lookup table size increases 1.56 times.
In CASE 2, the main differences to CASE 1 are the TypeIV_II and TypeIV_III structures in the 9^{th} round. Specifically, the sizes of TypeIV_IIA and TypeIV_IIB in the 9^{th} round are 12,288 (=4×4×3×2×128) bytes, while the size of Type_IIC is 1,048,576 bytes (=4×4×65536, 1MB). In addition, the TypeIV_IIIA size in the 9^{th} round becomes 8,192 (=4×4×2×2×128) bytes, and the TypeIV_IIIB size becomes 1MB. It is important to notice that the TypeV-N size is the same as TypeV. Then the total size is 2,904,064 bytes and increases by 5.58 times compared to the unprotected implementation as follows.

Table 2: Added tables in our implementation.

Type	Bijection length (bit)		CASE
	Input decoding	Output encoding	
TypeII-M	4	4	1,2,3
TypeII-MN	8	4	3
TypeIV_IIA & B	4	4	1,2,3
TypeIV_IIC	4	8	2,3
TypeIII-N	8	4	2,3
TypeIV_IIIA	4	4	2,3
TypeIV_IIIB	4	8	2,3
TypeV-N	8	·	2,3

- TypeII-M : 9×4×4×256×2×4 = 294,912 bytes.
- TypeIV_II : 8×4×4×(3×2×128 + 4×2×128) + 4×4×(2×3×2×128 + 256×256) = 1,302,528 bytes.
- TypeIII + TypeIII-N : 147,456 bytes.
- TypeIV_III : 8×4×4×3×2×128 + 1,056,768 = 1,155,072 bytes.
- TypeV-N : 4×4×256 = 4,096 bytes.

In CASE 3, the protection technique with the 8-bit random bijections used at the boundary of the 9^{th} and the final round of CASE 2 is also applied in the first round. This gives us the following, and the total size is 4,993,024 bytes that is 9.59 times larger than the unprotected implementation.

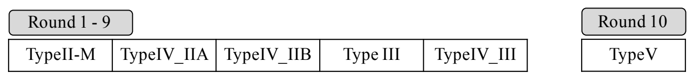
- TypeII-M + TypeII-MN : 9×4×4×256×2×4 = 294,912 bytes.
- TypeIV_II : 7×4×4×(3×2×128 + 4×2×128) + 2×(4×4×(2×3×2×128 + 256×256)) = 2,347,008 bytes.
- TypeIII + TypeIII-N : 147,456 bytes.
- TypeIV_III : 7×4×4×3×2×128 + 2,113,536 = 2,199,552 bytes.
- TypeV-N : 4×4×256 = 4,096 bytes.

The number of lookups. Since most of operations are table lookups except for ShiftRows, we compare the number of lookups. During each execution, the lookups for each table in the unprotected WB-AES implementation are counted as follows.

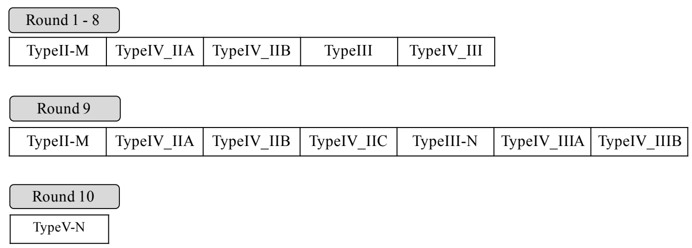
- TypeII : 9×4×4 = 144.
- TypeIV_II : 9×4×4×3×2 = 864.
- TypeIII : 9×4×4 = 144.
- TypeIV_III : 9×4×4×3×2 = 864.
- TypeV : 4×4 = 16.

Then, there are 2,032 lookups in total. Compared to this, the only differences in CASE 1 are

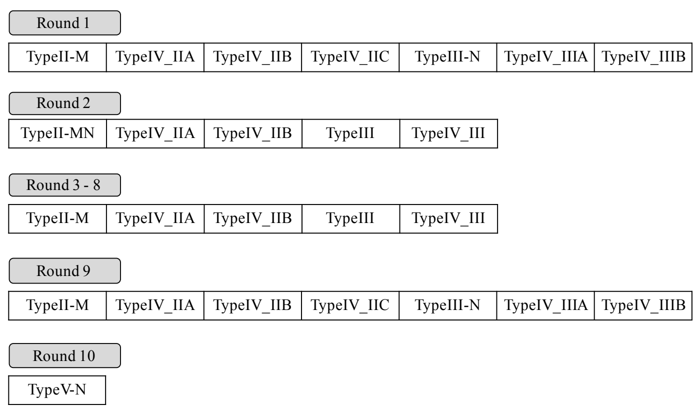
- TypeII-M : 9×4×4×2 = 288.
- TypeIV_II : 9×4×4×(3×2 + 4×2) = 2016.



(a) CASE 1



(b) CASE 2



(c) CASE 3

Fig. 13: Table lookup sequences in the CASE 1-3 implementations.

Then 3,328 lookups are performed during each execution of our masked WB-AES implementation, and thus the number of table lookups increases by 1.63 times compared to the unprotected one. The numbers of lookups in CASE 2 and CASE 3 are 3,296 and 3,264, respectively. These are 1.62 and 1.6 times, respectively, compared to the unprotected implementation. We can find that the number of table lookups decrease as the protection is enhanced because the number of the 8-bit unit XOR table lookup increases. Therefore, it is necessary to make a careful choice according to the security requirement level and available resources in the device to which this countermeasure is applied.

Figure 14 shows the memory accesses performed by our CASE 1 implementation on the stack. One can see repeated memory access patterns from round 1 to round 9. In the final round, memory access is relatively small due to the absence of MixColumns.

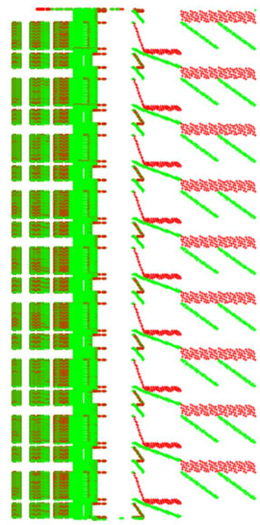


Fig. 14: Visualization of a software execution trace of our WB-AES implementation. Green: addresses of memory locations being read, Red: being written.

4 Security Analysis and Experimental Results

In this section, we begin with the problematic encoding, which is composed of invertible linear transformations and two concatenated 4-bit random bijections. Let us denote that encoding by λ, and let $\delta = \Pr[y_i = \lambda(y)_j]$, given i, j in the range of 0 to 7, and for all $y \in \mathrm{GF}(2^8)$, where $\lambda(y)$: $\{0,1\}^8 \rightarrow \{0,1\}^8$ and y_i means the i^{th} bit of y. Based on the fact that DCA and power analysis on the previous white-box implementations are possible due to the imbalance in λ, we can conclude that δ is noticeably greater (or less) than 1/2 enough to cause the correlation between y_i and $\lambda(y)_j$. If $\delta = 0$ as an extreme example, then y_i and $\lambda(y)_j$ are negatively correlated to each other. To make y_i uncorrelated to a particular bit of the λ output, our proposed method applies random masks for each value of $y \in \mathrm{GF}(2^8)$. If the mask is picked uniformly at random, then 256 masks are applied. What is important over here is that if the mask m is random, $(m \oplus y)$ and its j^{th} bit are also random. This gives us the following observation that

$$\hat{\delta} = \Pr[y_i = \lambda(y \oplus m)_j] = 1/2,$$

where m is random for each y. Consequently, y_i and $\lambda(y \oplus m)_j$ will be uncorrelated to each other, and we can conclude that mono-bit CPA attacks are hardly to succeed.

In terms of the Walsh transforms of *Definition 1*, by randomly masking the key-intermediate values before encoding them, we have

$$\begin{aligned} W_{f_i}(\omega) = &\ \Sigma_{x\in\{0,1\}^8}(-1)^{f_i(x\oplus m\in_R\{0,1\}^8)\oplus x\cdot\omega} \\ &\Leftrightarrow \Sigma_{x\in\{0,1\}^8}(-1)^{f_i(m'\in_R\{0,1\}^8)\oplus x\cdot\omega}, \end{aligned}$$

and this gives us

$$\Delta^f_{CorrectSubKey} \approx \Delta^f_{WrongSubKey},$$

because m is picked uniformly at random for each x and thus $f(m' \in_R \{0,1\}^8)$ will not correlate to $x \cdot \omega$. Figure 15 shows the sum of imbalances Δ^f_k at the TypeII-M lookup values and now we can see there is no distinguishable peak for the correct subkey.

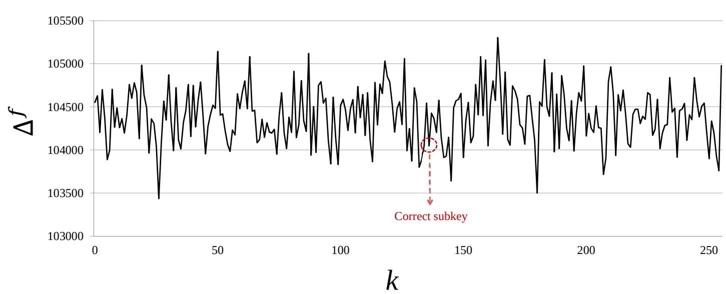


Fig. 15: Sum of all imbalances Δ^f_k at the TypeII-M output in the CASE 1 implementation.

In the CASE 2 and CASE 3 implementations, the non-linear encoding for each subbyte of the 1^{st} and 9^{th} round outputs is performed with the 8-bit random bijections. Thus, $g(p)$ now involves non-linear encodings by 8-bit random bijections instead of two concatenated 4-bit ones. To show that

$$\Delta^g_{CorrectSubKey} \approx \Delta^g_{WrongSubKey},$$

we calculated Δ^g_k for the TypeV-N input and Figure 16 shows that there is no spike at the correct subkey. This is because 8-bit random bijections are used to eliminate correlation before and after non-linear encodings.
One might choose random masks with the HW of 4, but in this case the number of masks is reduced compared to using a full range of masks. There are mainly two reasons why a CPA attack based on a model other than the mono-bit model is unlikely to be achieved on our proposed implementation. As aforementioned, the randomly chosen encoding randomly changes the HW and the bit-to-bit

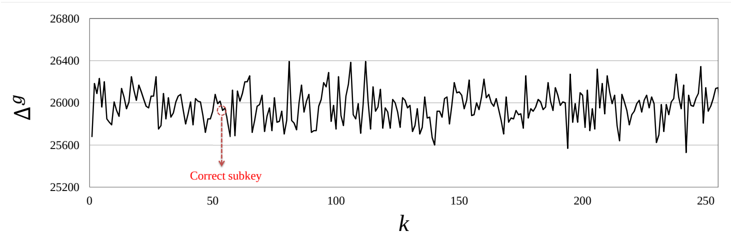


Fig. 16: Sum of all imbalances Δ_k^g at the TypeV-N input in the CASE 2 and CASE 3 implementations.

correlation before and after the encoding also varies from case to case. Furthermore, masking before the encoding makes the HW of the encoded output more unpredictable. Then we have:

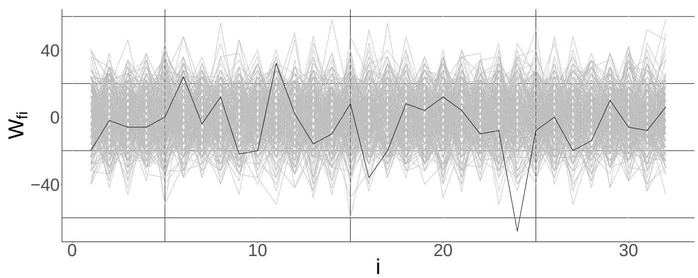
$$\hat{\delta} = \Pr[\mathrm{HW}(y) = \mathrm{HW}(\lambda(y \oplus m))] = 1/9,$$

which makes CPA based on the HW-model unsuccessful. For this reason, we focus on mono-bit CPA in the following experiments.

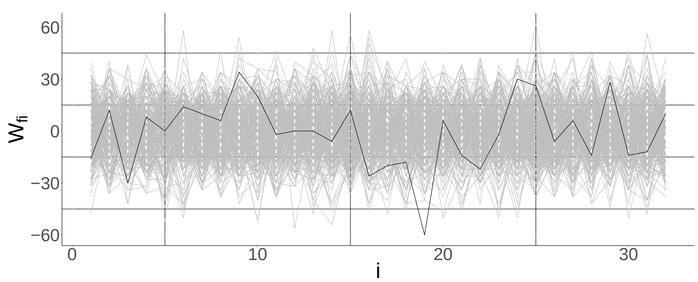
DCA and Results. We have generated 20 target instances of our CASE 1 implementation to be attacked by DCA. For more accurate attacks, 10,000 software traces were generated with random plaintexts for each target instance. DCA was performed with mono-bit CPA attacks on the SubBytes output in the first round. The entire first round was observed in order to check whether the key is leaked in the masked values or in the process of unmasking them. We note that the expected number of successful guessing subkeys is 1.25 (= 320/256), and the probability of the successful attack is 0.39%. Let's call our 20 attacks ATK #1, . . . , ATK #20. Consequently, 4 out of 320 correct subkeys were ranked at the top in at least one list (sum or highest) as shown in Table 3. (All DCA ranking tables can be found in [19].) However, the following analysis shows that the revealed subkeys were accidentally found. The main reasons for this conclusion are that two key leakages occurred at the points where the SubBytes outputs multiplied by 02 were looked up, and the other two occurred at the points where the encoded masks were looked up.
We provide the full DCA rankings of ATK #12, ATK #14, ATK #17, and ATK #18 in Table 4, 5, 6 and 7, respectively, in order to find out which target bit of the hypothetical value correlates to the key. It is important to notice again that the mono-bit CPA ranking itself does not determine the subkey, but is determined based on the sum of the correlation coefficients for each target bit or the highest correlation coefficient. We determined ω based on the correlated target bits for the leaked subkeys, and calculated their Walsh transforms.
For ATK #12 and ATK #17, let's see Table 4, Figure 17a and Table 6, Figure 17b, respectively. In case of ATK #12, we can know that the 2^{nd}-bit-based CPA revealed the 15^{th} subkey, and its leak point was $f_{24}(\cdot)$. To understand how

Table 3: Sum/Highest DCA ranking of the correct subkey. If the correct one is not in the top 10, we leave it blank.

ATK # \ SubKey	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1																
2								2/5								
3																
4																
5																
6				8/4												
7																
8								3/			/6					
9					5/3							/6	7/			
10											9/		/3			
11												/4				
12		5/3		5/											1/1	/2
13			9/					3/						6/		
14									/1							/4
15										/6						
16								/7								
17															1/9	
18									1/2							
19																
20			3/													

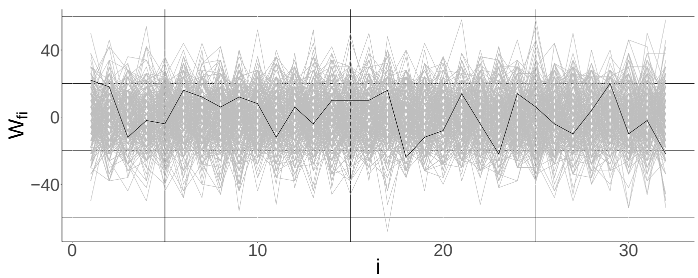


(a) On the 15^{th} subkey in ATK #12 with $\omega = 2$

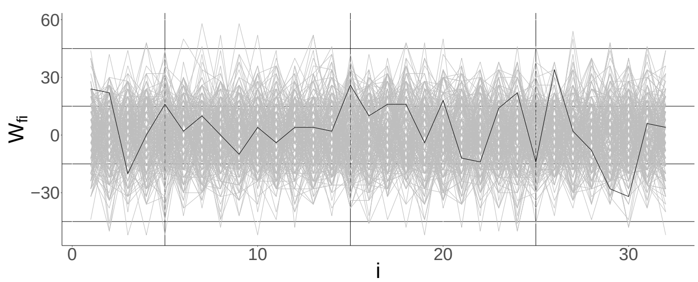


(b) On the 15^{th} subkey in ATK #17 with $\omega = 64$.

Fig. 17: Walsh transforms for $f_{i\in\{1,\ldots,32\}}(\cdot)$ of ATK #12 and ATK #17. Black line: correct key, gray line: wrong key candidates.



(a) On the 9^{th} subkey in ATK #14 with $\omega = 128$.



(b) On the 9^{th} subkey in ATK #18 with $\omega = 16$.

Fig. 18: Walsh transforms for $f_{i\in\{1,\ldots,32\}}(\cdot)$ of ATK #14 and ATK #18. Black line: correct key, gray line: wrong key candidates.

Table 4: DCA ranking of ATK #12.

TargetBit \ SubKey	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	20	19	156	151	228	101	158	89	78	232	199	110	141	210	58	64
2	169	1	41	30	188	189	24	149	196	162	101	237	103	38	1	118
3	218	113	51	138	166	63	97	2	237	53	227	138	163	227	55	2
4	190	53	33	65	55	212	146	177	2	152	225	119	9	230	30	253
5	138	142	62	3	16	4	184	121	107	170	23	253	97	143	151	160
6	137	99	206	184	165	44	111	73	27	148	119	247	52	152	29	71
7	61	137	43	108	223	197	172	223	199	71	70	131	84	84	149	240
8	106	202	102	4	200	58	254	156	65	51	84	178	138	238	14	83

Table 5: DCA ranking of ATK #14.

TargetBit \ SubKey	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	161	95	150	236	231	231	104	198	197	249	150	33	212	68	196	135
2	18	66	108	57	124	79	47	78	142	11	252	240	122	70	212	187
3	213	198	95	107	159	85	99	98	149	201	172	92	160	144	63	193
4	237	249	221	10	45	157	10	168	107	1	209	194	242	17	177	249
5	129	218	167	91	176	124	113	168	83	59	228	52	183	43	9	204
6	46	232	85	205	244	212	18	9	37	221	250	131	237	66	76	1
7	227	249	212	94	237	45	227	129	194	208	103	131	46	165	145	228
8	46	193	137	249	124	250	111	21	1	97	31	128	247	106	115	215

to interpret $f_{24}(\cdot)$ related to the 15^{th} subkey in the first round, recall Section 3.2 that the 15^{th} subkey is involved into x of $Ty_2(x)$, where

$$Ty_2(x) = x \cdot [01\ 03\ 02\ 01]^T,$$

and x is the SubBytes output. Then this gives us $f_{24}(\cdot)$ is the LSB of the encoded byte of $(x \cdot 02) \oplus m \in_R \{0,1\}^8$, where $(x \cdot 02)$ is implemented as a 1-bit left shift followed by a conditional ($\oplus$ 0x1B) if the MSB of x was 1. Therefore, this key leak is considered accidental since this has nothing to do with the 2^{nd} bit of the target hypothetical SubBytes output. An explanation of the accidental key leak in ATK #17 can be given in the same way. When performing CPA with the 7^{th} bit of the hypothetical SubBytes output, $f_{19}(\cdot)$ is shown to be not first-order correlation immune, where $f_{19}(\cdot)$ is the 6^{th} of the encoded byte of $(x \cdot 02) \oplus m \in_R \{0,1\}^8$. They are irrelevant to each other.

In ATK #14 and ATK #18, we can not see any distinguishable imbalance that is likely to reveal the subkey as shown in Figure 18. Interestingly, we found that key leakages occurred at the unexpected points: during the lookup of the encoded masks as shown in Figure 19, where $\bar{f}_{i\in\{1,\ldots,32\}}(x)$ denotes 32 Boolean functions for the encoded masks and x is the SubBytes output. In other words, $\bar{f}(x)$ means the encoded random masks used to protect the 4-byte intermediate value of the

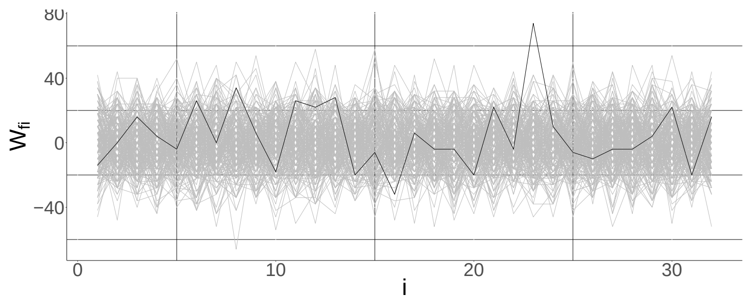
Table 6: DCA ranking of ATK #17.

TargetBit \ SubKey	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	54	242	59	27	127	171	9	174	249	72	135	151	31	28	12	256
2	23	33	64	195	85	104	154	216	226	31	222	137	173	225	132	113
3	151	145	221	80	126	238	11	134	236	181	224	250	154	30	12	203
4	255	237	62	63	20	217	160	218	225	101	197	125	207	134	16	211
5	209	93	107	204	11	194	92	254	220	18	110	223	106	154	38	224
6	197	132	211	252	151	173	7	50	71	49	39	29	212	20	3	177
7	138	161	220	246	16	60	251	46	223	199	35	158	196	129	1	209
8	159	192	204	13	120	237	231	253	202	71	72	45	142	251	10	238

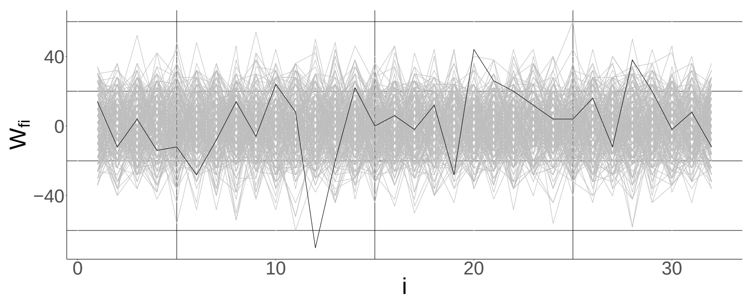
Table 7: DCA ranking of ATK #18.

TargetBit \ SubKey	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	174	52	230	105	91	122	229	39	84	194	213	221	118	38	158	32
2	158	176	107	84	22	190	56	61	33	228	197	123	44	125	97	16
3	202	30	239	254	181	142	201	23	21	190	147	117	22	242	185	248
4	8	132	252	126	232	29	95	20	41	126	12	254	72	155	166	91
5	134	62	79	110	163	5	6	88	1	256	24	88	137	196	174	122
6	243	147	88	27	68	184	72	212	133	246	196	83	176	145	18	239
7	193	222	162	168	45	26	225	234	242	73	144	92	181	6	34	167
8	252	192	67	39	141	31	21	129	119	147	14	215	151	158	154	160

MixColumns. Since the randomly generated mask is not key sensitive, these can not be the key leakages. Therefore, it can be concluded that there was no key leakage in the correct sense.



(a) On the 9^{th} byte of the state matrix in ATK #14 with $\omega = 128$



(b) On the 9^{th} byte of the state matrix in ATK #18 with $\omega = 16$.

Fig. 19: Walsh transforms for $\bar{f}_{i\in\{1,\ldots,32\}}(\cdot)$ of ATK #14 and ATK #18. Black line: correct key, gray line: wrong key candidates.

The histogram and normal distribution of the 2560 (=20×16×8) mono-bit CPA rankings are shown in Figure 20. The average ranking of the mono-bit CPA attacks for the correct subkey is 128.98 (S.D = 74.61). The highest value average of the mono-bit CPA correlation coefficient for the correct subkey was just 0.206 (S.D = 0.022). It is much lower than 0.557, that of the unprotected white-box implementation attacked with only 200 software traces. In conclusion, the correlation to the key is drastically reduced through Boolean masking applied before encoding, and our method can be used as an efficient countermeasure against DCA and power analysis by significantly mitigating key leakage caused by the encoding imbalance.

5 Conclusion and Discussion

In this paper, we proposed a masked white-box cryptographic implementation to protect DCA attacks. First, we generated 20 target instances according to

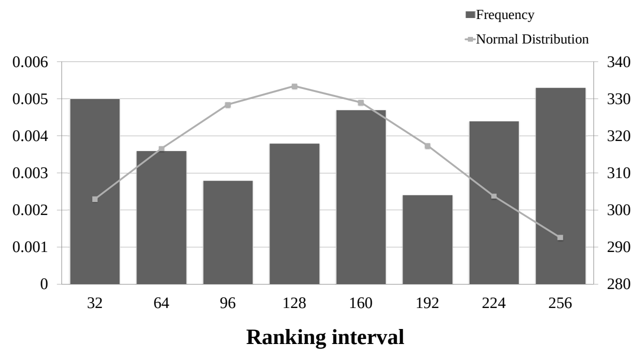


Fig. 20: CPA ranking histogram and normal distribution.

the unprotected WB-AES implementation and performed DCA on the SubBytes output in the first round with 200 software traces. As a result, an average of 14.3 subkeys were leaked and the average of the highest CPA correlation coefficient for the correct subkey was 0.557. In order to testify the problematic encoding imbalance we provided the sum of all imbalances that distinguishes the correct key from other key candidates.

To solve this problem, we applied masking to the intermediate value before applying the encoding during the white-box table generation. Based on this basic idea, a design method of the masked WB-AES implementation was suggested. To demonstrate its security, DCA was performed with 10,000 software traces for each of 20 instances. Although 4 out of the 320 subkeys were leaked as a result, we showed that they can not be seen as key leakages because a correlation has occurred at a point where the target intermediate value has nothing to do with the key value. The highest CPA correlation coefficient of the correct subkey was 0.206 in average. Collectively, we can conclude that our proposed method can practically defend DCA and power analysis on white-box cryptographic implementations.

We presented three variants based on the security requirement level for DCA and power analysis attacks. Compared to the unprotected WB-AES implementation, the lookup table size increased by approximately 1.56 to 9.59 times, and the number of lookups by about 1.6 times. Thus a careful choice has to be made where and how to apply this countermeasure. An additional attractive point is that there is no need for a random source at runtime. Of course, the storage requirements of our proposed method might be not suitable for low-cost and resource-constrained devices like IC cards, but are likely to be available to other smart devices.

While there are a variety of problems and limitations, white-box cryptographic implementations certainly have advantages in environments where hardware cryptographic equipment is not available. In addition, it is easy to update the key or the cryptographic logic compared to the hardware device. Another interesting

point is that the white-box cryptographic implementation for the symmetric key algorithm can be applied to asymmetric key applications because the encryption and the decryption lookup tables are different from each other. Currently, various companies [14] [2] [16] are trying to commercialize white-box cryptography, and more and more white-box solutions will be provided in the future. In the case of software-based cryptographic implementations, the secret keys that reside in memory are likely to leak if they do not have any protection. Therefore, the level of protection should also be chosen appropriately, taking into account the value of the protected information.
Directions for future work include developing various designs of other block ciphers and combining additional techniques to provide resistance to white-box attacks. Also, applying other kinds of masking techniques can be taken into account.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2017-0-00364, Development of training-based template attack software for security verification of secret data leakage of embedded secure element [the T&HO project]), and (No.2016-0-00399, Study on secure key hiding technology for IoT devices [the KeyHAS project]).

References

1. Akkar, M., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. pp. 309–318. No. Generators (2001), http://dx.doi.org/10.1007/3-540-44709-1_26
2. Axsan white-box cryptographic solution.: https://www.arxan.com/technology/white-box-cryptography/
3. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a White Box AES Implementation. In: Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers. pp. 227–240 (2004), http://dx.doi.org/10.1007/978-3-540-30564-4_16
4. Biryukov, A., Bouillaguet, C., Khovratovich, D.: Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key (Extended Abstract). In: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I. pp. 63–84 (2014), http://dx.doi.org/10.1007/978-3-662-45611-8_4
5. Blömer, J., Guajardo, J., Krummel, V.: Provably Secure Masking of AES. In: Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers. pp. 69–83 (2004), http://dx.doi.org/10.1007/978-3-540-30564-4_5

6. Bogdanov, A., Isobe, T.: White-Box Cryptography Revisited: Space-Hard Ciphers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015. pp. 1058–1069 (2015), http://doi.acm.org/10.1145/2810103.2813699
7. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential Computation Analysis: Hiding your White-Box Designs is Not Enough. vol. 2015, p. 753 (2015), http://dblp.uni-trier.de/db/journals/iacr/iacr2015.html#BosHMT15
8. Bottinelli, P., Bos, J.W.: Computational Aspects of Correlation Power Analysis (2015), https://eprint.iacr.org/2015/260
9. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004)
10. Bringer, J., Chabanne, H., Dottax, E.: White Box Cryptography: Another Attempt. IACR Cryptology ePrint Archive 2006, 468 (2006), http://eprint.iacr.org/2006/468
11. Chow, S., Eisen, P., Johnson, H., Oorschot, P.C.V.: White-Box Cryptography and an AES Implementation. In: Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002). pp. 250–270. Springer-Verlag (2002)
12. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A White-Box DES Implementation for DRM Applications. In: Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers. pp. 1–15 (2002), http://dx.doi.org/10.1007/978-3-540-44993-5_1
13. Coron, J., Goubin, L.: On Boolean and Arithmetic Masking against Differential Power Analysis. In: Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings. pp. 231–237 (2000), http://dx.doi.org/10.1007/3-540-44499-8_18
14. Gemalto white-box cryptographic solution.: https://sentinel.gemalto.com/software-monetization/white-box-cryptography/
15. Goubin, L., Masereel, J., Quisquater, M.: Cryptanalysis of White Box DES Implementations. In: Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers. pp. 278–295 (2007), http://dx.doi.org/10.1007/978-3-540-77360-3_18
16. InsideSecure white-box cryptographic solution.: https://www.insidesecure.com/Products/Application-Protection/Software-Protection/WhiteBox
17. Karroumi, M.: Protecting White-Box AES with Dual Ciphers. In: Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers. pp. 278–291 (2010), http://dx.doi.org/10.1007/978-3-642-24209-0_19
18. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 388–397 (1999), http://dx.doi.org/10.1007/3-540-48405-1_25
19. Lee, S.: A Masked White-box Cryptographic Implementation for Protecting against Differential Computation Analysis. Cryptology ePrint Archive, Report 2017/267 (2017), http://eprint.iacr.org/2017/267
20. Lee, S., Choi, D., Choi, Y.J.: Improved Shamirś CRT-RSA Algorithm: Revisit with the Modulus Chaining Method. ETRI Journal 3(3) (Apr 2014)

21. Lee, S., Choi, D., Choi, Y.J.: Conditional Re-encoding Method for Cryptanalysis-Resistant White-Box AES. ETRI Journal 5(5) (Oct 2015), http://dx.doi.org/10.4218/etrij.15.0114.0025
22. Lee, S., Jho, N.: One-Bit to Four-Bit Dual Conversion for Security Enhancement against Power Analysis. IEICE Transactions 99-A(10), 1833–1842 (2016), http://search.ieice.org/bin/summary.php?id=e99-a_10_1833
23. Lepoint, T., Rivain, M., Mulder, Y.D., Roelse, P., Preneel, B.: Two Attacks on a White-Box AES Implementation. In: Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers. pp. 265–285 (2013), http://dx.doi.org/10.1007/978-3-662-43414-7_14
24. Luk, C., Cohn, R.S., Muth, R., Patil, H., Klauser, A., Lowney, P.G., Wallace, S., Reddi, V.J., Hazelwood, K.M.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In: Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005. pp. 190–200 (2005), http://doi.acm.org/10.1145/1065010.1065034
25. Masked WB-AES CASE1 sample binary.: https://github.com/SideChannelMarvels/Deadpool/tree/master/wbs_aes_lee_case1
26. Messerges, T.S.: Securing the AES Finalists Against Power Analysis Attacks. In: Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings. pp. 150–164 (2000), http://dx.doi.org/10.1007/3-540-44706-7_11
27. Michiels, W., Gorissen, P., Hollmann, H.D.L.: Cryptanalysis of a Generic Class of White-Box Implementations. In: Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers. pp. 414–428 (2008), http://dx.doi.org/10.1007/978-3-642-04159-4_27
28. Minaud, B., Derbez, P., Fouque, P., Karpman, P.: Key-recovery attacks on ASASA. In: Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II. pp. 3–27 (2015), http://dx.doi.org/10.1007/978-3-662-48800-3_1
29. Muir, J.A.: A Tutorial on White-box AES. IACR Cryptology ePrint Archive 2013, 104 (2013), http://eprint.iacr.org/2013/104
30. de Mulder, Y.: White-Box Cryptography: Analysis of White-Box AES Implementations. In: Ph.D thesis, KU (2002)
31. Mulder, Y.D., Roelse, P., Preneel, B.: Cryptanalysis of the Xiao - Lai White-Box AES Implementation. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. pp. 34–49 (2012), http://dx.doi.org/10.1007/978-3-642-35999-6_3
32. Mulder, Y.D., Wyseur, B., Preneel, B.: Cryptanalysis of a Perturbated White-Box AES Implementation. In: Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings. pp. 292–310 (2010), http://dx.doi.org/10.1007/978-3-642-17401-8_21
33. Nethercote, N., Seward, J.: Valgrind: a Framework for Heavyweight Dynamic Binary Instrumentation. In: Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007. pp. 89–100 (2007), http://doi.acm.org/10.1145/1250734.1250746

Table 8: DCA ranking of ATK #1. If the correct key is not in the top 10, we leave it blank.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	53	62	138	179	245	167	214	146	85	57	223	244	32	38	169	152
2	36	12	70	17	160	241	244	19	148	184	113	119	68	195	96	20
3	190	238	226	76	80	250	183	58	10	4	193	113	49	252	232	85
4	52	168	234	153	235	92	20	177	70	19	232	84	213	245	193	187
5	223	113	193	239	44	253	241	69	134	34	93	123	158	163	151	165
6	42	75	168	256	199	39	120	181	57	122	43	194	205	176	170	89
7	179	170	236	215	230	98	152	82	52	250	124	122	206	79	88	234
8	198	111	149	158	79	97	81	55	107	153	87	96	219	240	166	18
sum																
highest																

34. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-Channel Attacks and Glitches. In: Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings. pp. 529–545 (2006), `http://dx.doi.org/10.1007/11935308_38`
35. Sanfelix, E., Mune, C., de Haas, J.: Unboxing the White-Box: Practical Attacks against Obfuscated Ciphers. In: Presented at BlackHat Europe 2015 (2015), `https://www.blackhat.com/eu-15/briefings.html`
36. Sasdrich, P., Moradi, A., Güneysu, T.: White-Box Cryptography in the Gray Box - - A Hardware Implementation and its Side Channels -. In: Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. pp. 185–203 (2016), `http://dx.doi.org/10.1007/978-3-662-52993-5_10`
37. SCARF homepage: http://www.k-scarf.or.kr/
38. Wyseur, B.: White-Box Cryptography. In: Encyclopedia of Cryptography and Security, 2nd Ed. pp. 1386–1387 (2011), `http://dx.doi.org/10.1007/978-1-4419-5906-5_627`
39. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In: Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers. pp. 264–277 (2007), `http://dx.doi.org/10.1007/978-3-540-77360-3_17`
40. Xiao, Y., Lai, X.: A Secure Implementation of White-box AES. In: The Second Internationial Conference on Computer Science and Its Applications - CSA 2009. vol. 2009, pp. 1–6 (2009)

A DCA Ranking Tables

The following tables represent the DCA results for ATK #1 - ATK #20, except for ATK #12, ATK #14, ATK #17 and ATK #18 that were provided in Section 4. If the correct key is not in the top 10, we leave it blank.

Table 9: DCA ranking of ATK #2.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	44	229	36	54	233	67	37	74	23	47	71	160	203	195	208	87
2	15	223	161	247	229	211	76	165	205	188	78	179	45	188	61	169
3	171	219	117	156	171	82	176	127	113	90	41	64	138	125	108	129
4	81	184	62	202	56	50	211	108	198	53	217	71	76	41	155	115
5	186	31	161	19	76	61	206	32	202	71	33	102	123	131	15	177
6	256	238	49	80	16	232	185	34	73	236	130	110	178	242	2	32
7	87	64	4	59	157	76	225	30	106	171	253	99	34	27	254	29
8	186	148	164	29	166	98	18	2	7	113	202	45	115	63	118	54
sum	2															
highest	5															

Table 10: DCA ranking of ATK #3.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	252	90	243	76	164	242	174	236	251	179	171	2	55	225	128	210
2	154	114	193	230	57	77	119	231	185	155	125	2	46	167	77	248
3	21	242	235	206	127	55	247	256	77	38	199	52	174	247	121	99
4	80	52	73	208	35	211	178	50	79	86	230	147	18	135	31	61
5	220	7	108	110	7	80	24	208	255	99	4	157	237	225	213	45
6	229	189	140	60	8	30	222	33	113	46	37	255	189	115	204	35
7	189	13	52	128	205	193	129	175	96	39	24	123	171	133	82	127
8	70	200	122	204	213	166	235	6	13	240	27	110	37	50	23	203
sum																
highest																

Table 11: DCA ranking of ATK #4.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	230	44	114	119	100	13	4	130	140	185	84	213	32	78	51	139
2	15	229	171	220	248	131	225	127	223	69	200	178	104	33	131	170
3	93	16	254	254	180	36	249	208	12	188	217	191	194	252	158	140
4	236	162	250	37	215	50	240	140	25	190	31	78	192	84	191	3
5	22	121	217	239	181	24	199	12	249	213	225	15	248	219	41	152
6	187	163	51	148	185	18	123	218	181	63	204	13	223	144	89	114
7	190	183	128	59	39	189	4	219	65	125	48	59	254	214	26	234
8	220	76	110	143	250	208	168	212	64	25	15	85	182	141	41	135
sum																
highest																

Table 12: DCA ranking of ATK #5.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	256	31	195	189	210	39	18	192	147	35	150	246	239	190	75	72
2	183	238	88	185	212	105	91	64	210	244	45	95	129	253	196	74
3	110	41	68	116	133	67	119	203	203	188	204	181	106	165	85	219
4	169	171	18	20	25	91	124	252	91	184	111	175	95	143	93	179
5	41	220	155	35	147	30	115	73	16	106	58	142	136	146	32	161
6	57	231	174	103	21	235	227	94	180	61	44	190	31	127	240	199
7	40	179	174	74	139	129	59	24	67	1	24	134	65	94	204	213
8	227	222	81	201	164	72	96	116	199	151	238	36	14	179	113	46
sum																
highest																

Table 13: DCA ranking of ATK #6.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	114	102	69	21	109	115	107	53	191	36	183	31	147	137	155	142
2	113	253	168	36	160	244	19	242	171	224	20	235	231	47	15	98
3	249	74	138	13	19	35	169	62	249	189	214	94	95	247	106	197
4	63	12	84	210	200	77	160	24	244	229	104	215	128	59	21	72
5	65	235	66	1	173	189	192	191	63	71	104	156	101	113	156	46
6	27	244	188	211	97	212	215	159	22	106	230	127	54	163	196	209
7	5	212	73	117	170	46	174	127	249	60	158	37	10	250	219	95
8	78	66	117	252	123	130	75	203	5	22	164	97	147	136	138	28
sum				8												
highest				4												

Table 14: DCA ranking of ATK #7.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	215	221	253	175	255	255	177	238	175	43	111	131	247	174	86	79
2	73	30	242	15	28	84	64	73	86	215	148	240	155	46	38	95
3	116	115	217	47	207	20	7	106	76	167	2	192	67	245	148	218
4	94	32	230	106	242	77	139	78	256	22	125	23	164	41	214	55
5	174	110	145	246	105	15	85	34	154	57	31	151	61	48	118	12
6	252	109	155	95	187	144	249	85	164	82	236	41	221	191	181	142
7	134	241	71	166	256	237	184	26	72	241	171	205	144	164	190	50
8	199	87	86	238	40	132	152	77	33	73	157	19	223	143	155	208
sum																
highest																

Table 15: DCA ranking of ATK #8.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	27	214	75	143	120	75	210	5	71	241	62	92	109	108	54	16
2	140	80	18	219	232	147	18	17	214	52	58	169	61	14	196	12
3	157	208	153	91	28	204	138	25	77	212	51	100	98	221	220	235
4	190	189	185	27	236	143	125	166	7	240	223	249	106	15	161	12
5	146	111	46	28	213	70	102	217	91	35	2	75	30	114	54	114
6	145	230	48	195	151	136	225	206	15	50	219	3	9	56	129	104
7	8	43	60	229	80	57	187	15	213	199	245	10	216	176	147	201
8	130	224	249	155	159	44	167	30	125	121	77	109	54	176	146	44
sum								3								
highest											6					

Table 16: DCA ranking of ATK #9.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	187	224	96	237	34	254	86	246	74	107	175	80	40	117	227	159
2	74	164	132	181	232	112	71	7	122	54	31	150	7	231	71	241
3	118	77	41	122	112	135	2	97	151	210	34	221	27	20	44	147
4	149	227	134	91	27	244	139	30	228	106	123	2	154	106	24	87
5	170	154	229	186	23	37	68	93	7	73	220	171	139	130	4	151
6	252	171	31	168	234	160	212	197	38	93	226	36	223	168	140	120
7	10	26	200	211	116	193	237	227	175	194	256	83	97	256	28	87
8	205	179	217	47	1	240	249	51	184	126	41	167	4	140	167	46
sum					5								7			
highest					3							6				

Table 17: DCA ranking of ATK #10.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	18	109	75	207	123	197	124	249	95	3	101	169	206	146	122	19
2	180	179	95	223	12	110	217	113	179	201	183	106	206	64	49	166
3	170	173	204	49	24	245	72	153	115	121	84	122	162	96	158	163
4	25	205	140	237	183	102	227	111	82	208	86	212	169	175	105	7
5	225	155	131	227	18	217	116	191	168	110	49	16	123	138	227	57
6	138	29	52	242	180	27	206	151	100	87	15	236	2	202	213	214
7	115	177	173	171	211	51	166	211	211	202	26	211	132	139	138	91
8	68	150	50	49	158	62	218	27	239	240	34	143	6	57	25	19
sum											9					
highest													3			

Table 18: DCA ranking of ATK #11.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	38	94	170	176	215	90	50	27	231	249	13	108	64	14	134	12
2	237	49	159	51	54	183	119	123	187	20	177	217	133	253	23	207
3	169	34	253	113	129	25	38	225	111	187	144	58	131	220	88	71
4	197	80	68	45	212	97	139	218	89	211	78	242	81	176	107	57
5	238	82	94	41	107	200	242	25	129	63	14	2	165	146	85	167
6	173	150	221	243	215	179	197	122	110	86	225	112	113	59	166	100
7	206	156	112	70	97	23	178	242	91	170	107	176	63	134	233	220
8	243	125	248	111	18	207	126	121	233	83	69	67	137	209	72	36
sum																
highest												4				

Table 19: DCA ranking of ATK #13.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	9	64	98	151	39	157	200	9	125	108	104	88	97	28	1	85
2	23	178	51	190	47	152	72	172	216	217	227	237	154	4	217	104
3	186	29	21	135	207	147	131	22	52	50	190	134	210	68	160	146
4	171	77	116	234	70	80	60	15	89	230	218	231	58	132	116	147
5	119	211	249	157	98	72	18	56	124	255	203	6	68	23	211	15
6	57	167	44	109	76	88	209	164	72	8	152	109	138	94	188	178
7	243	72	70	23	254	213	73	67	165	228	74	149	129	137	225	250
8	154	148	2	246	127	146	179	33	229	114	169	88	250	135	152	16
sum			9					3						6		
highest																

Table 20: DCA ranking of ATK #15.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	212	109	141	19	99	141	19	141	240	99	227	56	43	194	135	54
2	189	233	193	146	95	81	166	136	135	123	88	214	97	86	43	167
3	226	103	232	250	254	160	61	128	35	194	89	228	172	192	86	150
4	161	195	3	255	109	254	96	53	199	47	18	111	139	236	120	2
5	198	197	46	242	124	255	141	113	165	81	250	243	255	251	192	114
6	121	202	246	208	256	179	35	176	71	1	229	222	47	49	10	182
7	35	107	136	123	42	49	126	199	72	198	9	191	140	253	106	174
8	191	85	34	240	14	27	38	130	33	66	37	63	216	62	32	38
sum																
highest										6						

Table 21: DCA ranking of ATK #16.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	249	211	108	234	8	193	122	61	184	168	145	42	223	252	53	124
2	35	243	32	170	155	176	116	147	37	256	70	72	22	189	253	214
3	248	202	140	63	154	162	221	128	73	123	235	101	196	103	33	70
4	232	113	92	92	103	110	18	28	197	87	137	231	84	186	47	129
5	40	200	163	185	165	6	159	3	24	38	66	125	233	156	127	145
6	105	136	28	130	8	123	180	175	86	126	34	210	22	65	192	99
7	106	141	52	102	139	152	67	108	147	83	21	42	243	193	38	80
8	128	123	216	11	90	58	114	125	146	208	141	52	101	50	17	99
sum																
highest								7								

Table 22: DCA ranking of ATK #19.

TargetBit \ KeyByte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	96	105	114	66	156	117	3	147	249	12	42	173	31	150	56	224
2	95	215	13	72	225	161	25	136	135	66	206	153	45	6	69	143
3	151	97	136	229	91	217	100	26	120	56	106	145	110	83	171	75
4	59	22	173	31	125	5	47	15	181	66	153	197	7	240	20	200
5	105	153	124	102	51	90	249	238	137	79	219	99	207	83	184	249
6	256	36	56	45	66	122	211	243	13	219	61	167	207	255	186	88
7	28	143	135	148	187	51	190	207	188	237	19	219	209	124	108	233
8	193	122	197	185	3	160	191	76	134	161	231	251	139	46	167	51
sum																
highest																

Table 23: DCA ranking of ATK #20.

KeyByte / TargetBit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	210	127	7	125	111	66	88	202	105	80	18	250	164	46	203	59
2	217	151	52	237	218	70	112	143	106	125	180	143	45	79	8	10
3	172	75	111	77	32	44	146	225	138	107	42	155	77	7	149	176
4	38	53	7	234	159	240	150	39	188	98	155	116	143	217	220	177
5	147	220	154	69	134	158	106	13	200	178	191	101	159	146	217	14
6	41	135	3	49	96	197	227	186	136	247	246	55	88	186	94	215
7	247	184	29	214	151	73	226	191	184	106	37	34	78	17	232	73
8	62	152	233	157	185	130	256	216	154	192	232	109	95	39	157	216
sum			3													
highest																