A Key Leakage Preventive White-box Cryptographic Implementation

Seungkwang Lee, Nam-su Jho, Myungchul Kim

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

Abstract. A white-box cryptographic implementation is to defend against white-box attacks that allow access and modification of memory or internal resources in the computing device. In particular, linear and non-linear transformations applied to this table-based cryptographic implementation is used to prevent key-dependent intermediate values from being seen by white-box attackers. However, it has been shown that there is a correlation before and after the linear and non-linear transformations so that even a gray-box attacker can reveal secret keys hidden in a white-box cryptographic implementation. In this paper, we focus on the problem of linear transformations including the characteristics of block invertible binary matrices and the distribution of intermediate values. Our experimental results and proof show that the balanced distribution of the key-dependent intermediate value is the main cause of key leakage. Based on this observation, we find out that a random byte insertion in the intermediate values before linear transformations can eliminate a problematic correlation to the key, and propose our white-box AES implementation using this principle. Our proposed implementations reduce the memory requirement by at most 33 percent compared to the masked implementations and also slightly reduce the number of table lookups. In addition, our method is a non-masking technique and does not require a static or dynamic random source, unlike the existing gray-box (power analysis) countermeasures.

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

1 Introduction

From a secret key point of view, a block cipher can be seen as a secret bijective function between a plaintext set and a ciphertext set. One possible implementation of this function is a lookup table of the ciphertext for each of its corresponding plaintext. Since implementing a cipher as one lookup table is impractical because of its huge size, it is usually implemented as a series of lookup tables. This table-based implementation is also used in the white-box cryptographic implementation so that a secret key is not revealed by a white-box attacker observing internal memory and computing resources. The important thing to remember over here is that the white-box cryptography generates key-instantiated lookup tables and protects each table with linear and non-linear transformations in order to prevent a key leakage from lookup values. Due to these transformations, a white-box attacker had been supposed to be unable to recognize a secret key via static or dynamic analysis.

So far, this white-box cryptography has been exploited by highly skilled attacks except for code lifting attacks [42] that steal the entire lookup table. For example, after the first two white-box DES (WB-DES) [12] and AES (WB-AES) [11] implementations were published, a number of practical cryptanalysis techniques [16][43] [4][24][29] have been introduced to extract the secret key from the white-box lookup tables. Many variants of WB-DES and WB-AES implementations [9][45] [19][23][25] were proposed and many were known to be practically broken [31][32]. Not only these white-box implementations of well-known standard block ciphers, dedicated white-box ciphers [5][6][30] have been also studied. One of dedicated cipher's shortcomings, however, is the lack of interoperability with the straightforward implementation of block ciphers. Recently, Differential Fault Analysis (DFA) [36] was introduced, where an attacker is able to inject a fault at a desired location in memory. Here, those white-box attacks rely on an in-depth understanding of a target implementation so that an attacker is able to gain read/write access to precise internal states during the execution. Thus commercial white-box cryptography [2][14][17][41] focuses on making a barrier to the full control of an attacker and is often combined with additional protection techniques including obfuscation, enveloping, hardware ID binding, and anti-debug protections. An explanation of these protections, primarily taking into account an untrusted environment, could be that the white-box implementation was without a doubt considered to defend against gray-box attacks, also known as side-channel attacks.

In contrast to white-box attacks, gray-box attacks are dependent on non-invasive information such as power consumption obtained while a target device performs cryptographic operations. For example, Differential Power Analysis (DPA) [21] is based on power consumption and this is one of the most well-known techniques to reveal the secret key imbedded in IC cards. Specifically, DPA is generally based on the fact that power consumption of a device is proportional or inversely proportional to the Hamming weight (HW) of data it processes. Thus a power analysis attacker collects a number of power traces with random plaintexts and finds a correct key that computes hypothetical values most highly correlated to the collected traces at a particular point. Currently, white-box cryptography can be easily broken by power analysis [7][37] without detailed knowledge of the target implementation. This means linear and non-linear transformations applied to lookup tables have no effect on hiding key-sensitive intermediate values.

In light of this vulnerability, finding software countermeasures to counteract power analysis has become essential for the further growth of white-box cryptography. Here are some examples. First, control flow obfuscation shuffles the order of table lookup only if it does not make any difference in the final result. The use of run-time random source for the large enough disarrangement and the re-ordering overhead are costly. Second, table location randomization is used to counteract address-based DPA, named Differential Computation Analysis (DCA) [7]. Compared to DPA and its variants such as Correlation Power Analysis (CPA) [8], DCA is a more skilled attack that uses software execution traces containing both sensitive data and memory address accesses; for this reason, DCA shows better analysis performance, but is in fact considered a white-box attack because of the direct access to memory in the process of collecting software execution traces. Of course, table location randomization is useless if value-based DCA is mounted. Third, adding an arbitrary number of dummy lookups can be one of the disarrangement methods to randomize the time instance at which the target intermediate value is computed for each execution. Indeed, it is known to be resistant to DCA and Zero Difference Enumeration [3] attacks. However, all these run-time randomization methods are expensive, and the execution time of the WB-AES encryption can be slowed up to 16 times [3]. In addition, the random source can always be compromised by a white-box attacker. Last but not least, there is a masked white-box implementation [22] for protecting against DCA and other DPA-like attacks. The key idea behind is to apply masking to sensitive intermediate values before transformations during the table generation, and thus no run-time random source is required. However, this method did not address the fundamental reason for the imbalance pertaining to the transformations but applied the existing masking to eliminate the vulnerability at a high cost.

1.1 Our contribution

- We find out that the key leakage after linear transformations is largely due to the balanced distribution of intermediate values, and offer a simple proof of it.
- We also find out that a random byte insertion in the intermediate value before linear transformations prevents the key leakage. Furthermore, our experimental results show that inserting position does not make difference and inserting more than one byte provides no additional effect.
- We provide a non-masking design of WB-AES based the above principle without the need for static or dynamic random sources. The memory requirement and the number of lookups are decreased compared to masked WB-AES implementations.

1.2 Organization of the Paper

The rest of this paper is organized as follows. Section 2 reviews some basic concepts including power analysis, the principle of a white-box cryptographic implementation, and the key leakage issue. In Section 3, we analyze the invertible linear transforms used in the white-box cryptography to see why the key-dependent intermediate values are still correlated to the key even after linear transformations. Based on this analysis, we propose our solution resistant to DCA and DPA-like attacks in Section 4. Specifically, a WB-AES implementation is newly implemented for concrete demonstration. We then evaluate its security and performance in Section 5. Finally, Section 6 concludes this paper.

2 Background

In this section, we introduce the basic concept of power analysis and a WB-AES implementation first appeared in [11], and then investigate key leakage points with the Walsh transforms.

2.1 Power Analysis

Gray-box attacks, which were recently successful on white-box cryptography, are precisely power analysis such as DPA and CPA. An explanation of successful DPA and CPA on the white-box cryptographic implementation could be that attacker's correct hypothetical value will correlate to the target table lookup value. Note that DCA improves the efficiency of DPA and CPA attack since there is no measurement noise in the software execution traces, unlike the power consumption traces.

After collecting the traces with random plaintexts, DPA and CPA perform statistical analysis in different ways. DPA uses the selection function D to split the collected traces into two sets based on the attacker's hypothetical values. If the attacker's hypothetical key is correct (and therefore the hypothetical value is correct), then the trace separation by D is also accurate and there will be a peak in the differential trace.

In contrast, CPA uses a leakage model including the HW and the Hamming distance instead of the selection function D. When attacking a white-box implementation, the bit (mono-bit) model is appropriate because HW-based CPA attacks are unlikely to be successful due to the disturbed HW by linear and nonlinear transformations. Given N power traces $V_{1..N}[1..\kappa]$ containing κ samples each, CPA will estimate the power consumption at each point of each trace using attacker's hypothetical intermediate value. For K different key candidates, let $\mathcal{E}_{n,k^*}$ $(1 \leq n \leq N,\ 0 \leq k^* < K)$ denote the power estimate in the n^{th} trace with the hypothetical key k^*. To measure a correlation between hypothetical power consumption and measured power traces, the estimator r is defined as follows [26]:

$$r_{k^*,j} = \frac{\sum_{n=1}^{N}(\mathcal{E}_{n,k^*} - \overline{\mathcal{E}_k^*}) \cdot (V_n[j] - \overline{V[j]})}{\sqrt{\sum_{n=1}^{N}(\mathcal{E}_{n,k^*} - \overline{\mathcal{E}_k^*})^2 \cdot \sum_{n=1}^{N}(V_n[j] - \overline{V[j]})^2}},$$

where $\overline{\mathcal{E}_k^*}$ and $\overline{V[j]}$ are sample means of $\mathcal{E}_k^*$ and $V[j]$, respectively. If there exists a correlation, a noticeable peak will be found in the correlation plot for the correct key.

Power analysis countermeasures can be categorized into masking and hiding, where masking breaks the correlation between power signals and the processed data while hiding reduces the signal to noise ratio. Maksing [1][13][15]

[27][33][39] randomizes every key-dependent intermediate value by precomputing a new masked lookup table for each execution of encryption. To protect against higher-order DPA attacks [18][28][40], where an attacker exploits the joint key leakage from several intermediate values, higher-order DPA countermeasures have been studied [38][35] [20][10][34]. One of the most used hiding techniques, on the other hand, is introducing random delay. When the target cryptographic operation occurs uniformly distributed across n time instants due to random delay, the number of power traces for a successful DPA grows in n^2 only if DPA is performed straightforwardly. Here we can see these countermeasures are strongly dependent on expensive run-time random source, and also result in slow execution of cryptographic algorithm.

2.2 WB-AES Implementation

In this section, we briefly explain the initial WB-AES implementation [11]. For the WB-AES implementation with a 128-bit key, the AES algorithm is re-written as follows:

state ← *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})
ciphertext ← state,

where k^r means the 4×4 round key matrix for round r, and $\boldsymbol{k}^r_{i,j}$ indicates that the ShiftRows is applied to $k^r_{i,j}$. A WB-AES implementation is currently based on the table-based implementation which combines the above operations except for ShiftRows into a set of lookup tables and applies linear and non-linear transformations. Linear transformations use two types of invertible matrices. One is multiplied to partial MixColumns outputs and the other is multiplied to each round input byte. Non-linear transformations consist of two 4-bit concatenated random bijections to reduce the total table size. More specifically, if 8-bit random bijections are used in non-linear transformations, the XOR lookup table size will increase significantly because an XOR table has to take two 8-bit inputs instead of two 4-bit ones. On the condition that there is no external encoding (encoding means linear and non-linear transformations) for better comparability with non-WB-AES implementations, we need four types of the lookup tables: *TypeII*, *TypeIII*, *TypeIV* and *TypeV*. From now on, we explain how to generate them.

TypeII. The lookup values of $\mathit{Type\ II}^r_{i,j}$ provide the encoded result of AddRoundKey, SubBytes and decomposed multiplications of MixColumns, where

$1 \leq r \leq 9$, and $0 \leq i, j \leq 3$. The first step of generating *TypeII* is to combine AddRoundKey and SubBytes into *T-boxes*, 160 8×8 lookup tables, as follows:

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

Note that *TypeII* uses T^1 - T^9 and *TypeV* uses T^{10} later. What is important over here is that we must decode the input of *TypeII* from round 2 based on the index change due to ShiftRows since it is encoded output of the previous round. On the other hand, the input to *TypeII* in the first round is not required to be decoded because we do not use the external encoding. Let $L^r_{i,j}$ denote 144 (=9×4×4) 8×8 binary invertible matrices used to linearly transform each byte of the round out, where $r \in [1, 9]$, $0 \leq i,\ j \leq 3$. In round $r \geq 2$, the inverse linear transformation (after inverse non-linear transformation) on the encoded *TypeII* input p' (previous round output) at the index $\{i,\ j\}$ is performed as follows:

1. Adjust the proper index after ShiftRows to find the corresponding inverse matrix of the linear transformation; $\{i,\ j'\} \leftarrow \{i,\ (j+i) \bmod 4\ \}$
2. $p \leftarrow (L^{r-1}_{i,j'})^{-1} \cdot p'$.

Then we know that $T^r_{i,j}(p)$ gives us an input x to the MixColumns step except for $T^{10}_{i,j}(p)$ because there is no MixColumns in the final round. Let's decompose the matrix multiplication with a column vector in MixColumns 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 four terms y_0, y_1, y_2, y_3 at the right-hand side, we define Ty_i tables:

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

The next step is to perform linear transformations on $Ty_{i\in\{0,1,2,3\}}(x)$ with a given invertible 32×32 matrix M, conduct non-linear transformations and store them in *TypeII* as illustrated in Fig 1. Because of its 4-byte outputs the intermediate values after visiting *TypeII* will be store in a 4×4×4 matrix. Of course, this dimensional structure can be different depending on the programmer's choice.

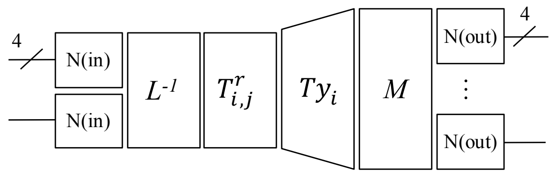


Fig. 1: A schematic diagram of *TypeII* generation. N: non-linear transformations.

TypeIII. During the *TypeIII* generation (Fig. 2), we perform the inverse linear transformation with M^{-1} and apply the linear transformation with L. These two linear transformations result in one-byte *Type II* input in the next round. After combining intermediate values by looking up *TypeIV_III*, the linear transformation by M is canceled out and the linear transformation by L protects the round output in the state matrix. In turn, this will be canceled out in *TypeII* input decoding. After visiting *TypeIII*, we have the intermediate values in a 4×4×4 matrix.

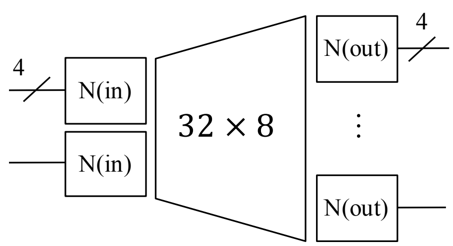


Fig. 2: A schematic diagram of *TypeIII* generation.

TypeIV. This performs XOR operations in order to combine intermediate values. Thus *TypeIV* depicted in Fig. 3 is visited after *TypeII* or *TypeIII* lookups; we call them *TypeIV_II* and *TypeIV_III*. *TypeIV_II* and *TypeIV_III* combine 4×4×4 intermediate values into a 4×4 state matrix.

All two 4-bit inputs to be XORed must be linearly transformed at the same offset of the same invertible matrix so that we do not need to linearly transform the input when generating *TypeIV* because it satisfies the distributive property of multiplication over addition. Fig. 4 and Fig. 5 simplify the lookup flows of *TypeIV* following *TypeII* and *TypeIII*.

TypeV. This includes T^{10} and is looked up in the final round. As depicted in Fig. 6 the lookup values of *TypeV* are not encoded because those are the ciphertext, and as stated previously we assume that there is no external encoding.

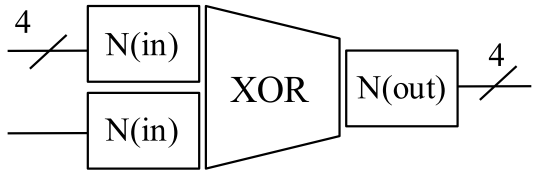


Fig. 3: A schematic diagram of *TypeIV* generation.

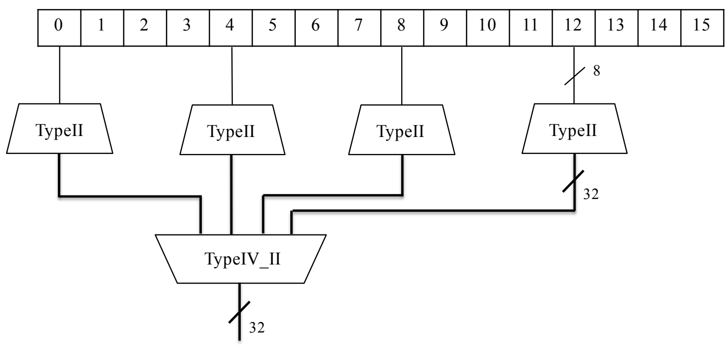


Fig. 4: *TypeII* and *TypeIV_II* lookups.

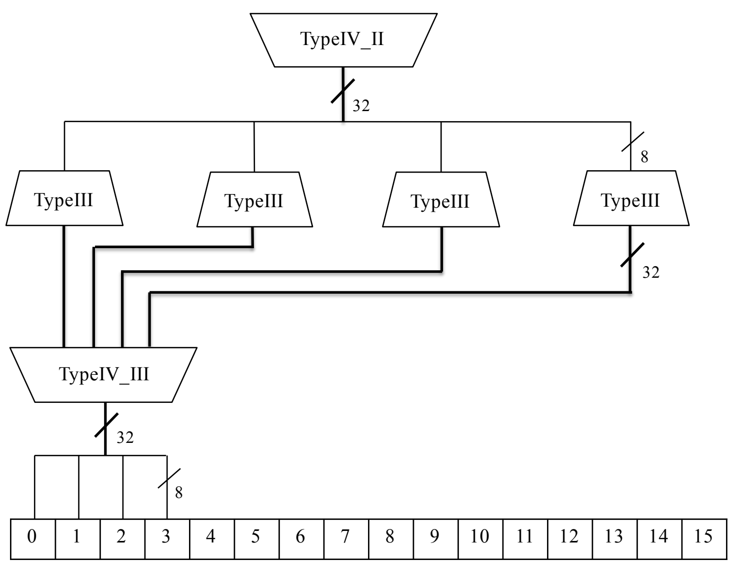


Fig. 5: *TypeIII* and *TypeIV_III* lookups.

Then, an AES encryption can be performed by only ShiftRows and table lookups of these four types of lookup tables.

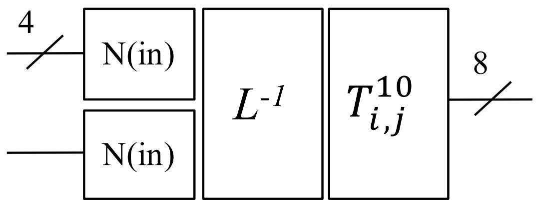


Fig. 6: A schematic diagram of *TypeV* generation.

This WB-AES implementation has three key leakage points that could be successfully analyzed by gray-box attacks. First, an attacker can perform a DPA or CPA attack on the SubBytes output, that is the *TypeII* output, in the first round. Then it requires to test 2^8 candidates for each subkey of the first round key provided that there is no protection for key leakage. Second, as explained in [22], the final round input (or the ninth round output) can be attacked with 2^{16} candidates for each subkey of the final round key due to the absence of MixColumns. Lastly, it is not impossible to attack the first round output by searching 2^{32} candidates for each 4-byte column vector of the first round key matrix. Then attacker's target values will be *TypeIV_II* or *TypeIV_III* outputs in the first round. In the next section, we explain the use of the Walsh transforms to detect key leakages and demonstrate it on this WB-AES implementation.

2.3 Detecting Key Leakage by the Walsh Transforms

We can quantify or visualize a correlation using the Walsh transforms if a target lookup table is given. To understand how the Walsh transform can be used to quantify a correlation between the input and output of a target lookup table, we use the following definitions from [37].

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.*

Then we know that $W_f(\omega)$ quantifies the imbalances in the encoding, and the large absolute value of $W_f(\omega)$ means the strong correlation between $f(x)$ and $x \cdot \omega$. Using this property, we calculate the correlation between the table lookup values and hypothetical values.

Let's demonstrate the key leakage from the *TypeII* lookup value in the first round. Given a subkey $k^0_{0,2} = \mathit{0x}88$, and for every input value $p \in \mathrm{GF}(2^8)$, we know that

$$\begin{aligned} x \quad &= S(p \oplus k^0_{0,j}) \\ y_0(x) &= \begin{bmatrix} 2 \cdot x & x & x & 3 \cdot x \end{bmatrix}^T \end{aligned}$$

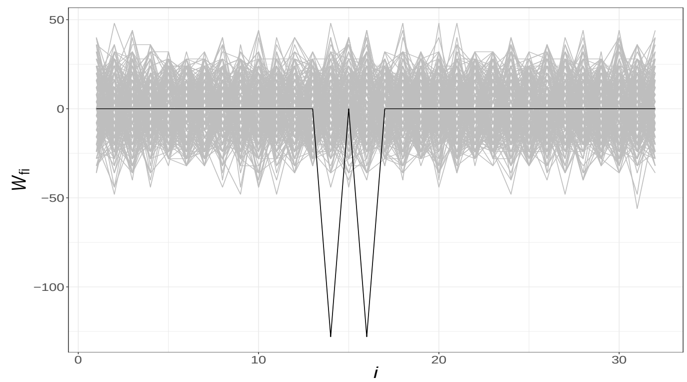
where S represents the AES SubBytes and the multiplication by 2 is implemented as a 1-bit left shift followed by a conditional ($\oplus$ 0x1B) if the MSB of the operand was 1. Then $f(x)$ here denotes *TypeII* lookup values which are computed from linear and non-linear transformations on $y(x)$, and we have 32 Boolean functions $f_{i \in \{1,\ldots,32\}}(x)$: $\{0,1\}^8 \rightarrow \{0,1\}$. To find a correct key, we calculate the Walsh transforms W_{f_i} and sum all the imbalances for each key candidate and ω such that $\mathrm{HW}(\omega) = 1$ as follows:

$$\Delta^f_{k \in \{0,1\}^8} = \sum_{\omega=1,2,4,\ldots,128} \sum_{i=1,\ldots,32} |W_{f_i}(\omega)|.$$

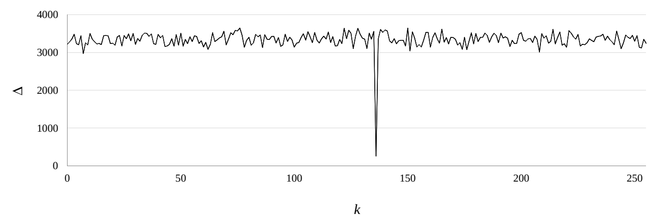
The reason why we only select ω of $\mathrm{HW}(\omega) = 1$ is that the HW-based key leakage model is not effective to detect the correlation between the in/output of the encoding.

The Walsh transforms and their sum of all imbalances are given in Fig. 7. As we can see in Fig. 7a, the Walsh transforms with $\omega = 4$ of the correct key ($\mathit{0x}88$) produce 0 except two points; the W_{f14} and W_{f16} of the correct key are -128, and their absolute value (128) is the most highest value. In contrast the maximum and the average values of $|W_{f_i}(\omega)|$ of wrong key candidates are 56 and about 13.13 (the standard deviation is about 9.35), respectively. This gives us that $f_{14}(\cdot)$ and $f_{16}(\cdot)$ cause key leakages and thus power analysis using the 3^{rd} bit (when the LSB is the 1^{st} bit) of attacker's hypothetical SubBytes outputs is able to recover this subkey. $\Delta^f_{k=\mathit{0x}88}$ is 256 ($= |-128| + |-128|$) which is obviously distinguishable from that of other key candidates as shown in Fig. 7b while $\Delta^f_{k \neq \mathit{0x}88}$ are about 2900-3700. This simply shows us how to use the sum of all imbalances for recovering the correct key.

In the similar way, we can detect a key leakage at the final round input. In this scenario, we give the first subkey of the ninth round key ($\mathit{0x}54$) and let the attacker guess the first subkey ($\mathit{0x}13$) of the final round. Given a subbyte of the ciphertext, x becomes the attacker's hypothetical input value and 8 Boolean functions $f_{i \in \{1,\ldots,8\}}(x)$: $\{0,1\}^8 \rightarrow \{0,1\}$ indicate the encoded input to *TypeV*. On the condition that there is no external encoding on the ciphertext and the attacker knows the first subkey of the ninth and final round keys, $\Delta^f_{k=\mathit{0x}13}$ in the final round is 4096; this is much smaller than $\Delta^f_{k \neq \mathit{0x}13}$ in the range 25900 - 26500 as shown in Fig. 8. In the following section, we analyze the linear transformations used in the white-box lookup table generation, and provide a clue for a secure implementation.



(a) Walsh transforms for $f_{i\in\{1,\cdots,32\}}(\cdot)$ with $\omega = 4$ for all key candidates. Gray: wrong key candidates; Black: correct key.



(b) Sum of all imbalances for all key candidates.

Fig. 7: Key leakage detection using the Walsh transforms.

3 Analysis of Linear Transformations

As we know that two 4-bit concatenated random bijections for non-linear transformations offer a limited hiding capability compared to 8-bit to 8-bit random bijections, linear transformations play an important role for key leakage protection. The following provides our crucial observation on the linear transformation used in the white-box cryptography.

3.1 Key Leakage Statistics after Linear Transformations

We begin with an experimental result of a key leakage at the linear transformation demonstrated by the sum of imbalance depicted in Fig. 9, where the Walsh transforms use

$$f(x) = M \cdot y_{i\in\{0,1,2,3\}}(x),$$

for x of each y_i computed from four subkeys, the 9^{th} to 12^{th} subkeys ($0x$88, $0x$99, $0x$AA, $0x$BB) of the first round in this experiment, and a 32×32 binary invertible matrix M. Unlike in the case of Fig. 7b of a key leakage from the linear and non-linear transformations, this shows a key leakage from linear transformations without non-linear transformations. We can see that linear transformations

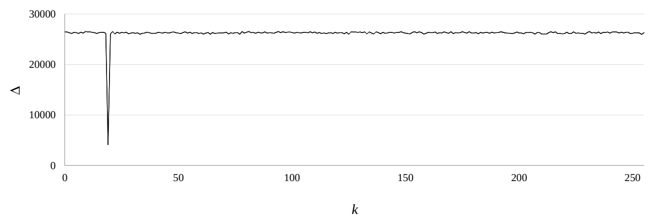


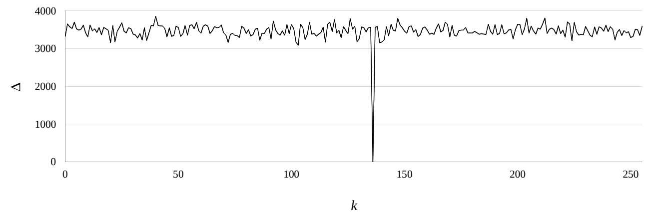
Fig. 8: Sum of the imbalance for all key candidates in the final round input.

with M can hide three subkeys $0x88$, $0x$AA, and $0x$BB, but expose one subkey $0x99$ from $y_1(x)$. This gives us two facts. First, as we will show later in this section, linear transformations produce well-balanced output with an overwhelming probability, but this is not always guarantee a reliable protection on secret keys. Second, the correct key can be recovered by the Walsh transforms even if its sum of imbalances is 0 indicating no correlation because it is distinguishable.
TABLE 1 and Fig. 10 show our experimental results of linear transformations on $y_{i \in \{0,1,2,3\}}(x)$ using 1000 randomly generated invertible matrices. For HW(ω) = 1, $Wf_i(\omega)$ = 0 with approximately 99.7% and 0.3 % of $Wf_i(\omega) = 256$; the average of $|W_{f_i}(\omega)|$ is approximately 0.7. We will proof later there is no other $Wf_i(\omega)$ values. Here, both cases (0 and 256) will lead to two different types of key leakages due to the distinguishable Walsh transform value and the noticeably high correlation coefficient, as pointed out previously. For there are 8 values of $\omega \in$ GF(2^8) such that HW(ω) = 1 and y_0 - y_3 output 32-bit values, 1024 Wf_i will be tested to see if there exists a key leakage from the linear transformation using a given matrix M. Consequently, there probably exist about 3 peaks of the correct key distinguishable from wrong key candidates, and the 3 peaks can reveal 1 to 3 subkeys. Each of y_0, y_1, y_2, and y_3 shows around 1/2 probability of $\Delta^f_{k^c} = 0$, and only about 5% of matrices do not leak any subkeys after linear transformations, where k^c means the correct key. In most cases, 1 to 3 out of four subkeys are shown to be exposed.
From now on, we are going to analyze this problematic characteristic of the linear transformations that produce extreme Wf_i values of 0 or 256. The first thing we want to investigate is whether the invertible matrix is responsible for this matter.

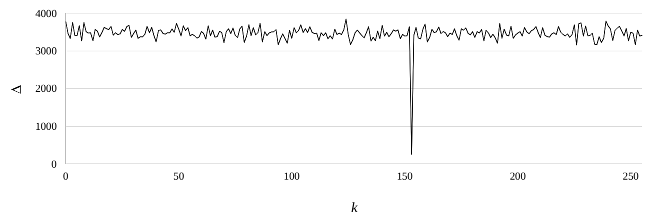
3.2 Analysis of Block Invertible Square Matrix

In [11], the authors choose M as a non-singular matrix with submatrices of full rank with a reference to [44] for maximizing information diffusion. To begin with, we briefly review the definition of a block invertible square matrix.

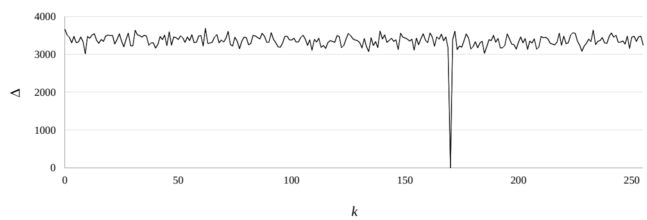
Definition 3. *If all the blocks $B_{i,j}$ in a block matrix ${}^{n}_{m}M[{}^{p}B]$ are invertible, matrix M is called an (m, n, p) block invertible matrix. Furthermore, if $m = n$,*



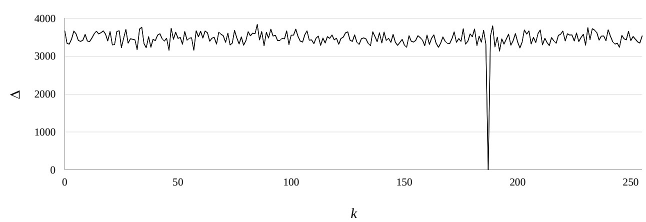
(a) On $M \cdot y_0(x)$



(b) On $M \cdot y_1(x)$



(c) On $M \cdot y_2(x)$



(d) On $M \cdot y_3(x)$

Fig. 9: Sum of the imbalance of $W_{f_i}(\omega)$ for all key candidates on each $y_{i\in\{0,1,2,3\}}(x)$ with only linear transformations.

and M is invertible then M is called an $(m,\ p)$ block invertible square matrix, where ${}^{n}_{m}M[{}^{p}B]$ denotes an $n \times m$ matrix M with nm/p^2 blocks (submatrices), and $B_{i,j}$ denotes the block in row i and column j of blocks [44].

Generating $(n,\ 2)$ block invertible square matrices begins with a $(2,\ 2)$ block invertible square matrix and extends by $(4,\ 2)$, $(6,\ 2)$, $\ldots$, and repeats it $(n-2)/2$

Table 1: Experimental results of linear transformations with 1000 randomly generated block invertible matrices. k^c: correct key.

	Vectors to be transformed			
Number of	y_0	y_1	y_2	y_3
$Wf_i(\omega) = 0$	255,206	255,205	255,309	255,203
$Wf_i(\omega) = 256$	794	795	691	797
$\Delta^f_{k^c} = 0$	475	489	520	464
$\Delta^f_{k^c} = 256$	333	307	316	343
$\Delta^f_{k^c} = 512$	132	144	122	146
$\Delta^f_{k^c} > 512$	60	60	42	47

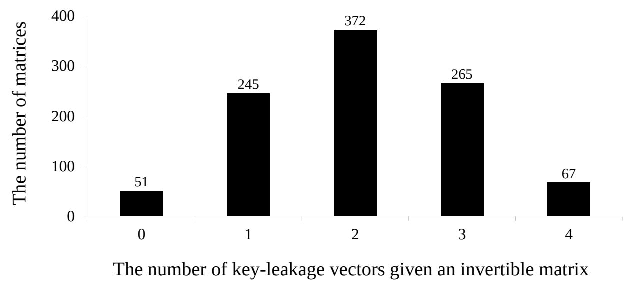


Fig. 10: The number of block invertible matrices (y-axis) vs. the number of key-leakage vectors among $y_{i\in\{0,1,2,3\}}$ given a block invertible matrix (x-axis).

times. The important point over here is that every 2×2 submatrix in a (n, 2) block invertible square matrix should be invertible by the definition and all 2×2 invertible matrices in GF(2) are as follows:

$$\begin{vmatrix}1 & 0\\0 & 1\end{vmatrix} \quad \begin{vmatrix}1 & 1\\1 & 0\end{vmatrix} \quad \begin{vmatrix}0 & 1\\1 & 1\end{vmatrix} \quad \begin{vmatrix}0 & 1\\1 & 0\end{vmatrix} \quad \begin{vmatrix}1 & 1\\0 & 1\end{vmatrix} \quad \begin{vmatrix}1 & 0\\1 & 1\end{vmatrix}$$

At a glance, the number of 1s in the 4 out of 6 matrices is greater than 0s. By the principle of constructing a block invertible square matrix, the HW of each row and column in an (n, 2) block invertible matrix will be greater than $n/2$. For example, let's assume that a (4, 2) matrix is initialized with

$$\begin{vmatrix}1 & 0\\0 & 1\end{vmatrix},$$

then its resulting matrix will be

$$\begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{vmatrix}.$$

In the case of an initialization with

$$\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix},$$

we will have

$$\begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{vmatrix}.$$

During the generation of a (32, 2) matrix through this process, 1s appear more frequently. We have performed the following experiment to check if this overweight HW of the invertible block square matrix is the main reason for key leakage. We randomly generated a balanced *non-invertible* 32×32 matrix M^b, such that $f(x) = M^b \cdot y_{i\in\{0,1,2,3\}}(x)$, where M^b has the HW of 16 for each row and column, and used it to compute the sum of imbalances. As shown in Fig. 11, there still exist key leakages from y_1 and y_2 with $\Delta^f_{k^c} = 256$. For this reason, we can conclude that the matrix HW itself is not the cause of key leakages from linear transformations.

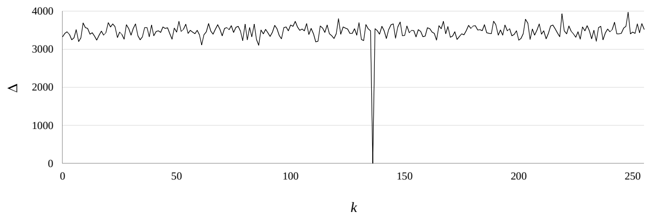
3.3 Analysis of Key-dependent Intermediate Values

The next key-leakage point to be analyzed is y. From Definition 1 and 2, we know that a balanced correlation immune function is strongly dependent on the distribution of $f_i(x) \oplus x \cdot \omega$. Since a matrix characteristic is not responsible for the key leakage as we analyzed previously, the distribution of y is convinced to mainly decide the distribution of $f_i(x) \oplus x \cdot \omega$. Here recall that given a key-dependent value $x \in \mathrm{GF}(2^8)$ and 1000 randomly generated invertible matrices M, $W_{f_i}(\omega) = 0$ with approximately 99.7% while only 0.3% of $W_{f_i}(\omega) = 256$, where $\mathrm{HW}(\omega) = 1$. The following proof explains the reason behind.

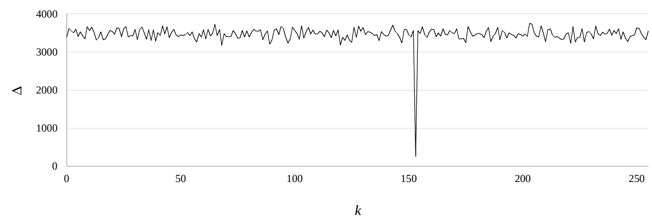
Lemma 1. *Assume that a 256×8 binary matrix* $\mathbf{H}$ *is defined as*

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & \dots \\ \vdots & \ddots & \\ h_{256,1} & & h_{256,8} \end{bmatrix}$$

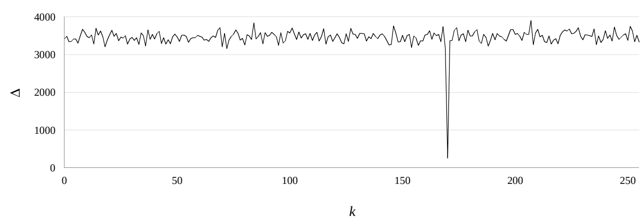
where i^{th} *row vector* $\mathbf{h}_{i,*} = \langle h_{i,1}, h_{i,2}, \dots, h_{i,8}\rangle$ *is an element of* $GF(2^8)$ *and* $\mathbf{h}_{i,*} \neq \mathbf{h}_{j,*}$ *for all* $i \neq j$. *Then the HW of XORs of arbitrary chosen column vectors from* H *is 0 or 128. In other words,* $HW(\mathbf{h}_{*,j_1} \oplus \mathbf{h}_{*,j_2} \oplus \cdots \oplus \mathbf{h}_{*,j_n}) =$ *0 or 128, where* n *is a random positive integer and* $j_i \in \{1, 2, \dots, 8\}$.



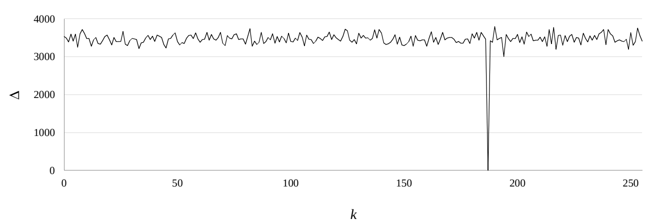
(a) On $M^b \cdot y_0(x)$



(b) On $M^b \cdot y_1(x)$



(c) On $M^b \cdot y_2(x)$



(d) On $M^b \cdot y_3(x)$

Fig. 11: Sum of the imbalance for all key candidates on each $y_{i\in\{0,1,2,3\}}(x)$ multiplied with a balanced matrix M^b.

Proof : Let $\mathcal{J}$ be a set of randomly chosen indices from $\{1, 2, \ldots, 8\}$. Note that for any duplicated indices α and α' in $\mathcal{J}$, i.e. $\alpha = \alpha'$, removing the duplicated indices from $\mathcal{J}$ makes no change to the result HW.

$$\oplus_{j\in\mathcal{J}} \mathbf{h}_{*,j} = \left(\oplus_{j\in\mathcal{J}-\{\alpha,\alpha'\}} \mathbf{h}_{*,j} \right) \oplus \mathbf{h}_{*,\alpha} \oplus \mathbf{h}_{*,\alpha'}$$

$$= \big(\oplus_{j \in \mathcal{J}-\{\alpha,\alpha'\}} \mathbf{h}_{*,j} \big) \oplus \mathbf{0} = \oplus_{j \in \mathcal{J}-\{\alpha,\alpha'\}} \mathbf{h}_{*,j}.$$

Therefore without loss of generality we can assume that $\mathcal{J}$ contains no duplicated indices and moreover $\left|\mathcal{J}\right| = n \leq 8$.
Now we can define following partitions of indices:

$$\mathcal{I}_{b_1,b_2,\ldots,b_n} = \{\ell \in \mathcal{I} | h_{\ell,j_i} = b_i \text{ for all } j_i \in \mathcal{J}\},$$

where $\mathcal{I} = \{1, 2, \ldots, 256\}$, and $b_i \in \{0, 1\}$. Here all $\mathcal{I}_{b_1,b_2,\ldots,b_n}$ are disjoint to the others and $\cup \mathcal{I}_{b_1,b_2,\ldots,b_n} = \mathcal{I}$. To complete the proof, we need that for any choice of b_i's, $\left|\mathcal{I}_{b_1,b_2,\ldots,b_n}\right| = 256/2^n = 2^{8-n}$. This can be shown easily as followings. Suppose that $\left|\mathcal{I}_{b_1,b_2,\ldots,b_n}\right| = t > 2^{8-n}$. It means that there are t row vectors in $\mathbf{H}$ satisfying the condition j_i-th bit of the vector equals to b_i. In other words, n bits are determined by choice of b_i's and only $8-n$ bits are remained free. From the condition of t is larger than 2^{8-n} and the pigeon hole principle in mathematics, there must exist at least two indices ℓ and ℓ' in $\mathcal{I}_{b_1,b_2,\ldots,b_n}$, where all bits of $\mathbf{h}_{\ell,*}$ are completely same to the bits of $\mathbf{h}_{\ell',*}$. It contradicts to the assumption $\mathbf{h}_{i,*} \neq \mathbf{h}_{j,*}$ for any $i \neq j$.
From the definition of HW, we can deduce $HW(\oplus_{j \in \mathcal{J}} \mathbf{h}_{*,j})$ is summation of $\left|\mathcal{I}_{b_1,b_2,\ldots,b_n}\right|$ where $\oplus_{i=1,\ldots,n} b_i = 1$.

$$\begin{aligned} HW(\oplus_{j \in \mathcal{J}} \mathbf{h}_{*,j}) &= \Sigma_{\oplus_{i=1,\ldots,n} b_i = 1} \left|\mathcal{I}_{b_1,b_2,\ldots,b_n}\right| \\ &= \Sigma_{\oplus_{i=1,\ldots,n} b_i = 1} 2^{8-n} = \Sigma_{2^{n-1}} 2^{8-n} \\ &= 2^{n-1} \cdot 2^{8-n} = 2^7 = 128. \end{aligned}$$

Note that if $\mathcal{J}$ is empty after de-duplication then the final HW becomes 0. It concludes the proof of lemma.

Note that $W_{f_i}(w)$ is defined as $\sum_{x \in GF(2^8)} (-1)^{f_i(x) \oplus w \cdot x} = \sum_{x \in \{0,1\}^8} (-1)^{M_{i,*} \cdot y(x) \oplus w \cdot x}$, where $M_{i,*}$ is i^{th} row of the matrix M and $y(x)$ is one of $y_0(x)$ - $y_3(x)$ depending on the target subkey. For convenience, let $y(x) = y_0(x)$, a 32×1 matrix $[2 \cdot x \;\; x \;\; x \;\; 3 \cdot x]^T$. If we define $\mathbf{Y}(x)$ as a 32×256 matrix $[2 \cdot \mathbf{H} \;\; \mathbf{H} \;\; \mathbf{H} \;\; 3 \cdot \mathbf{H}]^T$, where the $\mathbf{H}$ is the matrix defined at the lemma 1, it is easy to show that each column vector of $2 \cdot \mathbf{H}$ or $3 \cdot \mathbf{H}$ can be defined with XORs of some column vectors of $\mathbf{H}$ based on the property of $GF(2^8)$. Then the above equation can be re-written as

$$\sum_{j=\{1,2,\ldots 256\}} (-1)^{B_j(M_{i,*} \cdot \mathbf{Y}(x) \oplus (w \cdot \mathbf{H}^T))},$$

where $B_j(v)$ means the j^{th} bit of the vector v. Since the exponents of the equation can have only two values 0 or 1, the summation over $\{1, 2, \ldots, 256\}$ can be re-written with the number of exponents which are 1.

$$W_{f_i}(w) = 256 - (2 \times HW(M_{i,*} \cdot \mathbf{Y}(x) \oplus (w \cdot \mathbf{H}^T)))$$

Note that all row vectors of the matrix $\mathbf{Y}(x)$ is represented by XORing of column vectors of $\mathbf{H}$. Therefore $M_{i,*} \cdot \mathbf{Y}(x) \oplus (w \cdot \mathbf{H}^T)$ can be also represented by XORing

of column vectors of $\mathbf{H}$. From the lemma 1, it deduces that the HW of $M_{i,*} \cdot \mathbf{Y}(x) \oplus (w \cdot \mathbf{H}^T)$ is 0 or 128. Finally, $W_{f_i}(w) = 256 - (2 \times HW(M_{i,*} \cdot \mathbf{Y}(x) \oplus (w \cdot \mathbf{H}^T)))$ becomes 256 or 0. What is remarkable point over here is that the probability of $W_{f_i}(w) = 256$ is very small but not zero. Specifically, it happens when all column indices of $\mathbf{H}$ are canceled each other when the summation is computed with the randomly chosen matrix M.
As mentioned already, our experiment showed that $W_{f_i}(w) = 256$ with 0.3% in the calculation with the correct key, while the wrong key candidates produced $|W_{f_i}(\omega)| = 56$ at maximum and 13.13 in average. For this reason, 1024 tests of $W_{f_i}(w)$ given a matrix M are likely to cause key leakages with overwhelming probability. Based on these findings, we propose a novel approach for a secure implementation in the following section.

4 Proposed Method

Our analysis in the previous section shows that a balanced distribution of the intermediate values is the main reason behind the key leakage. In order to make it unbalanced, our key idea is to insert random bytes in the intermediate values before linear transformations. From now on, we answer to the following questions.

- Where is the appropriate position for random bytes to be inserted?
- How many random bytes must be inserted?
- How to apply this key idea to a new WB-AES implementation?

We begin with an analysis of the inserting position and the required number of random bytes to be inserted.

4.1 Inserting A Random Byte in the Intermediate Values

First, we will insert a random byte at a particular position in the 4-byte intermediate value $y_{i\in\{0,1,2,3\}}(x)$ and then perform a linear transformation with a 40×40 binary block invertible matrix M^* to check if any key leakage occurs. Among the five inserting positions ρ_1 - ρ_5 of y_0, for example,

$$\left[\rho_1 \; 2 \cdot x \; \rho_2 \; x \; \rho_3 \; x \; \rho_4 \; 3 \cdot x \; \rho_5\right]^T$$

we select ρ_i, where $i \in [1, 5]$, and then insert different $\gamma \in_R \mathrm{GF}(2^8)$ at ρ_i for each $x \in \mathrm{GF}(2^8)$. Let $y_0^*(x)$ denote $y_0(x)$ after the random byte insertion, and $f^*(x)$ denote $y_0^*(x) \cdot M^*$. Then we can define the Walsh transforms with respect to f^*:

$$W_{f_i^*}(\omega) = \Sigma_{x\in\{0,1\}^8}(-1)^{f_i^*(x)\oplus x\cdot\omega}$$

for 40 Boolean functions

$$f^*_{i\in\{1,\ldots,40\}}(x) : \{0,1\}^8 \to \{0,1\}.$$

With 1000 randomly generated M^*, we computed $W_{f_i^*}(\omega)$ with respect to y_0 - $y_3(x)$ for each position ρ_i. As a result, TABLE 2 gives us that the correct key results in $W_{f_i^*}(\omega) = 0$ with approximately 5% and the average $|W_{f_i^*}(\omega)|$ is about 12.7. Recall that, without the random byte insertion, $W_{f_i}(\omega) = 0$ with approximately 99.7% and the average of $|W_{f_i}(\omega)|$ is approximately 0.7.

	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5
% of $W_{f_i^*}(\omega) = 0$	5.05 (0.03)	5.06 (0.07)	4.93 (0.05)	5.0 (0.05)	5.04 (0.04)
Average of $\lvert W_{f_i^*}(\omega)\rvert$	12.73 (0.02)	12.75 (0.01)	12.76 (0.01)	12.73 (0.01)	12.76 (0.01)
Similarity with $W_{f_i^\gamma}$	> 0.999				

Table 2: $W_{f_i^*}$ after inserting a random byte at each inserting position (the standard deviation in parenthesis), and the cosine similarity of the distributions between $W_{f_i^*}$ and $W_{f_i^\gamma}$.

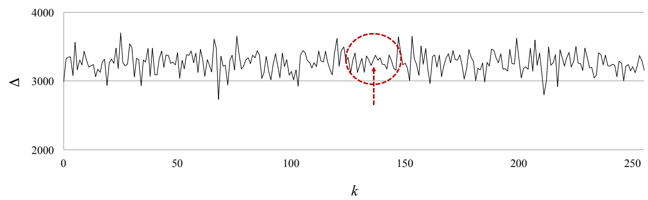
To see the effect of the random byte insertion, we conducted an additional experiment as follows.

1. Let $y^\gamma(x) = [\gamma 1\ \gamma 2\ \gamma 3\ \gamma 4\ \gamma 5]^T$ for each $x \in \mathrm{GF}(2^8)$. In other words, replace all the key-dependent intermediate values with random bytes.
2. $f^\gamma(x) = M^* \cdot y^\gamma(x)$.
3. Repeat step (1) - (2) with 1000 random M^* matrices, and accumulate the number of occurrences of each value of $W_{f_i^\gamma}(\omega)$.
4. Compute % of $W_{f_i^\gamma}(\omega) = 0$ and the average $|W_{f_i^\gamma}(\omega)|$.
5. Compute the cosine similarity between the distributions of $W_{f_i^\gamma}(\omega)$ and $W_{f_i^*}(\omega)$ for each of y_0 - y_3 and for each ρ_i.

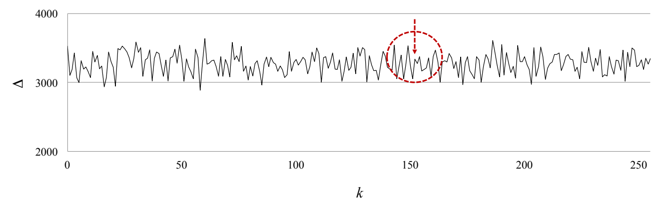
As a result, we have $W_{f_i^\gamma}(\omega) = 0$ with approximately 5%, the average $|W_{f_i^\gamma}(\omega)|$ is approximately 12.74, and the cosine similarity between their distributions is always larger than 0.999. The cosine similarity larger than 0.99 means they show very similar distribution. We note that the cosine similarity between the distributions of $W_{f_i^\gamma}(\omega)$ and $W_{f_i}(\omega)$ is about 0.25.
In order to visualize this effect of inserting a random byte, we select ρ_5 and calcalculate the sum of the imbalances of $W_{f_i^*}(\omega)$ for each key candidate with ω such that $\mathrm{HW}(\omega) = 1$ as follows:

$$\Delta^{f^*}_{k\in\{0,1\}^8} = \sum_{\omega=1,2,\ldots,128} \sum_{i=1,\ldots,40} |W_{f_i^*}(\omega)|,$$

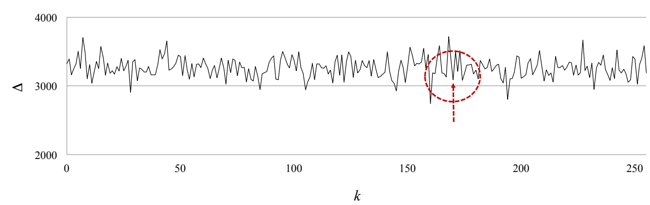
Fig. 12 shows $\Delta^{f^*}_{k\in\{0,1\}^8}$ and we can see that the correct subkeys $0x88$ - $0x\mathrm{BB}$ are no longer distinguishable from other candidates.
In addition, it is noticeable that inserting more than one random byte in the intermediate values does not increase the imbalance; they show a similar level of the imbalance of the one-byte insertion. Thus, we can conclude that inserting a random byte into anywhere among ρ_1-ρ_5 can effectively prevent the key leakage



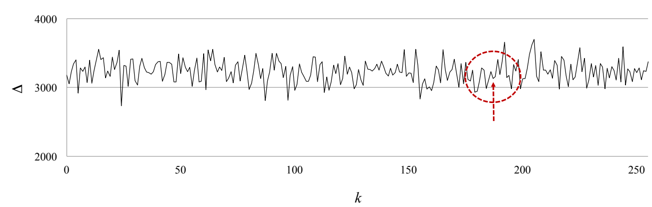
(a) On $M^* \cdot y_0(x)$



(b) On $M^* \cdot y_1(x)$



(c) On $M^* \cdot y_2(x)$



(d) On $M^* \cdot y_3(x)$

Fig. 12: Sum of the imbalance of $W_{f_i^*}(\omega)$ for all key candidates. Red arrow: the correct key.

after the linear transformation. Based on this observation, we explain how to implement a key leakage preventive WB-AES algorithm with 128-bit key size in the following.

4.2 Secure WB-AES Implementation

In this section, we explain how to apply the random byte insertion before linear transformations in order to protect the first and final rounds of WB-AES. Note that any attack on the inner rounds which require large complexity of 2^{128} key candidates is not practical. Our WB-AES uses the table name (number) used in previous studies to minimize confusion but only mark * in its superscript at the beginning of the name if we modify its structure or size. Then, a set of lookup tables in our proposed WB-AES is basically composed of four types: **TypeII*, **TypeIII*, **TypeIV* and *TypeV*.

***TypeII**. In our WB-AES, we pick $\gamma \in_R$ GF(2^8) for each $x \in$ GF(2^8), and define $Ty_i^*(x)$:

$$Ty_0^*(x) = [2x\ x\ x\ 3x\ \gamma \in_R GF(2^8)]^T$$
$$Ty_1^*(x) = [3x\ 2x\ x\ x\ \gamma \in_R GF(2^8)]^T$$
$$Ty_2^*(x) = [x\ 3x\ 2x\ x\ \gamma \in_R GF(2^8)]^T$$
$$Ty_3^*(x) = [x\ x\ 3x\ 2x\ \gamma \in_R GF(2^8)]^T.$$

The next step is to perform linear transformations on $Ty_i^*(x)$ with M^* and apply non-linear transformations as illustrated in Fig 13 and Algorithm 1. Note that the input decoding from round 2 takes into account the index change due to ShiftRows. During the execution of WB-AES, its lookup values will contain 5-byte outputs including an encoded random byte and thus the intermediate values after visiting **TypeII* will be store in a 5×4×4 matrix.

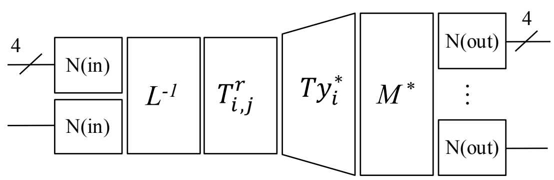


Fig. 13: A schematic diagram of **TypeII* generation.

***TypeIII**. During the **TypeIII* generation shown in Fig. 14, we perform the inverse linear transformation with $(M^*)^{-1}$ and apply the linear transformation with L. Now we need 180 (=9×5×4) 8 × 8 binary invertible matrices $L^r_{i,j}$, where $r \in [1, 9]$, $0 \leq i \leq 4$, and $0 \leq j \leq 3$ because **TypeIII* gives an additional byte in the intermediate values. Algorithm 2 describes the generation of **TypeIII*. After visiting **TypeIII*, we have the intermediate values in a 5×4×5 matrix.

***TypeIV**. **TypeIV_II* and **Type IV_III* are used to XOR the lookup values of **TypeII* and **TypeIII*, respectively. **TypeIV_II* combines 5×4×4 intermediate values into a 5×4 state matrix and **TypeIV_III* combines 5×4×5 intermediate values into a 5×4 state matrix. We extract the first four rows of this resulting 5×4 intermediate matrix into a 4×4 state matrix and use it as the next round input. Fig. 15 and Fig. 16 simplify the lookup flows of **TypeIV* following **TypeII* and **TypeIII*.

for r = *1 to 9* **do**
  for i = *0 to 3* **do**
    for j = *0 to 3* **do**
      for p' = *0 to 255* **do**
        if $r == 1$ **then**
          $p = p'$;
        else
          $\{i,\, j'\} = \{i,\, (j+i) \bmod 4\ \}$
          $p' = \text{inverse-non-linear-transform}(p')$
          $p = (L^{r-1}_{i,j'})^{-1} \cdot p'$
        end
        $x = T^{r}_{i,j}(p)$
        $y = Ty_i(x)$
        $y^* = y \mid\mid \gamma \in_R \mathrm{GF}(2^8)$
        $f^* = M^* \cdot y^*$
        for k = *0 to 4* **do**
          $f^*[k] = \text{non-linear-transform}(f^*[k])$
          $^*TypeII^{r}_{i,j,p',k} = f^*[k]$
        end
      end
    end
  end
end

Algorithm 1: **TypeII* Generation.

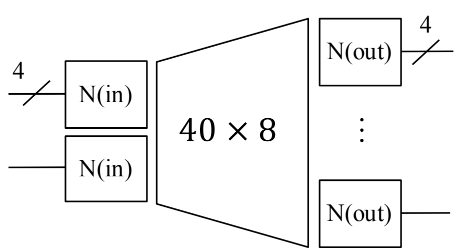


Fig. 14: A schematic diagram of **TypeIII* generation.

```
for r = 1 to 9 do
    for i = 0 to 4 do
        for j = 0 to 3 do
            for p' = 0 to 255 do
                p = inverse-non-linear-transform(p′)
                g*[5] = {0, 0, 0, 0, 0}
                g*[i] = p
                g* = (M*)^-1 · g*
                for k = 0 to 4 do
                    g* = L^r_{k,j} · g*
                    g*[k] = non-linear-transform(g*[k])
                    *TypeIII^r_{i,j,p′,k} = g*[k]
                end
            end
        end
    end
end
```

Algorithm 2: **TypeIII* Generation.

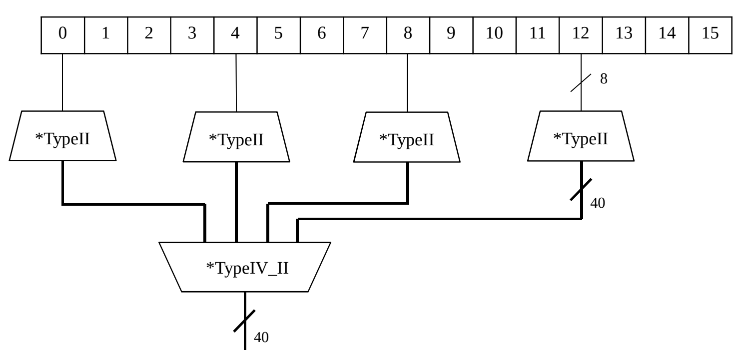


Fig. 15: **TypeII* and **TypeIV_II* lookups.

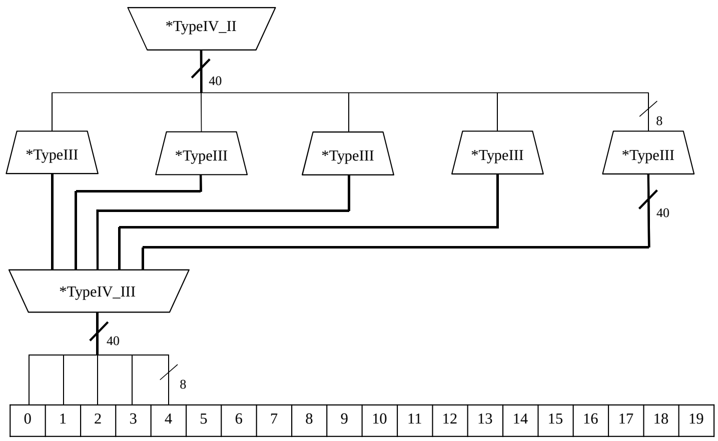


Fig. 16: **TypeIII* and **TypeIV_III* lookups.

By using **TypeII*, **TypeIV_II*, **TypeIII*, and **TypeIV_III* in the first round, we can protect against gray-box attacks using SubBytes outputs on **TypeII* lookup values. Next, we explain how to protect the first round output and final round input.

4.3 More Protection and Our Variants

As explained in Section 2, there can be three key leakage points in WB-AES. It is noticeable that the higher the security level, the higher the cost for a white-box cryptographic implementation. In this point of view, the authors in [22] proposed three different variants of masked WB-AES implementations, aptly named CASE 1, 2 and 3, depending on the explained attack complexity.

- CASE 1: Protecting first round *TypeII* outputs
- CASE 2: CASE 1 + protecting the whole final round
- CASE 3: CASE 2 + protecting the whole first round.

So far, we have introduced our method to protect *TypeII* outputs in the first round. Our WB-AES of inserting a random byte before linear transformations protects **TypeII* and **TypeIV_II* outputs, but does not provide any protection on each round output since combining the intermediate values by **TypeIV_III* cancels out the linear transformation by M^* and apply another linear transformation by L on each byte of the round output. As a result, attacks on **TypeIV_III* are still possible, and thus the next is to protect the final round input and the first round output.
Suppose that we apply the same technique of a random byte insertion using a 16×16 invertible matrix L^*. This choice leads to the following additional costs:

- The **TypeIII* size increases.

- The $^*TypeIV\_III$ size increases.
- A temporary storage for intermediate values increases.
- The number of table lookup increases.
- A 16-bit to 8-bit mapping table is needed to cancel out L^* and apply L corresponding to the $^*TypeII$ input decoding in the next round; the mapping table should be protected by a secure non-linear transformation with 8-bit to 8-bit random bijections.

Due to these additional costs, it is cost effective to partially adapt 8-bit to 8-bit random bijections at the end of $^*TypeIV\_III$ like $TypeIV\_IIIC$ used in [22], rather than inserting a random byte in $^*TypeIII$.

Let us have three variants of CASE 1, 2, and 3 like in the case of the masked WB-AES variants. Then we already have the CASE 1 implementation including $^*TypeII$, $^*TypeIII$, and $^*TypeIV$ in the first round.

Next, we describe how to implement the CASE 2 implementation. We append a postfix 'N' to the end of $^*TypeIV\_III$ indicating the use of 8-bit to 8-bit random bijections for non-linear transformations. Fig. 17 shows a diagram of $^*TypeIV\_IIIN$, and Fig. 18 provides a simple lookup flow of $^*TypeIII$, $^*TypeIV\_III$ and $^*TypeIV\_IIIN$ to produce the final round input at the ninth round. Then the inverse non-linear transformation in the $^*TypeVN$ generation must be the corresponding inverse 8-bit to 8-bit bijections as shown in Fig. 19

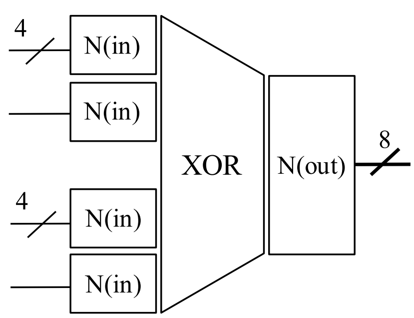


Fig. 17: A schematic diagram of $^*TypeIV\_IIIN$ generation.

In the CASE 3 implementation, $^*TypeIV\_IIIN$ is used in the first round and thus $^*TypeIIN$ in the second round must have the corresponding 8-bit to 8-bit inverse bijections as shown in Fig. 20. Note that a random byte insertion is not applied to $^*TypeIIN$ because this gives the inner round intermediate values which require an attacker to completely know the first round key. Then the table lookup sequences for each variant are given in Fig. 21.

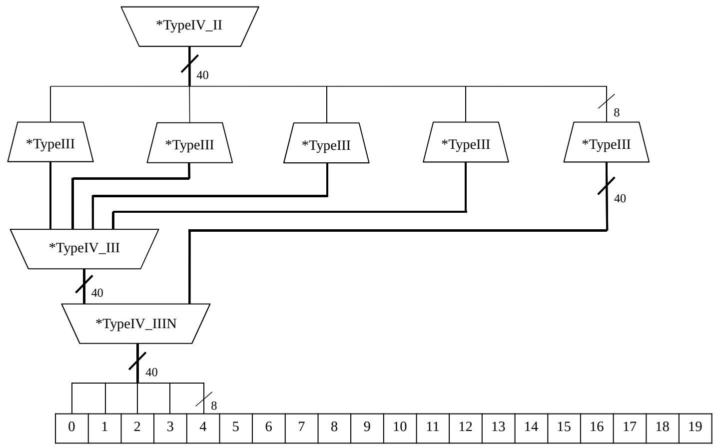


Fig. 18: * *TypeIV_III* and * *TypeIV_IIIN* lookups.

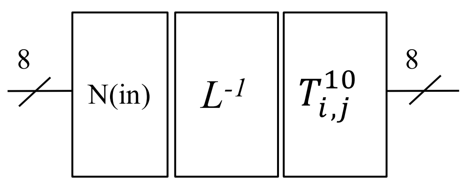


Fig. 19: A schematic diagram of * *TypeVN* generation.

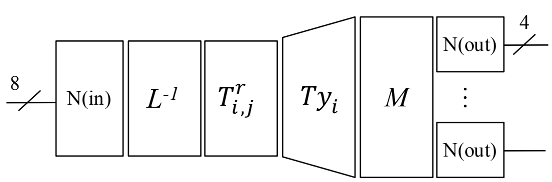
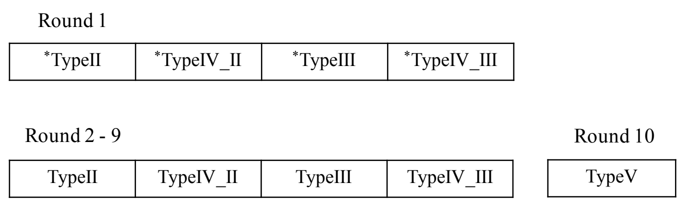
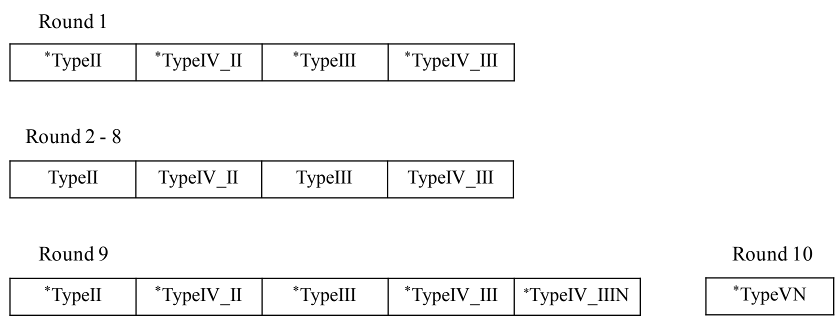


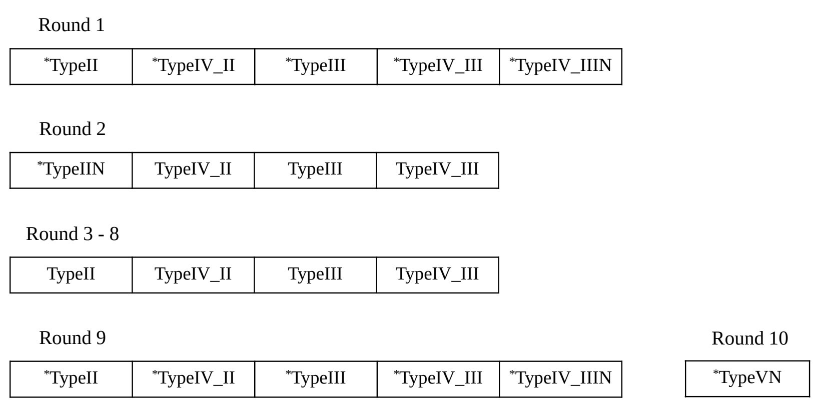
Fig. 20: A schematic diagram of * *TypeIIN* generation.



(a) CASE 1



(b) CASE 2



(c) CASE 3

Fig. 21: Table lookup sequences in the CASE 1-3 implementations.

5 Evaluation

In this section, we evaluate the security and performance of our proposed method. To verify the key leakage protection, we use the Walsh transform and DCA attacks. The performance evaluation compares the lookup table size and the number of lookups of WB-AES with the initial WB-AES and the masked WB-AES implementations [11][22].

5.1 Security

We demonstrate the prevention of key leakages from **TypeII* and **TypeIV_IIIN* in our WB-AES implementation. For the 8^{th} subkey and attacker's hypothetical SubBytes output x, let 40 Boolean functions $f^*_{i\in\{1,\ldots,40\}}(x)$: $\{0,1\}^8 \rightarrow \{0,1\}$ denote **TypeII* lookup values in the first round. Then we have the sum of imbalance:

$$\Delta^{f^*}_{k\in\{0,1\}^8} = \sum_{\omega=1,2,\ldots,128} \sum_{i=1,\ldots,40} |W_{f_i}(\omega)|,$$

where $\mathrm{HW}(\omega) = 1$. We can see there is no longer distinguishable peak at the right subkey (*0x*88) in Fig. 23a. This gives us that power analysis such as DPA and CPA as well as DCA attacks using the SubBytes output in the first round will not be successful on our WB-AES implementation. In order to show the DCA result, we collected 1,000 software execution traces by DCA techniques and then performed a CPA attack, and Fig. 22 shows memory read/write accesses during the execution of our WB-AES. As a result, DCA was not able to recover any key from our WB-AES as shown in TABLE 3, and the last two rows provide the highest correlation coefficient and the correct key's correlation coefficient values, respectively. We know that the coefficient value around 0.2 of the correct key means that there is no meaningful correlation between correct hypothetical values and lookup values.

Fig. 22: Visualization of a software execution trace of our WB-AES implementation. Green: read memory addresses, Red: write memory address.

Table 3: DCA ranking for our WB-AES implementation when conducting mono-bit CPA on the SubBytes output in the first round with 1000 software traces.

TargetBit \ SubKey	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	211	105	33	180	126	226	93	225	251	176	67	230	129	130	150	172
2	256	150	250	13	126	20	212	25	110	200	162	71	77	159	76	192
3	11	59	28	154	232	139	86	67	171	85	205	197	106	114	96	245
4	71	161	52	4	138	249	37	19	36	189	220	121	34	25	31	101
5	219	41	249	237	172	93	112	255	199	191	15	103	140	90	94	158
6	154	246	143	53	135	159	48	229	99	5	32	98	251	149	207	179
7	165	167	237	50	238	185	156	208	127	54	56	157	145	59	70	251
8	253	199	248	173	122	208	186	208	129	56	43	110	113	202	69	92
Highest coeff.	0.33	0.29	0.29	0.28	0.29	0.30	0.31	0.28	0.29	0.30	0.30	0.35	0.30	0.31	0.33	0.30
Key's coeff.	0.22	0.19	0.20	0.25	0.16	0.21	0.20	0.21	0.19	0.24	0.23	0.18	0.19	0.21	0.21	0.17

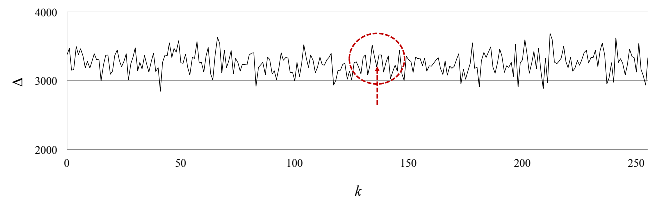
In addition, we conducted the same experiment as in Sectin 2 with the attacker's hypothetical input value x and 8 Boolean functions $f_{i\in\{1,\ldots,8\}}(x)$: $\{0,1\}^8 \rightarrow \{0,1\}$ for the encoded final round input so that we verify the key leakage prevention at **TypeIV_IIIN* outputs. As a result, there is no peak at the correct key (*0x*13) as shown in Fig. 23b.

5.2 Performance

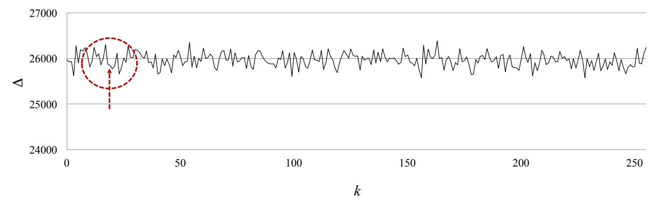
Our performance evaluation in this section compares the total lookup table size and the number of table lookups with the initial WB-AES and masked WB-AES implementations [11][22].

Table size. Before table size comparsions, we need to mention two things: 1) Because there is no variant in the initial WB-AES implementation, we denote it CASE 1 (without CASE 2 and CASE 3) for convenient comparison with the masked WB-AES and our WB-AES implementations; 2) It is necessary to recalculate the masked WB-AES table size and table lookups for the pair comparison because the authors calculated for the case of their masking technique applied to both outer and inner rounds while we applied our technique to only outer rounds.
The table size of CASE 1 is 544,768 bytes in total and this is calculated as follows:

- **TypeII*: 4×4×256×5 = 20,480
- **TypeIV_II*: 5×4×3×2×128 = 15,360
- **TypeIII*: 5×4×256×5 = 25,600
- **TypeIV_III*: 5×4×4×2×128 = 20,480
- *TypeII*: 8×4×4×256×4 = 131,072
- *TypeIII*: 8×4×4×256×4 = 131,072



(a) On $^*TypeII$ lookup values



(b) On $^*TypeVN$ input values

Fig. 23: Sum of the imbalance for all key candidates in our WB-AES.

- $TypeIV$: 2×8×4×4×3×2×128 = 196,608
- $TypeV$: 4×4×256 = 4,096 bytes.

In CASE 2, the total size is 1,874,944 bytes computed by rewriting the table size in the final round as follows:

- $^*TypeII$: 4×4×256×5 = 20,480
- $^*TypeIV\_II$: 5×4×3×2×128 = 15,360
- $^*TypeIII$: 5×4×256×5 = 25,600
- $TypeIV\_III$: 5×4×3×2×128 = 15,360
- $^*TypeIV\_IIIN$: 5×4×256×256 = 1,310,720.

Lastly, the total size of CASE 3 is 3,180,544 bytes computed by rewriting the $^*TypeIV\_III$ size in the first round as follows:

- $^*TypeIV\_III$: 5×4×3×2×128 = 15,360
- $^*TypeIV\_IIIN$: 5×4×256×256 = 1,310,720.

TABLE 4 shows the comparisons with the previous implementations. Our WB-AES implementation reduces approximately 33% of the memory requirements of masked WB-AES in CASE 2 and CASE 3.

Table lookup. The number of table lookups of CASE 1 is 2124 in total and this is calculated as follows:

- $^*TypeII$: 4×4 = 16
- $^*TypeIV\_II$: 5×4×3×2 = 120
- $^*TypeIII$: 5×4 = 20
- $^*TypeIV\_III$: 5×4×4×2 = 160

	CASE 1	CASE 2	CASE 3
Initial WB-AES	520,192	-	-
Masked WB-AES	548,864	2,774,688	4,763,648
Our WB-AES	544,768	1,874,944	3,180,544

Table 4: Table size comparison (byte) with the previous WB-AES implementations.

- *TypeII*: $8 \times 4 \times 4 = 128$
- *TypeIII*: $8 \times 4 \times 4 = 128$
- *TypeIV*: $2 \times 8 \times 4 \times 4 \times 3 \times 2 = 1536$
- *TypeV*: $4 \times 4 = 16$.

In addition, the numbers of lookups in CASE 2 and CASE 3 are 2196 and 2176. Here, the use of 8-bit to 8-bit bijections in the XOR tables slightly reduces the number of lookups. Compared to the masked implementations, our WB-AES requires slightly less table lookups.

	CASE 1	CASE 2	CASE 3
Initial WB-AES	2,032	-	-
Masked WB-AES	2,176	2,288	2,256
Our WB-AES	2,124	2,196	2,176

Table 5: Table lookup comparison with the previous WB-AES implementations.

In summary, our WB-AES implementation can prevent key leakages with at most 33 percent less memory requirements than the masked implementations. We remark that the white-box cryptographic applications have been limited due to the high memory requirements that mainly determine the cost. For this reason, the reduced table size can contribute to the widespread use of white-box cryptography.

6 Conclusion and Discussion

In this paper, our analysis shows the well distributed intermediate values cause the key leakage after linear transformation in the white-box implementation. Based on this analysis, we introduce a novel approach of inserting a random byte into intermediate values before linear transformations to prevent key leakages and propose a WB-AES implementation using this principle. To protect the outer round output, we partially apply non-linear transformations using 8-bit to 8-bit random bijections and provide three different variants of WB-AES depending on the security levels. In conclusion, our WB-AES, as a non-masking

implementation, does not require any static or dynamic random source and decreases the memory requirement by at most 33% with slightly reduced table lookups, compared to the masked WB-AES.
As mentioned already, there are many proprietary white-box cryptographic solutions available at the market. Because there are a number of issues, including vulnerabilities to highly skilled white-box attacks, white-box cryptography is being combined with additional protection, and commercial companies do not need to disclose their technology for this part. Thus, much research on defense techniques has not been made publicly, compared with attacks on white-box cryptography. We think this study can provide useful information in this regard. The following aspects should be improved or added for more complete white-box cryptographic implementation. First, the memory requirement of the lookup table has to be improved for the widespread use of white-box cryptography. Second, countermeasures on gray- and white-box attacks ought to be combined together at low cost. We know that the memory and computational costs are definitely a disadvantage of white-box cryptography, but it is certainly providing the most secure way to protect a secret key in the software-based implementation. Provided that the above challenges are addressed, white box cryptography will be widely used for security enhancement in many low-cost devices in the future.

References

1. Akkar, M.L., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems. pp. 309–318. CHES '01, Springer-Verlag, London, UK, UK (2001)
2. Axsan white-box cryptographic solution.: `https://www.arxan.com/technology/white-box-cryptography/`
3. Banik, S., Bogdanov, A., Isobe, T., Jepsen, M.B.: Analysis of Software Countermeasures for Whitebox Encryption. vol. 2017, pp. 307–328 (2017), `http://tosc.iacr.org/index.php/ToSC/article/view/596`
4. 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)
5. 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)
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)
7. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.:
8. 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)

9. Bringer, J., Chabanne, H., Dottax, E.: White Box Cryptography: Another Attempt. IACR Cryptology ePrint Archive 2006, 468 (2006)
10. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Fast software encryption: 19th international workshop, fse 2012, washington, dc, usa, march 19-21, 2012. revised selected papers. pp. 366–384. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-34047-5_21
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)
13. Coron, J., Paar, C., Hardware, C., sebastien Coron, J., Goubin, L.: On boolean and arithmetic masking against differential power analysis (2000)
14. Gemalto white-box cryptographic solution: https://sentinel.gemalto.com/software-monetization/white-box-cryptography/
15. Golić, J.D., Tymen, C.: Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers. pp. 198–212. Springer Berlin Heidelberg, Berlin, Heidelberg (2003), http://dx.doi.org/10.1007/3-540-36400-5_16
16. 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)
17. InsideSecure white-box cryptographic solution: https://www.insidesecure.com/Products/Application-Protection/Software-Protection/WhiteBox
18. Joye, M., Paillier, P., Schoenmakers, B.: Cryptographic hardware and embedded systems – ches 2005: 7th international workshop, edinburgh, uk, august 29 – september 1, 2005. proceedings. pp. 293–308. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11545262_22
19. 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)
20. Kim, H., Hong, S., Lim, J.: A Fast and Provably Secure Higher-order Masking of AES S-box. In: Proceedings of the 13th international conference on Cryptographic hardware and embedded systems. pp. 95–107. CHES'11, Springer-Verlag, Berlin, Heidelberg (2011)
21. 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)
22. Lee, S., Kim, T., Kang, Y.: A Masked White-Box Cryptographic Implementation for Protecting Against Differential Computation Analysis. IEEE Transactions on Information Forensics and Security 13(10), 2602–2615 (Oct 2018)
23. Lee, S., Choi, D., Choi, Y.J.: Conditional Re-encoding Method for Cryptanalysis-Resistant White-Box AES. vol. 5. Electronics and Telecommunications Research Institute (Oct 2015), http://dx.doi.org/10.4218/etrij.15.0114.0025

24. 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)
25. Link, H.E., Neumann, W.D.: Clarifying Obfuscation: Improving the Security of White-box DES. In: International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II. vol. 1, pp. 679–684 Vol. 1 (2005)
26. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security) (2007)
27. Messerges, T.S.: Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings
28. Messerges, T.S.: Cryptographic Hardware and Embedded Systems — CHES 2000: Second International Workshop Worcester, MA, USA, August 17–18, 2000 Proceedings, chap. Using Second-Order Power Analysis to Attack DPA Resistant Software, pp. 238–251. Springer Berlin Heidelberg, Berlin, Heidelberg (2000), http://dx.doi.org/10.1007/3-540-44499-8_19
29. 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)
30. 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)
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)
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)
33. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A Side-channel Analysis Resistant Description of the AES S-box. In: Proceedings of the 12th international conference on Fast Software Encryption. pp. 413–423. FSE'05, Springer-Verlag, Berlin, Heidelberg (2005)
34. Prouff, E., Rivain, M.: Advances in cryptology – eurocrypt 2013: 32nd annual international conference on the theory and applications of cryptographic techniques, athens, greece, may 26-30, 2013. proceedings. pp. 142–159. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-38348-9_9
35. Rivain, M., Prouff, E.: Provably Secure Higher-order Masking of AES. In: Proceedings of the 12th international conference on Cryptographic hardware and embedded systems. pp. 413–427. CHES'10, Springer-Verlag, Berlin, Heidelberg (2010)
36. 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
37. 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)
38. Schramm, K., Paar, C.: Higher Order Masking of the AES. In: Proceedings of the 2006 The Cryptographers' Track at the RSA conference on Topics in Cryptology. pp. 208–225. CT-RSA'06, Springer-Verlag, Berlin, Heidelberg (2006)
39. Trichina, E., Seta, D., Germani, L.: Cryptographic hardware and embedded systems - ches 2002: 4th international workshop redwood shores, ca, usa, august 13–15, 2002 revised papers. pp. 187–197. Springer Berlin Heidelberg, Berlin, Heidelberg (2003), http://dx.doi.org/10.1007/3-540-36400-5_15
40. Waddle, J., Wagner, D.: Cryptographic hardware and embedded systems - ches 2004: 6th international workshop cambridge, ma, usa, august 11-13, 2004. proceedings. pp. 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-28632-5_1
41. WhiteboxCRYPTO: https://www.microsemi.com/document-portal/doc_view/135631-whiteboxcrypto-product-overview-rev4
42. 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
43. 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)
44. Xiao, J., Zhou, Y.: Generating Large Non-Singular Matrices over an Arbitrary Field with Blocks of Full Rank (2002), http://eprint.iacr.org/2002/096
45. 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)