IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 4, pp. 493–522. DOI:10.46586/tches.v2023.i4.493-522

Revisiting the Computation Analysis against Internal Encodings in White-Box Implementations

Yufeng Tang, Zheng Gong(✉), Bin Li and Liangju Zhao

School of Computer Science, South China Normal University, Guangzhou, China
yuft@m.scnu.edu.cn,cis.gong@gmail.com

Abstract. White-box implementations aim to prevent the key extraction of the cryptographic algorithm even if the attacker has full access to the execution environment. To obfuscate the round functions, Chow *et al.* proposed a pivotal principle of white-box implementations to convert the round functions as look-up tables which are encoded by random *internal encodings*. These encodings consist of a linear mapping and a non-linear nibble permutation. At CHES 2016, Bos *et al.* introduced *differential computation analysis* (DCA) to extract the secret key from the runtime information, such as accessed memory and registers. Following this attack, many computation analysis methods were proposed to break the white-box implementations by leveraging some properties of the linear internal encodings, such as Hamming weight and imbalance. Therefore, it becomes an alternative choice to use a non-linear byte encoding to thwart DCA. At CHES 2021, Carlet *et al.* proposed a structural attack and revealed the weakness of the non-linear byte encodings which are combined with a non-invertible linear mapping. However, such a structural attack requires the details of the implementation, which relies on extra reverse engineering efforts in practice. To the best of our knowledge, it still lacks a thorough investigation of whether the non-linear byte encodings can resist the computation analyses.
In this paper, we revisit the proposed computation analyses by investigating their capabilities against internal encodings with different algebraic degrees. Particularly, the algebraic degree of encodings is leveraged to explain the key leakage on the non-linear encodings. Based on this observation, we propose a new *algebraic degree computation analysis* (ADCA), which targets the mappings from the inputs to each sample of the computation traces. Different from the previous computation analyses, ADCA is a higher-degree attack that can distinguish the correct key by matching the algebraic degrees of the mappings. The experimental results prove that ADCA can break the internal encodings from degree 1 to 6 with the lowest time complexity. Instead of running different computation analyses separately, ADCA can be used as a generic tool to attack the white-box implementations.

Keywords: White-Box Implementation · Computation Analysis · Internal Encoding · DIBO Function · Boolean Function

1 Introduction

In the classical setting of cryptanalysis, a cryptographic algorithm is evaluated in a *black-box* attack context. The black-box adversary is capable of arbitrarily choosing the inputs and receiving the corresponding outputs of the cryptographic primitive. However, in practice, the adversary can access the *side-channel information* such as timing or power consumption. In this *gray-box* model, the adversary can perform *side-channel attacks*, such as *power analysis* during the execution of algorithms. For software implementations

Received: 2023-04-15 Accepted: 2023-06-15 Published: 2023-08-31

that suffer severe challenges in an untrusted execution environment, the adversary has full access to the cryptographic implementation. It implies that the adversary can observe and manipulate the executed primitive. Such a *white-box* attack context was introduced by Chow *et al.* [CEJvO02a]. Table 1 illustrates the different attacker capabilities in various attack models.

Table 1: The different attacker capabilities in black-box, gray-box, and white-box models.

Model	Access to Cryptographic Algorithm	Example Attack Method
Black-Box	inputs and outputs	differential and linear cryptanalysis
Gray-Box	side-channel information	statistical analysis of power traces
White-Box	full access	breakpoint instruction and memory dumps

To prevent the key extraction in the white-box model, Chow *et al.* proposed the first white-box implementations of AES and DES [CEJvO02a, CEJvO02b]. The fundamental idea behind them is to conceal the secret key in a network of look-up tables (LUTs). The entries and outputs of these tables are protected by randomly generated bijective encodings. This principle for generating the white-box implementation of a cryptographic algorithm is called the CEJO framework. In particular, there are two kinds of encodings. *Internal encodings* are applied within the implementation and can be canceled pairwise between the successive tables. The use of linear encodings is to introduce *diffusion* while the application of non-linear 4-bit (nibble) encodings is to ensure *confusion*. *External encodings* are applied to the plaintext and ciphertext for protecting the inputs and outputs of the algorithm. The cryptographic functionality is implemented as $E'_k = G \circ E_k \circ F^{-1}$ instead of the standard implementation E_k for the introduction of the external encodings F and G. Following the CEJO framework, many white-box implementations have been proposed to protect the block ciphers [LN05, BCD06, XL09, Kar10], but all these implementations have been broken by *structural attacks* [BGE04, WMGP07, MWP10, MRP12, LRM$^+$13]. In practice, the white-box implementations are commonly protected with code obfuscation and binary protection. Thus, structural attacks require extra reverse engineering efforts to reveal the details of the internal encodings.

Inspired by the *differential power analysis* (DPA) [KJJ99], Bos *et al.* [BHMT16] proposed the *differential computation analysis* (DCA) on white-box implementations. DCA collects the runtime computed values as *computation traces* and recovers the secret key by the attack model of DPA. Since external encodings will change the input/output specification of the original cipher, DCA mainly focuses on internal encodings. Different from structural attacks, DCA can be mounted automatically to gather noise-free leakage information without the knowledge of encoding details and reverse engineering. Moreover, as a gray-box attack, a DCA adversary does not exploit the full power in the white-box attack context. It has been demonstrated that many published white-box implementations are vulnerable to DCA. To explain the experimental success of DCA, Sasdrich *et al.* [SMG16] proposed the *spectral analysis* (SA). It computes the Walsh transforms of the cryptographic functions to distinguish the correct key candidate. Bock *et al.* [BBMT18, BBB$^+$19] provided the statistical analysis on the attack results of DCA and proposed an improved DCA (IDCA). It has been pointed out that both the Hamming weight of a row in the linear encoding and the use of non-linear nibble encodings cause the key leakage. This observation inspires the countermeasures with non-linear byte encodings. Lee *et al.* [LJK20] demonstrated that the key leakage depends on the imbalanced linear encodings, which denies the impact of the Hamming weight of linear encodings. Banik *et al.* [BBIJ17] proposed the *zero difference enumeration* (ZDE) attack to detect the equivalent values of the intermediate states by choosing special pairs of plaintexts. At CHES 2019, Rivain and Wang [RW19] proposed *collision attack* (CA) and *mutual information analysis* (MIA) as the variants of DCA. Zeyad *et al.* [ZMAB19] proposed the *bucketing computational analysis* (BCA) to attack white-box implementations. Similar to ZDE, BCA sorts the computation traces based on key-dependent intermediate values. At CHES 2021, Carlet *et al.* [CGM21] proposed a structural attack based on an improvement of SA (ISA) to defeat non-linear byte encodings. The mapping of internal encodings is formalized as a diffused-input-blocked-output (DIBO) function. DIBO consists of a linear mapping and a non-linear function which is a concatenation of small permutations. The analysis of ISA indicates that the combination of a non-invertible linear encoding and a non-linear byte encoding cannot prevent the attack of ISA. Therefore, they suggested choosing an invertible linear encoding instead of a non-invertible one to resist ISA.

Without loss of generality, DCA and other DCA-like attacks can be categorized as *computation analysis*. These attacks perform statistical analysis on the collected computation traces to recover the secret key. However, the proposed computation analyses have various attack capabilities. Most of them focus on the properties of linear encodings, such as Hamming weight [BBB+19], imbalance [LJK20], and invertibility [CGM21]. For defeating the computation analysis, an intuition is to expand the non-linear nibble encodings as byte encodings. As a structural attack, ISA demonstrated that the combined use of non-invertible linear encoding is vulnerable to cryptanalysis. However, it is unclear whether the non-linear byte encodings can defeat the computation analyses. A comparison among the proposed computation analyses on their capabilities also has not been fully investigated.

Our Contribution. In this paper, our contribution is threefold:

1. **Review of the computation analyses against internal encodings.** To evaluate the capabilities of the proposed different computation analyses, we revisit the definitions of their various distinguishers. Moreover, the reasons behind their successes against the internal encodings are also investigated. Especially, it is the first time to formalize the time complexities and summarize the effectiveness of different attacks.

2. **New algebraic degree computation analysis.** By leveraging the properties of non-linear encodings, we investigate the impact of the algebraic degree of the combined encodings against the computation analysis. We propose the *algebraic degree computation analysis* (ADCA) as an automatic side-channel attack to break the white-box implementations. ADCA is a *higher-degree* analysis that can distinguish the correct key by matching the algebraic degrees of the mappings from the inputs to each sample of traces. For $1 \leq d \leq 6$, a degree-d ADCA can defeat the internal encodings with an algebraic degree at most d. The theoretical analysis demonstrates that the algebraic degree of encodings mainly causes the key leakage. Therefore, the weakness of internal encoding depends on the algebraic degree of the combined mapping instead of the properties of its linear part.

3. **Experiment of breaking differently constructed encodings.** To compare the attack capabilities, we refine a structure from the DIBO function and mount the different computation analyses to recover the secret key. The two target functions are constructed by various (non-)linear encodings and random encodings with different algebraic degrees, respectively. By a thorough experiment of breaking the internal encodings with the published computation analyses, Table 2 summarizes the attack results. The results demonstrate that SA and ISA can break the same maximal cases as ADCA but with higher time complexities 2^{27} and 2^{32}, respectively. We note that ISA is a structural attack and SA does not reveal the weaknesses of different combinations of (non-)linear encodings. Thus, ADCA can be a generalized tool

to break the most cases of encodings (degrees from 1 to 6) with the lowest time complexity $2^{21.32} \sim 2^{24.07}$. The source code of the experiment is publicly available [1].

Table 2: The comparison among the computation analyses on breaking the refined structure based on 14 cases (refer to Table 11 in Section 5.2) of (non-)linear encodings and 7 cases (refer to Table 12 in Section 5.2) of random encodings.

Computation Analysis	Time Complexity	The Number of Broken Cases	
		(Non-)Linear	Degrees from 1 to 7
DCA [BHMT16]	2^{22}	3	0
IDCA [BBB+19]	2^{27}	9	2
CPA [RW19]	2^{22}	3	0
CA [RW19]	2^{29}	4	0
MIA [RW19]	2^{22}	3	0
SA [SMG16]	2^{27}	11	6
MSA [LJK20]	2^{22}	4	1
ISA [CGM21]	2^{32}	11	6
ADCA (Section 4)	$2^{21.32} \sim 2^{24.07}$	11	6

In particular, as the countermeasures against DCA, the masking scheme [SEL21] splits the sensitive variables into secret shares while the shuffling implementation [BU21] shuffles the location of sensitive variables. We note that these countermeasures rely on secret designs without the application of the internal encodings. Hence, the higher-order DCA attacks [TGCX23] on masking and shuffling implementations are out of the scope of this paper. Moreover, we also do not consider the computation analysis on the dedicated white-box ciphers [BI15, BIT16] which are also constructed without the internal encodings.

Organization. The remainder of this paper is organized as follows. Section 2 describes the notions, notations, and definitions of computation analysis. In Section 3, an adaptive white-box adversary model and a refined structure from the DIBO function are introduced for the analysis. The distinguishers of different computation analyses are also revisited. Section 4 proposes the new ADCA attack on white-box implementations. Section 5 illustrates the experimental results of different computation analyses on various internal encodings. Section 6 concludes this paper.

2 Preliminaries

2.1 Basic Notions and Notations

Throughout this paper, the finite field with order 2 is represented by $\mathbb{F}_2$ while $\mathbb{F}_2^n$ denotes the n-dimensional vector space over $\mathbb{F}_2$. The bitwise addition modulo 2, i.e., exclusive-OR (`XOR`) operation between two vectors is denoted by $\oplus$. An addition by a constant c is represented as a function: $\oplus_c : x \mapsto x \oplus c$. The symbol $\|$ denotes the concatenation of vectors. Let $a = (a_1, \cdots, a_n)$ and $b = (b_1, \cdots, b_n)$ be elements of $\mathbb{F}_2^n$, the inner product of them is $a \cdot b = a_1b_1 \oplus \cdots \oplus a_nb_n$. The identity function is denoted as F_{id}. An (n, m)-bit function represents a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. If $n = m$, it is called an n-bit function. The composition of two functions F and G is denoted by $F \circ G$ and the concatenation of them is represented by $(F, G)(x, y) = (F(x), G(y))$. The intersection between two sets A and B is defined by $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$, and they are disjoint if $A \cap B = \emptyset$.

[1]https://github.com/scnucrypto/Revisit_Computation_Analysis

The union of two sets A and B is defined as $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$. The following paragraphs introduce the theoretical notions required for the description of computation analysis. Without a specific description, the term degree represents the algebraic degree and the non-linear internal encodings are shortened to nibble/byte encodings for simplicity in the remainder of this paper.

Pearson's Correlation Coefficient. Let $\mathtt{Cov}$ denote the *covariance* between two random variables X and Y. The function $E(X)$ is the *expectation* of X, and σ_X denotes the *standard deviation* of X. *Pearson's correlation coefficient* is a measure of the *linear* correlation between X and Y and is defined by the following equation.

$$\mathtt{Cor}(X,Y) = \frac{\mathtt{Cov}(X,Y)}{\sigma_X \cdot \sigma_Y} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - (E(X))^2}\sqrt{E(Y^2) - (E(Y))^2}}$$

The computed correlation coefficient satisfies $-1 \leq \mathtt{Cor}(X,Y) \leq 1$. The correlation of X and Y are negatively or positively linear if $\mathtt{Cor}(X,Y) = -1$ or $\mathtt{Cor}(X,Y) = 1$. X and Y are *linearly* independent if $\mathtt{Cor}(X,Y) = 0$.

Boolean function. A *Boolean function* f with n variables is an $(n,1)$-bit function. The *weight* of f is denoted by $\mathtt{wt}(f) = |\{x \in \mathbb{F}_2^n : f(x) = 1\}|$, i.e., the *Hamming weight* ($\mathtt{HW}$) of its truth table. A Boolean function f is balanced if $\mathtt{wt}(f) = 2^{n-1}$. The *imbalance* of f is defined as follows.

$$B(f) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} = 2^n - 2 \cdot \mathtt{wt}(f)$$

Definition 1 (*Algebraic Normal Form*)**.** Every Boolean function f has a unique *algebraic normal form* (ANF) which can be represented as an n-variable polynomial over $\mathbb{F}_2$, of the form

$$f(x_1, \cdots, x_n) = \sum_{I \subseteq \{1, \cdots, n\}} a_I x^I,$$

where the monomial $x^I = \prod_{i \in I} x_i$ and $a_I = 0$ or 1.

Definition 2 (*Algebraic Degree*)**.** The algebraic degree of a Boolean function f is the maximal degree of the monomials of its ANF, which is denoted by

$$d_{alg}(f) = \mathtt{max}\{|I| \mid I \subseteq \{1, \cdots, n\}, a_I \neq 0\},$$

where $|I|$ is the size of I. A zero function has an algebraic degree 0. A Boolean function f is *affine* if its algebraic degree equals 1. An affine Boolean function is *linear* if $f(0) = 0$.

Let $\mathcal{X}$ be a set of variables over $\mathbb{F}_2$ such that $\mathcal{X} = \{x_1, x_2, \cdots x_{|\mathcal{X}|}\}$. We define the d-th degree closure of $\mathcal{X}$ to be a set obtained by composing any element of degree at most d with elements from $\mathcal{X}$:

$$\mathcal{X}_d = \{f \circ (x_1, x_2, \cdots x_{|\mathcal{X}|}) \mid \forall f : \mathbb{F}_2^{|\mathcal{X}|} \mapsto \mathbb{F}_2,\ d_{alg}(f) \leq d,\ x_i \in \mathcal{X}\}.$$

For instance,

- $\mathcal{X}_1 = \{1\} \cup \{x_i \mid x_i \in \mathcal{X}\}$,
- $\mathcal{X}_2 = \{1\} \cup \{x_i x_j \mid x_i, x_j \in \mathcal{X}\}$, which also includes $\mathcal{X}^{(1)}$ when $i = j$.

Boolean Correlation. Let f, g be two $(n,1)$-bit functions, define

$$\begin{aligned} N_b^f &= |\{x \in \mathbb{F}_2^n, b \in \mathbb{F}_2 : f(x) = b\}|, \\ N_{b_1 b_2}^{f,g} &= |\{x \in \mathbb{F}_2^n, b_1, b_2 \in \mathbb{F}_2 : f(x) = b_1, g(x) = b_2\}|. \end{aligned}$$

For a uniform random input $x \in \mathbb{F}_2^n$, the correlation coefficient between $f(x)$ and $g(x)$ can be described as follows.

$$\mathtt{Cor}(f,g) = \frac{N_{11}^{f,g} N_{00}^{f,g} - N_{10}^{f,g} N_{01}^{f,g}}{\sqrt{N_1^f N_0^f N_1^g N_0^g}}$$

Vectorial Boolean Function. A *vectorial* Boolean function F is an (n,m)-bit function which is also called *multi-output* Boolean function. The corresponding Boolean functions $F_1, F_2, \cdots, F_m$ satisfying $F(x) = (F_1(x), F_2(x), \cdots, F_m(x))$ are called *coordinate functions* of F. The function F is *balanced* if each output $y = F(x) \in \mathbb{F}_2^m$ has 2^{n-m} preimages. A balanced (n,n)-bit function is an n-bit permutation. The degree of F is defined to be the maximal degree of its coordinate functions. In particular, if an n-bit permutation P has the maximal degree $n-1$, the degree of its inverse P^{-1} is also $n-1$.

Walsh Transform. The *Walsh transform* of an n-variable Boolean function f maps every element $u \in \mathbb{F}_2^n$ to

$$W_f(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus u \cdot x}.$$

A larger absolute value of $W_f(u)$ implies a higher correlation between $f(x)$ and $u \cdot x$. The Walsh transform of a (n,m)-bit function F maps every pair $(u,v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ to the value at u of the Walsh transform of the Boolean function, which satisfies

$$W_F(u,v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus u \cdot x}.$$

Definition 3 (*Correlation Immune*)**.** If the Walsh transform W_f of a Boolean function f satisfies $W_f(u) = 0$, for $0 \leq \mathtt{HW}(u) \leq m$, it is called a balanced m-th order correlation immune function. Let F be an (n,m)-bit function. The function F is correlation immune at order t iff $W_F(u,v) = 0$ for every $u \in \mathbb{F}_2^n$, $\mathtt{HW}(u) \leq t$ and for every $v \in \mathbb{F}_2^m$.

Mutual Information. For two random variables X and Y, the *mutual information* is a measure of the dependence between them. It quantifies the information obtained about X by observing Y. It can be calculated as follows.

$$I(X;Y) = \sum_{x \in \mathcal{X}} \mathtt{Pr}(X = x, Y = y) \cdot \mathtt{log}_2 \left(\frac{\mathtt{Pr}(X = x, Y = y)}{\mathtt{Pr}(X = x) \cdot \mathtt{Pr}(Y = y)} \right)$$

2.2 Internal Encodings

By using input and output encodings I and O respectively, a table T can be transformed into T' as follows.

$$T' = O \circ T \circ I^{-1}$$

Because of the pairwise invertible encodings, the composition of two encoded tables T' and R' yields a new mapping such that

$$R' \circ T' = (O_R \circ R \circ I_R^{-1}) \circ (O_T \circ T \circ I_T^{-1}) = O_R \circ (R \circ T) \circ I_T^{-1},$$

where $I_R^{-1} \circ O_T = F_{id}$. In the case of AES, the first-round encryption starts by `AddRoundKey`, `SubBytes`, and `MixColumns`. By means of *matrix partitioning*, the multiplication of `MixColumns` M can be decomposed into four 32-bit vectors:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_0 \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus x_1 \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus x_2 \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus x_3 \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}$$
$$= M_1(x_0) \oplus M_2(x_1) \oplus M_3(x_2) \oplus M_4(x_3).$$

Each M_j for $1 \leq j \leq 4$ denotes an $(8, 32)$-bit mapping. For the sake of simplicity, we omit the index of the function. Let S denote the Sbox. An $(8, 32)$-bit key-dependent T-box is defined as follows.

$$T(x) = M \circ S(x \oplus k)$$

By applying the internal encodings, a linear mapping ϕ and a non-linear permutation B are used to protect the output of T-box. In particular, B can be represented by the concatenation of small random permutations. Consequently, the obfuscated function O_k can be defined by

$$x \mapsto O_k(x) = B \circ \phi \circ T(x) = B \circ \phi \circ M \circ S(x \oplus k). \qquad (1)$$

Definition 4 (*Diffused-Input-Blocked-Output Function* [CGM21])**.** Let n and n_0 denote two positive integers such that n is a multiple of n_0. The Diffused-Input-Blocked-Output (DIBO) functions are defined as $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n : F = B \circ \phi$, where ϕ is a linear permutation of $\mathbb{F}_2^n$ and B consists of n/n_0 n_0-bit functions $B_1, \cdots, B_{n/n_0}$ such that

$$B(x_1, \cdots, x_n) = (B_1(x_1, \cdots, x_{n_0}), B_2(x_{n_0+1}, \cdots, x_{2n_0}), \cdots, B_{\frac{n}{n_0}}(x_{n-n_0+1}, \cdots, x_n)).$$

The definition of the DIBO function formalizes the internal encodings used in white-box implementations. Considering the bit width of `MixColumns`, the dimension of ϕ is 32, i.e., $n = 32$. For the linear property of the decomposition of M, the outputs of the blocked non-linear encodings are followed by `XOR` operations. A type of `XOR` tables is introduced to `XOR` the encoded vectors without the exposure of encodings. The `XOR` table with two inputs needs to be practical for the size of $2^{2\times n_0} \times n_0$ bits. Hence, most of the white-box implementations utilize a nibble encoding for $n_0 = 4$ or a byte encoding for $n_0 = 8$. Figure 1 depicts the structure of the function $O_k(x)$ for the nibble and byte encodings.

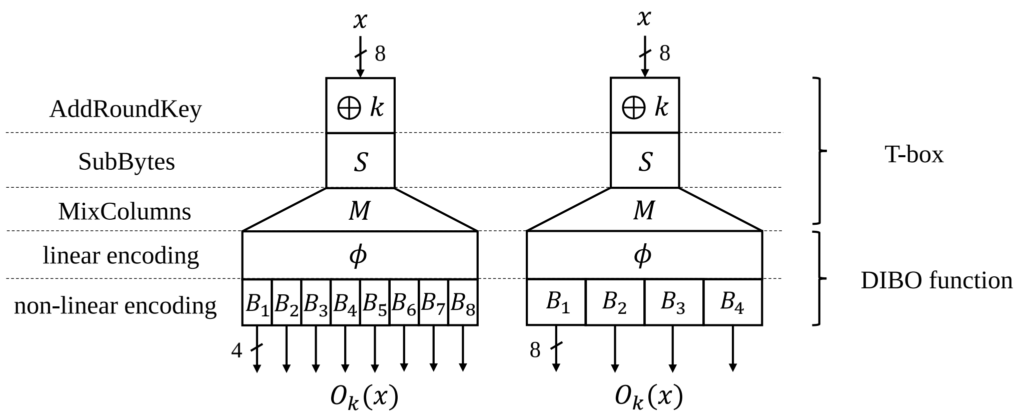

Figure 1: The structure of T-box and DIBO function for nibble encoding (left) and byte encoding (right).

2.3 Computation Analysis

Computation analysis is inspired by physical side-channel attacks. With the help of dynamic binary instrumentation tools, such as Intel PIN, it collects the computation traces by invoking the implementation several times. These traces consist of the accessed memory addresses, data, and associated instructions. Indeed, this process aims to record the patterns corresponding to LUTs. Let $\boldsymbol{v} = (v_1, v_2, \cdots, v_T)$ denote a computation trace that is composed of T samples. The adversary collects N computation traces $(\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \cdots \boldsymbol{v}^{(N)})$ related to N inputs $(x^{(1)}, x^{(2)}, \cdots, x^{(N)})$. Due to the absence of external encodings, each input $x^{(t)}$ for $t \in [N]$ is a state of plaintext. By interpreting the traces as an $N \times T$ matrix, each row of it is a computation trace $\boldsymbol{v}^{(t)}$ and each column of it consists of N intermediate values for the same location over different computations. A *distinguisher* D which maps the inputs $\{x^{(t)}\}$ and the traces $\{\boldsymbol{v}^{(t)}\}$ to a distinguishing score, is defined as

$$\delta_k = D\left((x^{(1)}, \cdots, x^{(N)}), (\boldsymbol{v}^{(1)}, \cdots \boldsymbol{v}^{(N)})\right).$$

In this context, the adversary makes a key guess k to predict a sensitive intermediate variable $s = \varphi_k(x)$ and computes the dependency between the predictions and the traces. The highest score δ_{k^*} is selected as the candidate for the most likely correct key k^*. In the literature, many distinguishers have been proposed in various computation analyses. The following section revisits those proposals along with their time complexities and attack capabilities.

3 Revisiting the Distinguishers of Computation Analysis

In the literature, many computation analyses have been proposed to reveal the weaknesses of the internal encodings. However, these attacks depend on the different weaknesses of linear encodings, such as `HW` and invertibility. Most of the computation analyses only compare their capabilities to the original DCA. It lacks a thorough comparison of all the proposals. Without detailed information on the internal encodings, an adversary needs to mount the analyses separately. In this section, we revisit the adversary models and target functions of the computation analyses. Moreover, the distinguishers, reasons for key leakage, and time complexities of the published attacks are compared.

3.1 The Adversary Models

Most of the existing computation analyses focus on the key leakage of white-box implementations in a gray-box attack context. These attacks exploit the computation traces including table lookups to analyze the correlation between the encoded sensitive variables and the hypothetical key-dependent values. Rivain and Wang [RW19] introduce a passive adversary for computation analysis. The assumed adversary can only record the runtime information, such as accessed memory to recover the secret key. However, this attack model seems too weak for white-box adversaries. *Differential data analysis* (DDA) [AH16] assumes that the adversary can access the entire table lookups within the implementation to perform the power analysis. Nevertheless, for the white-box implementations based on Boolean circuits (without LUTs) [SEL21, BU21], DDA will be failed because it does not collect the outputs of functions. Moreover, DDA collects all the intermediate values of the implementation. We note that only the outputs of some specific tables/functions are helpful for detecting key leakage. Thus, the recorded values of DDA are redundant for analysis. Tang *et al.* [TGS$^+$21] introduced an adaptive side-channel attack context to break a masked white-box AES. It focuses on the abilities of an adversary to adaptively choose inputs based on the pre-collected computation traces. Carlet *et al.* [CGM21] considered the spectral analysis in a white-box context. The analysis computes the Walsh transform by the outputs of tables instead of computation traces, which is a structural attack on the DIBO function of white-box implementations. Hence, different analyses might be assumed to be mounted in different contexts. To fairly evaluate the proposed computation analyses, we propose an adaptive white-box adversary which is described as follows.

- The adversary can invoke the white-box implementation many times by choosing arbitrary inputs.
- The adversary is capable of collecting multiple intermediate values during each execution with the ability of a white-box attacker. The recorded values consist of output states of a specific table or a particular function.
- The adversary exploits the collected intermediate values as computation traces and mounts the existing computation analyses to recover the secret key.

In this context, the adversary can pinpoint a specific table/function to recover its embedded key by analyzing the structure of the obtained mapping. There are three reasons to inspire such a new attack model as follows.

1. A precise simulation of the computation trace. The attack model focuses on analyzing the table lookups or the outputs of functions, which are the primarily collected objects for computation analysis. Thus, it is no need to record multiple traces.
2. A structural analysis of the internal encodings, such as nibble and byte encodings. The obtained mapping from the inputs to the table lookups helps to mount a structural attack to analyze the encodings and recover the secret key.
3. A combination of different attack contexts of different analyses. The attack model helps us to compare the attack capability of different computation analyses in the same context.

3.2 A Refined Structure from DIBO Function

As illustrated in Equation (1) in Section 2.2, an obfuscated key-dependent function O_k is a mapping from $\mathbb{F}_2^8$ to $\mathbb{F}_2^{32}$. A DIBO function $B \circ \phi$ within O_k is a combined function of linear and non-linear mappings from $\mathbb{F}_2^{32}$ to $\mathbb{F}_2^{32}$. Most of the computation analyses simultaneously evaluate all the 32-bit outputs of O_k to compute the correlation with the sensitive variables. However, the blocked outputs of the DIBO function are independent of each other since the non-linear encodings are randomly generated. Thus, we can refine a smaller structure from O_k by its blocked outputs to reduce the length of computation traces.

The linear layer M of the block cipher can be merged into the linear mapping ϕ of the DIBO function, which can be defined by

$$(L_1, L_2, L_3, L_4) : \mathbb{F}_2^8 \mapsto (\mathbb{F}_2^8)^4 : x \mapsto (L_1, L_2, L_3, L_4)(x) = (\phi_1, \phi_2, \phi_3, \phi_4)(\texttt{02}x \| x \| x \| \texttt{03}x).$$

We note that each L_i for $1 \leq i \leq 4$ might not be bijective because the decomposed matrix ϕ_i might not be invertible. Let N_i for $1 \leq i \leq 4$ denote the non-linear encodings. Each N_i is a byte encoding or a concatenation of two nibble encodings. The function N_i represents both the byte and nibble cases in a DIBO function. Hence, the obfuscated function O_k can be redefined as

$$O_k : \mathbb{F}_2^8 \mapsto (\mathbb{F}_2^8)^4 : x \mapsto O_k(x) = (N_1 \circ L_1, N_2 \circ L_2, N_3 \circ L_3, N_4 \circ L_4)(S(x \oplus k)).$$

The function O_k is a concatenation of four independent functions $(\overline{O_k})_i$ for $1 \leq i \leq 4$ such that

$$(\overline{O_k})_i : \mathbb{F}_2^8 \mapsto \mathbb{F}_2^8 : x \mapsto (\overline{O_k})_i(x) = N_i \circ L_i \circ S \circ \oplus_k.$$

By omitting the index, a refined structure of the key-dependent function $\overline{O_k}$ can be defined by

$$\overline{O_k}(x) = N \circ L \circ S(x \oplus k).$$

We note that there are two reasons for introducing such a new structure.

1. An adaptive white-box attacker can only collect the 1-byte output of $\overline{O_k}$ by randomly choosing x to analyze its underlying key without recording the 4-byte outputs of O_k.
2. It is more practical to simulate and analyze the functions L and N over $\mathbb{F}_2^8$ than the functions ϕ and B over $\mathbb{F}_2^{32}$.

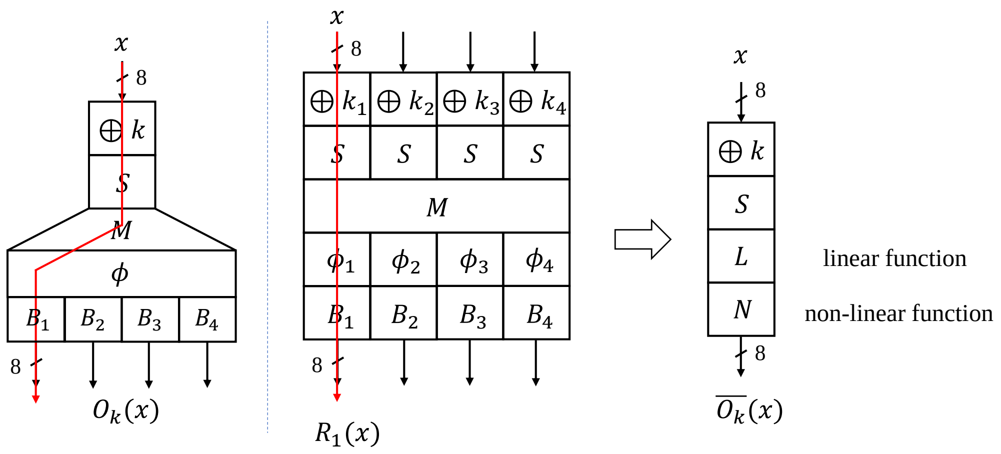

Figure 2: A refined structure to transform the target functions O_k and R_1 into an 8-bit key-dependent function $\overline{O_k}$.

The refined structure is depicted in Figure 2. In addition, $\overline{O_k}$ can also represent the mapping from the inputs to the first-round outputs. Let $(x_1, x_2, x_3, x_4) \in (\mathbb{F}_2^8)^4$ denote 4 inputs of an AES subround. The linear encodings of the outputs are denoted by $(\phi_1, \phi_2, \phi_3, \phi_4) : (\mathbb{F}_2^8)^4 \mapsto (\mathbb{F}_2^8)^4$ and the non-linear encodings are represented by $(B_1, B_2, B_3, B_4) : (\mathbb{F}_2^8)^4 \mapsto (\mathbb{F}_2^8)^4$. For $1 \le i \le 4$, each non-linear encoding B_i is an 8-bit permutation or a concatenation of two 4-bit permutations. By combining the LUTs in the first round, a function of the subround $R_{(k_1,k_2,k_3,k_4)} : (\mathbb{F}_2^8)^4 \mapsto (\mathbb{F}_2^8)^4$ is defined as follows.

$$R_{(k_1,k_2,k_3,k_4)}(x) = (B_1, B_2, B_3, B_4) \circ (\phi_1, \phi_2, \phi_3, \phi_4) \circ M \circ (S \circ S \circ S \circ S) \circ (\oplus_{k_1}, \oplus_{k_2}, \oplus_{k_3}, \oplus_{k_4})$$

Let $R_i : (\mathbb{F}_2^8)^4 \mapsto \mathbb{F}_2^8$ denote the coordinate functions of $R_{(k_1,k_2,k_3,k_4)}$ such that

$$R_i(x_1, x_2, x_3, x_4) = B_i \circ \phi_i \left(\bigoplus_{j=1}^{4} M_{i,j} \cdot S(x_j \oplus k_j) \right),$$

where $M_{i,j} (1 \le i, j \le 4)$ are the coefficients of `MixColumns`. If the input bytes (x_2, x_3, x_4) are fixed to be constant values, R_1 can be transformed as

$$R_1(x_1) = B_1 \circ \phi_1 \left(M_{1,1} \cdot S(x_1 \oplus k_1) \oplus c \right),$$

where c is an unknown constant value computed by the fixed inputs (x_2, x_3, x_4). Hence, by combining $M_{i,j}$ and c into the linear encoding ϕ_i as an affine function L, one can obtain a mapping $N \circ L \circ S \circ \oplus_k$. Since $N \circ L$ can represent a combined random encoding of a linear mapping and a permutation, the mapping $N \circ L \circ S \circ \oplus_k$ is similar to $\overline{O_k}$.

3.3 The Proposed Computation Analyses and Their Distinguishers

We assume that the target function is $\overline{O_k} : \mathbb{F}_2^8 \mapsto \mathbb{F}_2^8 : \overline{O_k}(x) = N \circ L \circ S(x \oplus k^*)$ for the correct key $k^* \in \mathbb{F}_2^8$. Based on the abilities of an adaptive white-box adversary, the attacker can freely choose arbitrary N inputs x. The collected computation trace is denoted as $\boldsymbol{v} = (v_1, v_2, \cdots, v_T)$ with T samples. The sensitive variables are defined as the 8 Sbox outputs S_i for $1 \leq i \leq 8$. Particularly, since ZDE requires the detection of collisions between multiple pairs of intermediate values, the target function is not applicable to the analysis of ZDE. Moreover, BCA relies on the bijection of the nibble encodings, which will fail to break the byte encodings. According to the same method of collision, the results of ZDE and BCA can refer to CA. Although ISA is rather a structural attack than a computation analysis, it is claimed that ISA improves the capability of breaking the byte encodings. Thus, ISA is also included in the comparison of the proposed computation analyses.

Differential Computation Analysis. DCA [BHMT16] is the software counterpart of DPA to break the white-box implementations. It analyzes the execution traces by using the *difference of means* method. For a key guess k, DCA divides the computation traces into two distinct sets A_0 or A_1 based on the i-th output bit of the Sbox, i.e., $b = S_i(x \oplus k)$ as follows.

$$\text{For } b \in \{0,1\} \ A_b = \{\boldsymbol{v}^{(t)} | 1 \leq t \leq N, S_i(x \oplus k) = b\}$$

The mean trace is computed by

$$\overline{A_b} = \frac{\sum_{\boldsymbol{v} \in A_b} \boldsymbol{v}}{|A_b|}.$$

And the difference of means is defined as

$$\Delta = |\overline{A_0} - \overline{A_1}|.$$

This difference of means is calculated for the target bit with the key hypothesis. DCA then repeats the computation for 8 target bits of Sbox outputs and each key guess over $\mathbb{F}_2^8$. The corresponding key k^* of the obtained difference of means trace Δ^{k^*} with the highest peak is the most possible correct key. Therefore, the time complexity of DCA is $\mathcal{O}(|k| \cdot |i| \cdot T \cdot N)$.

Improved DCA. Since DCA has not fully revealed the reason behind its successful key recovery, Bock *et al.* [BBMT18, BBB+19] studied the internal encodings against DCA attack. They also introduced some improvements for DCA. Theorem 1 is recalled for the property of linear encodings. It points out that an identity row in linear encoding is the main cause of key leakage in DCA.

Theorem 1 ([BBB+19])**.** *When $S \circ \oplus_k$ is encoded via an invertible matrix A, the DCA attack returns a difference of means value equal to* 1 *for the correct key guess if and only if the matrix A has at least one row i with* $\texttt{HW} = 1$. *Otherwise, the DCA attack returns a difference of means value equal to* 0 *for the correct key guess.*

To counteract linear encodings, IDCA exhaustively searches all possible linear combinations LC of the Sbox output, i.e., to compute a target bit $b = LC \cdot S(x \oplus k)$. The following process is to sort the traces according to the result for each possible LC. Thus, it can compute a high correlation between the computation traces and the recovered encoded bit. Theorem 2 is recalled for the statistical analysis of nibble encodings. For the correct key guess under nibble encodings, it turns out the difference of means curve only consists of 5 possible values.

Theorem 2 ([BBB+19]). *When $S \circ \oplus_k$ is encoded via the nibble encodings, the difference of means curve obtained for the correct key hypothesis k consists only of values equal to* 0, 0.25, 0.5, 0.75 *or* 1.

Based on Theorem 2, IDCA selects the key hypothesis with an enough high peak value (at least 0.3) to stand out from other candidates. Otherwise, IDCA looks for the values converging to 0.25 or 0 and selects its corresponding key guess as the best one. Due to the calculation of all linear combinations, the time complexity of IDCA has an extra factor $\mathcal{O}(2^8)$. However, IDCA need not select a target bit of Sbox output since each linear combination of the Sbox output is a predicted sensitive variable. Thus, the time complexity of IDCA is $\mathcal{O}(2^8 \cdot |k| \cdot T \cdot N)$. Since the use of nibble encoding cannot prevent key leakage, they suggested exploiting the byte encoding to protect LUTs.

Correlation Power Analysis. Rivain and Wang [RW19] formalized *Correlation Power Analysis* (CPA) to attack white-box implementations. CPA assumes that the function of an intermediate variable is a non-injection and the encodings are bijections. For a key guess k, CPA computes the correlation between the predicted sensitive variable $\varphi_i(x) = S_i(x \oplus k), 1 \leq i \leq 8$ and each sample of the computation traces $v_j \in \boldsymbol{v}, j \in [T]$. The CPA distinguisher is derived from the maximal absolute value of the correlation as follows.

$$\delta_k^{\texttt{CPA}} = \texttt{arg max}\,|\texttt{Cor}(\varphi_i(x), v_j)|$$

In practice, CPA selects the key guess k^* with the maximal value δ_{k^*} by computing all the correlations between every bit of the Sbox output and each sample of the traces. Hence, the time complexity of CPA is $\mathcal{O}(|k| \cdot |i| \cdot T \cdot N)$.

Collision Attack. CA [RW19] computes a *collision computation trace* for each pair of inputs $(x^{(t_1)}, x^{(t_2)})$ where $t_1, t_2 \in [N]$, $t_1 \neq t_2$ and their related traces $(\boldsymbol{v}^{(t_1)}, \boldsymbol{v}^{(t_2)})$ as

$$\boldsymbol{w}^{(t_1,t_2)} = (w_1^{(t_1,t_2)}, w_2^{(t_1,t_2)}, \cdots, w_T^{(t_1,t_2)}).$$

Each sample $w_j^{(t_1,t_2)} = v_j^{(t_1)} \odot v_j^{(t_2)}$ for $j \in [T]$. The operator $\odot$ is defined as follows.

$$a \odot b = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

The *collision prediction* for a key guess k is computed by

$$\varphi_k(x^{(t_1)}, x^{(t_2)}) = \varphi_i(x^{(t_1)}) \odot \varphi_i(x^{(t_2)}),$$

where $\varphi_i(x) = S_i(x \oplus k), 1 \leq i \leq 8$ is a sensitive variable. The CA distinguisher computes the maximal absolute value of the correlation between the collision prediction and each sample of the corresponding collision traces, such that

$$\delta_k^{\texttt{CA}} = \texttt{arg max}\left|\texttt{Cor}\left(\varphi_k(x^{(t_1)}, x^{(t_2)}), w_j^{(t_1,t_2)}\right)\right|.$$

The main idea behind the CA distinguisher is that if some sensitive variables collide for a pair of inputs, the collision can also be obtained between their encoded variables. The time complexity of CA is $\mathcal{O}\left(|k| \cdot |i| \cdot T \cdot \binom{N}{2}\right)$. The phase $\binom{N}{2}$ is the number of all the pairs among N inputs.

Mutual Information Analysis. MIA [RW19] calculates the maximal mutual information between the predicted sensitive variable $\varphi_i(x) = S_i(x \oplus k)$ for $1 \leq i \leq 8$ and each sample of the computation traces $v_j \in \boldsymbol{v}$ for $j \in [T]$. The time complexity of MIA is $\mathcal{O}(|k| \cdot |i| \cdot T \cdot N)$. The MIA distinguisher is defined as follows.

$$\delta_k^{\texttt{MIA}} = \texttt{arg max}\ I(\varphi_i(x), v_j).$$

Spectral Analysis. SA [SMG16] detects the key leakage by computing the Walsh transform of functions. For a key guess k, SA derives a new function g_k by choosing the inputs $x' = S^{-1}(x) \oplus k$ of the target function $\overline{O_k}$, such that

$$g_k(x) = \overline{O_k}(x') = N \circ L \circ S \circ \oplus_{k^*} \circ \oplus_k \circ S^{-1}.$$

If the key candidate $k = k^*$, the function can be transformed to $g_{k^*}(x) = N \circ L$. The SA distinguisher is defined as follows.

$$\delta_k^{\mathtt{SA}} = \mathtt{arg\ min} \sum_{u \in \mathbb{F}_2^8} \sum_{i=1}^{8} |W_{(g_k)_i}(u)|$$

This distinguisher sums all the imbalances for each key candidate k and the inputs u. The correct key is distinguishable among other candidates since it has more correlation immune functions. In practice, SA computes the Walsh transform of the Boolean functions that map the inputs to each sample of traces. Thus, the time complexity of SA is $\mathcal{O}(|k| \cdot |u| \cdot T \cdot N)$.

Modified Spectral Analysis. Lee *et al.* [LJK20] modified the SA distinguisher to only choose u with $\mathtt{HW}(u) = 1$. The modified spectral analysis (MSA) considers a mono-bit model instead of a multi-bit one. It detects the correlation between a coordinate function g_i and an input bit of x. The MSA distinguisher is described as follows.

$$\delta_k^{\mathtt{MSA}} = \mathtt{arg\ min} \sum_{u=1,2,4,\cdots,128} \sum_{i=1}^{8} |W_{(g_k)_i}(u)|$$

Similar to SA, MSA calculates the Walsh transform of the mapping from the inputs to each sample of the traces. Thus, the time complexity of MSA is also $\mathcal{O}(|k| \cdot |u| \cdot T \cdot N)$, where u only has 8 different values without the space size of $\mathbb{F}_2^8$. Based on MSA, Lee *et al.* demonstrated that the key leakage depends on the imbalance of linear encodings instead of HW of it. The Walsh transform of $(g_k)_i$ computes the imbalances of the function $h(x) = (g_k)_i(x) \oplus u \cdot x$, such that

$$B(h) = \sum_{x \in \mathbb{F}_2^8} (-1)^{h(x)} = 2^8 - 2 \cdot \mathtt{wt}(h).$$

We note that $B(h) = 0$ for a balanced linear encoding. This implies that $\mathtt{wt}(h) = 128$. However, MSA points out that the key-dependent distribution of the intermediate values leads to $\mathtt{wt}(h) = 0$. Thus, it results in the Walsh transform value 256 which helps to distinguish the correct key.

Improved Spectral Analysis. ISA [CGM21] analyzes the correlation immune of the target functions and computes the Walsh transforms. As a structural attack, ISA leverages the maximum number of zeros in the Walsh transforms to distinguish the correct key. The ISA distinguisher is defined as follows.

$$\delta_k^{\mathtt{ISA}} = \mathtt{arg\ max} \# \{W_{(g_k)_i}(u, v) = 0 \mid u, v \in \mathbb{F}_2^8\}$$

Different from SA, ISA computes the Walsh transform of a vectorial Boolean function instead of its coordinates. The introduction of v is to calculate the component functions of g_k. Moreover, ISA considers attacking the byte encodings. The time complexity of ISA is $\mathcal{O}(|k| \cdot |u| \cdot |v| \cdot N)$. The analysis of ISA reveals that the success of ISA depends on the rank of the linear encodings. If L is invertible with full rank, ISA will fail in attacking the function g_k. Contrarily, if L is non-invertible, ISA will succeed even with byte encodings.

3.4 Discussion

Computation analysis is inspired by the side-channel techniques to perform statistical analysis on the intermediate values of white-box implementations. Hence, the assumed attack context is a gray-box one. Particularly, as a structural attack, ISA counts the zero values of Walsh transforms, which can only be mounted in a white-box attack context. To explain the reasons for their success, some computation analyses focus on the ineffectiveness of the internal encodings. Based on the collisions of encoded values, CPA, CA, and MIA require that the target function is a non-injection and the encoding $F = N \circ L$ is a bijection. We note that the target function $\overline{O_k}$ can be an 8-bit injection because the linear part L of the encoding F can be a non-invertible mapping. Hence, the assumptions do not hold in our analysis. Besides, the properties of linear encodings are leveraged to explain the key leakage. IDCA highlights the impact for `HW` of the rows in a linear encoding while MSA reveals the weakness of the imbalance of linear encodings. Even with the presence of byte encodings, ISA demonstrates that the non-invertible linear encoding L reduces the security against cryptanalysis.

Table 3: The comparison among the published computation analyses.

Distinguisher	Attack Context	Method	Analysis of Key Leakage	Time Complexity
DCA [BHMT16]	gray-box	correlation computation	-	2^{22}
IDCA [BBB$^+$19]			$\mathtt{HW} = 1$ of L	2^{27}
CPA [RW19]			non-injection of φ bijection of F	2^{22}
CA [RW19]				2^{29}
MIA [RW19]				2^{22}
SA [SMG16]		spectral analysis	-	2^{27}
MSA [LJK20]			imbalance of L	2^{22}
ISA [CGM21]	white-box		non-invertibility of L	2^{32}

Table 3 summarizes the different attack contexts, methods, analyses of key leakage, and time complexities of the published distinguishers. In the adaptive white-box context, we do not restrict the ability of an attacker to invoke the implementation many times. Thus, we consider $N = 2^8$ for the calculation of time complexity. In our experimental results, ISA has the highest time complexity 2^{32} while DCA, CPA, MIA, and MSA have the lowest one 2^{22}. Based on their methods, the distinguishers can be classified into two categories as follows.

1. Correlation computation, such as DCA, IDCA, CPA, CA, and MIA. These distinguishers compute the correlations between the samples of traces and the sensitive variables based on some key guesses. The correct key can be selected by the first rank of the computed correlations.

2. Spectral analysis, such as SA, MSA, and ISA. These distinguishers calculate the Walsh transforms of the target functions by choosing the inputs which are related to some key guesses. The correct key can be extracted by counting the maximal or minimal Walsh transforms to detect the correlation immune functions.

4 Algebraic Degree Computation Analysis

The published computation analyses mainly focus on correlation computation and spectral analysis of internal encodings. There are two reasons that motivate a novel distinguisher based on a new method.

1. To reveal the vulnerability of the non-linear encodings. The previous attacks have been proposed based on the properties of linear encodings, such as HW, imbalance, and invertibility. Nevertheless, a comprehensive analysis of non-linear encodings is missing. Besides, there is no analysis of a randomly generated encoding that is not constructed by linear and non-linear parts.

2. A new distinguisher with a flexible time complexity for breaking different constructed encodings and with a distinct computation for the number of required traces. The previous attacks have a fixed time complexity which is not related to the construction of encodings. Thus, the time complexities of breaking the nibble encodings and the byte encodings are identical. They also do not reveal the number of required computation traces.

This section introduces a new computation analysis that exploits the algebraic degree of the combination of linear and non-linear encodings to distinguish the correct key.

4.1 An Overview of ADCA

By the composition of linear mapping L and non-linear permutation N, the encoding F is defined as $F = N \circ L$. We note that N is an 8-bit non-linear permutation or a concatenation of two 4-bit ones. Thus, the target function with a correct key k^* can be transformed as

$$\overline{O_{k^*}}(x) = F \circ S \circ \oplus_{k*}.$$

For each key hypothesis $k \in \mathbb{F}_2^8$, the attacker can construct a guess function $A_k : \mathbb{F}_2^8 \mapsto \mathbb{F}_2^8$ by choosing the inputs $x' = S^{-1}(x) \oplus k$ of function $\overline{O_{k^*}}$, such that

$$A_k(x) = \overline{O_{k^*}}(x') = F \circ S \circ \oplus_{k*} \circ \oplus_k \circ S^{-1}.$$

If $k = k^*$, one can obtain $A_{k*}(x) = F$ which is the function of internal encodings. The differences between the guess functions $A_k(x)$ and $A_{k*}(x)$ are depicted in Figure 3.

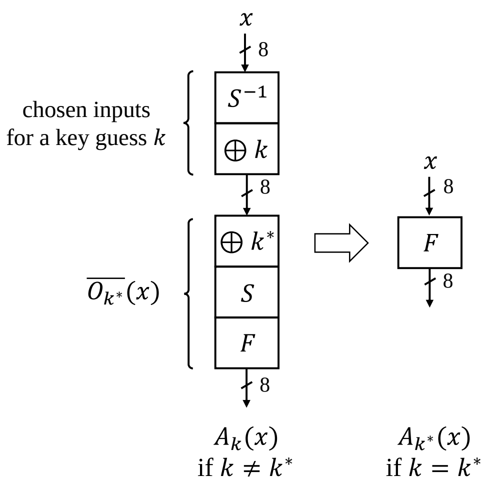

Figure 3: The construction of A_k if $k \neq k^*$ (left) and if $k = k^*$ (right).

We note that the attack strategy is to distinguish $A_{k*}(x)$ from $A_k(x)$. De Mulder [Mul14] discussed an *algebraic degree attack* (ADA) to defeat a white-box AES implementation without external encodings. It exploits a collision-based attack which is proposed by Lepoint *et al.* [LRM+13]. We note that a 4-bit permutation has the maximal degree 3. Thus, for the case of nibble encodings, the degree of $A_{k*}(x)$ will be 3 at most. Differently, an AES Sbox has a degree 7. Hence, the degree of $S(S^{-1}(x) \oplus c)$ (for a non-zero constant c) is greater than 3, which implies that the degree of $A_k(x)$ will be greater than the one of $A_{k*}(x)$. Thus, it can distinguish $A_{k*}(x)$ by a 4-th order derivative of the function $A_k(x)$. Following this method, the attacker can compute the degrees of $A_k(x)$ for different key candidates $k \in \mathbb{F}_2^8$. Therefore, the adversary can distinguish a function $A_{k^*}(x)$ for its lower degree. Without loss of generality, we propose a new computation analysis for white-box implementations as *algebraic degree computation analysis* (ADCA). The main process of ADCA consists of the following steps.

1. Collecting the computation traces by choosing inputs $x' = S^{-1}(x) \oplus k$ for a key guess $k \in \mathbb{F}_2^8$. The obtained traces $\boldsymbol{v} = (v_1, v_2, \cdots, v_T)$ contain the outputs of coordinate functions $(A_k(x))_i$ where $1 \leq i \leq 8$.
2. For each Boolean function $(f_k)_j : \mathbb{F}_2^8 \mapsto \mathbb{F}_2 : (f_k)_j(x) = v_j$ where $j \in [T]$ and $k \in \mathbb{F}_2^8$, computing the degree of $d_{alg}\left((f_k)_j\right)$.
3. For a given degree d, searching the correct key k^* that has the maximum numbers of $d_{alg}\left((f_{k^*})_j\right) \leq d$ where $j \in [T]$.

4.2 The Distinguisher of ADCA

For a correct key k^*, A_{k*} is transformed into a function of internal encodings, such that $A_{k*} = F$. For an incorrect key guess $k \neq k^*$, the target function $A_k = F \circ S \circ c \circ S^{-1}$ where c is a non-zero constant. If $d_{alg}(F) < d_{alg}(A_k)$, it implies $d_{alg}(A_{k*}) < d_{alg}(A_k)$. Thus, A_{k*} can be distinguished from A_k with a lower degree. However, the different constructions of L and N might result in various degrees of F, especially in each coordinate of F. Moreover, the degree of a vectorial Boolean function cannot precisely represent each coordinate function. Hence, ADCA needs to match the degree of each coordinate function. It computes the degrees of coordinate functions $(A_k)_i : \mathbb{F}_2^8 \mapsto \mathbb{F}_2$ where $1 \leq i \leq 8$ instead of the degree of a vectorial Boolean function $A_k : \mathbb{F}_2^8 \mapsto \mathbb{F}_2^8$.

For the case of nibble encodings, $d_{alg}(N) \leq 3$. Thus, each coordinate function of A_{k*} satisfies $d_{alg}\left((A_{k*})_i\right) < 7$, which can be distinguished from A_k. Although $d_{alg}(N) = 7$ for most cases of byte encodings, ISA reveals that the combined non-invertible linear mapping cannot resist ISA. Hence, $d_{alg}\left((A_{k*})_i\right) < 7$ is also satisfied for the case of the non-invertible linear encodings. The intuition is that the introduction of a non-invertible linear function reduces the degree of the combined mapping. Theorem 3 illustrates the relation between the invertibility of the linear part and the degree of the combined encodings.

Theorem 3. *Let L denote an n-bit non-invertible linear mapping and N be an n-bit permutation of which the coordinate functions $N_i : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ have $d_{alg}(N_i) = n - 1$ for $i \in [n]$. Given the combined function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n : F = N \circ L$, the algebraic degree of its coordinate function $F_i : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ satisfies $d_{alg}(F_i) \leq n - 1$.*

Proof. For an n-bit non-invertible matrix L with rank m ($1 \leq m < n$), there exist $n - m$ rows that can be represented by the linear combination of other rows. Let $l_{i,j}$ for $1 \leq i, j \leq n$ be the entries of L, where i denotes the row and j is the column of the entry. Let $L[i]$ denote the i-th row of L. Without loss of generality, we suppose that $L[n]$ can be represented by the other $n - 1$ rows for the given scalars $a_1, \cdots, a_{n-1} \in \{0, 1\}$, such that

$$L[n] = \sum_{1 \leq i \leq n-1} a_i \cdot L[i] = \sum_{a_i = 1, 1 \leq j \leq n} l_{i,j}.$$

For the linear mapping $y = L(x)$, each output coordinate y_i for $1 \leq i \leq n$ can be computed by

$$y_i = L[i] \cdot (x_1, \cdots, x_n) = \sum_{1 \leq j \leq n} l_{i,j} \cdot x_j.$$

Hence, the n-th output y_n can be represented as follows.

$$\begin{aligned} y_n = L[n] \cdot (x_1, \cdots x_n) &= \left(\sum_{1 \le i \le n-1} a_i \cdot L[i] \right) \cdot (x_1, \cdots, x_n) \\ &= \sum_{a_i=1, 1 \le i \le n-1} (L[i] \cdot (x_1, \cdots, x_n)) = \sum_{a_i=1, 1 \le i \le n-1} y_i. \end{aligned}$$

This indicates that y_n is the sum of other output coordinates. For the non-linear permutation $z = N(y)$, the coordinate functions are defined as

$$z_i = N_i(y_1, \cdots, y_n) = N_i(y_1, \cdots, \sum_{a_i=1, 1 \le i \le n-1} y_i).$$

Since each coordinate function N_i has the degree $n-1$, the monomial with the maximal degree of its ANF consists of $n-1$ variables. Hence, when the $(n-1)$-variable monomial contains y_n, its degree will be 0, $n-2$, or $n-1$. For example, when $y_n = y_2 \oplus y_3$ that consists of an even number of variables without y_1, the monomial $y_2 y_3 \cdots y_{n-1} y_n$ can be represented by

$$y_2 y_3 \cdots y_{n-1} y_n = y_2 y_3 \cdots y_{n-1}(y_2 \oplus y_3) = y_2 y_3 \cdots y_{n-1} \oplus y_2 y_3 \cdots y_{n-1} = 0.$$

For $y_n = y_2$ that contains an odd number of variables without y_1, we have

$$y_2 y_3 \cdots y_{n-1} y_n = y_2 y_3 \cdots y_{n-1} y_2 = y_2 y_3 \cdots y_{n-1}.$$

For $y_n = y_1 \oplus y_2$ that includes y_1, the representation is

$$y_2 y_3 \cdots y_{n-1} y_n = y_2 y_3 \cdots y_{n-1}(y_1 \oplus y_2) = y_1 y_2 y_3 \cdots y_{n-1} \oplus y_2 y_3 \cdots y_{n-1}.$$

As the ANF of each N_i contains multiple monomials with $n-1$ variables, the resulting degree of each N_i is most likely $n-2$. In the matrix L with rank m, $n-m$ rows can be simultaneously represented by other m rows. Thus, $n-m$ input variables of N can be represented by other m variables. This indicates that each coordinate function F_i $(1 \le i \le n)$ has the degree $(n-1)-(n-m) = m-1 < n-1$ with a high probability. □

To further explore the probability of the degrees of different combined encodings, Table 4 illustrates the number of the coordinate functions with $d_{alg}(F_i) < 7$ and $d_{alg}((A_k)_i) < 7$. The results are counted by $10,000$ tests of randomly generated L and N. The percentage depicts the number of tests that satisfy the corresponding degrees.

- For the nibble encodings, Case 1 demonstrates that all the coordinate functions satisfy $d_{alg}((A_{k*})_i) < 7$. This indicates that the combined (non-)invertible and nibble encodings have a degree lower than 7. For an incorrect key guess, nearly all coordinates of A_k have a degree 7. Hence, the difference between the degree of each coordinate of A_k and A_{k^*} $(k \neq k^*)$ can be helpful to distinguish the correct key k^*.
- For the combination of a non-invertible linear encoding and a byte encoding, Case 2 indicates that most of the coordinate functions satisfy $d_{alg}((A_{k*})_i) < 7$. We note that the correct key can be distinguished if the number of $d_{alg}((A_{k*})_i) < 7$ is larger than the number of $d_{alg}((A_k)_i) < 7$. If $\#\{d_{alg}((A_{k*})_i) < 7\} \le \#\{d_{alg}((A_k)_i) < 7\}$, it implies for an incorrect key guess. Table 5 illustrates the probability of a failed key recovery in Case 2. The results indicate that the probability of the indistinguishability between A_{k*} and A_k is 0.37902%. Nevertheless, for a DIBO function to be attackable by an ADCA distinguisher, at least one combined encoding F out of four ones have

Table 4: The number of coordinates satisfying $d_{alg}(F_i) < 7$ or $d_{alg}((A_k)_i) < 7$ for different L and N.

Case	Construction of L	Construction of N	$d_{alg}(F_i) < 7$ $d_{alg}((A_{k*})_i) < 7$	$d_{alg}((A_k)_i) < 7$
1	(non-)invertible	nibble	8 (100%)	2 (0.07%) 1 (3.15%) 0 (96.82%)
2	non-invertible	byte	8 (18.19%) 7 (2.51%) 6 (8.69%) 5 (17.97%) 4 (22.33%) 3 (18.38%) 2 (9.20%) 1 (2.42%) 0 (0.31%)	2 (0.04%) 1 (2.66%) 0 (97.30%)
3	invertible	byte	2 (0.05%) 1 (3.12%) 0 (96.83%)	2 (0.03%) 1 (2.96%) 0 (97.04%)

$\#\{d_{alg}((A_{k*})_i) < 7\} > \#\{d_{alg}((A_k)_i) < 7\}$. Hence, the proportion of vulnerable construction is

$$1 - (0.0037902)^4 = 0.9999999998.$$

Thus, when using byte encodings combined with non-invertible linear ones, the difference between the degrees of A_{k*} and A_k still helps to distinguish the correct key with an overwhelming probability.

Table 5: The probability of an incorrect key guess for Case 2.

The Number of Coordinates		Probability
$d_{alg}((A_{k*})_i) < 7$	$d_{alg}((A_k)_i) < 7$	
2	2	$9.2\% \times 0.04\% = 0.00368\%$
1	2	$2.42\% \times 0.04\% = 0.000968\%$
	1	$2.42\% \times 2.66\% = 0.064372\%$
0	2	$0.31\% \times 0.04\% = 0.000124\%$
	1	$0.31\% \times 2.66\% = 0.008246\%$
	0	$0.31\% \times 97.30\% = 0.30163\%$
Total		0.37902%

- For the combination of an invertible linear encoding and a byte encoding, Case 3 indicates that most of the coordinate functions have $d_{alg}((A_{k*})_i) = 7$. It is hard to distinguish the correct key k^*. However, there is a fraction of the distribution satisfying $\#\{d_{alg}((A_{k*})_i) < 7\} > \#\{d_{alg}((A_k)_i) < 7\}$, which still can recover the secret key. Table 6 illustrates the probability of a successful key recovery in Case 3. The results indicate that the probability of a successful ADCA on breaking the byte encoding combined with an invertible linear encoding is 3.077648%.

Computation of Algebraic Degree. For each sample of the computation traces $\boldsymbol{v} = (v_1, v_2, \cdots, v_T)$, an ADCA attacker can obtain a mapping $(f_k)_j : \mathbb{F}_2^8 \mapsto \mathbb{F}_2 : (f_k)_j(x) = v_j$

Table 6: The probability of a correct key guess for Case 3.

The Number of Coordinates		Probability
$d_{alg}\left((A_{k*})_i\right) < 7$	$d_{alg}\left((A_k)_i\right) < 7$	
2	1	$0.05\% \times 2.96\% = 0.00148\%$
	0	$0.05\% \times 97.04\% = 0.04852\%$
1	0	$3.12\% \times 97.04\% = 3.027648\%$
Total		3.077648%

where $j \in [T]$. To compute the degree of $(f_k)_j$, ADCA constructs a system of linear equations based on ANF. Let $\mathcal{X}$ denote a set of the input variables such that $\mathcal{X} = \{x_1, x_2, \cdots, x_8\}$ where each $x_i \in \mathbb{F}_2$. Consequently, we consider that an attacker collects N computation traces $(\boldsymbol{v}^{(1)}), (\boldsymbol{v}^{(2)}), \cdots, (\boldsymbol{v}^{(N)})$ corresponding to N inputs $(x^{(1)}, x^{(2)}, \cdots, x^{(N)})$ (either random or adaptively chosen). The coordinate bits of each input $x^{(t)}$ for $t \in [N]$ form its related set $\mathcal{X}^{(t)} = \{x_1^{(t)}, x_2^{(t)}, \cdots, x_8^{(t)}\}$. For $1 \le d \le 7$, An ADCA attacker computes the d-th degree closure of each set $\mathcal{X}_d^{(t)}$ such that

$$\mathcal{X}_d^{(t)} = \{1, x_1^{(t)}, x_2^{(t)}, \cdots, x_{9-d}^{(t)} \cdots x_8^{(t)}\}.$$

The cardinality of the set $\mathcal{X}_d^{(t)}$ is $p = \binom{8}{0} + \binom{8}{1} + \cdots + \binom{8}{d}$. We note that the ANF of each function $(f_k)_j$ can be represented as follows.

$$(f_k)_j(x_1, x_2, \cdots, x_8) = a_1 \oplus a_2 x_1 \oplus a_3 x_2 \oplus \cdots \oplus a_p x_{9-d} \cdots x_8,$$

where $a_1, a_2, \cdots, a_p$ are the coefficients over $\mathbb{F}_2$. For a degree d and every key guess $k \in \mathbb{F}_2^8$, an ADCA distinguisher can solve the following system of linear equations.

$$\begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_{9-d}^{(1)} \cdots x_8^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_{9-d}^{(2)} \cdots x_8^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \cdots & x_{9-d}^{(N)} \cdots x_8^{(N)} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} (f_k)_j\left(x^{(1)}\right) \\ (f_k)_j\left(x^{(2)}\right) \\ \vdots \\ (f_k)_j\left(x^{(N)}\right) \end{pmatrix} = \begin{pmatrix} v_j^{(1)} \\ v_j^{(2)} \\ \vdots \\ v_j^{(N)} \end{pmatrix}$$

If the system is solvable for the degree d, then $d_{alg}\left((f_k)_j\right) \le d$. Otherwise, $d_{alg}\left((f_k)_j\right) > d$. This linear system attempts to represent the function $(f_k)_j$ by an ANF with degree d. If d is lower than the correct degree d^* of $(f_k)_j$, the linear system is unsolvable because it lacks the corresponding monomials in degree-d^* closure. If $d = d^*$, the resulting vector $(a_1, a_2, \cdots, a_p)$ is the coefficients of the ANF of $(f_k)_j$. If $d > d^*$, the linear system is also solvable and the corresponding coefficients of the degree-d closure are equal to 0. Since a solvable linear system has p unknown variables $a_1, \cdots a_p$, it requires at least p equations. Thus, the number of inputs and computation traces need to satisfy $N \ge p$. For ADCA with degrees from 1 to 7, the minimum number of required traces is depicted in Table 7. We note that the number is computed equally to the value of p.

Table 7: The minimum number of required traces for ADCA with degrees from 1 to 7.

Degree	1	2	3	4	5	6	7
Minimum Number of Traces (p)	9	37	93	163	219	247	255

ADCA Distinguisher. Since $\boldsymbol{v}$ contains the outputs of A_k, the mappings $(f_k)_j$ include the coordinate functions $(A_k)_i$ for $j \in [T]$ and $1 \le i \le 8$. A degree-d ADCA distinguisher

$\delta_k^{\texttt{ADCA}}$ can be defined as follows.

$$\delta_k^{\texttt{ADCA}} = \texttt{arg max} \#\{d_{alg}\left((f_k)_j\right) \leq d \mid j \in [T]\}$$

To distinguish the correct key, the attacker can select a degree d such that $1 \leq d \leq 6$. For Case 1 in Table 4, the function F has a degree at most 3 if the mapping N is a nibble encoding. Therefore, the degree $d_{alg}\left((f_k)_j\right) \leq 3$ can be used to distinguish A_{k*} and A_k. To attack both Case 1 and Case 2, the ADCA distinguisher needs to be initialized with $d = 6$. We note that if $d_{alg}(F) < 7$, $d_{alg}\left((A_{k^*})_i\right) < 7$ might hold for the correct key $k^* \in \mathbb{F}_2^8$ and $d_{alg}\left((A_k)_i\right) = 7$ might be valid for an incorrect key $k \in \mathbb{F}_2^8$. Namely, there are many coordinate functions of f_{k^*} which have degrees at most 6, while there are almost no coordinate functions of f_k such that $d_{alg}(f_k) < 7$. Hence, the distinguisher will try to solve the linear system of degree 6 for each key guess. For the correct key k^*, the corresponding coordinates $(f_{k^*})_j$ have the maximum number of functions such that $d_{alg}\left((f_{k^*})_j\right) \leq 6$. To enhance the attack ability of ADCA on breaking Case 2, we also propose the other two distinguishers.

1. The first is to compute the total number of coordinate functions $(f_k)_j$ which satisfy $d_{alg}\left((f_k)_j\right) \leq d$ for every $1 \leq d \leq 6$ and each key $k \in \mathbb{F}_2^8$. The key guess corresponding to the maximum number turns out to be the correct key. Such an improved ADCA distinguisher $(\delta_k^{\texttt{ADCA}})_{I1}$ is depicted as follows.

$$(\delta_k^{\texttt{ADCA}})_{I1} = \texttt{arg max} \sum_{1 \leq d \leq 6} \#\{d_{alg}\left((f_k)_j\right) \leq d \mid j \in [T]\}$$

2. Case 2 illustrates that the probability for at least one coordinate function of A_{k^*} with $d_{alg}\left((A_{k*})_i\right) < 7$ is 99.69%. For an incorrect key guess, at least one coordinate function of A_k with $d_{alg}\left((A_k)_i\right) < 7$ has a much lower probability 2.7%. Hence, for a given degree d, the key guesses corresponding to $d_{alg}\left((f_k)_j\right) \leq d$ most likely contain the correct key. The second enhanced ADCA distinguisher $(\delta_k^{\texttt{ADCA}})_{I2}$ is described as follows.

$$(\delta_k^{\texttt{ADCA}})_{I2} \in \{d_{alg}\left((f_k)_j\right) \leq d \mid j \in [T]\}$$

Attack Complexity. The distinguisher $\delta_k^{\texttt{ADCA}}$ constructs a system of linear equations to detect the degree of each coordinate function $(A_k)_j$ for every key guess $k \in \mathbb{F}_2^8$ and $j \in [T]$. Let D denote an $N \times p$ $(N \geq p)$ matrix consisting of the closures of input variables, $\boldsymbol{a}$ be the resulting vector for the coefficients of ANF, and $\bar{\boldsymbol{v}}$ denote the vector of the same samples in different traces. The linear system can be simply represented as $D \cdot \boldsymbol{a} = \bar{\boldsymbol{v}}$. For a selected degree d, the matrix D can be pre-computed by calculating the degree-d closures for the variables of N inputs. This process does not involve the computation of traces and the prediction of keys. To verify the solvability of the linear system, an ADCA distinguisher can compute the rank of the matrix D and the one of the augmented matrix $(D \mid \bar{\boldsymbol{v}})$. Let $r(D)$ denote the rank of the matrix D. If $r(D) \geq r(D \mid \bar{\boldsymbol{v}})$, then the linear system is solvable. If $r(D) < r(D \mid \bar{\boldsymbol{v}})$, the linear system has no solution. Since the matrix D is pre-computed, its rank can also be pre-computed. Moreover, the operational steps for the row reduction to its echelon form can be pre-stored in the program. This process has the time complexity of $\mathcal{O}(N \cdot p^2)$, which is the same as the Gaussian Elimination.

In the ADCA attack process, the attacker collects the computation traces and computes $\bar{\boldsymbol{v}}$ based on every sample v_j in different traces. We note that the row reduction of $(D \mid \bar{\boldsymbol{v}})$ to its echelon form can be reduced to the row reduction on D. Since the operational steps for computing $r(D)$ are pre-stored, these steps can be operated on $\bar{\boldsymbol{v}}$. Let r denote the operational steps for the row reduction on $\bar{\boldsymbol{v}}$. Based on the Gauss Elimination of D, r has the maximal value $N \cdot p$. Subsequently, the attacker can compute $r(D \mid \bar{\boldsymbol{v}})$ based on the row reduced D and $\bar{\boldsymbol{v}}$. The solvability of the linear system can be verified by the comparison between the ranks of D and $(D \mid \bar{\boldsymbol{v}})$. Thus, the theoretical time complexity of ADCA is $\mathcal{O}(r \cdot T \cdot |k|) = \mathcal{O}(N \cdot p \cdot T \cdot |k|)$. Since p is the number of the degree-d closure, the time complexity of ADCA is related to its degree. Particularly, we note that the operational steps of the Gauss Elimination for $\bar{\boldsymbol{v}}$ are pre-computed. In practice, the values r are fixed for different degrees, which are irrelevant to the collected traces. For attacking the refined structure of a DIBO function, we have $N = 2^8$, $T = 8$, and $|k| = 2^8$. Table 8 illustrates the exact operating times r for the Gauss Elimination on the trace vector $\bar{\boldsymbol{v}}$ and the resulting time complexities of ADCA with different degrees.

Table 8: The time complexities of ADCA with different degrees.

Degree	1	2	3	4	5	6	7
r	$1,277$ $\approx 2^{10.32}$	$3,177$ $\approx 2^{11.63}$	$5,474$ $\approx 2^{12.42}$	$7,563$ $\approx 2^{12.88}$	$8,426$ $\approx 2^{13.04}$	$8,604$ $\approx 2^{13.07}$	$8,792$ $\approx 2^{13.10}$
Time Complexity	$2^{21.32}$	$2^{22.63}$	$2^{23.42}$	$2^{23.88}$	$2^{24.04}$	$2^{24.07}$	$2^{24.10}$

4.3 ADCA against Internal Encodings

ADCA is a new analysis on the degrees of internal encodings. It matches the degrees of the mappings from the inputs to the intermediate variables. Based on our analysis, the degree of the target function $A_k(x) = F \circ S \circ \oplus_{k*} \circ \oplus_k \circ S^{-1}$ dependents on the key guess k and the construction of the encoding F. The detailed dependency is described as follows.

- If the key guess is correct (i.e., $k = k^*$), it implies that $A_{k^*}(x) = F$. Thus, $d_{alg}(A_{k^*}) = d_{alg}(F)$. This indicates that the degree of the function A_{k^*} depends on the encoding F. Due to $1 \leq d_{alg}(F) \leq 7$ based on different generations, the function A_{k^*} has the degree $1 \leq d_{alg}(A_{k^*}) \leq 7$.

- If the key guess is incorrect (i.e., $k \neq k^*$), the function A_k has a more complicated structure which includes an Sbox and its inverse. Thus, the degree of A_k is most likely 7.

To precisely detect the differences between the degrees of A_{k^*} and A_k, ADCA computes the degree of each coordinate function. ADCA distinguishes the function A_{k^*} based on the maximum number of $d_{alg}((A_{k^*})_j) < 7$. For the different degrees of the encoding F, the number of the coordinate functions with a degree less than 7 is illustrated in Table 9. The results are counted by $10,000$ tests of randomly generated F with $1 \leq d_{alg}(F_i) \leq 7$. The percentage describes the number of tests that satisfy the corresponding degrees. The results demonstrate that if $1 \leq d_{alg}(F) \leq 6$, all coordinate functions of A_{k*} satisfying $d_{alg}((A_{k*})_i) < 7$ while almost all coordinate functions of A_k have degree 7. Thus, ADCA can distinguish the functions A_{k*} and A_k if $d_{alg}(F) \leq 6$. However, if $d_{alg}(F) = 7$, ADCA might be failed since both A_{k^*} and A_k have a similar number of degree-7 coordinate functions. Thus, the essential condition for a successful computation analysis is that the degrees of the internal encodings are less than 7. The different encodings might have various degrees which lead to different security levels. If the white-box implementation is protected by nibble encodings, the combined encoding F has a degree at most 3. ADCA might defeat this implementation with an overwhelming probability. Although a randomly generated byte permutation has $d_{alg} = 7$, its combination with a non-invertible linear mapping might reduce to a lower degree. Thus, it is still vulnerable to ADCA.

Table 9: The number of coordinates satisfying $d_{alg}(F_i) < 7$ or $d_{alg}((A_k)_i) < 7$ for different degrees of F.

$d_{alg}(F)$	$d_{alg}(F_i) < 7$ $d_{alg}((A_{k*})_i) < 7$	$d_{alg}((A_k)_i) < 7$	Attack of ADCA
1	8 (100%)	0 (100%)	success
2	8 (100%)	2 (0.02%) 1 (2.41%) 0 (97.57%)	success
3	8 (100%)	2 (0.05%) 1 (2.90%) 0 (97.05%)	success
4	8 (100%)	2 (0.02%) 1 (1.92%) 0 (98.06%)	success
5	8 (100%)	2 (0.01%) 1 (2.12%) 0 (97.87%)	success
6	8 (100%)	2 (0.02%) 1 (2.83%) 0 (97.15%)	success
7	2 (0.04%) 1 (2.95%) 0 (97.01%)	2 (0.05%) 1 (3.06%) 0 (96.89%)	failure

5 Comparison and Experimental Results

5.1 The Comparison between ADCA and The Published Distinguishers

ADCA is a higher-degree computation analysis to match the degrees of each Boolean function from the inputs to the samples of traces. Similar to DCA, an ADCA adversary only needs to analyze the accessed memory during the execution without the knowledge of encoding details. Any modification of the implementation is not necessarily required. Thus, it is a gray-box attack to perform statistical analysis on the computation traces. ADCA can also be mounted automatically without reverse engineering efforts. For the encodings with degrees from 1 to 6, the time complexity of a successful ADCA ranges from $2^{21.32}$ to $2^{24.07}$. For instance, a degree-3 ADCA can break the CEJO framework consisting of the nibble encodings with the time complexity $2^{23.42}$. To defeat most cases of nibble/byte encodings, a degree-6 ADCA is proposed as a general attack with the time complexity $2^{24.07}$. Differently, the time complexities of SA and ISA are 2^{27} and 2^{32} respectively, which are higher than ADCA. The summarized properties of ADCA are illustrated in Table 10.

Table 10: The properties of ADCA with degrees from 1 to 6.

Distinguisher	Attack Context	Method	Analysis of Key Leakage	Time Complexity
ADCA (degree 1 to 6)	gray-box	detection of algebraic degree	$d_{alg}(F) \leq 6$	$2^{21.32} \sim 2^{24.07}$

The main differences between ADCA and other computation analyses are described as follows.

- ADCA defeats the internal encodings by leveraging their degrees. The previous methods are based on correlation computation and spectral analysis. It is the first computation analysis based on the detection of degrees. ADCA explores the combined encodings without the partition of linear and non-linear parts. Thus, it focuses on the nonlinearity of the encodings instead of the HW, imbalance, and invertibility of linear encodings. Based on the theoretical analysis, ADCA can distinguish the correct key of the white-box implementations if the encodings are constructed with a degree lower than 7.
- ADCA is a higher-degree computation analysis. For $1 \leq d \leq 6$, a degree-d ADCA can defeat the internal encodings with a degree no more than d. It matches the degrees of the mappings from the inputs to each sample of traces by solving a system of linear equations. Moreover, the time complexity of ADCA is related to the attacking degree. Thus, ADCA has a flexible time complexity. In addition, ADCA has an explicit number of required traces for a practical attack on the encodings with different degrees.

We note that ADCA pre-computes the coefficients of a linear system, which does not depend on any key guess and trace sample. This process helps to reduce the time complexity of the Gauss Elimination. Differently, the processes of SA/ISA depend on the key guess and trace sample, which need to be calculated on-the-fly. Although ADCA is similar to ADA, the attack contexts and the definition of distinguishers are different. For ADA, it is a structural attack on the CEJO framework with the nibble encodings. ADA focuses on the degree of a specific LUT and uses its higher-order derivative to distinguish the correct key. However, the target function of ADCA is the mapping from the inputs to each sample of the computation traces. An ADCA attacker does not need to pinpoint a specific function and only focuses on the statistical analysis of traces. Furthermore, ADCA verifies the degree of a Boolean function instead of a vectorial Boolean function. A degree-d ADCA solves a system of linear equations to detect if the degree of the Boolean function is no more than d. We note that this process is not a precise computation but an estimate of the range of the degree.

Following the techniques of internal encodings, a countermeasure of ADCA is to combine an invertible linear mapping and a byte encoding. Although this countermeasure is the same as the proposed one of ISA, the core ideas behind ISA and ADCA are different. ISA highlights the invertibility of linear encoding without revealing the impact of the byte encoding. ADCA focuses on the degree of the combined encoding instead of the properties of its (non-)linear part. Since the combined byte encoding with an invertible linear mapping has degree 7, such an encoding can mitigate ADCA. Thus, we also propose another countermeasure by only applying a random byte permutation and making sure that the degrees of all the coordinate functions are 7.

ADCA constructs a system of linear equations to verify the degrees of Boolean functions. This process of solving the linear system is similar to *linear decoding analysis* (LDA) [GPRW20]. However, the target implementations of ADCA and LDA are different. LDA is proposed to break the linear masking scheme. This white-box implementation is implemented as a Boolean circuit without any LUT and internal encoding. The linear masking splits each sensitive variable x into n shares satisfying $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$. Let $(v_1, v_2, \cdots, v_N)$ denote a computation trace. LDA attempts to solve the following linear system to recover the n linear shares of the sensitive variable x.

$$\bigoplus_{i=1}^{T} a_i \cdot v_i = x$$

The resulting vector $(a_1, a_2, \cdots, a_T)$ indicates the locations of shares in the trace. If the system is solvable, the key guess corresponding to x is most likely correct. Different from LDA, ADCA defeats the internal encodings of white-box implementations. It constructs a linear system by the closure of input variables and the same samples in different traces. Moreover, the solvability of the linear system is related to the degrees of Boolean functions instead of the shares of masking schemes.

5.2 The Experimental Results on Breaking Internal Encodings

To compare the attack capabilities of various distinguishers, we perform an experiment to mount the different computation analyses on breaking different encodings. The target function is the refined structure $\overline{O_k}$ (refer to Figure 2). The encodings are generated by the optimized white-box matrix library `WBMatrix` [TGS+22]. Based on the proposed internal encodings, we generate the linear encodings by different `HW` and invertibility whilst the non-linear ones are constructed by the nibble or byte permutations. Simple linear and non-linear encodings are also provided for the experiment. Each attack case is executed over $1,000$ times on randomly generated encodings. The numbers of the broken tests for different computation analyses are illustrated in Table 11. As a general attack, ADCA is mounted with degree 6 in various cases. Moreover, to validate the influence of different degrees of encodings, Table 12 depicts the results of various computation analyses against the encodings with degrees from 1 to 7. Particularly, the attack degrees of ADCA are equal to the corresponding degrees of encodings. From the attack results, we can classify the distinguishers into two categories as follows.

- SA, ISA, and ADCA break the maximum number of attack cases (11 out of 14 and 6 out of 7, respectively in two tests) with various encodings. The broken cases for different combinations include the constructions of (1) linear encodings with nibble encodings, (2) non-invertible linear encodings with byte encodings, (3) linear encodings, and (4) nibble encodings. The broken cases for different degrees contain random encodings with degrees from 1 to 6. Thus, the encoding constructed by a non-linear byte function with the combination of an invertible linear mapping, or a degree-7 random permutation can resist SA, ISA, and ADCA attacks.
- Other computation analyses, such as DCA, IDCA, CPA, CA, MIA, and MSA, can break no more than 9 and 2 cases, respectively in two experiments. Among these attacks, IDCA has the maximum number of broken cases. The results also reveal that the attack capabilities of DCA, CPA, CA, and MIA are equal since they have a similar number of broken tests in each attack case.

The results of SA, ISA, and ADCA prove that the nibble encodings cannot prevent the computation analysis. Moreover, the linear encodings with $\texttt{HW} > 1$ are still vulnerable to ADCA. Thus, the analysis of `HW` cannot fully reveal the cause of key leakage. From the failed attacks of CPA, CA, and MIA, even the nibble encoding introduces a non-injection of the sensitive variable, these distinguishers also cannot defeat this type of internal encoding. Hence, the models of CPA, CA, and MIA still cannot fully interpret the ineffectiveness of internal encodings. The results of MSA support that the mono-bit model reduces the capability of SA. Thus, the poor performance of MSA cannot validate the reason for key leakage by the imbalance of linear encodings. Besides, the similar results of SA and ISA demonstrate that the higher time complexity of ISA does not enhance the attack capability of SA. The failed attack on the combined encodings (an invertible linear mapping with a byte encoding) confirms the analysis of ISA on the invertibility of linear encodings. Although an 8-bit random permutation has a degree 7, its combination of a non-invertible linear mapping will reduce the degree of the coordinate functions with a high probability.

Table 11: The successful attacks (each out of 1,000 tests) for different distinguishers δ against various internal encodings F.

F / δ	Non-Linear Nibble				Non-Linear Byte				Without Non-Linear				Without Linear	
	Invertible Linear		Non-Invertible Linear		Invertible Linear		Non-Invertible Linear		Invertible Linear		Non-Invertible Linear		Non-linear	
	`HW` > 1	`HW` $= 1$	`HW` > 1	`HW` $= 1$	`HW` > 1	`HW` $= 1$	`HW` > 1	`HW` $= 1$	`HW` > 1	`HW` $= 1$	`HW` > 1	`HW` $= 1$	Nibble	Byte
DCA [BHMT16]	605	959	590	971	6	7	182	215	173	1,000	186	1,000	1000	2
IDCA [BBB+19]	1,000	1,000	1,000	1,000	2	0	511	493	1,000	1,000	1,000	1,000	1,000	0
CPA [RW19]	601	959	582	964	5	8	178	214	173	1,000	186	1,000	1,000	3
CA [RW19]	601	959	582	964	5	8	178	214	173	1,000	186	1,000	1,000	3
MIA [RW19]	601	959	582	964	5	8	178	214	173	1,000	186	1,000	1,000	3
SA [SMG16]	1,000	1,000	1,000	1,000	2	3	1,000	1,000	1,000	1,000	1,000	1,000	1,000	3
MSA [LJK20]	969	906	967	882	2	5	618	464	999	984	1,000	872	0	5
ISA [CGM21]	1,000	1,000	1,000	1,000	1	6	1,000	1,000	1,000	1,000	1,000	1,000	1,000	5
ADCA (Our Proposal)	1,000	1,000	1,000	1,000	2	4	1,000	1,000	1,000	1,000	1,000	1,000	1,000	5

Table 12: The successful attacks (each out of 1,000 tests) for various distinguishers δ against the encodings F with degrees from 1 to 7.

F / δ	Degree						
	1	2	3	4	5	6	7
DCA [BHMT16]	223	827	535	530	579	170	8
IDCA [BBB$^+$19]	1,000	977	951	925	1,000	478	2
CPA [RW19]	223	840	540	521	578	162	7
CA [RW19]	223	840	540	521	578	162	7
MIA [RW19]	223	840	540	521	578	162	7
SA [SMG16]	1,000	1,000	1,000	1,000	1,000	1,000	3
MSA [LJK20]	1,000	968	712	786	835	438	1
ISA [CGM21]	1,000	1,000	1,000	1,000	1,000	1,000	3
ADCA (Our Proposal)	1,000	1,000	1,000	1,000	1,000	1,000	3

Thus, the analysis of ADCA on the degrees of internal encodings also explains the results of ISA.

For the case of degree 7 in Table 12, the broken tests of ADCA are less than the ones of DCA. To verify their capabilities, we mount DCA and ADCA extra 10,000 times for breaking the degree-7 encodings. The results indicate that DCA (40 broken tests) can break less than ADCA (44 broken tests). Due to the randomness of encodings, the results of degree 7 are different in each experiment. Since ISA has a time complexity of nearly 2^{42} for $1,000$ tests, it is not practical to run more than $1,000$ tests for ISA. We consider that it is acceptable for the comparison among the proposed attacks with $1,000$ tests. As the number of the broken tests is far less than $1,000$, the degree-7 encodings can be a countermeasure against the computation analysis with a high probability. Although SA and ISA can break the same maximal cases as ADCA, they have higher time complexities 2^{27} and 2^{32}, respectively. We note that ISA might also require reverse engineering efforts since it is a structural attack. For SA, it does not reveal the weaknesses of different combinations of linear and non-linear encodings. Thus, ADCA can be a general attack to break the most cases of encodings (degrees from 1 to 6) with the lowest time complexity $2^{21.32} \sim 2^{24.07}$.

6 Conclusion

This paper revisits the proposed computation analyses on a refined structure from the DIBO function. Besides, the flaws in the previous results on the weakness of internal encodings are discussed. By leveraging the nonlinearity of encodings, we propose a new ADCA attack. ADCA distinguishes the correct key by matching the degrees of the mappings from the inputs to the samples of traces. The analysis of ADCA reveals that the key leakage of internal encodings depends on their degrees. The encodings with a degree 7 can resist the computation analysis. To compare the attack capability, we mount the various distinguishers on different encodings. The results prove that ADCA can break the maximum number of encoding cases with the lowest time complexity. Moreover, the impact of the degrees of internal encodings against ADCA is validated in the experiment.

To defeat the degree-7 encoding, an interesting case is to verify if the proposed computation analyses are vulnerable to the $\mathbb{F}_2^{16} \mapsto \mathbb{F}_2^8$ mapping which has a larger space of inputs. It is also a challenge to find an internal encoding that can be provably secure against all the proposed computation analyses.

Acknowledgments

We would like to thank Jian Guo and the anonymous reviewers for their valuable comments. We are also grateful to Qunxiong Zheng for the helpful discussions on Theorem 3. This work was supported by National Natural Science Foundation of China (62072192), National Defense Technology 173 Basic Improvement Project (2121-JCJQ-JJ-0931), National Cryptography Development Fund (MMJJ20180205), Guangdong Basic and Applied Basic Research Foundation (2022A1515140090), and Natural Science Foundation of Jiangsu Province, China (Grant No. BK20220075).

References

[AH16] Hyunjin Ahn and Dong-Guk Han. Multilateral white-box cryptanalysis: Case study on WB-AES of CHES challenge 2016. *IACR Cryptol. ePrint Arch.*, page 807, 2016.

[BBB$^+$19] Estuardo Alpirez Bock, Joppe W. Bos, Chris Brzuska, Charles Hubain, Wil Michiels, Cristofaro Mune, Eloi Sanfelix Gonzalez, Philippe Teuwen, and Alexander Treff. White-box cryptography: Don't forget about grey-box attacks. *J. Cryptol.*, 32(4):1095–1143, 2019.

[BBIJ17] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, and Martin Bjerregaard Jepsen. Analysis of software countermeasures for whitebox encryption. *IACR Trans. Symmetric Cryptol.*, 2017(1):307–328, 2017.

[BBMT18] Estuardo Alpirez Bock, Chris Brzuska, Wil Michiels, and Alexander Treff. On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 103–120. Springer, 2018.

[BCD06] Julien Bringer, Herve Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. Cryptology ePrint Archive, Paper 2006/468, 2006. https://eprint.iacr.org/2006/468.

[BGE04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2004.

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In

Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.

[BI15] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1058–1069. ACM, 2015.

[BIT16] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158, 2016.

[BU21] Alex Biryukov and Aleksei Udovenko. Dummy shuffling against algebraic attacks in white-box implementations. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 219–248. Springer, 2021.

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[CGM21] Claude Carlet, Sylvain Guilley, and Sihem Mesnager. Structural attack (and repair) of diffused-input-blocked-output white-box cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):57–87, 2021.

[GPRW20] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *J. Cryptogr. Eng.*, 10(1):49–66, 2020.

[Kar10] Mohamed Karroumi. Protecting white-box AES with dual ciphers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers*, volume 6829 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2010.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[LJK20] Seungkwang Lee, Nam-Su Jho, and Myungchul Kim. On the linear transformation in white-box cryptography. *IEEE Access*, 8:51684–51691, 2020.

[LN05] Hamilton E. Link and William D. Neumann. Clarifying obfuscation: Improving the security of white-box DES. In *International Symposium on Information Technology: Coding and Computing (ITCC 2005), Volume 1, 4-6 April 2005, Las Vegas, Nevada, USA*, pages 679–684. IEEE Computer Society, 2005.

[LRM$^+$13] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box AES implementation. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 265–285. Springer, 2013.

[MRP12] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the xiao - lai white-box AES implementation. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2012.

[Mul14] Yoni De Mulder. *White-Box Cryptography: Analysis of White-Box AES Implementations (White-Box Cryptografie: Analyse van White-Box AES implementaties)*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2014.

[MWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box AES implementation. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2010.

[RW19] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):225–255, 2019.

[SEL21] Okan Seker, Thomas Eisenbarth, and Maciej Liskiewicz. A white-box masking scheme resisting computational and algebraic attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):61–105, 2021.

[SMG16] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box - - A hardware implementation and its side channels -. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2016.

[TGCX23] Yufeng Tang, Zheng Gong, Jinhai Chen, and Nanjiang Xie. Higher-order DCA attacks on white-box implementations with masking and shuffling countermeasures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):369–400, 2023.

[TGS+21] Yufeng Tang, Zheng Gong, Tao Sun, Jinhai Chen, and Fan Zhang. Adaptive side-channel analysis model and its applications to white-box block cipher implementations. In Yu Yu and Moti Yung, editors, *Information Security and Cryptology - 17th International Conference, Inscrypt 2021, Virtual Event, August 12-14, 2021, Revised Selected Papers*, volume 13007 of *Lecture Notes in Computer Science*, pages 399–417. Springer, 2021.

[TGS+22] Yufeng Tang, Zheng Gong, Tao Sun, Jinhai Chen, and Zhe Liu. WBMatrix: An optimized matrix library for white-box block cipher implementations. *IEEE Trans. Computers*, 71(12):3375–3388, 2022.

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 264–277. Springer, 2007.

[XL09] Yaying Xiao and Xuejia Lai. A secure implementation of white-box AES. In *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6. IEEE, 2009.

[ZMAB19] Mohamed Zeyad, Houssem Maghrebi, Davide Alessio, and Boris Batteux. Another look on bucketing attack to defeat white-box implementations. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 99–117. Springer, 2019.