IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 369–400. DOI:10.46586/tches.v2023.i1.369-400

Higher-Order DCA Attacks on White-Box Implementations with Masking and Shuffling Countermeasures

Yufeng Tang, Zheng Gong(✉), Jinhai Chen and Nanjiang Xie

School of Computer Science, South China Normal University, Guangzhou, China
cis.gong@gmail.com

Abstract. On white-box implementations, it has been proven that differential computation analysis (DCA) can recover secret keys without time-costly reverse engineering. At CHES 2021, Seker *et al.* combined linear and non-linear masking protections (SEL masking) to prevent sensitive variables from being predicted by DCA. At Eurocrypt 2021, Biryukov and Udovenko introduced a public dummy shuffling construction (BU shuffling) to protect sensitive functions. In this paper, we extend higher-order DCA (HO-DCA) to higher-degree context for exploiting the vulnerabilities against the state-of-the-art countermeasures. The data-dependency HO-DCA (DDHO-DCA), which is proposed at CHES 2020, is improved to successfully recover the correct key of SEL masking. In specific, our improved DDHO-DCA can also enhance the attack result of #100 which is the third winning challenge in WhibOx 2019. Since the XOR phase plays the same role as linear masking, we prove that a specific BU shuffling is vulnerable to HO-DCA attacks. Furthermore, we demonstrate that the combination of SEL masking and the specific BU shuffling still cannot defeat our higher-degree HO-DCA and improved DDHO-DCA attacks.

Keywords: White-box Implementation · Masking · Dummy Shuffling · Higher-order DCA · Data-dependency Attack

1 Introduction

White-box cryptography aims to securely execute cryptographic primitives even in a hostile environment where an adversary has full control over the implementation. In the white-box context, the attacker can get full access to the memory and modify all resources during the execution. In 2002, Chow *et al.* [CEJvO02a, CEJvO02b] proposed the seminal white-box implementations of AES and DES, which are categorized as the CEJO framework. The principle of the CEJO framework is to convert full-round operations into a series of lookup tables (LUTs) with embedded keys. Therefore, each table is obfuscated by (non)linear encodings to protect the secret keys. However, Billet et al. [BGE04] exhibited a structural attack (which is called BGE analysis) on Chow *et al.*'s white-box AES to retrieve the key and encodings. Although many improved white-box implementations are proposed [BCD06, XL09, Kar10], all of them are broken latter by BGE analysis and its successors [MWP10, MRP12, MRP13].

However, BGE analysis requires construction details, which implies extra efforts on reverse engineering in practice. To avoid these time-costly efforts, it is inspired to apply *side-channel analysis* (SCA) to evaluate the security of white-box implementations. At CHES 2016, Bos *et al.* [BHMT16] introduced *differential computation analysis* (DCA) as a software counterpart of *differential power analysis* (DPA). DCA records the computation traces in a noise-free context and exploits them to recover the secret key under the

Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

DPA model. It can be fully automated and breaks many publicly available white-box implementations. Hereafter, many enhanced DCA attacks [BBIJ17, RW19, ZMAB19] have been proposed to reduce the amount of required traces and plaintexts.

To resist DCA, a natural idea is to adapt the obfuscation methods, such as *masking* and *shuffling* countermeasures from the gray-box model [BRVW19]. Due to the introduction of secret shares, the white-box implementation is represented as a bitsliced Boolean circuit. Linear masking splits a sensitive variable into multiple shares and processes them securely without leaking any information. However, it has been proven that linear masking is vulnerable to *higher-order* DCA (HO-DCA) [BRVW19] and *algebraic* DCA [BU18] (also called *linear decoding analysis* (LDA) [GPRW20]). At Asiacrypt 2018, Biryukov and Udovenko [BU18] proposed a non-linear masking scheme against algebraic attacks. The non-linear masking ensures that applying any linear function to the intermediate vectors will not recover any sensitive variable. Unfortunately, this proposal is also broken by DCA based on the recent studies [GRW20, SEL21]. At CHES 2020, Goubin *et al.* [GRW20] discussed three cases to combine linear and non-linear masking schemes and demonstrated their weakness against HO-DCA, *higher-degree decoding analysis* (HDDA), and *data-dependency* HO-DCA (DDHO-DCA), respectively. At CHES 2021, Seker *et al.* [SEL21] proposed a combined masking scheme (SEL masking) to defeat algebraic DCA.

To add more randomness in the implementations, shuffling desynchronizes the computation traces by switching the order of several independent operations (e.g., Sbox). At CHES 2017, Banik *et al.* [BBIJ17] analyzed three cases of software countermeasures to counteract DCA by shuffling the order of table accesses. The cases are control-flow obfuscation, table location randomization, and dummy operations. They also proposed *zero difference enumeration* attack and broke these countermeasures. Bogdanov *et al.* [BRVW19] distinguished two dimensions, namely time and memory. *Time shuffle* randomizes the order of operations while *memory shuffle* rearranges the addresses of intermediate states. However, a DCA adversary can reorder the traces by the accessed memory addresses when the order of computations is shuffled in time. Hence, both time and memory shuffles are necessary to resist DCA. Bogdanov *et al.* also adapted the *template attacks* [CRR02] as *multivariate HO-DCA* against a masked and shuffled implementation. Goubin *et al.* [GRW20] introduced *horizontal* shuffling to randomize the computation in memory. *Vertical* shuffling permutes the order of successive operations. It can shuffle both the order of computations and memory locations. However, they also applied an *integrated HO-DCA* to defeat the horizontal shuffling. At Eurocrypt 2021, Biryukov and Udovenko [BU21] defined *dummy shuffling* by adding dummy operations to hide sensitive function and distinguished *hidden* and *public* shuffling. They also proposed a public dummy shuffling construction (BU shuffling) which has an explicit separation of each sensitive functions. The main function computes on the correct input while the dummy function computes on a randomly generated input. An XOR phase is then introduced to retrieve the real output.

To motivate the white-box AES implementations, WhibOx 2017 [PCY+] and 2019 [FMIS+] contests were organized for publicly examining the protection/attacking methods. Although the submitted schemes are not required to reveal their construction details, the enhanced SCA techniques are successfully applied on the challenges. Goubin *et al.* [GPRW20] studied the ranking first challenge in WhibOx 2017 and proposed LDA to break it. Bock and Tref [BT20] discussed the robustness of the challenges in WhibOx 2017 to defeat DCA and fault attacks. Goubin *et al.* [GRW20] presented DDHO-DCA and integrated HO-DCA against the three winning implementations with masking/shuffling countermeasure in WhibOx 2019.

Our Contribution. For the improved white-box countermeasures, DCA-like attacks become the pivotal threats to the implementations. Both SEL masking and BU shuffling are proposed to resist DCA and algebraic DCA attacks. However, the practical security

of these schemes against HO-DCA attacks is still pending for more intensive analysis. In specific, the security of dummy shuffling has not been practically analyzed. This paper aims to apply HO-DCA attacks against masking and shuffling countermeasures in white-box implementations. Our contributions are threefold.

1. **HO-DCA attacks against masking and shuffling countermeasures.** We revisit the techniques of HO-DCA attacks including HO-DCA, HDDA, DDHO-DCA, and integrated HO-DCA. Subsequently, we implement a bitsliced AES protected by SEL masking and analyze its security against HO-DCA, HDDA, and DDHO-DCA. We also implement an AES implementation protected by a specific BU shuffling. In this construction, each sensitive function is a key-embedded Sbox. Due to the masks XOR to zero, the XOR phase of the specific BU shuffling is reduced to linear masking. Hence, HO-DCA and algebraic DCA have been proven to defeat such a specific BU shuffling. Furthermore, we discuss a practical way to combine SEL masking and specific BU shuffling. The analysis results support that HO-DCA attacks can still defeat the combined countermeasures. The analyzed implementations and HO-DCA attacks are both formalized in a generalized form. The time complexities on computing the traces of HO-DCA attacks are provided as well.

2. **Higher-degree HO-DCA attack.** HO-DCA only recovers the linear shares of masking schemes. For defeating the non-linear masking, we propose a higher-degree HO-DCA (HDHO-DCA) attack. A degree-d HDHO-DCA extends the probed traces by multiplying all the d tuples and then applies HO-DCA on the higher-degree traces. This attack also generalizes HDDA to deal with linear masking and thus it is a generic attack against masking. We prove that HDHO-DCA not only defeats SEL masking but also breaks the combination of masking and dummy shuffling countermeasures.

3. **Improved data-dependency HO-DCA attack.** Although HDHO-DCA is a generic attack of masking, it is a brute-force approach by computing all the products and sums of shares. To reduce the time complexity, we mount DDHO-DCA on SEL masking by tracing the co-operands of variables through an AND instruction. However, our experimental results show that DDHO-DCA fails to break a certain AES implementation with SEL masking. A comprehensive study reveals that the target variables and the tracing values of DDHO-DCA are incorrect. To fill up this gap, we propose three paths to improve DDHO-DCA, namely targeting the input variable of AND gates, exploiting the variables with the same multipliers, and tracing the nodes in the second round. The improved attack successfully retrieves the correct key of SEL masking and enhances the attack results of challenge #100 in WhibOx 2019. We also introduce two security notions to mitigate the improved DDHO-DCA.

For summarizing the cryptanalysis results, Table 1 illustrates the time complexity of HO-DCA attacks to break SEL masking, a specific BU shuffling, and their combined countermeasure. HO-DCA attacks can be divided into two phases that are computing higher-order traces and analyzing traces for key recovery. We note that the time complexity of key recovery is negligible compared with the one of trace computation. Thus, the time complexity is used to denote the costs of computing higher-degree/higher-order traces throughout this paper. The results indicate that the time complexity of our improved DDHO-DCA only depends on the numbers of AND gadgets m and gates s. Hence, it is independent of the order of masking and the degree of shuffling. Compared with HDHO-DCA, the improved DDHO-DCA has an advantage on the time complexity. The source code of the implementations and HO-DCA attacks is open-sourced [1].

[1] https://github.com/scnucrypto/HO-DCA

Table 1: HO-DCA attacks on SEL masking, a specific BU shuffling, and their combination.

Countermeasures \ Attacks	HDDA [GRW20] [BU21]	Integrated HO-DCA [GRW20]	HDHO-DCA (brute-force attack) (Section 3.1)	Improved DDHO-DCA (Section 3.2)
SEL masking [SEL21]	$\mathcal{O}\left((t+q)^{2.8}\right)$	not applicable	$\mathcal{O}\left(\binom{t+q}{n+1}\right)$	$\mathcal{O}(ms)$
A specific BU shuffling (Section 4.1)	$\mathcal{O}(t^{2.8})$	$\mathcal{O}(t)$	$\mathcal{O}(p)$	not applicable
Combined masking and shuffling (Section 4.3)	$\mathcal{O}\left((p+g)^{2.8}\right)^{\dagger}$ $\mathcal{O}\left((t+q)^{2.8}\right)$	$\mathcal{O}(t)$	$\mathcal{O}\left(\binom{p+g}{n+1}\right)^{\dagger}$ $\mathcal{O}\left(\binom{t+q}{h^{d+1}+eh}\right)$	$\mathcal{O}(ms)$

Note: The dimension of a computation trace is t, the masking degree is $d+1$, the masking order is n, and the shuffling degree is h. The tracing function contains m AND gadgets and each gadget consists of s AND gates. Let $p = \binom{t}{h}$, $q = \binom{t}{d+1}$, and $g = \binom{p}{d+1}$. The symbol † represents that the attack contains a pre-processing step.

Organization. The remainder of this paper is organized as follows. Section 2 reviews HO-DCA and the masking/shuffling countermeasures in white-box implementations. Section 3 studies HO-DCA against SEL masking. A higher-degree HO-DCA is proposed. It also improves DDHO-DCA to break SEL masking and challenge #100 in WhibOx 2019. In Section 4, HO-DCA attacks are analyzed to defeat a specific BU shuffling and its combination with SEL masking. Section 5 concludes this paper.

2 Preliminaries

2.1 Computational and Algebraic Attacks

Differential Computation Analysis. Bos *et al.* [BHMT16] proposed DCA (*computational attack* in more general sense) to break white-box implementations without the knowledge of the underlying construction details. DCA adapts the technique of DPA to the white-box context. In the noise-free context, DCA successfully breaks many public white-box implementations. It firstly invokes the implementation several times with different plaintexts. During each execution, the *software computation traces* are collected through a dynamic instrumentation tool, such as Intel PIN. A DCA attacker then makes a key guess to predict the sensitive intermediate variables based on the same plaintexts. Subsequently, the correlation between the predictions and each sample of the computation traces is computed. A key guess with the highest correlation is probably a correct key. The function for calculating an intermediate state is defined as a selection function. The principle for distinguishing a key candidate based on the correlation between computation traces and predicted variables is called a distinguisher. The following paragraphs recall the definitions of DCA distinguisher.

Definition 1 (Pearson's Correlation Coefficient [RW19])**.** Let $\mathrm{Cov}(X,Y)$ be the *covariance* between X and Y, σ_X denote the *standard deviation* of X, and $E(X)$ represent the *expectation* of X. *Pearson's correlation coefficient* is a measure of linear correlation between two random variables X and Y, which can be denoted by $\mathrm{Cor}(X,Y)$ as follows.

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

$\mathrm{Cor}(X,Y)$ has a value between -1 and 1. For $\mathrm{Cor}(X,Y) = -1$ and 1, X and Y have

negative and positive correlations, respectively. If $\mathrm{Cor}(X, Y) = 0$, X and Y are linearly independent.

Definition 2 (Boolean Correlation [RW19])**.** Let $\mathbb{F}_2$ denote the finite field with order 2 and $\mathbb{F}_2^n$ be the n-dimensional vector space over $\mathbb{F}_2$. Let $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ represent two n-variable *boolean functions* such that

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

For a uniformly distributed random input $x \in \mathbb{F}_2^n$, Pearson's correlation between $f(x)$ and $g(x)$ can be described as follows.

$$\mathrm{Cor}(f, g) = \frac{N_{11}^{f,g} N_{00}^{f,g} - N_{10}^{f,g} N_{01}^{f,g}}{\sqrt{N_1^f N_0^f N_1^g N_0^g}} \tag{1}$$

The *bias* of a boolean function f is defined by

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

where $\mathrm{wt}(f)$ denotes the *Hamming weight* of the truth-table of f, i.e., $\mathrm{wt}(f) = |\{x \in \mathbb{F}_2^n : f(x) = 1\}|$. If a boolean function f equally outputs 0 and 1, f is *balanced* and $\mathrm{wt}(f) = 2^{n-1}$. For two balanced n-ary boolean functions f and g, Equation (1) can be simplified to

$$\mathrm{Cor}(f, g) = \frac{1}{2^n} B(f + g). \tag{3}$$

Definition 3 (DCA Distinguisher [RW19])**.** Let φ denote a selection function for a hypothesized key k^*, whilst its correct key is k. A computation trace $\{v_1, \cdots, v_t\}$ consists of t samples. A common selection function is defined as the output of Sbox such that $\varphi(x, k^*) = S(x \oplus k^*)$. The DCA distinguisher is calculated as the maximal absolute value of the correlation between each trace sample v_i and j-th output of φ, that is

$$\delta_k = \max_{1 \le i \le t} |\mathrm{Cor}(\varphi_j, v_i)|.$$

Algebraic DCA. Biryukov and Udovenko [BU18] introduced the algebraic DCA to defeat masking countermeasures. Let $(v_1, \cdots, v_t)$ denote a computation trace containing n linear shares of a sensitive variable x. The algebraic attack finds a linear combination of shares by solving the equations and recovers a constant vector $(a_1, \cdots, a_t) \in \{0, 1\}^t$ such that

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

If the system is solvable, the key guess is most likely correct. This attack is also independently discussed as *linear decoding analysis* (LDA) by Goubin *et al.* [GPRW20]. The time complexity of algebraic DCA is $\mathcal{O}(|\mathcal{K}| \cdot t^{2.8})$ where $\mathcal{K}$ is the subkey space and $\mathcal{O}(t^{2.8})$ is the time complexity of using the Strassen algorithm [Str69] to solve the linear system. Notably, it is independent of the masking order.

2.2 Higher-Order DCA Attacks

Higher-Order DCA. Bogdanov *et al.* [BRVW19] introduced HO-DCA against masking countermeasures in white-box implementations. HO-DCA consists of a pre-processing

step and a first-order DCA. Let $(v_1, \cdots, v_t)$ denote a computation trace. An order-n DCA computes $\binom{t}{n}$ combinations of the trace and form an order-n trace $(v_{j_1}, \cdots, v_{j_p})$. Each node v_{j_i} for $1 \leq\ i \leq p$ is the sum of n nodes in $(v_1, \cdots, v_t)$. HO-DCA recovers the sum of n linear shares by enumerating all the subsets with n nodes in the computation traces. Thus, a sensitive variable protected by linear masking can be retrieved from the higher-order traces.

Higher-Degree Decoding Analysis. To extend the algebraic DCA to deal with non-linear masking, HDDA has been proposed [GRW20, BU21]. HDDA multiplies all the d tuples in the computation trace and applies the algebraic DCA to an extended higher-degree trace. For a degree-d HDDA, the higher-degree traces contain $q = \binom{t}{d}$ samples. The extended traces consist of $(t + q)$ nodes. Thus, the time complexity of the algebraic DCA is $\mathcal{O}(|\mathcal{K}| \cdot (t + q)^{2.8})$.

Data-Dependency Higher-Order DCA. Goubin *et al.* [GRW20] proposed DDHO-DCA to attack the three winning challenges in WhibOx 2019 competition. The basic principle of DDHO-DCA is that the linear shares of some variables can be recovered by targeting the multipliers of an intermediate variable. This attack prevents the exponential growth of the linear masking order. Algorithm 1 is recalled for deriving the sets of co-operands for AND gates in a boolean circuit.

Algorithm 1 DetectCoOperands($\mathcal{C}$) [GRW20]

Input: A boolean circuit $\mathcal{C}$.
Output: An associative array M mapping a gate in $\mathcal{C}$ to a set of gates in $\mathcal{C}$.

```
M ← empty associative array
for g ∈ AND_GATES(C) do
    g1, g2 ← the two incoming gates of g
    if M does not have key g1 then
        M[g1] ← ∅
    if M does not have key g2 then
        M[g2] ← ∅
    M[g1] ← M[g1] ∪ {g2}
    M[g2] ← M[g2] ∪ {g1}
return M
```

The obtained array M contains the variables of which the sum is correlated to a sensitive variable. Therefore, a *data-dependency trace* can be derived to compute the bitwise sum of each set. For each g in M, $\bigoplus_{e \in M[g]} e$ forms the nodes in a data-dependency trace. In this case, one of the nodes in the trace might be the sum of linear shares. Therefore, DCA can be applied to recover the secret key.

Integrated Higher-Order DCA. Goubin *et al.* [GRW20] adapted the integrated attacks [CCD00, RPD09] with HO-DCA to defeat against shuffling countermeasure in white-box implementations. The principle is to combine t (shuffled) samples of traces and compute the correlation between the sum and the predicted value. The following lemma indicates that the obtained correlation is reduced by a factor $\sqrt{t}$ instead of t.

Lemma 1 ([GRW20]). *Let $X_i \in \mathbb{F}_2$ for $1 \leq i \leq t$ be t mutually independent and uniformly random variables. Let $Y \in \mathbb{F}_2$ be a random variable satisfying* $\mathrm{Cor}(Y, X_j) = \rho$ *for some $j \in \{1, \cdots, t\}$ and Y is independent of remaining X_i. We have*

$$\mathrm{Cor}(Y, \sum_i X_i) = \frac{1}{\sqrt{t}} \mathrm{Cor}(Y, X_j) = \frac{\rho}{\sqrt{t}}.$$

2.3 Masking and Shuffling Countermeasures

Linear Masking. Linear masking is also called Boolean masking. Ishai *et al.* [ISW03] defined that an order-n linear masking (ISW masking) splits a sensitive variable x into $n+1$ shares satisfying

$$x = x_0 \oplus x_1 \oplus \cdots \oplus x_n.$$

The variables $x_0, \cdots, x_{n-1}$ are generated uniformly and independently, and $x_n = x \oplus x_0 \oplus \cdots \oplus x_{n-1}$. This masking ensures that any combination of less than $n+1$ shares cannot reveal any information about x. However, algebraic DCA can break it by utilizing elementary linear algebra. An order-$(n+1)$ HO-DCA can also recover the sensitive variable of this linear masking.

Non-Linear Masking. Biryukov and Udovenko [BU18] proposed a non-linear masking scheme (BU masking). It splits a sensitive variable x into three shares a, b, and c satisfying

$$x = ab \oplus c,$$

where a and b are randomly generated bits, and c is computed as $c = x \oplus ab$ while $\Pr[(x \oplus ab) = x] = \frac{3}{4}$. Both Goubin *et al.* [GRW20] and Seker *et al.* [SEL21] proved that it is vulnerable to DCA.

Combination of Linear and Non-Linear Masking. Linear and non-linear masking schemes are vulnerable to algebraic DCA and DCA attacks, respectively. Thus, the combination of linear and non-linear masking is intuitively secure against computational and algebraic attacks. Goubin *et al.* [GRW20] discussed three possible cases to combine linear and non-linear masking (GRW masking), but none of them derives secure gadgets for each encoding. Let $x \in \mathbb{F}_2$ be a sensitive variable and $a, b, a_i, b_i, c_i \in \mathbb{F}_2$ denote the shares for $1 \le i \le n$. The three cases of GRW masking are listed as follows.

- Case 1. $x = (a_1 \oplus a_2 \oplus \cdots \oplus a_n)(b_1 \oplus b_2 \oplus \cdots \oplus b_n) \oplus (c_1 \oplus c_2 \oplus \cdots \oplus c_n)$,
- Case 2. $x = (a_1 b_1 \oplus c_1) \oplus (a_2 b_2 \oplus c_2) \oplus \cdots \oplus (a_n b_n \oplus c_n)$,
- Case 3. $x = ab \oplus c_1 \oplus c_2 \oplus \cdots \oplus c_n$.

In specific, Case 1 applies a linear masking on top of a non-linear masking while Case 2 applies a non-linear masking on top of a linear masking. For Case 3, it replaces the first share c_0 in ISW masking by a non-linear share ab. By analyzing the correlation between the sensitive variable and the sum of the linear shares, Goubin *et al.* revealed the weakness of the three combinations. The results imply that these cases cannot provide resistance against HO-DCA, DDHO-DCA, and integrated HO-DCA.

To resist computational and algebraic attacks, Seker *et al.* [SEL21] proposed the SEL masking which consists of linear shares of order n and non-linear shares of degree d. It follows two security notions, *probing security* [ISW03, KHL11] and *prediction security* [BU18]. The former model states that every tuple of n or less intermediate variables cannot reveal any information about the sensitive variable. The latter model focuses on the probability of an adversary to predict the secret value of any degree-d function over intermediate values. Let $x \in \mathbb{F}_2$ be a sensitive variable. The encoding phase of SEL masking can be described by

$$x = \prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i. \tag{4}$$

The variables $\tilde{x}_0, \cdots, \tilde{x}_d, x_1, \cdots, x_{n-1} \in \mathbb{F}_2$ are chosen randomly and independently, and $x_n = x \oplus \prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n-1} x_i$. Seker *et al.* also defined an XOR gadget, an AND gadget,

and a Refresh gadget to perform computation on encoded variables. The AND gadget is recalled in Algorithm 2 in which Refresh function represents the Refresh gadget of SEL masking. To resist the algebraic attacks, the refreshed masks prevent the inputs of the gadget from being fixed by the adversary.

Algorithm 2 AND gadget for combined masking [SEL21]

Input: $(\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]}$ satisfying $\prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i = x$ and $(\tilde{y}_j)_{j\in[0,d]}, (y_i)_{i\in[1,n]}$ satisfying $\prod_{j=0}^{d} \tilde{y}_j \oplus \bigoplus_{i=1}^{n} y_i = y$.

Output: $(\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]}$ satisfying $\prod_{j=0}^{d} \tilde{z}_j \oplus \bigoplus_{i=1}^{n} z_i = xy$.

$(\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]} \leftarrow \mathbf{Refresh}((\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]})$
$(\tilde{y}_j)_{j\in[0,d]}, (y_i)_{i\in[1,n]} \leftarrow \mathbf{Refresh}((\tilde{y}_j)_{j\in[0,d]}, (y_i)_{i\in[1,n]})$
for $0 \le i \le d$ **do**
    $\tilde{z}_i = \tilde{x}_i \tilde{y}_{i'}$ ▷ $i' = i + 1 \bmod (d+1)$
    for $1 \le j \le n$ **do**
        $r^{i,j} \leftarrow \mathbf{rand}(0,1)$
        $\tilde{z}_i = \tilde{z}_i \oplus r^{i,j}$
for $0 \le i \le n$ **do**
    for $i < j \le n$ **do**
        if $i = 0$ **then**
            $r_{j,0} \leftarrow \mathbf{And}[n,d]$
        else
            $r_{i,j} \leftarrow \mathbf{rand}(0,1)$
            $r_{j,i} \leftarrow (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$
for $1 \le i \le n$ **do**
    $z_i \leftarrow x_i y_i$
    for $0 \le j \le n$ and $j \ne i$ **do**
        $z_i \leftarrow z_i \oplus r_{i,j}$
return $(\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]}$

For the And$[n,d]$ function in AND gadget, Seker *et al.* only provided the detailed transformation for $d = 1$ and $d = 2$ which are described as follows.

$$
\begin{aligned}
\mathbf{And}[n,1] : r_{j,0} = {} & \tilde{x}_1(\tilde{x}_0 y_j \oplus r^{0,j}\tilde{y}_0) \oplus \\
& \tilde{y}_1(\tilde{y}_0 x_j \oplus r^{1,j}\tilde{x}_0) \oplus r^{1,j}(r^{0,1} \oplus \cdots \oplus r^{0,n}).\\
\mathbf{And}[n,2] : r_{j,0} = {} & \tilde{x}_0[\tilde{x}_2(\tilde{x}_1 y_j \oplus r^{0,j}\tilde{y}_0) \oplus r^{1,j} v \tilde{y}_1] \oplus \\
& \tilde{y}_0[\tilde{y}_1(\tilde{y}_2 x_j \oplus r^{1,j}\tilde{x}_2) \oplus r^{0,j} u \tilde{x}_2] \oplus \\
& \tilde{x}_0\tilde{y}_1(r^{1,j}\tilde{x}_2\tilde{y}_0 \oplus r^{2,j}\tilde{x}_1\tilde{y}_2) \oplus r^{0,j}\tilde{x}_1\tilde{y}_2(v \oplus \tilde{x}_2\tilde{y}_0) \oplus \\
& \tilde{x}_2\tilde{y}_0(r^{0,j}\tilde{x}_0 \oplus r^{1,j}\tilde{y}_1) \oplus uvr^{0,j},\\
u = {} & r^{1,1} \oplus \cdots \oplus r^{1,n},\\
v = {} & r^{2,1} \oplus \cdots \oplus r^{2,n}.
\end{aligned}
$$

Therefore, this specific SEL masking aims to resist arbitrary order DCA and at most degree-2 algebraic DCA.

Dummy Shuffling. Biryukov and Udovenko [BU21] introduced dummy shuffling to hide the real computation among several redundant computations. The target of dummy shuffling is a computational *slot* which is a sensitive function of the implementation that can be computed separately and independently. Dummy shuffling is performed in *input-shuffling* (shuffles the main and dummy inputs), *evaluation* (computes the slots on

main/dummy inputs), and *output-selection* (extracts the main output) phases. Based on the abilities of an adversary, the dummy shuffling techniques can be classified into *hidden/public* dummy shuffling. An attacker is hard to isolate any single (main/dummy) slot among a group of slots in a *hidden dummy shuffling* implementation, whilst any slot can be clearly separated and isolated in a *public dummy shuffling* implementation. However, it is still difficult to predict the location of a main slot. BU shuffling is a construction of public dummy shuffling with a single main slot and multiple dummy slots. It has a clear slot separation and the outputs of all the slots are simply XORed to extract the main output. A variant of this construction was implemented by the third winning challenge #100 in WhibOx 2019 contest [FMIS+].

3 Security Evaluation on Combined Masking

In this section, we analyze HO-DCA against SEL masking, extend HO-DCA to higher-degree context, and enhance the results of DDHO-DCA.

Target Implementation. We assume that the target implementation is possibly a Boolean circuit protected by SEL masking. It is represented by a directed acyclic graph where the vertices are *gate* and the edges between two nodes are *wires*. A gate might compute the NOT, XOR, and AND of the input wire(s) or output a constant value (0 or 1). The output of each gate can be connected to several following gates and can be associated with a Boolean circuit of the plaintexts. The *intermediate variables* are the values assigned to the wires that are neither input wires nor output ones. An XOR or AND *gadget* is a sub-circuit that securely process with $2 \cdot (n + d + 1)$ inputs (i.e., input shares). It relates to the XOR or AND operation between two unmasked sensitive variables. Each gadget has $n + d + 1$ outputs which are the shares of the resulting value. The required random values depend on the inputs of the implementation. A *computation trace* consists of several intermediate variables of the circuit. It is assumed that the attacker is able to locate a t-dimensional sub-trace which is denoted by a t-large *window*. Such a window might contain the shares of the target sensitive variable.

3.1 Higher-Order DCA Attacks on SEL Masking

To resist the algebraic DCA, SEL masking replaces the first share x_0 in ISW masking by non-linear shares $\tilde{x}_0, \cdots, \tilde{x}_d$ satisfying $\prod_{j=0}^{d} \tilde{x}_j = x_0$. For $d = 1$ and $d = 2$, the encoding schemes of SEL masking are specified in Equation (5) and (6), respectively. The corresponding constructions are named $(n, 1)$-masking and $(n, 2)$-masking.

$$x = \tilde{x}_0\tilde{x}_1 \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_n \tag{5}$$

$$x = \tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_n \tag{6}$$

We note that Equation (5) is equal to Case 3 of GRW masking. In the following paragraphs, the security of $(n, 1)$-masking and $(n, 2)$-masking is evaluated via HO-DCA attacks.

HO-DCA on SEL Masking. For $(n, 1)$-masking and $(n, 2)$-masking, Equation (5) and (6) imply that $x_1 \oplus x_2 \oplus \cdots \oplus x_n = x \oplus \tilde{x}_0\tilde{x}_1$ and $x_1 \oplus x_2 \oplus \cdots \oplus x_n = x \oplus \tilde{x}_0\tilde{x}_1\tilde{x}_2$, respectively. For any $x, \tilde{x}_0, \tilde{x}_1, \tilde{x}_2 \in \mathbb{F}_2$, $\Pr[(x \oplus \tilde{x}_0\tilde{x}_1) = x] = \frac{3}{4}$ and $\Pr[(x \oplus \tilde{x}_0\tilde{x}_1\tilde{x}_2) = x] = \frac{7}{8}$. A correlation can be found between the sensitive variable x and the sum of the intermediate variables $x_1 \oplus x_2 \oplus \cdots \oplus x_n$. Thus, there exists one node in the higher-order traces which is highly correlated to a predictable vector. The following lemma helps analyze the expected correlation of HO-DCA to SEL masking.

Lemma 2. *Let $X, X_0, X_1, \cdots, X_d \in \mathbb{F}_2$ be mutually independent and uniformly random variables. We have*

$$\mathrm{Cor}(X, X \oplus \prod_{j=0}^{d} X_j) = 1 - \frac{1}{2^d}.$$

Proof. Since X and $X \oplus \prod_{j=0}^{d} X_i$ are both balanced, from Equation (2) and (3), we have

$$\mathrm{Cor}(X, X \oplus \prod_{j=0}^{d} X_j) = \mathrm{Cor}(0, \prod_{j=0}^{d} X_j) = \frac{1}{2^{d+1}}(2^{d+1} - 2 \cdot \mathrm{wt}(\prod_{j=0}^{d} X_j)).$$

For a $(d+1)$-ary function $\prod_{j=0}^{d} X_j$, it outputs 1 only when the input variables are all equal to 1 such that $\mathrm{wt}(\prod_{j=0}^{d} X_j) = 1$. Hence, $\mathrm{Cor}(X, X \oplus \prod_{j=0}^{d} X_j) = 1 - \frac{1}{2^d}$. □

From Lemma 2, it indicates that for $(n, 1)$-masking,

$$\mathrm{Cor}(x, x_1 \oplus x_2 \oplus \cdots \oplus x_n) = \mathrm{Cor}(x, x \oplus \tilde{x}_0\tilde{x}_1) = \frac{1}{2}.$$

For $(n, 2)$-masking,

$$\mathrm{Cor}(x, x_1 \oplus x_2 \oplus \cdots \oplus x_n) = \mathrm{Cor}(x, x \oplus \tilde{x}_0\tilde{x}_1\tilde{x}_2) = \frac{3}{4}.$$

The strong correlation implies that an HO-DCA attacker is able to recover the secret key with high probability by targeting the n linear shares $x_1, x_2, \cdots, x_n$. This process, on a t-large window, has time complexity $\mathcal{O}\left(\binom{t}{n}\right)$ without locating the non-linear shares of degree-d. Besides, Lemma 2 also shows that the correlation nearly approaches 1 when the masking has a high degree d.

To extend HO-DCA to defeat non-linear masking, we introduce a *higher-degree* version of HO-DCA. A higher-degree HO-DCA (HDHO-DCA) with degree-d and order-n consists of a multiplication step followed by an HO-DCA. The adversary first multiplies all d points in the computation trace and mounts HO-DCA to combine n nodes in the higher-degree trace. For unknown d, the attacker can compute the products of all $2, 3, \cdots, \ell$ points among the nodes in the trace by guessing a possible degree satisfying $\ell \geq d$. A degree-d trace consists of $\binom{t}{d}$ points given by $(v_{j_1}, \cdots, v_{j_q})$. Each node v_{j_i} for $1 \leq i \leq q$ is the product of d nodes in $(v_1, \cdots, v_t)$. The multiplication of nodes helps to recover the product of non-linear shares. HDHO-DCA joints the higher-order trace with the original one and obtains $t + j_q$ points $(v_1, \cdots, v_t, v_{j_1}, \cdots, v_{j_q})$. Consequently, an order-n DCA combines the n nodes in the obtained trace to recover the sensitive variable. For defeating BU masking, HDHO-DCA enumerates all the subsets of the computation trace with cardinality 2 and multiplies the elements of each subset to obtain the degree-2 trace. Such a trace recovers the product of ab in BU masking. Since the variable c is recorded in the original trace, the connection of the degree-2 trace and the original trace contains the nodes ab and c. Thus, an order-2 DCA can recover the sensitive variable $ab \oplus c$ by adding all order-2 combinations of the trace. For (n, d)-masking, HDHO-DCA first obtains all the subsets with cardinality $d+1$ and multiplies the elements of each set. Such a trace recovers the product of non-linear shares, i.e., $\hat{x} = \prod_{j=0}^{d} \tilde{x}_j$, and computes $q = \binom{t}{d+1}$ items. Subsequently, HDHO-DCA combines the degree-$(d+1)$ trace with the original trace and mounts an order-$(n+1)$ DCA to compute $\binom{t+q}{n+1}$ items. Thus, $\hat{x} \oplus x_1 \oplus \cdots \oplus x_n = x$ is retrieved as a node in the trace.

HDDA on SEL Masking. HDDA first computes a degree-$(d+1)$ trace to multiply all the $(d+1)$-tuples in the computation trace. Similar to HDHO-DCA, this process aims to recover the product of non-linear shares and computes $q = \binom{t}{d+1}$ nodes. By combining the higher-degree trace with the original trace, there exist $n+1$ nodes of which the sum is a sensitive variable. Through a decoding phase by algebraic DCA, the linear shares can be recovered with the time complexity $\mathcal{O}(|\mathcal{K}| \cdot (t+q)^{2.8})$.

Data-Dependency HO-DCA on SEL Masking. DDHO-DCA aims to bypass the efforts of finding the masking shares. It recovers the linear shares of some sensitive variables by the multipliers of some intermediate variables.

Definition 4 (Multiplier)**.** A multiplier is a co-operand of a variable for an AND operation. Let v_g denote the output of a gate g in the circuit. The set $\{v_{g'}\}$, which includes the multipliers of v_g, is the co-operands such that $(v_g, v_{g'})$ will enter a subsequent AND gate.

Definition 5 (Data-Dependency Trace)**.** Let $\{v_{g'}\}$ be the multipliers of v_g. A data-dependency trace v consists of the sum of the multipliers for each gate g in the circuit such that $v = \{\oplus_{\{v_{g'_1}\}}, \cdots, \oplus_{\{v_{g'_t}\}}\}$.

Definition 6 (Data-Dependency Node)**.** Let $(v_1, \cdots, v_t)$ denote a data-dependency trace. Each sample v_i is called a data-dependency node. Each data-dependency node is the sum of multipliers for a variable.

In SEL masking, most of the AND gates appear in the AND gadgets. The set of multipliers of the input variables in the AND gadget for $(n, 1)$-masking and $(n, 2)$-masking are enumerated in Table 2 and Table 3, respectively. The multipliers in $\{\cdot\}$ are iterated over the order n. For the variables x_i and y_i in $(n, 1)$-masking, the full set and subset of multipliers that contain the linear shares have a correlation $\frac{1}{2}$. However, for such cases in $(n, 2)$-masking, the full set of multipliers only has a correlation $\frac{1}{4}$, which is still possible to distinguish the correct key. Although the Refresh gadget updates the shares of masking, the sum of the updated linear sharing is correlated to a sensitive variable. Therefore, applying Refresh gadgets cannot prevent this kind of flaw. The correlation holds for arbitrary high order n of linear masking shares. To get a high correlation for $(n, 2)$-masking, the adversary can obtain a subset of the multipliers for x_i and y_i by combining n variables among $n + 1$ variables. One of the combinations is $y_1 \oplus \cdots \oplus y_n$ or $x_1 \oplus \cdots \oplus x_n$. Thus, a correlation $\frac{3}{4}$ can be derived. This process implies that DDHO-DCA cannot recover the linear shares of SEL masking directly but can find the set consisting of the linear shares. The introduction of a non-linear share reduces the correlation in the case of $(n, 2)$-masking.

To verify the leakage, we implement a bitsliced AES-128 with BP Sbox which is proposed by Boyar and Peralta [BP09] (recalled in Appendix A). For a practical DDHO-DCA, Algorithm 1 is modified as Algorithm 3. It detects the co-operands of intermediate variables for AND gates. Since an operated gate in a Boolean circuit is hard to be tracked in the software, the new algorithm traces the memory address to record the intermediate variable. This attack uses 256 traces limited to 7168 and 8192 data-dependency nodes computed in the first round, respectively for $(5, 1)$-masking and $(5, 2)$-masking. The results are illustrated in Figure 1. The target variable is the 8-th output bit (i.e., s_8 in Appendix A) of the first Sbox in the initial round. Figure 1 depicts that the correct key candidate plotted in blue is hard to be distinguished from other candidates plotted in gray. This is not consistent with the correlation on the full multipliers of the variables x_i/y_i, especially for the case of $(n, 1)$-masking (refer to Table 2). The same negative result can be obtained by targeting any output bit of 16 Sboxes in the first round. This implies that the sum of the multipliers for any variable in the first round has a low correlation with any output bit of arbitrary Sbox. Such an observation stimulates a comprehensive study on DDHO-DCA and an exploration of its failure against SEL masking.

3.2 Improved Data-Dependency HO-DCA

DDHO-DCA exploits the multipliers of intermediate values to obtain a set consisting of linear shares. Nevertheless, the results reveal that an attacker cannot obtain a strong correlation when targeting any output of Sbox in the first round. This section explains the inefficiency of DDHO-DCA against SEL masking. Then we illustrate three paths to improve the data-dependency attack to recover the correct key with SEL masking.

Table 2: Multipliers for each variable in AND gadgets of $(n, 1)$-masking, for $1 \leq i \leq n$.

Variable	Multipliers	Correlation (full set)	Correlation (subset)
$\tilde{x}_0$	$\tilde{y}_1, \{y_i, r^{1,i}\}$	-	$\mathrm{Cor}(y, \tilde{y}_1 \oplus \bigoplus_{i=1}^{n} y_i) =$ $\mathrm{Cor}(y, \bigoplus_{i=1}^{n} y_i) = \frac{1}{2}$
$\tilde{x}_1$	$\tilde{y}_0, \{\tilde{x}_0 y_i \oplus r^{0,i}\tilde{y}_0\}$	-	-
x_1	$\tilde{y}_0, \{y_i\}$		
x_2	$\tilde{y}_0, \{y_i\}$	$\mathrm{Cor}(y, \tilde{y}_0 \oplus \bigoplus_{i=1}^{n} y_i) = \frac{1}{2}$	$\mathrm{Cor}(y, \bigoplus_{i=1}^{n} y_i) = \frac{1}{2}$
x_3	$\tilde{y}_0, \{y_i\}$		
$\tilde{y}_0$	$\tilde{x}_1, \{r^{0,i}, x_i\}$	-	$\mathrm{Cor}(x, \tilde{x}_1 \oplus \bigoplus_{i=1}^{n} x_i) =$ $\mathrm{Cor}(x, \bigoplus_{i=1}^{n} x_i) = \frac{1}{2}$
$\tilde{y}_1$	$\tilde{x}_0, \{\tilde{y}_0 x_i \oplus r^{1,i}\tilde{x}_0\}$	-	-
y_1	$\tilde{x}_0, \{x_i\}$		
y_2	$\tilde{x}_0, \{x_i\}$	$\mathrm{Cor}(x, \tilde{x}_0 \oplus \bigoplus_{i=1}^{n} x_i) = \frac{1}{2}$	$\mathrm{Cor}(x, \bigoplus_{i=1}^{n} x_i) = \frac{1}{2}$
y_3	$\tilde{x}_0, \{x_i\}$		

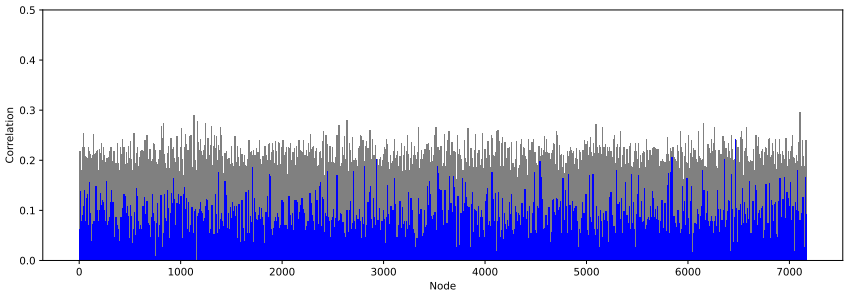
Table 3: Multipliers for each variable in AND gadgets of $(n, 2)$-masking, for $1 \leq i \leq n$.

Variable	Multipliers	Correlation (full set)	Correlation (subset)
$\tilde{x}_0$	$\tilde{y}_1, \{\tilde{x}_2(\tilde{x}_1 y_i \oplus r^{0,i}\tilde{y}_0)$ $\oplus r^{1,i} v \tilde{y}_1, \tilde{y}_1(r^{1,i}\tilde{x}_2\tilde{y}_0$ $\oplus r^{2,i}\tilde{x}_1\tilde{y}_2), r^{0,i}\}$	-	-
$\tilde{x}_1$	$\tilde{y}_2, \{y_i, r^{2,i}\tilde{y}_2,$ $r^{0,i}\tilde{y}_2(v \oplus \tilde{x}_2\tilde{y}_0)\}$	-	$\mathrm{Cor}(y, \tilde{y}_2 \oplus \bigoplus_{i=1}^{n} y_i) = \frac{1}{4}$ $\mathrm{Cor}(y, \bigoplus_{i=1}^{n} y_i) = \frac{3}{4}$
$\tilde{x}_2$	$\tilde{y}_0, \{\tilde{x}_1 y_i \oplus r^{0,i}\tilde{y}_0,$ $r^{1,i}, r^{0,i}u, r^{1,i}\tilde{y}_0, \tilde{y}_0,$ $\tilde{y}_0(r^{0,i}\tilde{x}_0 \oplus r^{1,i}\tilde{y}_1)\}$	-	-
x_1	$\tilde{y}_2, \{y_i\}$		
x_2	$\tilde{y}_2, \{y_i\}$	$\mathrm{Cor}(y, \tilde{y}_2 \oplus \bigoplus_{i=1}^{n} y_i) = \frac{1}{4}$	$\mathrm{Cor}(y, \bigoplus_{i=1}^{n} y_i) = \frac{3}{4}$
x_3	$\tilde{y}_2, \{y_i\}$		
$\tilde{y}_0$	$\tilde{x}_2, \{\tilde{y}_1(\tilde{y}_2 x_i \oplus r^{1,i}\tilde{x}_2)$ $\oplus r^{0,i} u \tilde{x}_2, r^{0,i}, r^{1,i}\tilde{x}_2,$ $\tilde{x}_2, \tilde{x}_2(r^{0,i}\tilde{x}_0 \oplus r^{1,i}\tilde{y}_1)\}$	-	-
$\tilde{y}_1$	$\tilde{x}_0, \{r^{1,i} v, \tilde{y}_2 x_i \oplus r^{1,i}\tilde{x}_2,$ $\tilde{x}_0(r^{1,i}\tilde{x}_2\tilde{y}_0 \oplus r^{2,i}\tilde{x}_1\tilde{y}_2),$ $r^{1,i}\}$	-	-
$\tilde{y}_2$	$\tilde{x}_1, \{x_i, r^{2,i}\tilde{x}_1,$ $r^{0,i}\tilde{x}_1(v \oplus \tilde{x}_2\tilde{y}_0)\}$	-	$\mathrm{Cor}(x, \tilde{x}_1 \oplus \bigoplus_{i=1}^{n} x_i) = \frac{1}{4}$ $\mathrm{Cor}(x, \bigoplus_{i=1}^{n} x_i) = \frac{3}{4}$
y_1	$\tilde{x}_1, \{x_i\}$		
y_2	$\tilde{x}_1, \{x_i\}$	$\mathrm{Cor}(x, \tilde{x}_1 \oplus \bigoplus_{i=1}^{n} x_i) = \frac{1}{4}$	$\mathrm{Cor}(x, \bigoplus_{i=1}^{n} x_i) = \frac{3}{4}$
y_3	$\tilde{x}_1, \{x_i\}$		

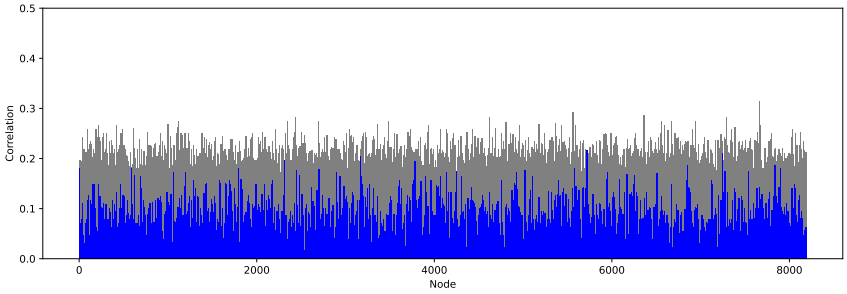
Algorithm 3 DetectCoOperands($\mathcal{C}$)

Input: A Boolean circuit $\mathcal{C}$.
Output: An associative array M mapping a variable in $\mathcal{C}$ to a set of variables in $\mathcal{C}$.

$M \leftarrow$ empty associative array
$T \leftarrow$ a mapping array from memory address to its value in $\mathcal{C}$
$A \leftarrow$ memory address array for the left operands of AND gates in $\mathcal{C}$
$B \leftarrow$ memory address array for the corresponding right operands in $\mathcal{C}$
for $0 \leq i \leq \text{length}(A)$ **do**
    if M does not have key $A[i]$ **then**
        $M[A[i]] \leftarrow \varnothing$
    if M does not have key $B[i]$ **then**
        $M[B[i]] \leftarrow \varnothing$
    $M[A[i]] \leftarrow M[A[i]] \cup T[B[i]]$
    $M[B[i]] \leftarrow M[B[i]] \cup T[A[i]]$
return M



(a) The correlation between data-dependency nodes and target bit of Sbox in $(5, 1)$-masking.



(b) The correlation between data-dependency nodes and target bit of Sbox in $(5, 2)$-masking.

Figure 1: The correlation curve when computing the data-dependency nodes in the first round and targeting the 8-th output bit of the first Sbox in the first round.

3.2.1 Targeting Inputs of AND Gates.

Deriving from the success of DCA, most of the attacks [BBMT18, RW19, GPRW20] select the outputs of a first-round Sbox as a hypothesized value. Similarly, DDHO-DCA targets an output bit of a first-round Sbox to compute the correlation with collected traces. However, the nodes in data-dependency traces consist of the sum of multipliers, which are correlated to the inputs of an AND gates in Sbox circuit. Let $\boldsymbol{x}$ denote a set of elements

$(x_1, \cdots, x_8)$ and $\bar{\boldsymbol{x}}$ be a set of secret shares. Figure 2 illustrates the relation among the multipliers, AND gates, and AND gadgets. On the left side, each AND gate (plotted in red) in the Sbox circuit is securely operated as an AND gadget in SEL masking. On the right side, DDHO-DCA records the variables (plotted in red) which are the inputs to an AND gate in AND gadgets. Since the obtained multipliers might contain the linear shares (e.g., $x_1, \cdots, x_n$ of $\bar{\boldsymbol{x}}$), the sum of them corresponds to the sum of the linear shares. Thus, the resulting sum is correlated to the inputs of AND gates in Sbox circuit.

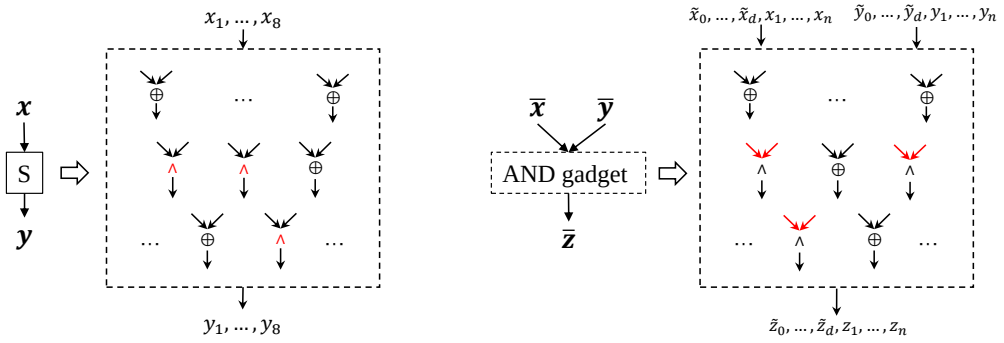


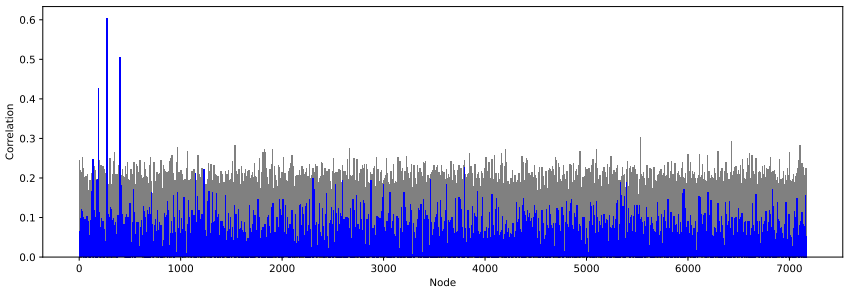
Figure 2: The circuit of Sbox (left) and AND gadget (right).

BP Sbox circuit has 32 AND operations and thus has 64 input values of AND gates. Among these inputs, some variables are operated repeatedly. Table 4 summarizes 36 various variables of the inputs and classifies them by their operating times. The results show that t_{29} is the most frequently used operand. Based on the variables listed in Table 4 and the output variables computed by the equations in Appendix A, any output of the Sbox is not directly assigned by any input of an AND gate. Table 8 in Appendix B indicates the low correlation between an input variable of AND gate and an output of Sbox circuit. For a successful attack, the hypothesized value of DDHO-DCA must be replaced by one of the inputs of AND gates in Sbox circuit.

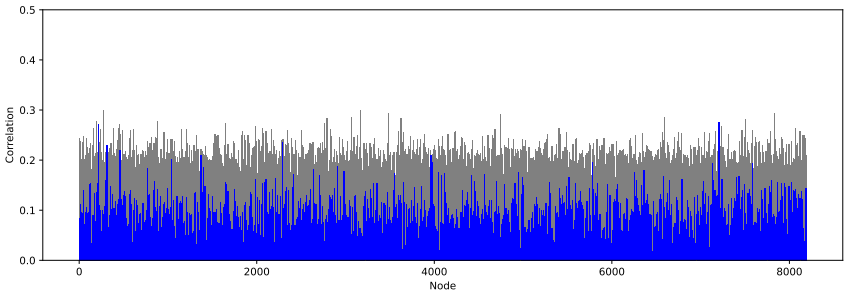
Table 4: Input variables of AND gates in Boyar and Peralta's Sbox circuit [BP09].

Operating times	Variables
1	$t_{21}, t_{23}, t_{24}, t_{25}, t_{27}, t_{30}, t_{31}, t_{35}, t_{38}$
2	$x_8, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9$ $y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{16}, y_{17}$ $t_{33}, t_{37}, t_{40}, t_{41}, t_{42}, t_{43}, t_{44}, t_{45}$
3	t_{29}

Figure 3 demonstrates the effectiveness of such an improved DDHO-DCA to defeat SEL masking when targeting the variable t_{29} in the first round. The correct key guess (plotted in blue) can be distinguished from incorrect key guesses (in gray) for $(5, 1)$-masking. Furthermore, the correlation curve has three peaks, which are corresponding to three times of multiplying with t_{29}. However, the situation is different at $(5, 2)$-masking. Although the peak of the correct key in Figure 3(b) is higher than the one of the correct key in Figure 1(b), it is not the highest peak among other candidates. This result implies a failed key recovery for the degree-2 case. As analyzed in Table 3, the correlation between the variable and the full set of its multipliers is $\frac{1}{4}$. The analysis might not be successful for this



(a) The correlation between data-dependency nodes and target bit of Sbox circuit in $(5, 1)$-masking.



(b) The correlation between data-dependency nodes and target bit of Sbox circuit in $(5, 2)$-masking.

Figure 3: The correlation curve when computing the data-dependency nodes in the first round and targeting t_{29} in the first Sbox in the first round.

Table 5: The vulnerable bits when targeting the input variables of AND gates for $(5, 2)$-masking with a full set of multipliers.

Key byte	Vulnerable variables	Key byte	Vulnerable variables
1	t_{31}, t_{43}, t_{45}	9	$t_{29}, t_{30}, t_{42}, t_{44}$
2	$t_{23}, t_{27}, t_{29}, t_{37}$	10	$t_{27}, t_{29}, t_{41}, t_{43}$
3	$t_{24}, t_{25}, t_{29}, t_{37}, t_{44}$	11	$t_{24}, t_{25}, t_{31}, t_{33}, t_{38}, t_{40}, t_{41}, t_{42}, t_{43}, t_{45}$
4	$t_{23}, t_{25}, t_{37}, t_{44}$	12	t_{31}, t_{37}, t_{41}
5	$t_{25}, t_{29}, t_{31}, t_{33}, t_{42}$	13	$t_{21}, t_{24}, t_{25}, t_{29}, t_{33}, t_{37}, t_{42}, t_{44}$
6	t_{38}	14	$t_{24}, t_{29}, t_{30}, t_{41}$
7	$t_{21}, t_{24}, t_{35}, t_{43}, t_{45}$	15	$t_{29}, t_{33}, t_{35}, t_{40}, t_{41}, t_{44}$
8	$t_{35}, t_{41}, t_{42}, t_{43}$	16	$t_{25}, t_{29}, t_{35}, t_{40}, t_{41}, t_{45}$

insignificant correlation. This leads us to target the subset of multipliers by combining the elements in the full set. In the next section, an improved analysis is proposed to locate the linear shares without the combination efforts. Our experiment on $(5, 1)$-masking explores that all the variables in Table 4 are helpful to distinguish a correct key. For $(5, 2)$-masking, there still exist vulnerable variables for an improved DDHO-DCA, which are summarized in Table 5.

Application to Break Challenge #100 of WhibOx 2019. Goubin *et al.* [GRW20] applied DDHO-DCA to the de-obfuscated challenge #100 for the full set of the multipliers. The target variables are the 8 output bits of Sbox in the first round. This attack only recovers

7 of the 16 key bytes. The further data-dependency attack exploits the subsets of the set of multipliers with cardinalities 2, 3, and 4. Using this attack, they recovered 8 more bytes of key and exhaustively searched the last key byte to recover the full key. Our improved DDHO-DCA which targets the inputs of an AND gate is more powerful to recover the first-round key. Since the implementation details of #100 are not published, we attempt to target the inputs of AND gates in BP Sbox. We note that an explicit input of AND gates corresponding to #100 might be helpful to obtain a strong correlation to distinguish the correct key. Table 6 hereafter illustrates the comparison on recovering 16 key bytes between targeting the output bits of Sbox and targeting the variables listed in Table 4. The variables in the table refer to the correct key guess ranked first in the correlation curve. The results show that our improved DDHO-DCA can recover the full key bytes with the full set of multipliers without the work factor on computing the subsets. Besides, for the case of one vulnerable bit when targeting the outputs of Sbox, the 8 bits of Sbox reveal 8 different key guesses. Thus, it is hard to distinguish the correct key from the incorrect keys. The worst case is that no bit of Sbox helps to obtain a correct key, while the attacker still needs to verify this situation and recover these key bytes with an exhaustive search. However, our experiment shows that for the target variables in Table 4, nearly half of them return a correct key while the other variables compute 0x00 as a possible key candidate. Hence, it is obvious to distinguish the correct key by our improved DDHO-DCA attack.

Table 6: The targeting vulnerable bits in DDHO-DCA and our improved attack.

Key byte	Sbox outputs [GRW20]	Input variables of AND gates	
		Different bits	Common bits
1	-	$t_{30}, t_{35}, t_{41}, t_{42}$	
2	-	$t_{30}, t_{35}, t_{41}, t_{42}$	
3	s_8	$t_{30}, t_{35}, t_{41}, t_{42}$	
4	s_4, s_1	$t_{30}, t_{35}, t_{41}, t_{42}$	
5	-	$t_{30}, t_{35}, t_{41}, t_{42}$	t_{21}, t_{23}
6	s_2	$t_{30}, t_{35}, t_{41}, t_{42}$	t_{24}, t_{25}
7	-	$t_{30}, t_{35}, t_{41}, t_{42}$	t_{27}, t_{29}
8	s_5, s_4	$t_{30}, t_{35}, \quad t_{42}$	t_{31}, t_{33}
9	-	$t_{30}, t_{35}, t_{41}, t_{42}$	t_{37}, t_{38}
10	s_8	$t_{30}, t_{35}, t_{41}, t_{42}$	t_{40}, t_{43}
11	s_3	$t_{30}, t_{35}, t_{41}, t_{42}$	t_{44}, t_{45}
12	-	$t_{30}, t_{35}, t_{41}, t_{42}$	
13	-	t_{35}, t_{41}, t_{42}	
14	s_8	$t_{30}, t_{35}, t_{41}, t_{42}$	
15	-	$t_{30}, t_{35}, t_{41}, t_{42}$	
16	-	$t_{30}, \quad t_{41}$	

3.2.2 Exploiting Variables with Same Multipliers

The previous section demonstrated that the sum of the full set of multipliers has a low correlation with the target input of AND gates for $(n, 2)$-masking. Table 3 in Section 3.1 computes the theoretical correlation between a variable and its multipliers. The results prove that the sum of the full set can only obtain the correlation value $\frac{1}{4}$, whilst combining the linear shares in the subset can obtain the value $\frac{3}{4}$. The reason is that the full set of multipliers contains a non-linear share. Thus, recovering the linear shares of $(n, 2)$-masking helps to measure the leakage of a sensitive variable. A feasible way is to enumerate the n elements among $n + 1$ elements in the full set. Before this process, the attacker needs to confirm the variable x_i or y_i by counting the number of its multipliers. If the variable

has $n+1$ multipliers, it could be a variable x_i or y_i. However, if the other variable in the Boolean circuit has $n+1$ multipliers, the work for the combination will fail. Hence, we propose a new method to find the linear shares of a sensitive variable in SEL masking.

Table 3 depicted that the linear shares x_i and y_i have the same multipliers $\{\tilde{y}_2, (y_i)_{1\leq i\leq n}\}$ and $\{\tilde{x}_1, (x_i)_{1\leq i\leq n}\}$. This observation implies that the multipliers of a linear share are equal to the ones of other linear shares. It can be demonstrated by the AND operations in line 14/16 and $\tilde{x}_1 y_j, \tilde{y}_2 x_j$ in And$[n, 2]$ in Algorithm 2. We note that this situation also holds for $(n, 1)$-masking. Thus, the linear shares in $(n, 2)$-masking multiply with the same vectors. Algorithm 4 describes an effective method to detect the variables with the same co-operands in a Boolean circuit. Since the multipliers of a variable consist of a set of binary values, the algorithm counts the hamming weight of a set to determine whether the two sets are equal or not. In each returned set, the variables have the same multipliers. For each set $N[i]$, the improved attack produces a new computation trace with the bitwise sum of it (i.e., $\bigoplus_{e\in N[i]} e$).

Algorithm 4 DetectSameCoOperands($\mathcal{C}$)

Input: A Boolean circuit $\mathcal{C}$.
Output: A set list N where the variables in each set have the same co-operands.

```
1: T ← a mapping array from memory address to its value in C
2: M ← DetectCoOperands(C)
3: for each key i of M do
4:     N[i] ← ∅
5:     for each key j of M and i < j do
6:         if #M[i] = #M[j] and #_0M[i] = #_0M[j] and #_1M[i] = #_1M[j] then
7:             N[i] ← T[j]
8:             M ← M − M[j]
9:     if N[i] ≠ ∅ then
10:        N[i] ← T[i]
11:        M ← M − M[i]
12:    else
13:        N ← N − N[i]
14: return N
```

Figure 4 illustrates this improvement on attacking $(5, 2)$-masking. The highest correlation peak of the correct key (plotted in blue) is greater than $\frac{3}{4}$. This demonstrates that the sum of the variables with the same multipliers is the sum of the linear shares. Thus, a strong correlation can be obtained by the improved DDHO-DCA.

3.2.3 Tracing Second-Round Data-Dependency Nodes

Table 4 illustrated that the 8-th input bit x_8 of Sbox is an input variable of the AND gates in BP Sbox. Since the ShiftRows, MixColumns, and AddRoundKey operations are byte-oriented, this implies that the 8-th output bit s_8 of Sbox is correlated to the next-round x_8. Therefore, targeting the first-round s_8 can obtain a correlation when tracing the next-round input x_8. Figure 5 depicts the correlation curve when targeting the 8-th output bit of the first Sbox with the next-round data-dependency traces. We recall that targeting any output of Sbox with the first-round data-dependency nodes cannot obtain a distinguishable correlation curve. However, Figure 5 depicts that both $(5, 1)$-masking and $(5, 2)$-masking can be defeated by tracing the data-dependency nodes in the next round. Hence, when targeting the output bits of Sbox, the data-dependency traces shall contain the nodes in the next round. We note that exploiting the sum of variables with the same multipliers can compute a higher correlation peak for $(5, 2)$-masking. Our further

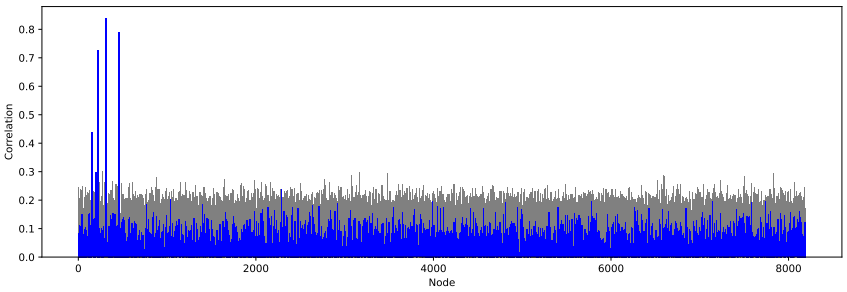
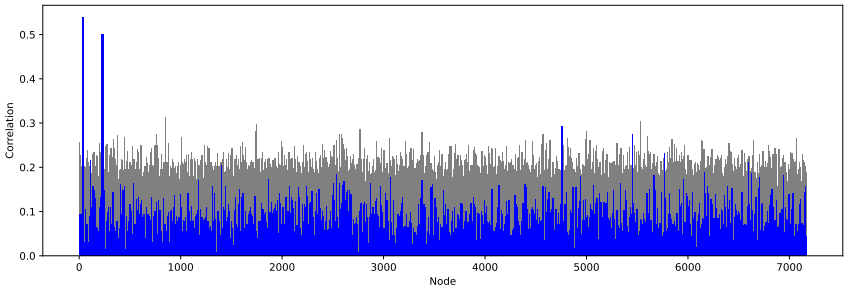
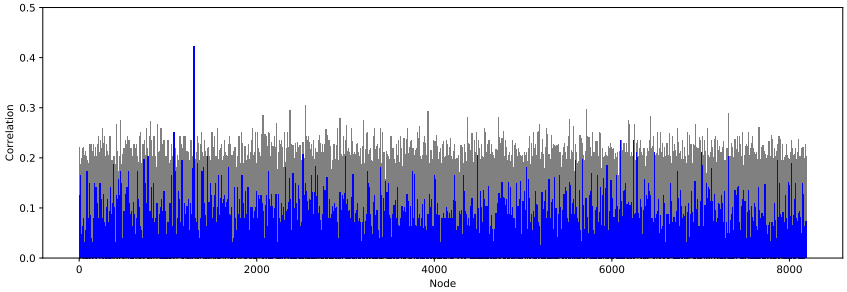


Figure 4: The correlation curve when computing the sum of variables with the same multiplies in the first round and targeting t_{29} in the first Sbox in the first round for $(5, 2)$-masking.



(a) The correlation between data-dependency nodes and target bit of Sbox in $(5, 1)$-masking.



(b) The correlation between data-dependency nodes and target bit of Sbox in $(5, 2)$-masking.

Figure 5: The correlation curve when computing the data-dependency nodes in the second round and targeting the 8-th output bit of the first Sbox in the first round.

experiment supports that targeting s_1 with the second-round traces can also distinguish the correct key for both $(5, 1)$-masking and $(5, 2)$-masking.

3.3 Generalization and Mitigation for Improved Attack

By targeting different variables and tracing the sum of intermediate vectors in the first/second round, Table 7 summarizes the results of DDHO-DCA to defeat SEL masking. Two strategies hereafter are summarized for the improvement of DDHO-DCA.

- Targeting the first-round inputs of AND gates in the Sbox circuit when computing the sum of the variables with the same multipliers in the first round.
- Targeting the first-round outputs of Sbox when computing the sum of the variables with the same multipliers in the second round.

The two improved data-dependency attacks can break degree-1/2 SEL masking with any arbitrary order n. Let m denote the number of AND gadgets that the data-dependency attack traces and s be the number of AND gates in each AND gadget. The time complexity of the improved DDHO-DCA is $\mathcal{O}(ms)$ which is independent of the masking order and the shuffling degree. It improves the time complexity over the generic HDHO-DCA attack. For mitigating the improved attacks, two security notions *gadget security* and *circuit security* are informally discussed for white-box cipher designers.

Table 7: The correlation when targeting various first-round variables and tracing different data-dependency nodes in the first/second round of defeating SEL masking.

Target variables	The sum of multipliers		The sum of variables with the same multipliers	
	First round	Second round	First round	Second round
First-round Sbox outputs	DDHO-DCA failed (d=1/2)	strong ($d = 1$) weak ($d = 2$)	failed (d=1/2)	strong ($d = 1$) strong ($d = 2$)
First-round inputs of AND gates	strong ($d = 1$) weak ($d = 2$)	failed (d=1/2)	strong ($d = 1$) strong ($d = 2$)	failed (d=1/2)

Gadget Security. Since the data-dependency attack targets the variables in an AND gadget, *gadget security* addresses the property against exploiting the information between a variable and its multipliers to recover the linear shares. We note that the co-operands in AND gadgets of SEL masking contain all linear shares of a sensitive variable. Thus, the intermediate variable in a secure AND gadget shall not multiply with all the elements of linear shares. Therefore, the multipliers have no leakage of linear shares. Moreover, the set of variables with the same multipliers in SEL masking is equal to the set of linear shares. Consequently, an attacker cannot distinguish a set of variables in a secure AND gadget based on their co-operands.

Circuit Security. The sum of multipliers observed by data-dependency attacks is correlated to an input variable of AND gates in the Sbox circuit. By targeting such variables in BP Sbox, all key bytes of WhibOx 2019 challenge #100 can be extracted. Hence, the intermediate variables, especially the inputs of AND gates, must be unpredictable in a secure Sbox circuit. This inspires that using a secret Sbox circuit might prevent the key leakage, although it might not follow Kerckhoffs's principle. We note that targeting s_8 (also the input variable x_8 in BP Sbox) can only recover 3 key bytes of #100, which implies that #100 might implement different Sbox circuits against the computational attack with a single predicted value. Nevertheless, the input variables of AND gates in a secure Sbox circuit must not be directly assigned by the Sbox inputs. In this case, the inputs of a secure Sbox circuit must be processed by enough XOR gates before applying an AND gate.

4 Security Evaluation on Dummy Shuffling

This section analyzes HO-DCA attacks on a specific BU shuffling and the countermeasure combined with SEL masking.

4.1 The Target Shuffling Implementation

BU shuffling is constructed with a single main slot and multiple dummy slots. It applies several redundant computations to hide the real operation. The main slot computes the function on the main input while the dummy slot computes the function on a pseudorandomly generated input. The output of a main slot is protected by a pseudorandom mask while the output of a dummy slot is replaced with a mask. All masks will be XORed to zero. Thus, it can retrieve the output of the main slot by an XOR phase.

Algorithm 5 The construction of an evaluated function

Input: g: an integer ≥ 1, the number of slots is $h = 2^g$;
$k \in \mathbb{F}_2^n$: a key state;
$S : \mathbb{F}_2^n \to \mathbb{F}_2^m$: an Sbox;
$G_{k_1}(x) : \mathbb{F}_2^n \to \mathbb{F}_2^g$: a PRF instance with key k_1;
$H_{k_2}(x) : \mathbb{F}_2^n \to \mathbb{F}_2^n$: a PRF instance with key k_2;
$F_{k_3}[x](h) : \mathbb{F}_2^n \times \mathbb{F}_2^g \to \mathbb{F}_2^m$: a TZS-PRF instance with key k_3.
Output: An evaluated function $E(x) : \mathbb{F}_2^n \to (\mathbb{F}_2^m)^h$.

$\bar{\boldsymbol{v}} \leftarrow$ an array with h entries
$x' \leftarrow x$
Input-shuffling:
for $1 \leq i \leq h$ **do**
  if $G_{k_1}(x) = i$ **then** ▷ A main slot.
    $\bar{\boldsymbol{v}}[i] \leftarrow x$
  else ▷ A dummy slot.
    $x' \leftarrow H_{k_2}(x')$
    $\bar{\boldsymbol{v}}[i] \leftarrow x'$
Evaluation slots:
for $1 \leq i \leq h$ **do**
  $\bar{\boldsymbol{v}}[i] \leftarrow S(\bar{\boldsymbol{v}}[i] \oplus k)$
  $m \leftarrow F_{k_3}[x](i)$ ▷ Mask.
  if $G_{k_1}(x) = i$ **then** ▷ A main slot.
    $\bar{\boldsymbol{v}}[i] \leftarrow \bar{\boldsymbol{v}}[i] \oplus m$
  else ▷ A dummy slot.
    $\bar{\boldsymbol{v}}[i] \leftarrow m$
$E(x) \leftarrow \bar{\boldsymbol{v}}[1]||\cdots||\bar{\boldsymbol{v}}[h]$
return E

However, BU shuffling does not provide an instance of the slot and does not reveal the details about the combination of input-shuffling and evaluation slots phases. Hence, it is implicit to implement BU shuffling. We note that a typical example of slot is the Sbox in a block cipher which uses the same Sbox in each round. The function of Sbox might be invoked multiple times independently in an implementation. Therefore, each slot in our analysis consists of an Sbox with a key addition. We propose a specific BU shuffling that protects each key-embedded Sbox with multiple dummy slots. Furthermore, an evaluated function is proposed to combine the input-shuffling and evaluation slots phases. It contains a cluster of slots, in which a main slot computes on the correct input while other slots compute on random inputs. Thus, it requires the input of main slot and outputs the results of main and dummy slots. Three parts of the evaluated function pseudorandomly depend on the input, that are the generation of dummy inputs, the location of the main slot, and the generation of masks. Hence, the construction requires three pseudorandom functions (PRFs). One of them is a tweakable zero-sum PRF ([BU21], Definition 15) of which the outputs sum to zero. The construction of an evaluated function is described in Algorithm 5. The resulting function maps an input state $x \in \mathbb{F}_2^n$ into t outputs $\{\bar{\boldsymbol{v}}[1], \cdots, \bar{\boldsymbol{v}}[h]\} \in (\mathbb{F}_2^m)^h$.

Such a construction can be also implemented as a LUT to prevent the exposure of the index of main slot. Figure 6 also depicts the evaluated function and the following output-selection phase in the specific BU shuffling. The symbol \$ denotes a dummy input. The dummy slot outputs a mask m_i while the output of the main slot is masked by m_j for $1 \leq i, j \leq h$ and $i \neq j$. All masks will be XORed to zero.

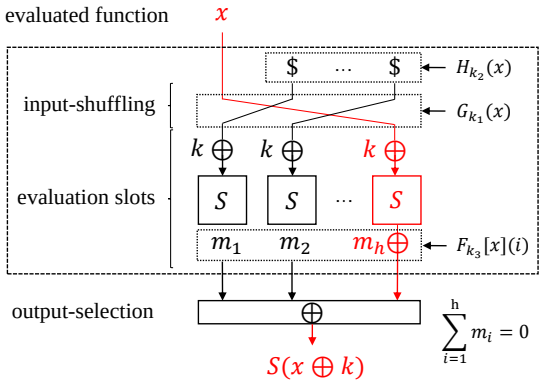


Figure 6: An evaluated function and the following output-selection phase.

Based on this construction, each Sbox in a block cipher can be replaced by an evaluated function. We note that the output-selection phase can be performed inside the following layers of the cipher. A detailed definition of an evaluated function with the AES Sbox in the specific BU shuffling is described as follows. Let $\boldsymbol{v} = \{v_1, \cdots, v_t\}$ represent the outputs of an evaluated function, where $v_i \in \mathbb{F}_2$ for $1 \leq i \leq t$. The outputs $\boldsymbol{v}$ can be split into h subsets and the cardinality of each subset is 8 such that $t = 8 \cdot h$. There exists a subset $\bar{\boldsymbol{v}} = \{v_i, \cdots, v_{i+7}\}$ for $1 \leq i < i + 7 \leq t$ with $\#\bar{\boldsymbol{v}} = 8$ consisting of consecutive bits that are the results of masked main slot. Let $x \in \mathbb{F}_2^8$ denote an input state and $E : \mathbb{F}_2^8 \to \mathbb{F}_2^t$ be an evaluated function consisting of main and dummy slots. For $1 \leq j \leq h$, $m_j \in \mathbb{F}_2^8$ represent the masks satisfied $\bigoplus_{j=1}^{h} m_j = 0$. Suppose that $S : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ represents the Sbox substitution, $k \in \mathbb{F}_2^8$ denotes a key byte, and $||$ is the concatenation of the bit strings. In this case, $\boldsymbol{v} \leftarrow E(x) = \bar{\boldsymbol{v}}_1 || \cdots || \bar{\boldsymbol{v}}_j || \cdots || \bar{\boldsymbol{v}}_h$ for $1 \leq j \leq h$ ($h \geq 2$) and $\bar{\boldsymbol{v}}_j \leftarrow S(x \oplus k) \oplus m_j$ when the j-th slot is a main slot. For other dummy slots $\bar{\boldsymbol{v}}_\ell \leftarrow m_\ell$ for $\ell \in [1, h]/j$. Notably, the output of a dummy slot is replaced by a mask. Thus, we only formalize the final assignment of $\bar{\boldsymbol{v}}_\ell$. The output-selection phase indicates that

$$\bigoplus_{j=1}^{h} \bar{\boldsymbol{v}}_j = S(x \oplus k). \tag{7}$$

Following the definitions of public dummy shuffling ([BU21], Definition 8, 9, and 10), the attack model for HO-DCA on the specific BU shuffling is described as follows.

- The adversary is able to invoke the implementation many times with arbitrarily chosen inputs.
- The adversary cannot fix the randomness inside the program and has no access to shuffled permutation and output-selection phase.
- The adversary is capable of targeting the function of each slot but is hard to distinguish the computation of main slot. Specifically, she can obtain the outputs of a group of slots (e.g., an evaluated function) that contains a main slot protected by several dummy slots.

We note that the attacker can track the returned values of the evaluated function by choosing the inputs. In the following analysis, we assume that the attacker computes the outputs of an evaluated function as computation traces and mounts HO-DCA attacks to recover the secret key of the main slot.

4.2 Higher-Order DCA Attacks on Specific BU Shuffling

HO-DCA/Algebraic DCA on Specific BU Shuffling. Following the attack model, an attacker can obtain the resulting sequence $\boldsymbol{v} = \{v_1, \cdots, v_t\}$ by invoking the evaluated function with chosen inputs. Let $s_1, \cdots, s_8 \in \mathbb{F}_2$ denote the output variables of Sbox such that $\{s_1, \cdots, s_8\} \leftarrow S(x \oplus k)$. There exist h bits $v_i, v_{i+8}, \cdots, v_{i+8(h-1)}$ in $\boldsymbol{v}$, which satisfy the sum of them is s_i for $1 \le i \le 8$. The main reason is that the masks XOR to zero. Based on $\bigoplus_{j=1}^{h} \bar{\boldsymbol{v}}_j = S(x \oplus k)$, the sum of one bit of each $\bar{\boldsymbol{v}}_j$ is equal to the corresponding bit of $S(x \oplus k)$. This reduces the problem of identifying the dummy slots in shuffling to retrieving the linear shares in masking. By combining h items among $\{v_1, \cdots, v_t\}$ to form $v_i \oplus v_{i+8} \oplus \cdots \oplus v_{i+8(h-1)} = s_i$, HO-DCA computes a h-order traces consisting in $\binom{t}{h}$ items. For algebraic DCA, it solves a linear equation $\bigoplus_{i=1}^{t} a_i v_i = s_j$ where $1 \le j \le 8$. A solution vector $\boldsymbol{a} = (a_1, \cdots, a_t)$ targets the j-th bit of each slot, where $a_j, a_{j+8}, \cdots, a_{j+8(h-1)} = 1$ and other variables are 0.

An example of HO-DCA/algebraic DCA is described as follows. Assuming that a main slot is protected by 3 dummy slots such that $h = 4$ and $t = 8 \cdot 4 = 32$. Within the output sequence $\boldsymbol{v} = \{v_1, \cdots, v_{32}\}$, we have $v_1 \oplus v_9 \oplus v_{17} \oplus v_{25} = s_1$. HO-DCA computes $\binom{32}{4}$ items to extend the traces and recover s_1. The solution vector of algebraic DCA is

$$\boldsymbol{a} = (1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0).$$

Integrated HO-DCA on Specific BU Shuffling. By locating the output $\boldsymbol{v}$ of an evaluated function, the attacker can compute the correlation between a predicted value $\bigoplus_{\ell=1}^{8} s_\ell$ and the sum of all samples of the obtained subtraces. Since the masks XOR to zero, the sum of n elements in $\boldsymbol{v}$ is equal to the sum of the output bits of Sbox. Derived from Equation (7), we have

$$\bigoplus_{i=1}^{t} (\boldsymbol{v})_i = \bigoplus_{i=1}^{t} v_i = \bigoplus_{\ell=1}^{8} (\bigoplus_{j=1}^{h} \bar{\boldsymbol{v}}_j)_\ell = \bigoplus_{\ell=1}^{8} (S(x \oplus k))_\ell = \bigoplus_{\ell=1}^{8} s_\ell. \tag{8}$$

Thus, the adversary can obtain a strong correlation by targeting the sum of Sbox outputs and computing the sum of traced function outputs. Moreover, evaluating all 16 inputs also helps to obtain a strong correlation by targeting the j-th Sbox. Based on Lemma 1, if the obtained trace also contains other p nodes except for an evaluated function, the correlation is reduced by a factor $\sqrt{p+1}$.

Assume the attacker locates all 16 evaluations and records the corresponding outputs $\boldsymbol{v}_i$ for $1 \le i \le 16$. Let t_i denote the length of sequence output from each function. We note that the number of slots in each evaluated function can be different. Based on Equation (8), the sum of all evaluated functions is

$$\begin{aligned} &\bigoplus_{i=1}^{t_1} (\boldsymbol{v}_1)_i \oplus \bigoplus_{i=1}^{t_2} (\boldsymbol{v}_2)_i \oplus \cdots \oplus \bigoplus_{i=1}^{t_{16}} (\boldsymbol{v}_{16})_i \\ =&\bigoplus_{\ell=1}^{8} s_\ell^1 \oplus \bigoplus_{\ell=1}^{8} s_\ell^2 \oplus \cdots \oplus \bigoplus_{\ell=1}^{8} s_\ell^{16}. \end{aligned}$$

By enumerating the j-th input whilst fixing others, the result is the sum of the j-th Sbox outputs with a constant c such that $\bigoplus_{\ell=1}^{8} s_\ell^1 \oplus \cdots \oplus \bigoplus_{\ell=1}^{8} s_\ell^{16} = \bigoplus_{\ell=1}^{8} s_\ell^j \oplus c$. This result

is highly correlated to $\bigoplus_{\ell=1}^{8} s_\ell^j$. Hence the j-th key byte can be recovered by targeting the sum of j-th Sbox outputs and computing the sum of all evaluated outputs. To verify this leakage, we exploit such a dummy shuffling implementation. The average number of dummy slots in each function is 2. Figure 7 depicts the correlation of each key guess by targeting the sum of the outputs of the first Sbox. Besides the results of combining the outputs of the first evaluated function, it can also represent the combination of all outputs of 16 evaluations. The correct key 0x56 ranks first with the correlation 1. Particularly, an integrated HO-DCA utilizes the sum of evaluated function(s) such that the attacker does not require the ability to split each slot within an evaluation. Thus, it also threatens hidden dummy shuffling.

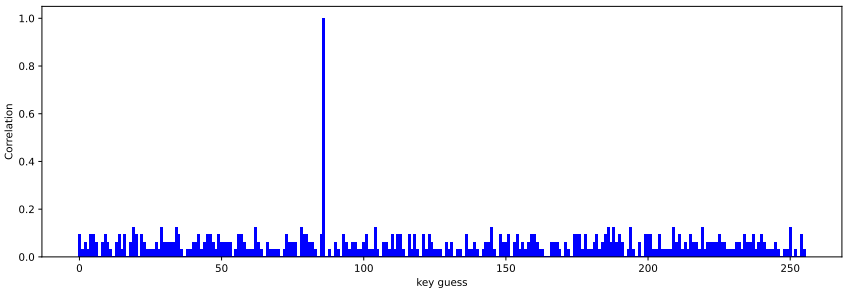


Figure 7: The correlation curve when computing the sum of the first or all evaluated function(s) and targeting the sum of the first Sbox outputs.

4.3 Attacks on The Combination of Masking and Shuffling

Section 4.2 investigated the vulnerability of a specific BU shuffling against HO-DCA attacks. These attacks exploit the sum of (sub)traces to predict a sensitive variable. It can work because the applied masks XOR to zero. The masked variable and the other masks form the linear shares as in a masking scheme. Thus, the masks can be removed by an addition among intermediate variables as HO-DCA against masking. A similar technique of protecting a sensitive vector by XORing a mask was proposed by Lee and Kim [LK20]. Their scheme hides the sensitive variables by introducing random masks. Then it retrieves the original result by an addition between masked value and masks. However, it cannot resist the adaptive side-channel analysis [TGS+21]. To mitigate HO-DCA attacks, an intuition is to combine the masking and shuffling. The main idea is to integrate two independent countermeasures for more effective protection against HO-DCA attacks. This section proposes a construction of combining SEL masking and the specific BU shuffling. Moreover, HO-DCA is also analyzed against this combined countermeasure.

Target Implementation. We assume that the target implementation is a Boolean circuit protected by SEL masking and the specific BU shuffling. For masking, the degree is $d+1$ and the order is e. The shuffling degree is h which implies that each evaluated function has h slots. Each original variable is split into $r = e + d + 1$ shares satisfying Equation (4). The evaluated function of Algorithm 5 maps input shares $\{x\} \in (\mathbb{F}_2^r)^n$ into h outputs $\{\bar{\boldsymbol{v}}[1], \cdots, \bar{\boldsymbol{v}}[h]\} \in ((\mathbb{F}_2^r)^m)^h$. It corresponds to $n \times r$ input shares and $h \times m \times r$ output shares. The Sbox is implemented as a bitsliced circuit. All the XOR and AND operations between original variables are replaced by the secure XOR and AND gadgets. The randomness in the evaluated function, e.g., dummy input shares, the location of the main slot, masks, and the refresh of gadgets all depend on the input.

More precisely, the combined countermeasure implements each slot in the evaluated function (following the specific BU shuffling) by applying secure gadgets (following SEL masking). A detail definition of such an evaluated function with AES Sbox is described as follows. Let $\boldsymbol{v} = \{v_1, \cdots, v_t\}$ represent the outputs of an evaluated function, where $v_i \in \mathbb{F}_2$ for $1 \leq i \leq t$. It can be split into h subsets and the cardinality of each subset is $8r$ such that $t = 8r \cdot h$. Every r variables $\bar{v} = \{v_1, \cdots, v_r\}$ correspond to the shares of a protected output satisfying the SEL masking and $r = e + d + 1$ such that $\boldsymbol{v} = \{\bar{v}_1, \cdots, \bar{v}_{8h}\}$. There exists a subset $\bar{\boldsymbol{v}} = \{\bar{v}_i, \cdots, \bar{v}_{i+7}\}$ for $1 \leq i < i + 7 \leq 8h$ with $\#\bar{\boldsymbol{v}} = 8$ consisting of consecutive shares that are the results of masked main slot. Let $x \in (\mathbb{F}_2^r)^8$ denote the input shares, $E : (\mathbb{F}_2^r)^8 \to ((\mathbb{F}_2^r)^8)^h$ be an evaluated function consisting of main and dummy slots, and $m_j \in (\mathbb{F}_2^r)^8$ for $1 \leq j \leq h$ represent the masks satisfied $\bigoplus_{j=1}^{h} m_j = 0$. Suppose that $S : (\mathbb{F}_2^r)^8 \to (\mathbb{F}_2^r)^8$ represents the Sbox circuit protected by SEL masking, $k \in (\mathbb{F}_2^r)^8$ denotes a key byte and $G_\oplus$ represents an XOR gadget. In this case, $\boldsymbol{v} \leftarrow E(x) = \bar{\boldsymbol{v}}_1 || \cdots || \bar{\boldsymbol{v}}_j || \cdots || \bar{\boldsymbol{v}}_h$ for $1 \leq j \leq h$ ($h \geq 2$) and $\bar{\boldsymbol{v}}_j \leftarrow S(G_\oplus(x, k)) \oplus m_j$ while the j-th slot is a main slot. For other dummy slots $\bar{\boldsymbol{v}}_\ell \leftarrow m_\ell$ where $\ell \in [1, h]/j$, the output-selection phase implies that

$$\bigoplus_{j=1}^{h} \bar{\boldsymbol{v}}_j = S(G_\oplus(x, k)).$$

This XOR phase indicates that the sum of a set of nodes in the computation trace can cancel the effect of dummy shuffling. Thus, it can retrieve the secret shares of masking. We note that SEL masking is vulnerable to HO-DCA attacks as shown in Section 3.1. Furthermore, HO-DCA combines the samples to recover the sensitive variable. Hence, the combined masking and shuffling countermeasure inherently cannot resist HO-DCA attacks.

HDHO-DCA/HDDA on Combined Masking and Shuffling. Suppose that an attacker obtains the resulting sequence $\boldsymbol{v} = \{v_1, \cdots, v_t\}$ by invoking the evaluated function with chosen inputs. HDHO-DCA and HDDA with $(d+1)$-th degree and $(h^{d+1} + eh)$-th order can directly break the combined countermeasure. Assume that $e = 2$, $d = 1$, $h = 2$, and $r = e + d + 1 = 4$. A sensitive variable $x = \tilde{x}_0 \tilde{x}_1 \oplus x_1 \oplus x_2$. Let m_i $(1 \leq i \leq r)$ denote the mask of each share. The obtained trace $\boldsymbol{v} = \{v_1, \cdots, v_t\} = \{\cdots, m_1, m_2, m_3, m_4, \cdots, (\tilde{x}_0 \oplus m_1), (\tilde{x}_1 \oplus m_2), (x_1 \oplus m_3), (x_2 \oplus m_4), \cdots\}$. Let $\tilde{x}_0 \oplus m_1 = a_1$, $\tilde{x}_1 \oplus m_2 = a_3$, $x_1 \oplus m_3 = a_3$, and $x_2 \oplus m_4 = a_4$ such that $\boldsymbol{v} = \{v_1, \cdots, v_t\} = \{\cdots, m_1, m_2, m_3, m_4, \cdots, a_1, a_2, a_3, a_4, \cdots\}$. We note that x can be unmasked as $x = (m_1 \oplus a_1)(m_2 \oplus a_2) \oplus m_3 \oplus a_3 \oplus m_4 \oplus a_4 = m_1 m_2 \oplus m_1 a_2 \oplus a_1 m_2 \oplus a_1 a_2 \oplus m_3 \oplus a_3 \oplus m_4 \oplus a_4$. The attacking steps of HDHO-DCA are illustrated as follows.

- Degree-$(d+1)$ extension. Multiplying all $d + 1 = 2$ tuples in $\boldsymbol{v}$ and extending it as $\boldsymbol{v}' = \{\cdots, m_1, m_2, m_3, m_4, \cdots, a_1, a_2, a_3, a_4, \cdots, m_1 m_2, m_1 a_2, \cdots, a_1 m_2, a_1 a_2, \cdots\}$. The higher-degree traces consist of $q = \binom{t}{d+1}$ items.
- Order-$(h^{d+1} + eh)$ extension. Computing the sum of every $h^{d+1} + eh = 8$ items in $\boldsymbol{v}'$. The higher-order traces consist of $\binom{t+q}{h^{d+1}+eh}$ nodes. The computing elements might contain the set $\{m_3, m_4, a_3, a_4, m_1 m_2, m_1 a_2, a_1 m_2, a_1 a_2\}$ such that $x = m_1 m_2 \oplus m_1 a_2 \oplus a_1 m_2 \oplus a_1 a_2 \oplus m_3 \oplus a_3 \oplus m_4 \oplus a_4$ can be recovered.

Since HDDA has the same degree-$(d+1)$ extension as HDHO-DCA, it solves the linear system on the traces $\boldsymbol{v}'$. Therefore, the time complexity of HDDA is $\mathcal{O}(|\mathcal{K}| \cdot (t+q)^{2.8})$.

For each node in the computation trace, since the masks XOR to zero, there exist other $h - 1$ samples such that the sum of the h nodes is equal to a share of the main output. A pre-processing step can be executed to defeat the XOR phase of BU shuffling. It consists of a combination of all h nodes in $\boldsymbol{v}$ resulting in $p = \binom{t}{h}$ items formed in $v_{i_1} \oplus \cdots \oplus v_{i_h}$ where

$1 \leq i_1 \leq \cdots \leq i_h \leq t$. This process removes the masks and exposes the shares of SEL masking. The following HDHO-DCA and HDDA attacks are similar to the analysis of SEL masking. We note that the multiplication phase of HDHO-DCA is the same as the one of HDDA. As the application of SEL masking, there exist $d+1$ nodes in $\{p\}$ of which the product is the non-linear part of a sensitive variable. Thus, the degree-$(d+1)$ computation trace consists of $g = \binom{p}{d+1}$ items. Consequently, the pre-processing $\{p\}$ is extended with degree-$(d+1)$ trace $\{g\}$. There exist $e+1$ nodes corresponding to the linear shares of SEL masking. For HDHO-DCA, the sum of every $e+1$ nodes is computed. Thus, the sensitive variable is retrieved as a node in the trace. The overall time complexity of HDHO-DCA is $\mathcal{O}\left(\binom{p+g}{e+1}\right)$. Suppose that $t = 10$, $e = 2$, $d = 1$, and $h = 2$, the computation times of the higher-order traces for a direct HDHO-DCA are $\binom{t+q}{h^{d+1}+eh}$=1,217,566,350. However, with the pre-processing step, the computation times can be reduced to $\binom{p+g}{e+1}$=184,251,045. Figure 8 depicts the correlation of each key guess when mounting HDHO-DCA with a pre-processing step on such a combined countermeasure. For this attack, the degree is 2 and the order is 3. The correct key 0x56 ranks first with a correlation greater than 0.4. For HDDA, the linear shares can be recovered by solving the linear system with the time complexity $\mathcal{O}(|\mathcal{K}| \cdot (p+g)^{2.8})$.

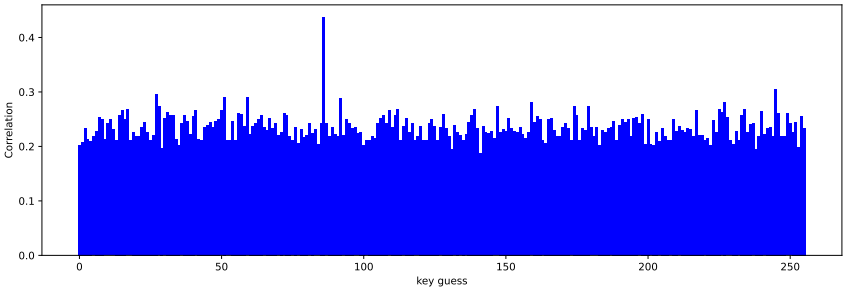


Figure 8: The correlation curve of degree-2 and order-3 HDHO-DCA on analyzing a key byte in the combined masking and shuffling.

Improved DDHO-DCA on Combined Masking and Shuffling. Because of the application of SEL masking, the improved DDHO-DCA will collect the multipliers of each variable in each slot computation. Consequently, it computes the traces corresponding to h slots in each evaluated function. Due to the specific BU shuffling, the trace of the main slot is shuffled with other dummy traces. Thus, it fails in calculating the correlations of main slots. But the outputs of dummy slots are replaced by masks. If a gate value is flipped in the dummy slot, it would not affect its final output. Furthermore, the ciphertext is also constant even if such a single fault has been injected in a dummy slot. After locating the main slot, the improved DDHO-DCA can defeat the SEL masking with the time complexity $\mathcal{O}(ms)$ (see Section 3.3) of computing higher-order traces. We note that the mask of the main output is independent of the gate values which DDHO-DCA tracks.

Integrated HO-DCA on Combined Masking and Shuffling. Integrated HO-DCA computes the sum of the samples of the obtained trace $\boldsymbol{v}$. Since the applied masks of the specific BU shuffling XOR to zero, this integrated process removes the masks. The resulting value is equal to the sum of the output shares of the main slot. This implies that the integrated HO-DCA is also applicable on the combined masking and shuffling. Suppose that the output shares of the main slot are $\mathbf{c} = \{c_1, \cdots, c_{mr}\}$ where m corresponds to an

Sbox $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ ($m = 8$ for the AES Sbox) and $r = e + d + 1$. The elements in $\mathbf{c}$ contain the linear shares of a sensitive variable. Based on Lemma 1 and 2, the correlation between the prediction and the sum of linear shares is reduced by a factor $\sqrt{mr - e}$.

5 Conclusion

This paper investigates the possibility of HO-DCA attacks on SEL masking and BU shuffling which are the state-of-the-art countermeasures for white-box implementations. To recover the non-linear shares, HDHO-DCA is introduced as a generic attack against masking. The improved DDHO-DCA demonstrates the vulnerabilities of SEL masking and enhances the attack results of the challenge #100 in WhibOx 2019. Since the XOR phase can be reduced to the linear masking, we showcase that a specific BU shuffling cannot resist HO-DCA attacks. Furthermore, the results support that the combination of SEL masking and the specific BU shuffling still cannot defeat our HDHO-DCA and improved DDHO-DCA attacks. It is still challenging to construct countermeasures for resisting these improved HO-DCA techniques. Future work consists in extending the improved DDHO-DCA to a higher-degree context and applying it to the combined masking and shuffling implementations. Another interesting research direction is to improve BU shuffling to avoid using the vulnerable XOR phase.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was by the National Key R&D Program of China (Grant No.2020AAA0107703), National Natural Science Foundation of China (62072192), National Defense Technology 173 Basic Improvement Project (2121-JCJQ-JJ-0931), National Cryptography Development Fund (MMJJ20180206) and CCF Tencent Open Fund.

References

[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.

[BP09] Joan Boyar and Rene Peralta. New logic minimization techniques with applications to cryptology. Cryptology ePrint Archive, Paper 2009/191, 2009. https://eprint.iacr.org/2009/191.

[BRVW19] Andrey Bogdanov, Matthieu Rivain, Philip S. Vejre, and Junwei Wang. Higher-order DCA against standard side-channel countermeasures. 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 118–141. Springer, 2019.

[BT20] Estuardo Alpirez Bock and Alexander Treff. Security assessment of white-box design submissions of the CHES 2017 CTF challenge. In Guido Marco Bertoni and Francesco Regazzoni, editors, *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Revised Selected Papers*, volume 12244 of *Lecture Notes in Computer Science*, pages 123–146. Springer, 2020.

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 373–402. Springer, 2018.

[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.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000.

[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.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[FMIS+] Yunsi Fei, Vincent Mooney III, Patrick Schaumont, Andrey Bogdanov, Stefan Kölbl, Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. CHES 2019 capture the flag challenge - the WhibOx contest edition 2. https://whibox.io/contests/2019/.

[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.

[GRW20] Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):454–482, 2020.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[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.

[KHL11] HeeSeok Kim, Seokhie Hong, and Jongin Lim. A fast and provably secure higher-order masking of AES s-box. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2011.

[LK20] Seungkwang Lee and Myungchul Kim. Improvement on a masked white-box cryptographic implementation. *IEEE Access*, 8:90992–91004, 2020.

[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.

[MRP13] Yoni De Mulder, Peter Roelse, and Bart Preneel. Revisiting the bge attack on a white-box aes implementation. Cryptology ePrint Archive, Paper 2013/450, 2013. https://eprint.iacr.org/2013/450.

[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.

[PCY+] Emmanuel Prouff, Chen-Mou Cheng, Bo-Yin Yang, Thomas Baignères, Matthieu Finiasz, Pascal Paillier, and Matthieu Rivain. CHES 2017 capture the flag challenge - the WhibOx contest, an ecrypt white-box cryptography competition. https://whibox.io/contests/2017/.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[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.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969.

[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.

[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.

A The Circuit Definition of AES Sbox

The circuit definition of AES Sbox proposed by Boyar and Peralta [BP09] consists of following equations. The inputs and outputs of Sbox are $x_1, x_2, \cdots, x_8 \in \mathbb{F}_2$ and $s_1, s_2, \cdots, s_8 \in \mathbb{F}_2$, respectively. For $1 \leq i \leq 21$, $0 \leq j \leq 67$, and $0 \leq \ell \leq 17$, the intermediate variables are $y_i, t_j, z_\ell \in \mathbb{F}_2$.

$$
\begin{aligned}
y_{14} &= x_4 \oplus x_6 & y_{13} &= x_1 \oplus x_7 & y_9 &= x_1 \oplus x_4 \\
y_8 &= x_1 \oplus x_6 & t_0 &= x_2 \oplus x_3 & y_1 &= t_0 \oplus x_8 \\
y_4 &= y_1 \oplus x_4 & y_{12} &= y_{13} \oplus y_{14} & y_2 &= y_1 \oplus x_1 \\
y_5 &= y_1 \oplus x_7 & y_3 &= y_5 \oplus y_8 & t_1 &= x_5 \oplus y_{12} \\
y_{15} &= t_1 \oplus x_6 & y_{20} &= t_1 \oplus x_2 & y_6 &= y_{15} \oplus x_8 \\
y_{10} &= y_{15} \oplus t_0 & y_{11} &= y_{20} \oplus y_9 & y_7 &= x_8 \oplus y_{11} \\
y_{17} &= y_{10} \oplus y_{11} & y_{19} &= y_{10} \oplus y_8 & y_{16} &= t_0 \oplus y_{11} \\
y_{21} &= y_{13} \oplus y_{16} & y_{18} &= x_1 \oplus y_{16}
\end{aligned}
$$

$$
\begin{aligned}
t_2 &= y_{12} \cdot y_{15} & t_3 &= y_3 \cdot y_6 & t_4 &= t_3 \oplus t_2 \\
t_5 &= y_4 \cdot x_8 & t_6 &= t_5 \oplus t_2 & t_7 &= y_{13} \cdot y_{16} \\
t_8 &= y_5 \cdot y_1 & t_9 &= t_8 \oplus t_7 & t_{10} &= y_2 \cdot y_7 \\
t_{11} &= t_{10} \oplus t_7 & t_{12} &= y_9 \cdot y_{11} & t_{13} &= y_{14} \cdot y_{17} \\
t_{14} &= t_{13} \oplus t_{12} & t_{15} &= y_8 \cdot y_{10} & t_{16} &= t_{15} \oplus t_{12} \\
t_{17} &= t_4 \oplus t_{14} & t_{18} &= t_6 \oplus t_{16} & t_{19} &= t_9 \oplus t_{14} \\
t_{20} &= t_{11} \oplus t_{16} & t_{21} &= t_{17} \oplus y_{20} & t_{22} &= t_{18} \oplus y_{19} \\
t_{23} &= t_{19} \oplus y_{21} & t_{24} &= t_{20} \oplus y_{18}
\end{aligned}
$$

$$
\begin{aligned}
t_{25} &= t_{21} \oplus t_{22} & t_{26} &= t_{21} \cdot t_{23} & t_{27} &= t_{24} \oplus t_{26} \\
t_{28} &= t_{25} \cdot t_{27} & t_{29} &= t_{28} \oplus t_{22} & t_{30} &= t_{23} \oplus t_{24} \\
t_{31} &= t_{22} \oplus t_{26} & t_{32} &= t_{31} \cdot t_{30} & t_{33} &= t_{32} \oplus t_{24} \\
t_{34} &= t_{23} \oplus t_{33} & t_{35} &= t_{27} \oplus t_{33} & t_{36} &= t_{24} \cdot t_{35} \\
t_{37} &= t_{36} \oplus t_{34} & t_{38} &= t_{27} \oplus t_{36} & t_{39} &= t_{29} \cdot t_{38} \\
t_{40} &= t_{25} \oplus t_{39}
\end{aligned}
$$

$$
\begin{aligned}
t_{41} &= t_{40} \oplus t_{37} & t_{42} &= t_{29} \oplus t_{33} & t_{43} &= t_{29} \oplus t_{40} \\
t_{44} &= t_{33} \oplus t_{37} & t_{45} &= t_{42} \oplus t_{41} & z_0 &= t_{44} \cdot y_{15} \\
z_1 &= t_{37} \cdot y_6 & z_2 &= t_{33} \cdot x_7 & z_3 &= t_{43} \cdot y_{16} \\
z_4 &= t_{40} \cdot y_1 & z_5 &= t_{29} \cdot y_7 & z_6 &= t_{42} \cdot y_{11} \\
z_7 &= t_{45} \cdot y_{17} & z_8 &= t_{41} \cdot y_{10} & z_9 &= t_{44} \cdot y_{12} \\
z_{10} &= t_{37} \cdot y_3 & z_{11} &= t_{33} \cdot y_4 & z_{12} &= t_{43} \cdot y_{13} \\
z_{13} &= t_{40} \cdot y_5 & z_{14} &= t_{29} \cdot y_2 & z_{15} &= t_{42} \cdot y_9 \\
z_{16} &= t_{45} \cdot y_{14} & z_{17} &= t_{41} \cdot y_8
\end{aligned}
$$

$$
\begin{aligned}
t_{46} &= z_{15} \oplus z_{16} & t_{47} &= z_{10} \oplus z_{11} & t_{48} &= z_5 \oplus z_{13} \\
t_{49} &= z_9 \oplus z_{10} & t_{50} &= z_2 \oplus z_{12} & t_{51} &= z_2 \oplus z_5 \\
t_{52} &= z_7 \oplus z_8 & t_{53} &= z_0 \oplus z_3 & t_{54} &= z_6 \oplus z_7 \\
t_{55} &= z_{16} \oplus z_{17} & t_{56} &= z_{12} \oplus t_{48} & t_{57} &= t_{50} \oplus t_{53} \\
t_{58} &= z_4 \oplus t_{46} & t_{59} &= z_3 \oplus t_{54} & t_{60} &= t_{46} \oplus t_{57} \\
t_{61} &= z_{14} \oplus t_{57} & t_{62} &= t_{52} \oplus t_{58} & t_{63} &= t_{49} \oplus t_{58} \\
t_{64} &= z_4 \oplus t_{59} & t_{65} &= t_{61} \oplus t_{62} & t_{66} &= z_1 \oplus t_{63} \\
s_1 &= t_{59} \oplus t_{63} & s_7 &= \overline{t_{56} \oplus t_{62}} & s_8 &= \overline{t_{48} \oplus t_{60}} \\
t_{67} &= t_{64} \oplus t_{65} & s_4 &= t_{53} \oplus t_{66} & s_5 &= t_{51} \oplus t_{66} \\
s_6 &= t_{47} \oplus t_{65} & s_2 &= \overline{t_{64} \oplus s_4} & s_3 &= \overline{t_{55} \oplus t_{67}}
\end{aligned}
$$

B The Correlations amongst Intermediates and Outputs of Sbox

Figure 8 illustrates the computed correlation between an input variable of AND gate and an output of Sbox circuit. The low correlations imply that targeting the outputs of Sbox will result in a failed DDHO-DCA attack.

Table 8: The correlations amongst the input variables of AND gates and the outputs of BP Sbox.

Input variables of AND gates	Sbox outputs							
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
y_{12}	0.0313	0.1094	0.0156	0.0781	0.0625	0.0156	0.0938	0.0156
y_{15}	0.0156	0.1094	0.0313	0.0469	0.0156	0.0938	0.0313	0.0625
y_3	0.0313	0.0469	0.0938	0.0469	0.0781	0.0000	0.0625	0.0938
y_6	0.0156	0.0469	0.0000	0.0781	0.0156	0.0469	0.0313	0.0313
y_4	0.0000	0.0313	0.0156	0.0625	0.0781	0.0469	0.0313	0.0469
x_8	0.0938	0.0313	0.0313	0.0625	0.0938	0.1094	0.0000	0.0938
y_{13}	0.0781	0.0938	0.0313	0.0469	0.0938	0.0469	0.0000	0.0469
y_{16}	0.0000	0.0938	0.0781	0.0938	0.0000	0.0156	0.0469	0.0156
y_5	0.0781	0.0313	0.0313	0.1094	0.1094	0.1094	0.0469	0.0000
y_1	0.0313	0.0313	0.0625	0.0313	0.0313	0.0938	0.1094	0.0469
y_2	0.0625	0.0625	0.0000	0.0313	0.0469	0.0313	0.0156	0.0156
y_7	0.0313	0.0625	0.0781	0.0625	0.0313	0.0156	0.0938	0.0938
y_9	0.0938	0.0000	0.1094	0.0313	0.0000	0.0469	0.0469	0.0625
y_{11}	0.0938	0.0000	0.0156	0.0938	0.0938	0.0625	0.0625	0.0313
y_{14}	0.0156	0.0156	0.0469	0.0938	0.0313	0.0938	0.0625	0.0938
y_{17}	0.1094	0.0156	0.0156	0.0469	0.1094	0.0781	0.0156	0.0469
y_8	0.0156	0.0469	0.0938	0.0625	0.0313	0.0156	0.0781	0.0313
y_{10}	0.0469	0.0469	0.0938	0.0156	0.0469	0.0781	0.0156	0.1094
t_{21}	0.0313	0.0313	0.0626	0.0626	0.0313	0.0313	0.0313	0.0626
t_{23}	0.0626	0.0313	0.0313	0.0313	0.0626	0.0313	0.1252	0.0313
t_{25}	0.0626	0.0313	0.0313	0.1252	0.0626	0.0313	0.0626	0.0313
t_{27}	0.0000	0.0626	0.0000	0.0313	0.0000	0.0626	0.0000	0.0000
t_{31}	0.1252	0.0313	0.0939	0.1566	0.1252	0.0313	0.0313	0.0939
t_{30}	0.0313	0.0626	0.0313	0.0313	0.0313	0.0626	0.1252	0.0313
t_{24}	0.0313	0.0313	0.0626	0.0626	0.0313	0.0313	0.0626	0.0626
t_{35}	0.0798	0.1436	0.0160	0.0160	0.0798	0.1436	0.0479	0.0160
t_{29}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{38}	0.0000	0.0939	0.0313	0.1252	0.0000	0.0939	0.0626	0.0313
t_{44}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{37}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{33}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{43}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{40}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{42}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{21}	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626	0.0626
t_{45}	0.1094	0.0156	0.0156	0.0469	0.1094	0.0781	0.0156	0.0469
t_{41}	0.0469	0.0469	0.0938	0.0156	0.0469	0.0781	0.0156	0.1094