A light white-box masking scheme using Dummy Shuffled Secure Multiplication

Alex Charlès[1] and Aleksei Udovenko[2]

[1] DCS, University of Luxembourg, Esch-sur-Alzette, Luxembourg
alex.charles@uni.lu
[2] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
aleksei@affine.group

Abstract. In white-box cryptography, early encoding-based countermeasures have been broken by the DCA attack, leading to the utilization of masking schemes against a surge of automated attacks. The recent filtering attack from CHES 2024 broke the last viable masking scheme from CHES 2021 resisting both computational and algebraic attacks, raising the need for new countermeasures.
In this work, we perform the first formal study of the combinations of existing countermeasures and demonstrate that applying Dummy Shuffling (EUROCRYPT 2021) then ISW masking (CRYPTO 2003) to a circuit carries algebraic, correlation, and filtering security - necessary conditions to withstand state-of-the-art automated attacks. We also show that applying these two countermeasures in the opposite order leads to a Higher-Order Filtering attack, highlighting the importance of the order of application of the combined countermeasures.
We also propose a new masking scheme called *S5*, standing for the Semi-Shuffled Secret Sharing Scheme, a scheme merging Dummy Shuffling and ISW in a single countermeasure more efficiently than a direct composition.

Keywords: White-box Cryptography · S5 · ISW · Dummy Shuffling · HDDA · FLDA · HODCA

1 Introduction

In 1999, Kocher, Jaffe, and Jun [22] showed that an implementation can be vulnerable if there is a leak of its side-channel information, such as electrical power consumption or timing of the execution, which startled the development of the side-channel field. Later, in 2002, Chow, Eisen, Johnson and van Oorshot [12,13] broadened the question by supposing that an attacker could fully access the implementation during the computation, which they called white-box. For example, a white-box attacker observing the intermediate values can find the state before and after adding the key and easily recover it. Therefore, the authors proposed to encode the intermediate values of the ciphers and perform computations through a set of lookup tables. These tables typically represent three consecutive operations composed in one unit: decode the encoded input, perform operations, and re-encode the output. While the attacker knows the lookup tables in the white-box context, the idea of [12] was that decomposing the tables would be difficult. Different encoding designs were proposed [31,21], but all were broken with a variety of attacks [4,14,23].

In 2016, Bos, Hubain, Michiels, and Teuwen [8] showed that a white-box adaptation of the *differential power analysis* from [22], which they called *differential computational analysis* (DCA), applied to breaking encoding-based white-box implementations (studied consequently in more details in [1,24]), without any adaptation on the encoding design. Masking schemes such as ISW [20] were employed to thwart this attack. Furthermore, [5,17] presented a new automated attack, *linear decoding analysis* (LDA), that can efficiently break linear masking schemes. In particular, it has been used during the WhibOx 2017 contest to break the winning white-box implementation of the AES [15].

To prevent the LDA attack, Biryukov and Udovenko proposed the first non-linear masking scheme [5], and another countermeasure called *Dummy Shuffling* [6]. Although these countermeasures alone are susceptible to the DCA attack, it was suggested that they can be combined with a correlation-resistant scheme like ISW. The combination would resist in theory both DCA and LDA by forcing the two attacks to use higher-order/degree variants (HODCA [7], HDDA [17]) which have an exponential cost on their order/degree. However, they did not provide details on how such a combination should be performed or provide a security analysis of the result.

Seker, Eisenbarth, and Liskiewicz [27] proposed a masking scheme *(SEL)* of degree up to three, generalizing the quadratic scheme from [5] and making it much lighter. Optimizations of higher order/degree attacks were found [18,29], while remaining slow for high-degree instances of the SEL masking scheme. However, recently, Charlès and Udovenko showed that all instances of this scheme are weak to LPN-based attacks [10] and *filtered linear decoding analysis* (FLDA) [11], making this scheme unusable by its own.

Our contribution The break of the SEL masking scheme by filtering attacks left no viable countermeasure. Therefore, in this study, we extend the work of [6] from EUROCRYPT 2021, by proposing a first formal study of the combinations of the two main countermeasures - ISW masking scheme and dummy shuffling - and a new, more efficient scheme called S5, resisting the main automated white-box attacks in the literature.

1. *(ISW and Dummy Shuffling Combination)* ISW and Dummy shuffling achieve correlation and algebraic resistances respectively, and we demonstrate that applying both countermeasures sequentially to a circuit carries both resistances and withstand state-of-the-art attacks. Furthermore, we point out the importance of the order of application of countermeasures, as applying ISW then Dummy Shuffling results in a different circuit structure from the reverse order, with different implementation sizes.
2. *(Higher-Order filtering)* We showed that performing ISW then Dummy Shuffling is susceptible to Higher-Order Filtering, while the other order of appli-

cation is resistant. This contradicts the earlier belief that the combination's order is unimportant.

3. *(Semi-Shuffled Secret-Sharing Scheme)* To lower the implementation cost of Dummy shuffling composed with ISW while having equivalent security properties, we propose *S5*, a new countermeasure extending the AND gadget of ISW by splitting the information of only one of its shares amongst different slots, using a structure similar to Dummy Shuffling.
4. *(Security proofs)* We extend the algebraic proof from [6] to S5 and the two combinations of countermeasures, and prove Strong Non-Interference (SNI) [3] of the S5 gadget up to 14 shares using the MaskVerif tool [2].
5. *(Benchmarks)* Finally, we provide theoretical estimations and experimental benchmarks for the gate costs of all three combined countermeasures. The supporting code implementing S5 and benchmarks is available at:

github.com/S5white-box/code

2 Notations and definitions

- The binary field is denoted by $\mathbb{F}_2$ and the vector space of dimension n over $\mathbb{F}_2$ is denoted by $\mathbb{F}_2^n$.
- For a vector $v \in \mathbb{F}_2^n$ (*resp.* a list L of n elements), we denote its i^{th} element, $1 \leq i \leq n$, by v_i (*resp.* L_i), such that $v = \{v_1, \cdots, v_n\}$ (resp. $L = \{L_1, \cdots, L_n\}$).
- We denote the addition in this binary field by "$\oplus$", also denoted "XOR", and we keep these notations when adding two vectors of $\mathbb{F}_2^n$.
- Similarly, we denote the multiplication in $\mathbb{F}_2$ by "$\cdot$", also denoted by "AND", and we keep these notations to multiply two vectors of $\mathbb{F}_2^n$ coordinate-wise.
- m vectors $\in \mathbb{F}_2^n$ (resp. lists of n elements) can be expressed as a $n \times m$ matrix (resp. two-dimensional list) M. The element of M of the $i^{th}, i \in \{1, \cdots n\}$ row and the $j^{th}, j \in \{1, \cdots m\}$ column is denoted by $M_{i,j}$.
- We denote the function that returns the number of elements contained in a given list, vector, or set X by $|X|$.
- For a Boolean function f we denote its weight (the number of preimages of 1) by $|f|$.
- We denote the matrix multiplication exponent by ω, which depends on the algorithm employed: $\omega \approx 2.8$ for the Strassen algorithm [28].
- A fresh randomness is denoted by \$, and is computed by a pseudo-random number generator in the white-box setting, which is outside of the scope of this paper. A bit variable v receiving a fresh random value is denoted by $v \leftarrow \$$.

3 The framework

3.1 Preliminaries

Circuit and masking schemes Any stateless implementation can be represented as a *circuit*, a Boolean representation using only bit variables that interact with each other with bitwise gates (AND, XOR and NOT), that a masking scheme can transform. A masking scheme encodes each bit variable v into $n > 1$ *shares* using an *encoding function*, such that the corresponding *decoding function* applied to these n bit variables retrieves the original bit variable v. Each of these n shares carries partial information of v, forcing an attacker to reverse the encoding function to retrieve full information of v.

To perform the bitwise gate without having to decode the n shares and thus leaking information on v, the gates are replaced by *gadgets*, functions that take as input the shares of two variables, perform operations without leaking information, and output the shares of the resulting variable.

For instance, instead of having $\text{XOR}(x, y) = z = x \oplus y$, we would have the shares $\{x_1, \cdots, x_n\}$ and $\{y_1, \cdots, y_n\}$ as input such that $\text{Decode}(\{x_1, \cdots, x_n\}) = x$ and $\text{Decode}(\{y_1, \cdots, y_n\}) = y$, and we would replace the XOR by its gadget such that $\text{GadgetXOR}(\{x_1, \cdots, x_n\}, \{y_1, \cdots, y_n\}) = \{z_1, \cdots, z_n\}$, such that $\text{Decode}(\{z_1, \cdots, z_n\}) = z = x \oplus y$.

In the next subsections, we show that the decoding function often XORs shares together to resist correlation attacks, and can perform other methods to resist algebraic attacks, which leads to the following definition:

Definition 1. *Let $\mathcal{D} : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be a decoding function of a masking scheme $\mathcal{M}$ and v a bit variable expressed over n shares. $\mathcal{D}$ can be expressed as $L(v) \oplus N(v) = \mathcal{D}(v)$, with $L : \mathbb{F}_2^a \mapsto \mathbb{F}_2, a \leq n$ a linear function and $N : \mathbb{F}_2^b \mapsto \mathbb{F}_2, b \leq n$ a nonlinear function.*
We denote L by the linear part of $\mathcal{M}$ and N by the nonlinear part of $\mathcal{M}$.

Example 1. The BU masking scheme (*c.f.* subsection 3.3) has decoding function $D(x_1, x_2, x_3) = x_1 \oplus x_2 \cdot x_3$. So the linear part of BU is x_1, and its nonlinear part is $x_2 \cdot x_3$.

Example 2. The ISW_ℓ masking scheme (*c.f.* subsection 3.3) has decoding function $D(x_1, \cdots, x_\ell) = x_1 \oplus \cdots \oplus x_\ell$. So the linear part of ISW_ℓ is D entirely and it has no nonlinear part.

Definition 2. *Let $\mathcal{D} : \mathbb{F}_2^n \mapsto \mathbb{F}^2$ be the decoding function of a masking scheme with n shares, and let $\mathcal{L}$ be the set of linear functions mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The noise rate of the masking scheme is given by:*

$$\tau = \min_{f \in \mathcal{L}} \left(\frac{\sum_{x \in \mathbb{F}_2^n} \mathcal{D}(x) \oplus f(x)}{2^n} \right)$$

Example 3. The BU masking scheme (*c.f.* subsection 3.3) has decoding function $\mathcal{D}(x_1, x_2, x_3) = x_1 \oplus x_2 \cdot x_3$. Choosing the linear function $f(x_1, x_2, x_3) = x_1$ shows that its noise rate is $\tau = \frac{1}{4}$.

Traces This paper proposes a study of masking schemes, mainly designed to resist (extended) grey-box attacks in the white-box model. These attacks are fully automated and only require input from traces of the implementation. In the side-channel model also called the grey-box model, a trace is the side-channel information leaked during the ciphering of a plaintext. In the white-box setting, since we have full access to the implementation, we have direct access to the bit values used to perform the encryption, without any measurement noise.

We can record every bit value that takes every bitwise gate to generate traces over different inputs. These bitwise gates are called *nodes*. Through T plaintexts, a *node* will take T different values in $\{0, 1\}$, which gives a vector of dimension T over $\mathbb{F}_2$, denoted *node vector*. We can represent all of the N node vectors $V_i, i \in \{1, \cdots, N\}$ in $\mathbb{F}_2^T$ in a $T \times N$ matrix as follows:

$$\begin{array}{c c} & \begin{array}{cccc} \text{node } 1 & \text{node } 2 & \cdots & \text{node } N \end{array} \\ \begin{array}{c} \text{trace } 1 \\ \text{trace } 2 \\ \vdots \\ \text{trace } T \end{array} & \begin{pmatrix} V_{1,1} & V_{1,2} & \dots & V_{1,N} \\ V_{2,1} & V_{2,2} & \dots & V_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ V_{T,1} & V_{T,2} & \dots & V_{T,N} \end{pmatrix} \end{array}$$

Selection function In combination with the traces, a selection function is needed to perform a grey-box attack. In the case of the AES, an attack often recovers the key byte by byte. Since the input and the AES Sbox are known, it is possible to brute-force the 256 key byte possibilities and deduce for each of them the first bit of the output of the AES Sbox of the first round. For another input, we can compute 256 new values for each possibility of the key bytes. Over T traces, we obtain 256 vectors in $\mathbb{F}_2^T$ per key byte, denoted by *node vectors*. The set of selection vectors of a selection function of a cryptographic primitive is called the *key space* and is denoted by $\mathcal{K}$. For the AES, we have $|\mathcal{K}| = 4096$.

Suppose an unprotected circuit of an AES implementation, therefore, one of its nodes will correspond to the first bit of the output of the first Sbox of the first round. With enough traces and a selection function, it is possible to observe one of the node vectors corresponding exactly to one of the selection vectors. With enough traces, we can be sure the key byte generated was the correct guess. This simple attack called exact matching explains the base principle of the grey-box attacks but is not enough to break protected circuits.

3.2 Grey-box attacks in the white-box context

Differential Computational Analysis (DCA) The first white-box implementations [12,13] used encoding as a countermeasure, making the nodes vectors different from the selection vectors. However, [8] showed in their *Differential Computational Analysis* attack (DCA) that some of the node vectors were correlating with some of the selection vectors even through the encodings [26,25,9], *i.e.* the ratio of matches (or mismatches) between some of the node vectors and some of the selection vectors was not one-half like for random values. By computing the correlation of each of the node vectors with the selection vectors, Bos *et al.* showed that the highest absolute correlation score achieved by the selection

vector amongst the 256 of a key byte corresponded to the correct guess, with enough traces.
To prevent a correlation attack, a countermeasure should have a non-null linear part.

Linear Decoding Analysis (LDA) To thwart correlation attacks, linear masking schemes have been employed, which forces an attacker to find a subset of node vectors that XORs to one of the selection vectors to retrieve the corresponding key byte. In [16], the authors pointed out that we can perform linear algebra in the white-box context since we have information on the traces without any noise. Therefore an attacker can try to observe if one of the selection vectors is a solution of a linear equation from all the node vectors. However, solving a linear equation over all the node vectors would be unrealistic in time and space, so we need to attack subsets of nodes.

The simplest method to attack relevant subsets is the sliding window method, which consists of taking the $\mathcal{W}$ consecutive node vectors ($\mathcal{W}$ stands for Window size), performing the attack, and sliding by $\mathcal{S} \leq \mathcal{W}$ nodes, to take the $\mathcal{S}$ to $\mathcal{S} + \mathcal{W}$ next nodes, and so on. In this work, all the attack complexities will be given in function of the subset size, denoted by $\mathcal{W}$ for this reason. However, other techniques to choose more efficient subsets also exist [18,29], but are out of the scope of this paper.
To prevent an algebraic attack, a countermeasure should have a non-null non-linear part.

Higher Order attacks (HDDA, HODCA) Some nonlinear masking schemes were employed to prevent such algebraic attacks. Still, it is possible to perform a Higher Order version of the DCA attack (HODCA) [7] against correlation-resistant schemes, and a Higher Degree version of the LDA attack (HDDA) [17] against masking schemes algebraic-resistant schemes. Both these techniques use the same idea of increasing the window by adding all the combinations of XOR (resp. AND) of node vectors before performing DCA (resp. LDA). This increase of the window adds $\sum_{i=0}^{\mathcal{O}} \binom{\mathcal{W}}{i}$ new vectors, with $\mathcal{O}$ the order or the degree, which is equivalent to an exponential increase. Therefore, these attacks are exponential in their order or degree.
To prevent higher-order attacks, a countermeasure should have tunable parameters to make these attacks impossible in practice.

White-Box Learning Parity With Noise (WBLPN) This attack presented in [10] showed that performing a grey-box attack in the white-box setting could be considered as solving a Learning Parity with Noise (LPN) problem. The LPN problem consists of solving linear equations in the presence of noise. In the white-box setting, we can consider that the nonlinear part of a masking scheme is a noise occurring following the noise rate τ probability. This attack is exponential in $\mathcal{W}$, but the lower τ is, the more efficient the attack becomes and can compete with the higher-order ones.
To prevent WBLPN attacks, the noise rate of a countermeasure should not be low.

Filtering attacks (FLDA, HOF) In [11], a new class of attacks called *filtering* was proposed, which nullifies one share by choosing the subset of traces where the node corresponding to the share is equal to zero. This methodology is very efficient against masking schemes that have a nonlinear part consisting of monomials of high degrees, as nullifying one of the shares of a monomial nullifies the whole nonlinear monomial ($x_1 \cdot x_2 \cdot x_3$ becomes equal to zero if we force one of the three variables to be equal to zero). The authors also proposed Higher-Order Filtering (HOF) for future countermeasure designs, which nullifies multiple shares simultaneously. This will be revealed as a useful attack in the following sections.
To prevent Filtering attacks, the countermeasure security should not be tempered with a low order of filtering.

To summarize, Table 1 shows all the available attacks and their best time and space complexities in the literature.

Table 1. Time and space complexities of grey-box attacks in the white-box context onto a subset of nodes of size $\mathcal{W}$. ω is the matrix multiplication exponent, $|\mathcal{K}|$ is the number of selection vectors (4096 for the AES), τ is the noise rate of the countermeasure, k_τ (resp. $T_{\mathcal{O},\tau}$) is a constant that depends on τ (resp. $\mathcal{O}$ and τ), $c_\tau = \frac{1}{1-\tau}$, $c'_\tau = \frac{-\ln(1-\tau)}{(1/2-\tau}$.

Attack	Reference	Time, $O(\cdot)$	Traces, $O(\cdot)$
DCA$_\tau$	[8]	$\mathcal{W}k_\tau\|\mathcal{K}\|$	k_τ
LDA	[16,11]	$\mathcal{W}^\omega + \|\mathcal{K}\|\mathcal{W}$	$\mathcal{W}$
HODCA$_{\mathcal{O},\tau}$	[7]	$\mathcal{W}^{\mathcal{O}}\|\mathcal{K}\|T_{\mathcal{O},\tau}$	$T_{\mathcal{O},\tau}$
HDDA$_d$	[17,11]	$\mathcal{W}^{d\omega} + \|\mathcal{K}\|\mathcal{W}^d$	$\mathcal{W}^d$
WBLPN$_\tau$	[10]	$\mathcal{W}^{\omega-1}\|\mathcal{K}\|c'_\tau c_\tau^{\mathcal{W}}$	$\mathcal{W}c'_d$
FLDA	[11]	$\mathcal{W}^{\omega+1} + \|\mathcal{K}\|\mathcal{W}^2$	$2\mathcal{W}$
HOF$_{\mathcal{O}}$-LDA	[11]	$\mathcal{W}^{\omega+\mathcal{O}} + \|\mathcal{K}\|\mathcal{W}^{\mathcal{O}+1}$	$2^{\mathcal{O}}\mathcal{W}$

To assess the security level of a countermeasure, we define λ, the *security parameter* in Definition 3. Since every countermeasure studied in this work has a high noise rate, we did consider the WBLPN attack in this definition.

Definition 3. *We say that a countermeasure achieves λ security if no automated attack with time complexity in $O(\mathcal{W}^\lambda)$ or less can break it with non-negligible probability, under the supposition that the key space $\mathcal{K}$ is reduced to one selection vector: $|\mathcal{K}| = 1$.*

3.3 Masking schemes

ISW masking scheme A well-known solution against correlation attacks has been proposed in [20] for the grey-box context: replacing every bit variable

v by ℓ shares $\{x_1 \cdots x_\ell\}$, such that $v = \bigoplus_{i=1}^{\ell} x_i$. That way, each node vector corresponding to the shares does not correlate with any selection vectors.

While the XOR gadget consists of XORing the shares coordinate-wise ($z_i = x_i \oplus y_i$ for $i \in \{1, \cdots, \ell\}$), the AND gadget is more complex: an SNI version of it is given in Algorithm 1, taken from [3].

Algorithm 1 SecMult

Inputs:
- $\{x_1, \cdots, x_\ell\}$ s.t. $\bigoplus_{i=1}^{\ell} x_i = x$
- $\{y_1, \cdots, y_\ell\}$ s.t. $\bigoplus_{i=1}^{\ell} y_i = y$
- $\frac{(\ell-1)\ell}{2}$ fresh randomness

Output: $\{z_1 \cdots z_\ell\}$ s.t. $\bigoplus_{i=1}^{\ell} z_i = x \cdot y$

```
1:  for $i \in \{1, \cdots, \ell\}$ do
2:      $z_i \leftarrow x_i \cdot y_i$
3:  end for
4:  for $i \in \{1, \cdots, \ell\}$ do
5:      for $j \in \{(i+1), \cdots, \ell\}$ do
6:          $r \leftarrow \$$
7:          $z_i \leftarrow z_i \oplus r$
8:          $z_j \leftarrow z_j \oplus ((r \oplus (x_i \cdot y_j)) \oplus (x_j \cdot y_i))$
9:      end for
10: end for
11: return $z$
```

This AND gadget, SecMult, can be interpreted as a matrix storing intermediate computations in its cells, and yielding XOR of each of its rows as the result.

Example 4. Let us suppose that we have three shares ($\ell = 3$). Therefore, we receive $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$ such that $x_1 \oplus x_2 \oplus x_3 = x$ and $y_1 \oplus y_2 \oplus y_3 = y$, and we want to compute $\{z_1, z_2, z_3\}$ such that $z_1 \oplus z_2 \oplus z_3 = x \cdot y$. We have:

$$x \cdot y = (x_1 \oplus x_2 \oplus x_3) \cdot (y_1 \oplus y_2 \oplus y_3) =$$

$$x_1 \cdot y_1 \tag{1}$$

$$\oplus\, x_2 \cdot y_1 \oplus x_2 \cdot y_2 \oplus x_1 \cdot y_2 \tag{2}$$

$$\oplus\, x_3 \cdot y_1 \oplus x_3 \cdot y_2 \oplus x_3 \cdot y_3 \oplus x_2 \cdot y_3 \oplus x_2 \cdot y_3 \oplus x_1 \cdot y_3 \tag{3}$$

Here, the first line of the equation will correspond to the first line of the matrix, the second to the second, and the third to the third. To ensure have an SNI gadget (*c.f.* Section 6, [3]), $\frac{(\ell-1)\ell}{2} = 3$ fresh randomness $(r_{1,2}, r_{1,3}, r_{2,3})$ are involved, as explained in Algorithm 1, which gives the following matrix:

$$\begin{array}{c} z_1 \leftarrow \\ z_2 \leftarrow \\ z_3 \leftarrow \end{array} \begin{pmatrix} x_1 \cdot y_1 & r_{1,2} & r_{1,3} \\ x_2 \cdot y_1 \oplus x_1 \cdot y_2 \oplus r_{1,2} & x_2 \cdot y_2 & r_{2,3} \\ x_1 \cdot y_3 \oplus x_3 \cdot y_1 \oplus r_{1,3} & x_2 \cdot y_3 \oplus x_3 \cdot y_2 \oplus r_{2,3} & x_3 \cdot y_3 \end{pmatrix}$$

Now, $z_i, i \in \{1, 2, 3\}$ gets the XOR of the elements of the i^{th} line of the matrix. We can observe that $z_1 \oplus z_2 \oplus z_3$ contains every element of the previous equation, and two occurrences of each fresh randomness which cancels out and is, therefore, equal to $x \cdot y$.

BU masking scheme To thwart the Linear Decoding Analysis (LDA) attack that breaks ISW, a nonlinear masking scheme was proposed in [5], achieving algebraic security. This masking scheme shares every bit variable v by three shares x_1, x_2, x_3 such that $v = x_1 \oplus x_2 \cdot x_3$. However, contrarily to ISW, this scheme would not resist correlation attacks as x_1 correlates with v, however, authors theorized that once combined with an algebraic-resistant scheme such as the ISW masking scheme, the resulting circuit would resist both correlation and algebraic attacks.

SEL masking scheme A first masking scheme resisting both algebraic and correlation attacks was proposed in [27], which has decoding function: $v = \bigoplus_{i=1}^{\ell} x_i \oplus \prod_{i=0}^{d} \tilde{x}_i$. The x_i are the linear shares and bring correlation resistance as for the ISW masking scheme, forcing HODCA to order ℓ, while the $\tilde{x}_i$ bring algebraic resistance as for BU masking scheme, forcing HDDA to be degree d. Since these two attacks are exponential in their order/degree, this masking scheme was enough to thwart them if ℓ and d were chosen big enough. However, it was later shown that filtering attacks [11] could break it in polynomial time.

3.4 The dummy shuffling countermeasure

In [6], the authors showed that the shuffling methodology used to increase measurement noise in the grey-box model (*c.f.* [19,30]) could be adapted in the white-box model to prevent algebraic attacks, by creating s exact copies of the implementation to protect, called *slots*. At each plaintext, one of which, the *main slot*, is chosen randomly to perform the real computation and is given the real input, while the $s - 1$ others, *the dummy slots*, are given random inputs. The *flags* are derived pseudorandomly from the input to choose which slot will be the main. For s slots, every bit input is passed through the input-shuffling function with $s - 1$ random bits of the dummy slots, which permutes them given the flags. Each slot then computes the function on its input, resulting in s outputs. Finally, the s output bits are passed through the output-selection function to recover the main slot output, which unshuffles them given the flags.

Definition 4. *Let f be the flags of a Dummy Shuffling implementation, and L a list of $n > 1$ elements:*

- *We define a function that shuffles a list L of $n > 1$ elements by $Shuffle_f(L)$.*
- *Similarly, we define a function that unshuffles a list L of $n > 1$ elements by $Unshuffle_f(L)$.*
- *We have $Shuffle_f \circ Unshuffle_f(L) = Unshuffle_f \circ Shuffle_f(L) = L$.*
- *We denote the function that returns the element of the main slot of a shuffled list L by $DecodeDS_f(L)$.*

Applying the Shuffle function with the flags f onto a list containing zero followed by random values creates *pre-shuffled randomness*. For each AND gate, a *pre-shuffled randomness* is created and XORed to refresh them to ensure algebraic security. The Dummy Shuffling Refresh function is depicted in Algorithm 2, given the flags f.

Algorithm 2 Dummy Shuffling Refresh gadget

```
Inputs:
  • $\{x_1, \cdots, x_\ell\}$ s.t. $\texttt{Decode}_f(\{x_1, \cdots, x_\ell\}) = x$
  • $s - 1$ fresh randomness
  • the flags $f$
Output: $\{\tilde{x}_1, \cdots, \tilde{x}_\ell\}$ s.t. $\texttt{Decode}_f(\{x_1, \cdots, x_\ell\}) = x$
1: $S \leftarrow \{0, \$, \cdots, \$\}$, s.t. $|S| = s$
2: $S \leftarrow \texttt{Shuffle}_f(S)$
3: $\{x_1 \cdots x_\ell\} \leftarrow \{x_1 \oplus S_1, \cdots, x_\ell \oplus S_\ell\}$
4: return $\{x_1, \cdots, x_\ell\}$
```

Definition 5. *A dummy shuffling implementation is performed over three main phases:*

1. ***Input-encoding:*** *This first phase of the implementation chooses the flags f randomly from the input, uses it to generate every pre-shuffled randomness that will be needed during the second phase and encodes every input of the algorithm, which creates s inputs over the s slots, with one of them being the main one and is unmodified, while the others are randomly generated.*
2. ***Evaluation:*** *The second phase evaluates every copy of the algorithm for their input. Only the algorithm located in the main slot performs the real computations. The refresh gadget after every AND gate takes its pre-shuffled randomness from the first phase and adds it to the gate's output.*
3. ***Output-selection:*** *This last phase concludes the implementation by getting every output of the s slots and applying the Unshuffle function with the flags to recover the output of the main slot. The other random outputs of the dummy slots are dismissed.*

The authors prove that Dummy Shuffling with refreshes achieves algebraic security for the evaluation phase. Currently, Dummy Shuffling is the only solution proven algebraically resistant to any degree, simply by increasing the number of slots (s slots implies performing HDDA of degree $d = s$ to break it), as the SEL masking scheme only has an algebraic proof up to the third degree.

Unlike the ISW masking scheme, instead of having ℓ node vectors XORing to one of the selection vectors, in that case, s vectors equal to one of the selection vectors for an unknown subset of traces. However, this scheme alone is weak to correlation attacks, so the authors theorized that it should be combined with the ISW masking scheme, yet did not provide any details on how exactly it should be done and what security it would achieve.

Conclusion To summarize, Table 2 shows all the available countermeasures and best known attacks against them.

Table 2. Different available white-box countermeasures with their gate overhead for XOR and AND operations, given with the lowest time complexity white-box attack known to thwart it. ω is the matrix multiplication exponent, $|\mathcal{K}|$ is the number of selection vectors (4096 for the AES), and k_τ is a small constant determined in function of the noise rate τ.

Scheme	Reference	XOR cost	AND cost	Best attack	Time $O(\cdot)$
ISW_ℓ	[20]	ℓ	$3\ell^2 - 2\ell$	LDA	$\mathcal{W}^\omega + \mathcal{W}\lvert\mathcal{K}\rvert$
BU	[5]	29	39	$\mathrm{DCA}_{1/4}$	$\mathcal{W}k_{1/4}\lvert\mathcal{K}\rvert$
DS_s	[6]	$s+1$	$6s+2$	$\mathrm{DCA}_{(s-1)/2s}$	$\mathcal{W}k_{(s-1)/2s}\lvert\mathcal{K}\rvert$
$\mathrm{SEL}_{\ell,2}$	[27]	$\ell+4$	$2\ell^2+5\ell-1$	FLDA	$\mathcal{W}^{\omega+1} + \lvert\mathcal{K}\rvert\mathcal{W}^2$
$\mathrm{SEL}_{\ell,3}$	[27]	$\ell+9$	$2\ell^2+15\ell-2$	FLDA	$\mathcal{W}^{\omega+1} + \lvert\mathcal{K}\rvert\mathcal{W}^2$

Since [11], at best, a countermeasure such as $\mathrm{SEL}_{3,2}$ can achieve $\lambda = \omega+1 \approx 4$ security (*c.f.* Definition 3), which, in practice, results in very efficient attacks taking little time to recover the full key.

The remaining solution that has only been theorized is to apply two countermeasures one after the other, one with algebraic resistance, and another one with correlation resistance, hoping for a resulting circuit to have both resistance properties.

4 Combining countermeasures

The idea suggested in [5,6,11] was to combine two countermeasures to prevent algebraic and correlation attacks. While it is clear that ISW is a good candidate for avoiding correlation attacks, there are two solutions to prevent algebraic attacks. The first is to use a nonlinear masking scheme based on high-degree monomials like BU or $\mathrm{SEL}_{1,d}$, the second is Dummy Shuffling.

Combining ISW with BU or SEL We decided to not focus on the high-degree monomial-based masking schemes as none of them can achieve algebraic security of arbitrary order, as in [27], the authors brought algebraic security up to degree three which is weak to HDDA of degree three ($O(\mathcal{W}^{3\omega-1} + |\mathcal{K}|\mathcal{W}^3)$), achieving at best 3ω-security. Furthermore, the noise rate of such schemes is lowering as the degree increases, which is a vulnerability that [10] highlighted.

4.1 ISW then Dummy Shuffling

A first method to combine these two countermeasures would be to apply one on a circuit $C' \leftarrow \mathrm{ISW}(C)$, then the second one $C'' \leftarrow \mathrm{DS}(C')$. We will denote the function that applies ISW and then Dummy Shuffling to a circuit by $\mathrm{DS} \circ \mathrm{ISW}$. Interestingly, the order of application of the two countermeasures matters.

Therefore, by opposition, we denote the function that applies Dummy Shuffling and then ISW to a circuit by ISW $\circ$ DS.

Applying the ISW masking scheme with ℓ linear shares replaces in the original circuit C every bitwise gate by gadgets and every bit variable by ℓ shares. Applying Dummy Shuffling with s slots to this modified circuit ISW(C) duplicates it s times, then generates the input-shuffling phase, the output-selection phase, and XORs pre-shuffled randomness to every AND gates constituting the ISW AND gadgets. The construction is illustrated in Figure 1.

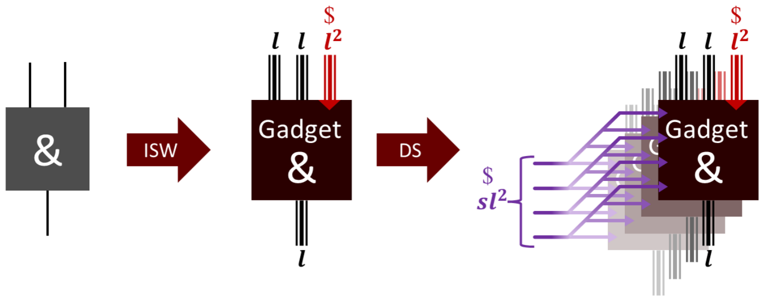

Fig. 1. Applying ISW and then Dummy Shuffling.

For such a scheme and given the flags f, a bit variable v will be shared of $s\ell$ shares $x_{i,j}, i \in \{1, \cdots, \ell\}, j \in \{1, \cdots, s\}$, such that:

$$\begin{aligned} v &= \text{Unshuffle}_f\left((x_{1,1} \oplus \cdots \oplus x_{\ell,1}), \cdots, (x_{1,s} \oplus \cdots \oplus x_{\ell,s})\right) \\ &= \text{Unshuffle}_f(x_{1,1}, \cdots, x_{1,s}) \oplus \cdots \oplus \text{Unshuffle}_f(x_{\ell,1}, \cdots, x_{\ell,s}) \end{aligned}$$

Correlation analysis In this case, determining the noise rate of the implementation would not give us information on how to perform the most efficient HODCA attack, as the noise rate would be computed using an algebraic function involving the $s\ell$ shares, making the corresponding HODCA of order $s\ell$.

Instead, we propose an order ℓ HODCA attack that consists of XORing the shares $\{x_{1,1} \oplus \cdots \oplus x_{\ell,1}\}$, which would result on an order ℓ HODCA. This XORing function matches the decoding function when the main slot corresponds to the XORed shares which happens with probability $\frac{1}{s}$, but also when the main slot does not correspond but the random value that the XORing function takes is correct, which happens with probability $\frac{s-1}{s} \cdot \frac{1}{2}$. Therefore, the two function mismatches with probability $p = \frac{s-1}{2s}$, resulting to an HODCA attack of order ℓ and noise p.

Algebraic analysis In [6], the authors showed that Dummy Shuffling resists HDDA of degree matching the number of dummy slots, here $s - 1$. Since

ISW does not bring any algebraic resistance, HDDA of degree $d = s$ can break DS $\circ$ ISW, which has a time complexity of $\mathcal{W}^{d\omega} + |\mathcal{K}|\mathcal{W}^d$.

Filtering analysis It is possible to find a better complexity that solely depends on s, by performing a Higher-Order Filtering attack. Indeed, [11] proposed such an algorithm to filter multiple nodes simultaneously, allowing fixing any node vector to a desired value.

With DS $\circ$ ISW, every XOR gates are refreshed using Algorithm 2, which will XOR random values to the dummy slots, and a zero to the main slot. Even if the main slot is unknown, we know it is not located where these random values from refresh are equal to one. So, using Higher-Order filtering of order $s - 1$, we can choose the subset of traces where the node vectors corresponding to these shuffled random values equal one.

For this filtered subset of traces where this condition holds we can ensure that the last random value left free of constraints will always be equal to zero, hence fixing the main slot to a single slot. Now that the main slot is always the same, we removed the Dummy Shuffling algebraic security, making DS $\circ$ ISW weak to an LDA attack. Performing HOF$_{s-1}$-LDA attack has a time complexity in $\mathcal{W}^{s-1+\omega} + |\mathcal{K}|\mathcal{W}^{s-1}$, which is better than HDDA of degree $d = s$.

In the end, to achieve λ security, DS$_s$ $\circ$ ISW$_\ell$ need to have for parameters $\ell = \lambda$ and $1 < s = \lambda - \omega + 1 \approx \lambda - 2$.

4.2 Dummy Shuffling then ISW

By applying Dummy Shuffling first, we duplicate the whole circuit C onto s slots, add the input-shuffling and the output-selection phases, and add pre-shuffled randomness to all the AND gates. Now, we apply ISW to this modified circuit $DS(C)$ and replace every XOR and AND gates by XOR and AND gadgets, and every variable by ℓ shares. Contrarily to DS $\circ$ ISW, we can observe that the input-shuffling and output-selection phases are here protected by ISW, adding a new layer of obfuscation. The construction is illustrated in Figure 2.

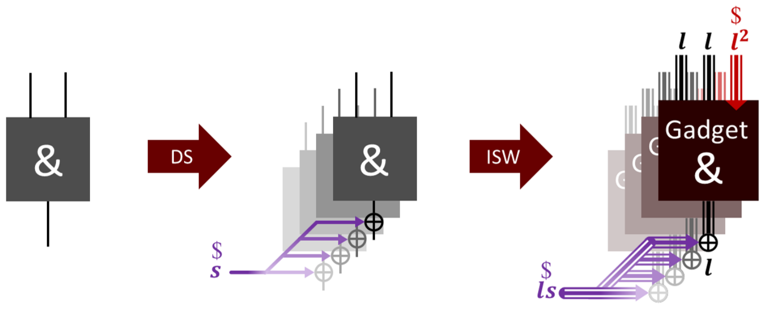

Fig. 2. Applying Dummy Shuffling and then ISW.

Security analysis Since ISW $\circ$ DS has the same decoding function as DS $\circ$ ISW, we can also perform the same HODCA$_{\ell,p}$ and HDDA$_s$ attacks. However, here the pre-shuffled randomness of Dummy Shuffling is shared, forcing the previous Higher Order Filtering LDA attack presented against DS $\circ$ ISW to be of order $\ell s - 1$, and therefore to be impracticable.

In the end, to achieve λ security, ISW$_\ell \circ$ DS$_s$ need to have for parameters $\ell = \lambda$ and $1 < s = \lceil \frac{\lambda}{\omega} \rceil \approx \lceil \frac{\lambda}{3} \rceil$, which is better than DS$_s \circ$ ISW$_\ell$. For instance, to achieve 10-security, DS $\circ$ ISW would need $\ell = 10$ and $s = 8$ while ISW $\circ$ DS would need $\ell = 10$ and $s = 4$, resulting in a lighter implementation.

Implementation size Surprisingly, for the same parameters ℓ and s, ISW $\circ$ DS has a different implementation size than ISW $\circ$ DS, as shown in Table 3. Indeed, ISW and Dummy Shuffling have different costs of transforming an AND and a XOR gate. Since transforming an AND gate creates new XOR and AND gates, applying them in different order changes the overall size of the implementation.

ISW$_\ell$∘DS$_s$

l \ s	2	3	4	5	6	7
2	0.2	1.0	1.2	1.6	2.1	2.5
3	0.7	1.7	2.1	2.8	3.5	4.2
4	1.1	2.7	3.2	4.2	5.3	6.3
5	1.6	3.7	4.5	6.0	7.5	8.9
6	2.1	5.0	6.0	8.0	10.0	12.0
7	2.7	6.5	7.8	10.4	12.9	15.5

DS$_s$∘ISW$_\ell$

l \ s	2	3	4	5	6	7
2	0.4	1.0	1.3	1.8	2.2	2.7
3	0.9	1.7	2.1	2.9	3.7	4.5
4	1.3	2.4	3.1	4.3	5.6	6.8
5	1.9	3.3	4.2	6.0	7.7	9.5
6	2.5	4.3	5.5	7.9	10.3	12.7
7	3.2	5.4	7.0	10.1	13.2	16.3

Table 3. Comparison of the implementation size (in million of gates) between ISW$_\ell$∘DS$_s$ and DS$_s$∘ISW$_\ell$ applied to a 10-round AES (31k gates), using the implementation given in the `wboxkit` tool[4].

Theorem 1. *Let C be a circuit constituted of $n_\oplus$ XOR gates and $n_\wedge$ AND gates.*

- *DS$_s$ $\circ$ ISW$_\ell(C)$ has an overhead of $(6s-4)(\ell^2-\ell)n_\wedge$ AND gates, and $s(2\ell^2-2\ell)n_\wedge + \ell s n_\oplus$ XOR gates.*
- *ISW$_\ell$ $\circ$ DS$_s(C)$ has an overhead of $(6s-4)(\ell^2-\ell)n_\wedge$ AND gates, and $(6s-4)(2\ell^2-2\ell)n_\wedge + \ell s n_\oplus$ XOR gates.*

Proof. ISW$_\ell$ transforms one AND gate into $\ell^2 - \ell$ AND gates and $2\ell^2 - 2\ell$ XOR gates, and transforms one XOR gate into ℓ XOR gates. For the whole implementation (input-encoding, evaluation, and output-selection phases) Dummy Shuffling transforms one AND gate into $6s-4$ AND gates, and transforms one XOR gates into s XOR gates [6].

Let us begin by DS$_s \circ$ ISW$_\ell(C)$: given a circuit C made of $n_\wedge$ AND gates and $n_\oplus$ XOR gates, ISW$_\ell(C)$ will have $(\ell^2-\ell)n_\wedge$ AND gates and $(2\ell^2-2\ell)n_\wedge + \ell n_\oplus$

[4] See `https://github.com/hellman/wboxkit`.

XOR gates. Finally, $\mathrm{DS}_s \circ \mathrm{ISW}_\ell(C)$ will have a total of $(6s-4)(\ell^2-\ell)n_\wedge$ AND gates, and $s\left((2\ell^2-2\ell)n_\wedge + \ell n_\oplus\right)$ XOR gates.

Likewise, for $\mathrm{ISW}_\ell \circ \mathrm{DS}_s(C)$ and for the same circuit C with $n_\wedge$ AND gates and $n_\oplus$ XOR gates, $\mathrm{DS}_s(C)$ will have $(6s-4)n_\wedge$ AND gates and $sn_\oplus$ XOR gates. Finally, $\mathrm{ISW}_\ell \circ \mathrm{DS}_s(C)$ will have $(6s-4)(\ell^2-\ell)n_\wedge$ AND gates and $lsn_\oplus + (6s-4)(2\ell^2-2\ell)n_\wedge$ XOR gates.

In theory, both $\mathrm{DS}_s \circ \mathrm{ISW}_\ell(C)$ and $\mathrm{ISW}_\ell \circ \mathrm{DS}_s(C)$ have the same number of AND gates, but have different amount of XOR gates. More precisely, ISW $\circ$ DS has $(6-\frac{4}{s})$ time more XOR gates. However, in practice, ISW $\circ$ DS has less than DS $\circ$ ISW. This can be explained as the pseudo-random generator used in white-box is not considered in the theory.

Conclusion On the whole, ISW $\circ$ DS has an equivalent implementation size as DS $\circ$ ISW given the same parameter ℓ and s, has the advantage of having obfuscated input-shuffling and output-selection phases, resists the same correlation and algebraic attacks, and has a better resistance against filtering attacks. Therefore, ISW $\circ$ DS should be preferred in any circumstances over DS $\circ$ ISW.

5 Semi-Shuffled Secret Sharing Scheme: S5

Even if ISW $\circ$ DS is a countermeasure able to achieve any given λ security, its implementation size is heavy due to the successive application of countermeasures that have not been studied to be combined.

We propose S5, which, similarly to the SEL masking scheme [27], proposes to replace a share of ISW with a nonlinear component, but here with Dummy Shuffling instead of a high-degree monomial. The design is unique as it merges a masking scheme that focuses on the gates with a countermeasure that acts on the whole implementation.

5.1 Definition of S5

Decoding function Instead of having copies of a whole implementation distributed over different slots like ISW $\circ$ DS, we would like to have ℓ shares as ISW_ℓ, with the real value of the last share taken by one of the s *slotted shares* chosen randomly for each different input. We denote the $S5$ masking schemes for ℓ linear shares with one of them shuffled amongst s slots by $S5_{\ell,s}$.

Given the $\ell+s-1$ shares $\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}$ of a bit variable x, the decoding function $\mathrm{DecodeDS}_f$ of Dummy Shuffling, and the flags f, $\mathrm{S5}_{\ell,s}$ has the following decoding function:

$$\mathtt{Decode}_f(\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}) = x_1 \oplus \cdots \oplus x_{\ell-1} \oplus \mathrm{DecodeDS}_f(x_{\ell,1}, \cdots, x_{\ell,s})$$

Encoding function As for ISW_ℓ, $\ell-1$ random values are required to encode a bit variable and fix the last share x_ℓ being equal to the sum of these random values plus the original value. Then, $s-1$ supplementary random values are required for the dummy slots as in Dummy shuffling. The procedure is depicted in Algorithm 3.

Algorithm 3 S5's Encoding function

```
Input:
  • A bit variable $z$ to share
  • $\ell + s - 2$ fresh random values
  • The flags $f$
Output: The shares $\{z_1, \cdots, z_\ell, z_{\ell,1}, \cdots, z_{\ell,s}\}$, such that
        $\mathrm{Decode}_f(\{z_1, \cdots, z_\ell, z_{\ell,1}, \cdots, z_{\ell,s}\}) = z$
1: for $i \in \{1 \cdots \ell - 1\}$ do
2:     $z_i \leftarrow \$$
3: end for
4: $z_{\ell,1} \leftarrow \bigoplus_{i=1}^{\ell-1} z_i$
5: for $i \in \{2 \cdots s\}$ do
6:     $z_{\ell,i} \leftarrow \$$
7: end for
8: $\{z_{\ell,1} \cdots z_{\ell,s}\} \leftarrow \mathtt{Shuffle}_f(\{z_{\ell,1} \cdots z_{\ell,s}\})$
9: return $\{z_1 \cdots z_{\ell-1}, z_{\ell,1} \cdots z_{\ell,s}\}$
```

Xor gadget Now that we can encode and decode our bit variables, we need to replace the bit gates with gadgets, such that they do not leak information while computing the correct output. The XOR gadget is simple to perform, as for both the ISW masking scheme and Dummy Shuffling, this gadget only consists of XORing the shares individually.

Let $\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}$ and $\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}$ be the shares of two bit variables x and y of an $\mathrm{S5}_{\ell,s}$ such that $\mathtt{Decode}_f(\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}) = x$ and $\mathtt{Decode}_f(\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}) = y$. We have:

$$\mathrm{Gadget}\textsc{xor}(\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}, \{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}) = \{(x_1 \oplus y_1) \cdots (x_{\ell-1} \oplus y_{\ell-1}), (x_{\ell,1} \oplus y_{\ell,1}) \cdots (x_{\ell,s} \oplus y_{\ell,s})\}$$

And gadget We presented SecMult, the ISW_ℓ AND gadget in Algorithm 1, and explained its matrix representation. $\mathrm{S5}_{\ell,s}$ AND gadget for one slot ($s = 1$) is exactly this same algorithm. For $s > 1$, to compute the values of all the slots, we will duplicate s time in the last row of the representation matrix.

Let $\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}$ and $\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}$ be the shares of two bit variables x and y of an $\mathrm{S5}_{\ell,s}$ such that $\mathtt{Decode}_f(\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}) = x$ and $\mathtt{Decode}_f(\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}) = y$. $\mathrm{S5}_{\ell,s}$ AND gadget has four steps:

▷ **Step 1:** Process the $\ell - 1$ first shares with the ISW AND gadget by applying $\mathrm{SecMult}(\{x_1 \cdots x_{\ell-1}\}, \{y_1 \cdots y_{\ell-1}\})$ as a regular $\mathrm{ISW}_{\ell-1}$ masking scheme.

- ▷ **Step 2:** For each shuffled share $i \in \{1, \cdots, s\}$, perform the last part of $\mathrm{SecMult}(\{x_1 \cdots x_{\ell-1}, x_{\ell,i}\}, \{y_1 \cdots y_{\ell-1}, y_{\ell,i}\})$ that corresponds to the last row of the matrix to compute only the necessary values to determine the last ℓ^{th} share of an ISW_ℓ masking scheme.
- ▷ **Step 3:** Recover the first $(\ell - 1)$ shares by XORing every elements of the first $(\ell - 1)$ line of the matrix.
- ▷ **Step 4:** Recover the last s shares by XORing every elements of the last s line of the matrix.

The AND gadget is depicted in Algorithm 4, and also includes refresh functions to achieve security properties discussed in Section 6.

Example 5. Let us take the $\mathrm{S5}_{\ell=3,s=3}$ AND gadget, without the refresh functions: we receive $\{x_1, x_2, x_{3,1}, x_{3,2}, x_{3,3}\}$ and $\{y_1, y_2, y_{3,1}, y_{3,2}, y_{3,3}\}$, the shares of two bit variables x and y such that $\mathtt{Decode}_f(\{x_1, x_2, x_{3,1}, x_{3,2}, x_{3,3}\}) = x$ and $\mathtt{Decode}_f(\{y_1, y_2, y_{3,1}, y_{3,2}, y_{3,3}\}) = y$. We want to compute $\{z_1, z_2, z_{3,1}, z_{3,2}, z_{3,3}\}$ such that $\mathtt{Decode}_f(\{z_1, z_2, z_{3,1}, z_{3,2}, z_{3,3}\}) = z = x \cdot y$.

Then, we can create a $(\ell+s-1) \times \ell$ matrix, here a 5×3 matrix M filled with zeroes to represent the computations. Step 1 begins by computing the elements of the first $\ell - 1 = 2$ columns and rows, that only depend on the linear part of the shares, namely x_1, x_2, y_1, and y_2. Then, Step 2 computes the rest of the operations that depend on the non-linear part: $x_{3,1}$, $x_{3,2}$, $x_{3,3}$, $y_{3,1}$, $y_{3,3}$, and $y_{3,3}$. Step 3 consists of XORing every element of the first $\ell - 1 = 2$ lines to compute the linear part of z: z_1 and z_2. Finally, the last Step 4 does the same for the last $s = 3$ lines and computes $z_{3,1}$, $z_{3,2}$, and $z_{3,3}$.

$$\begin{array}{r} z_1 \leftarrow \\ z_2 \leftarrow \\ z_{3,1} \leftarrow \\ z_{3,2} \leftarrow \\ z_{3,3} \leftarrow \end{array} \begin{pmatrix} x_1 \cdot y_1 & r_{1,2} & r_{1,3} \\ x_2 \cdot y_1 \oplus x_1 \cdot y_2 \oplus r_{1,2} & x_2 \cdot y_2 & r_{2,3} \\ x_1 \cdot y_{3,1} \oplus x_{3,1} \cdot y_1 \oplus r_{1,3} & x_2 \cdot y_{3,1} \oplus x_{3,1} \cdot y_2 \oplus r_{2,3} & x_{3,1} \cdot y_{3,1} \\ x_1 \cdot y_{3,2} \oplus x_{3,2} \cdot y_1 \oplus r_{1,3} & x_2 \cdot y_{3,2} \oplus x_{3,2} \cdot y_2 \oplus r_{2,3} & x_{3,2} \cdot y_{3,2} \\ x_1 \cdot y_{3,3} \oplus x_{3,3} \cdot y_1 \oplus r_{1,3} & x_2 \cdot y_{3,3} \oplus x_{3,3} \cdot y_2 \oplus r_{2,3} & x_{3,3} \cdot y_{3,3} \end{pmatrix}$$

One can observe that the different slots do not interact with each other, meaning that, given $i, j \in \{1, 2, 3\}, i \neq j$, $x_{3,i}$, $y_{3,i}$ and $z_{3,i}$ does not interact with $x_{3,j}$, $y_{3,j}$ and $z_{3,j}$. Therefore, given flags f and a corresponding main slot $m \in \{1, 2, 3\}$, $\mathrm{DecodeDS}_f(z_{3,1}, z_{3,2}, z_{3,3}) = z_{3,m}$ only depends on $x_{3,m}$ and $y_{3,m}$. Therefore:

$$\begin{aligned} z &= z_1 \oplus z_2 \oplus \mathrm{DecodeDS}_f(z_{3,1}, z_{3,2}, z_{3,3}) \\ &= z_1 \oplus z_2 \oplus z_{3,m} \\ &= \mathrm{SecMult}(\{x_1, x_2, x_{3,m}\}), (\{y_1, y_2, y_{3,m}\}) \\ &= (x_1 \oplus x_2 \oplus x_{3,m}) \cdot (y_1 \oplus y_2 \oplus y_{3,m}) \\ &= (x_1 \oplus x_2 \oplus \mathrm{DecodeDS}_f(x_{3,1}, x_{3,2}, x_{3,3})) \cdot (y_1 \oplus y_2 \oplus \mathrm{DecodeDS}_f(y_{3,1}, y_{3,2}, y_{3,3})) \\ &= x \cdot y \end{aligned}$$

Algorithm 4 S5's AND gadget

Inputs:
- $\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}$ s.t. $\texttt{Decode}_f(\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}) = x$
- $\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}$ s.t. $\texttt{Decode}_f(\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}) = y$
- One shared pre-shuffled randomness (for SPSR$_f$)
- Two pre-shuffled randomness (for Refresh$_f$)
- $\frac{\ell(\ell-1)}{2}$ fresh randomness

Output:
- $\{z_1 \cdots z_{\ell-1}, z_{\ell,1} \cdots z_{\ell,s}\}$ s.t. $\texttt{Decode}_f(\{z_1 \cdots z_{\ell-1}, z_{\ell,1} \cdots z_{\ell,s}\}) = x \cdot y$

```
 1: $\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\} \leftarrow$ Refresh($\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}$)
 2: $\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\} \leftarrow$ Refresh($\{y_1 \cdots y_{\ell-1}, y_{\ell,1} \cdots y_{\ell,s}\}$)
 3: $M \leftarrow (\ell + s - 1) \times \ell$ zero matrix

 4: for $i \in \{1 \cdots \ell - 1\}$ do                         ▷ Step 1: Handling linear shares
 5:     for $j \in \{i + 1 \cdots \ell - 1\}$ do
 6:         $M_{i,j} \leftarrow \$$
 7:         $M_{j,i} \leftarrow (M_{i,j} \oplus x_i \cdot y_j) \oplus x_j \cdot y_i$
 8:     end for
 9: end for
10: for $i \in \{1 \cdots \ell - 1\}$ do
11:     $z_i \leftarrow x_i \cdot y_i$
12: end for

13: for $i \in \{1 \cdots \ell - 1\}$ do                         ▷ Step 2: Handling shuffled shares
14:     $M_{i,\ell} \leftarrow \$$
15: end for
16: for $k \in \{1 \cdots s\}$ do
17:     for $i \in \{1 \cdots \ell - 1\}$ do
18:         $M_{k+\ell-1,i} \leftarrow (M_{i,\ell} \oplus x_i \cdot y_{\ell,k}) \oplus x_{\ell,k} \cdot y_i$
19:     end for
20: end for
21: for $k \in \{1 \cdots s\}$ do
22:     $z_{\ell,k} \leftarrow x_{\ell,k} \cdot y_{\ell,k}$
23: end for

24: for $i \in \{1 \cdots \ell - 1\}$ do              ▷ Step 3: Computing the linear part of the result
25:     for $j \in \{1 \cdots \ell\}$ do
26:         if $i \neq j$ then
27:             $z_i \leftarrow z_i \oplus M_{i,j}$
28:         end if
29:     end for
30: end for

31: $R \leftarrow$ SPSR$_f$                          ▷ Step 4: Computing the shuffled part of the result
32: for $i \in \{1 \cdots s\}$ do
33:     for $j \in \{1 \cdots \ell - 1\}$ do
34:         $z_{\ell,i} \leftarrow (z_{\ell,i} \oplus R_{i,j}) \oplus M_{i+\ell-1,j}$
35:     end for
36: end for

37: return $\{z_1 \cdots z_{\ell-1}, z_{\ell,1} \cdots z_{\ell,s}\}$
```

Randomness As explained in the following Section 6, to achieve SNI and algebraic security, S5 need to be refreshed. A refresh function takes for input the shares of a variable and adds randomness to every share without modifying its decoding value. S5 uses three type of randomness:

– *Fresh randomness:* The usual randomness produced by a pseudo-random number generator in the white-box setting. Used to generate the two other types of randomness, to encode the input values of a circuit, and in the AND gadget. A bit variable v receiving fresh randomness is denoted by $v \leftarrow \$$.
– *Pre-shuffled randomness:* This randomness is constituted of s bit values, that, once XORed to a bit variable over s slots, ensures that the main slot remains unmodified.
– *Shared pre-shuffled randomness:* Lastly, it is possible to share pre-shuffled randomness over $\ell - 1$ shares. We end up with a $s \times (\ell - 1)$ matrix R, such that the XOR of each of its lines is equal to one of the s bits of the pre-shuffled randomness.

Pre-shuffled randomness In Section 6, we show that the shuffled part of the two inputs of S5 AND gadget should be refreshed to achieve algebraic security, which can be done using the dummy shuffling refresh function. Generating a pre-shuffled randomness over s slots requires $s-1$ fresh randomness, without counting the number of randomness involved in the Decode and Shuffle function of Dummy Shuffling. Applying the shuffle function with the flags f onto a zero followed by the random values creates pre-shuffled randomness, which, once XORed with a bit variable over multiple slots, ensures that the main slot is not modified. Refreshing the shuffled part of S5 is depicted in Algorithm 5:

Algorithm 5 S5's shuffled refresh gadget

```
Inputs:
  • $\{x_1, \cdots, x_{\ell-1}, x_{\ell,1}, \cdots, x_{\ell,s}\}$ s.t. $\texttt{Decode}_f(\{x_1, \cdots, x_{\ell-1}, x_{\ell,1}, \cdots, x_{\ell,s}\}) = x$
  • $s - 1$ fresh randomness
  • the flags $f$
Output: $\{x_1, \cdots, x_{\ell-1}, \tilde{x}_{\ell,1}, \cdots, \tilde{x}_{\ell,s}\}$ s.t. $\texttt{Decode}_f(\{x_1, \cdots, x_{\ell-1}, \tilde{x}_{\ell,1}, \cdots, \tilde{x}_{\ell,s}\}) = x$
1: $S \leftarrow \{0, \$, \cdots, \$\}$, s.t. $|S| = s$
2: $S \leftarrow \texttt{Shuffle}_f(S)$
3: $\{x_1 \cdots x_{\ell-1}, \tilde{x}_{\ell,1} \cdots \tilde{x}_{\ell,s}\} \leftarrow \{x_1 \cdots x_{\ell-1}, x_{\ell,1} \oplus S_1, \cdots, x_{\ell,s} \oplus S_s\}$
4: return $\{x_1, \cdots, x_{\ell-1}, \tilde{x}_{\ell,1}, \cdots, \tilde{x}_{\ell,s}\}$
```

Shared pre-shuffled randomness In the $S5_{\ell,s}$ AND gadget, we create a $(\ell + s - 1) \times \ell$ matrix M, which has its last s rows filled with computations necessary to create the s shuffled shares of the output. Let us denote the $s \times (\ell - 1)$ matrix containing these last rows by S. In Section 6, we show that to achieve SNI security of the XOR gadget, we need to refresh every value of the S matrix. Let us denote the refreshed matrix by $\tilde{S}$.

The first constraint to such refresh function is that given the main slot $m \in \{1, \cdots, s\}$, we need to ensure that $\bigoplus_{i=0}^{\ell-1} S_{m,i} = \bigoplus_{i=0}^{\ell-1} \tilde{S}_{m,i}$, but we don't know the main slot without the shuffle or unshuffle functions, which we cannot use in the gadget as it would leak information on the flag. The second constraint is that the shuffled shares should not interact with each other.

A first idea would be to apply the refresh function of Algorithm 2 to each column of S. This would ensure SNI property and the previously exposed constraints, but performing $\ell - 1$ Shuffle call would be too heavy. Instead, shuffle one randomness gives a s-length vector r such that $r_m = 0$, which we can transform in a $(\ell + s - 1) \times \ell$ matrix R by sharing each of its s values over $\ell - 1$ shares. XORing S and R gives a refreshed matrix that respects the two constraints, and, as detailed in Section 6, ensures SNI security. The creation of this matrix denoted by *Shared pre-shuffled randomness* is given in Algorithm 6

Algorithm 6 S5's Shared pre-shuffled randomness (SPSR)

```
Inputs:
  • $s - 1 + \frac{\ell(\ell-1)}{2}$ fresh randomness
  • the flags $f$
Output: A two dimensional array $R$ containing $s$ vectors of $x$ elements such that
        $\mathrm{Decode}_f((R_{1,1} \oplus \cdots \oplus R_{1,x}), \cdots, (R_{s,1} \oplus \cdots \oplus R_{s,x})) = 0$
1: $S \leftarrow \{0, \$, \cdots, \$\}$, s.t. $|S| = s$
2: $S \leftarrow \mathtt{Shuffle}_f(S)$
3: for $k \in \{1, \cdots, \ell - 1\}$ do
4:     $R_i \leftarrow \{S_i, 0, \cdots, 0\}$, s.t. $|R_i| = \ell - 1$
5:     for $i \in \{1, \cdots, s\}$ do                    ▷ SNI refresh gadget 4b of [3]
6:         for $j \in \{i + 1, \cdots, s\}$ do
7:             $r \leftarrow \$$
8:             $R_{k,i} \leftarrow S_i \oplus r$
9:             $R_{k,j} \leftarrow S_j \oplus r$
10:        end for
11:    end for
12: end for
13: return $R$
```

Phases On the whole, S5 uses shuffle functions from Dummy Shuffling and therefore needs the three-phased structure depicted in Definition 5. The only difference is the addition of the shared pre-shuffled randomness, which needs to be pre-computed and shared in the first input-encoding phase.

5.2 S5 analysis

Higher-order attack analysis Firstly, we show in Section 6 that $\mathrm{S5}_{\ell,s}$ has the same algebraic resistance against algebraic attacks as dummy shuffling with s slots, and therefore is only broken by HDDA of degree $d \geq s$. By design, $\mathrm{S5}_{\ell,s}, \ell >$

1 resists the DCA attack, as the linear shares do not carry full information on the variable being shared. However, given τ, the noise rate of S5ℓ, s, HODCA$_{\mathcal{O},\tau}$ of order $\mathcal{O} = \ell$ can break it.

Proposition 1. *The noise rate of* $S5_{\ell,s}$ *is* $\tau = \frac{s-1}{2s}$.

Proof. Let a variable x being encoded to the shares $\{x_1, \cdots, x_{\ell-1}, x_{\ell,1}, \cdots, x_{\ell,s}\}$ by S5$_{\ell,s}$ with the flags f. By definition, we have:

$$\begin{aligned} x &= \texttt{Decode}_f(\{x_1 \cdots x_{\ell-1}, x_{\ell,1} \cdots x_{\ell,s}\}) \\ &= x_1 \oplus \cdots \oplus x_{\ell-1} \oplus \text{DecodeDS}_f\{x_{\ell,1}, \cdots, x_{\ell,s}\}) \end{aligned}$$

Choosing the linear function $f(x_1, \cdots, x_{\ell-1}, x_{\ell,1}) = x_1 \oplus \cdots \oplus x_{\ell-1} \oplus x_{\ell,1}$ will match the decoding function's output depending on $x_{\ell,1}$: when $x_{\ell,1}$ represents the main slot, the linear function is perfectly matching the decoding function, when $x_{\ell,1}$ is a dummy slot, the linear function will match the decoding function with one-half probability. Therefore, since $x_{\ell,1}$ is a dummy slot with probability $\frac{s-1}{s}$, the linear and decoding functions will have different outputs with probability $\frac{s-1}{2s} = \tau$. $\square$

Filtering analysis Let a variable x being encoded by S5$_{\ell,s}$ with the flags f to the shares $\{x_1, \cdots, x_{\ell-1}, x_{\ell,1}, \cdots, x_{\ell,s}\}$. Using a higher-order filtering attack, one can fix the s shuffled shares $\{x_{\ell,1}, \cdots, x_{\ell,s}\}$ to zero (or one) to ensure that, no matter where the main slot is located, its corresponding shuffled share would remain constant for all the subset of traces for whose this condition holds, making the scheme vulnerable to an LDA attack. So, HOF$_s$-LDA breaks S5$_{\ell,s}$ and has a time complexity in $O(\mathcal{W}^{\omega+s} + |\mathcal{K}|\mathcal{W}^{s+1})$, which is better than HDDA$_d$.

Security parameter Given a security parameter λ of Definition 3, we need to choose ℓ and s such that no attack in time complexity lesser than $\mathcal{W}^\lambda$ can break S5$_{\ell,s}$. Since, for the parameter ℓ, the best attack is HODCA$_{\mathcal{O},\tau}$ and has time complexity $\mathcal{W}^{\mathcal{O}}|\mathcal{K}|T_{\mathcal{O},\tau}$, we need to choose $1 < \ell = \lambda$. Similarily, since the best attack for the parameter s is HOF$_s$-LDA and has time complexity $O(\mathcal{W}^{\omega+s} + |\mathcal{K}|\mathcal{W}^{s+1})$, we need to choose $1 < s = \lambda - \omega$.

Implementation cost Let C be a circuit to protect with S5$_{\ell,s}$, and let $n_\oplus$ and $n_\wedge$ be its number of XOR and AND gates, respectively. As for Dummy Shuffling, the implementation is constituted of three phases. To estimate the implementation cost, we need to estimate the cost of each phase. For the first phase, we need to estimate the cost of the two pre-shuffled randomness and the shared pre-shuffled randomness used at each AND gate of C. However, since the number of inputs is negligible compared to $n_\wedge$, we do not need to estimate the cost of encoding the input.

We need to generate three pre-shuffled randomness with one used to generate a shared pre-shuffled randomness per AND gate in the original circuit C. In [6], the authors estimated the cost of performing an input shuffling to $4s \cdot n_\wedge$. Since we need three per AND gate, we end up with a cost of $12s \cdot n_\wedge$ to generate the pre-shuffled randomness.

One of these three pre-shuffled randomness needs to be shared, which is equivalent to estimate the cost of Algorithm 6, which performs $\ell-1$ times two XOR over $\sum_{i=1}^{s-1} i = \frac{\ell(\ell-1)}{2}$ combinations, for a total of $(\ell-1)s(s-1)$ operations. In total, the number of gates G_{IE} of the first input-encoding phase is $G_{IE} = (\ell-1)s(s+11)$.

For the second phase, we need to estimate the cost of the gadgets. The XOR gadget performs a XOR per couple of input shares, so costs $(\ell+s-1)n_{\oplus}$ operations. The four steps of $\mathrm{S5}_{\ell,s}$ AND gadget has different costs: $2\ell(\ell-1)+\ell-1 = \mathrm{step}_1$, $4s(\ell-1)+s = \mathrm{step}_2$, $\ell(\ell-1) = \mathrm{step}_3$, and $2s(\ell-1) = \mathrm{step}_4$. In total, the second phase costs:

$$\begin{aligned} G_E &= (\mathrm{step}_1 + \mathrm{step}_2 + \mathrm{step}_3 + \mathrm{step}_4)n_{\wedge} + (\ell+s-1)n_{\oplus} \\ &= ((\ell-1)(3\ell+6s)+\ell+s-1)\, n_{\wedge} + (\ell+s-1)n_{\oplus} \end{aligned}$$

Lastly, the output decoding phase applies the decoding function to every output, but since the number of outputs is negligible compared to the number of gates $n_{\oplus}$ and $n_{\wedge}$, we conclude that the total cost of the implementation to transform a circuit C to $\mathrm{S5}_{\ell,s}(C)$ is:

$$G_{IS} + G_E = ((\ell-1)(s(s+11)+3\ell+6s)+\ell+s-1)\, n_{\wedge} + (\ell+s-1)n_{\oplus}$$

5.3 S5 and ISW∘DS comparison

Theoretic implementation cost Let C be a circuit with $n_{\oplus}$ and $n_{\wedge}$ number of XOR and AND gates, respectively. We showed in Theorem 1 that the total implementation size for $\mathrm{ISW}_{\ell}\circ\mathrm{DS}_s(C)$ (the more secure order of application of the two countermeasures) has a total implementation size in $(6s-4)(3\ell^2-3\ell)n_{\wedge} + \ell s n_{\oplus}$ gates, compared to $((\ell-1)(s(s+11)+3\ell+6s)+\ell+s-1)\, n_{\wedge} + (\ell+s-1)n_{\oplus}$ for $\mathrm{S5}_{\ell,s}(C)$.

We can observe that to transform a AND gate, $\mathrm{S5}_{\ell,s}(C)$ scales quadratically with both ℓ and s, while $\mathrm{ISW}_{\ell}\circ\mathrm{DS}_s(C)$ scales quadratically with ℓ but only linearly with s. Moreover, the AND gate transformation of $\mathrm{ISW}_{\ell}\circ\mathrm{DS}_s(C)$ is less expensive when $\ell = s$ than $\mathrm{S5}_{\ell,s}(C)$. However, even if the XOR gate transformation scale linearly for both s and ℓ for both $\mathrm{S5}_{\ell,s}(C)$ and $\mathrm{ISW}_{\ell}\circ\mathrm{DS}_s(C)$, when $\ell = s$, it scales linearly with $\mathrm{S5}_{\ell,s}(C)$ and quadratically with $\mathrm{ISW}_{\ell}\circ\mathrm{DS}_s(C)$.

In-practice implementation cost Table 4 shows the implementation size differences for a 10-round AES. S5 has a lower Implementation size for similar parameters, with a proportional difference increasing with ℓ and s. This huge difference can be explained for two main reasons: the first is that there are more XOR gates than AND gates in the AES base implementation (62% XOR, 20% AND, 18% NOT).

The second is that for ISW and for Dummy Shuffling, the `wboxkit` tool creates a pseudo-random number generator (PRNG) to generate fresh randomness. In a circuit transformed by Dummy Shuffling, a first PNRG is created, and then, by applying ISW, it is encoded and a second one is created, which is much more heavy, while not accounted for in the theory. Whereas for S5, only one PRNG is created.

$\mathrm{ISW}_\ell \circ \mathrm{DS}_s$

l \ s	2	3	4	5	6	7
2	0.2	1.0	1.2	1.6	2.1	2.5
3	0.7	1.7	2.1	2.8	3.5	4.2
4	1.1	2.7	3.2	4.2	5.3	6.3
5	1.6	3.7	4.5	6.0	7.5	8.9
6	2.1	5.0	6.0	8.0	10.0	12.0
7	2.7	6.5	7.8	10.4	12.9	15.5

$\mathrm{S5}_{\ell,s}$

l \ s	2	3	4	5	6	7
2	0.0	0.4	0.5	0.6	0.8	0.9
3	0.3	0.6	0.7	0.9	1.1	1.3
4	0.6	0.9	1.0	1.3	1.5	1.7
5	0.8	1.2	1.4	1.7	2.0	2.3
6	1.2	1.6	1.9	2.3	2.6	3.0
7	1.6	2.1	2.5	2.9	3.4	3.9

Table 4. Comparison of the implementation size (in million of gates) between $\mathrm{ISW}_\ell \circ \mathrm{DS}_s$ and $\mathrm{S5}_{\ell,s}$ applied to a 10-round AES (31k gates), using the implementation given in the wboxkit tool (`https://github.com/hellman/wboxkit`), and our S5 implementation (`https://github.com/S5white-box/code`).

Security parameter Comparing both countermeasures for the same parameters is not relevant if we want to which one of the two is the more efficient given a known security parameter. Following Definition 3, for a security parameter λ, we have $\mathrm{S5}_{\lambda,\lambda-\omega}$ while having $\mathrm{ISW}_\lambda \circ \mathrm{DS}_{\lceil \lambda/3 \rceil}$, with both countermeasures parameters greater of equal than 2. While the s parameter of $\mathrm{ISW}_\ell \circ \mathrm{DS}_s$ scales better with λ than $\mathrm{S5}_{\ell,s}$, it becomes greater only for every $\lambda \geq 6$.

Conclusion Both $\mathrm{ISW}_\ell \circ \mathrm{DS}_s$ and $\mathrm{S5}_{\ell,s}$ are efficient countermeasures against state-of-the-art attacks, and have different characteristics so should both be considered as valuable options against gray-box attacks in the white-box context. While $\mathrm{S5}_{\ell,s}$ is more efficient for smaller (and more realistic) parameters, $\mathrm{ISW}_\ell \circ \mathrm{DS}_s$ show better scalability for security parameter $\lambda \geq 6$, while having the first input-encoding phase encoded by ISW.

We also recall that [6] described a bit-sliced implementation of dummy shuffling by filling a 64-bit CPU register with one variable from 64 slots. Contrarily to $\mathrm{ISW}_\ell \circ \mathrm{DS}_s$, $\mathrm{S5}_{\ell,s}$ can not benefit from this technique due to interactions of unslotted and slotted variables.

Lastly, even if S5 is more efficient in practice, we recommend both countermeasures as they might not have the same weaknesses for future attacks, as we showed that slight differences in the structure are enough to create a vulnerability (see Section 4).

6 Security analysis

In this section, we discuss and prove the security of the three combined schemes against the relevant trace-based gray-box attacks, mainly correlation (HODCA) and algebraic (HDDA).

Algebraic security First, we reproduce the relevant definitions and results about the dummy shuffling from [6]. The *error* of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is given by $\mathsf{err}\,(f) = \min(|f|\,, |f \oplus 1|)/2^n$.

Definition 6. *For an implementation $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$, the set $\mathcal{F}^{(d)}(C)$ denotes all functions obtained by combining intermediate functions computed in C with a function of degree at most d. Elements of this set are Boolean functions f mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2$.*

Definition 7 (Scheme [6]). *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a function. A* scheme *S* computing *F consists of*

1. *an* encoding function *$S.\mathsf{enc}(x, r_e) : \mathbb{F}_2^n \times \mathbb{F}_2^{|r_e|} \to \mathbb{F}_2^{n'}$;*
2. *an* implementation *$S.\mathsf{comp}(x', r_c) : \mathbb{F}_2^{n'} \times \mathbb{F}_2^{|r_c|} \to \mathbb{F}_2^{m'}$;*
3. *a* decoding function *$S.\mathsf{dec}(y') : \mathbb{F}_2^{m'} \to \mathbb{F}_2^m$.*

Definition 8 (τ-error-d-AS scheme [6]). *Let S be a scheme and let $d \geq 1$ be an integer. Let τ be the minimum error among all non-trivial functions from $\mathcal{F}^{(d)}(S.\mathsf{comp})$ composed with $S.\mathsf{enc} = S.\mathsf{enc}(x, r_e)$ for any* fixed *$x = \tilde{x} \in \mathbb{F}_2^n$:*

$$\tau = \min\left\{ \mathsf{err}\left(\, f(S.\mathsf{enc}(\tilde{x}, \cdot), \cdot)\,\right) \;\middle|\; f(x, r_c) \in \mathcal{F}^{(d)}(S.\mathsf{comp}) \setminus \{\mathbf{0}, \mathbf{1}\}\,, \quad \tilde{x} \in \mathbb{F}_2^n \right\},$$

where the error is computed over r_e, r_c. If $\tau > 0$, the scheme S is said to be degree-d algebraically secure with error τ (τ-error-d-AS).

Definitions of schemes First, we recall the basic dummy shuffling scheme (called "the evaluation-phase model", EPM) from [6].

Definition 9 (Scheme S_s^{DS}). *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation and let $s \geq 2$. Define the scheme $S_s^{DS}(C)$ with the following phases:*

- *$S5_{\ell,s}.\mathsf{enc}(x, r_e) : \mathbb{F}_2^n \times \mathbb{F}_2^{|r_e|} \to \mathbb{F}_2^{\tilde{n}}$ creates extra $s-1$ fully random inputs, derives random shuffling flags f from the input randomness r_e, and shuffles all the inputs using these flags (c.f. subsection 3.4). Note $\tilde{n} = n \times s$.*
- *$S5_{\ell,s}.\mathsf{comp}(x) : \mathbb{F}_2^{\tilde{n}} \to \mathbb{F}_2^{\tilde{m}}$ evaluates C at each of the s shuffled inputs (independently) and outputs all the s outputs. Here $\tilde{m} = m \times s$.*
- *$S5_{\ell,s}.\mathsf{dec}(x, f) : \mathbb{F}_2^{\tilde{m}} \times \mathbb{F}_2^{|f|} \to \mathbb{F}_2^m$ unshuffles the s outputs from the previous phase using the flags f (which need to be securely passed from the encoding phase) and outputs the right output. This phase is only defined for the correctness and is not covered by security analysis in [6].*

Before applying dummy shuffling, the circuit needs to be "refreshed".

Definition 10 (Refreshed Circuit). *Let $C(x) : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a Boolean circuit implementation with at most n_a AND gates. Define the* refreshed circuit *$\tilde{C}(x, r) : \mathbb{F}_2^n \times \mathbb{F}_2^{n_a} \to \mathbb{F}_2^m$ as follows. Replace each AND gate $a_k = z_i \wedge z_j$ in C, $1 \le k \le n_a$ by the circuit $a'_k = r_k \oplus a_k = r_k \oplus (z_i \wedge z_j)$, where r_k is the k-th extra bit; each wire using a_k is rewired to use a'_k.*

In our security proofs, we reduce the new combined schemes to the original dummy shuffling scheme S_s^{DS}, and then apply the main theorem from [6].

Theorem 2 ([6]). *Let C be an implementation and $s \ge 2$ an integer. The dummy shuffling scheme $S = S_s^{DS}(\tilde{C})$ is τ-error-d-AS for any $1 \le d \le s - 1$, with $\tau \ge 2^{-2d} \cdot (s - d)/s$.*

We now formally define the three schemes of countermeasure combinations considered in the paper. The main purpose of this is specifying the part of the implementation covered by the proof: the operations' gadgets excluding any shuffling/sharing or decoding operations (in line with existing state-of-the-art of dummy shuffling / ISW). For brevity, we will only describe the main phase (comp) of each scheme.

Definition 11 (Scheme $S5_{\ell,s}$). *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation and let $l \ge 2, s \ge 2$. Define the scheme $S5_{\ell,s}(C)$ with $S5_{\ell,s}.\mathsf{comp}(x, r_c) : \mathbb{F}_2^{\tilde{n}} \times \mathbb{F}_2^{|r_c|} \to \mathbb{F}_2^{\tilde{m}}$ as follows. The input x is consists of n lists of $s+\ell-1$ share each, as well ass $3n_a$ groups of preshuffled shared randomness (ℓ bits each). The computation proceeds by applying the $S5_{\ell,s}$ gadgets according to the circuit C, using fresh randomness from r_c and the preshared randomness from x.*

Remark 1. The S5's AND gadget already includes an equivalent of the refreshing, therefore $S5_{\ell,s}$ is intended to be applied directly to the original circuit C, without the refreshing circuit transformation.

Definition 12 (Scheme $S_{\ell,s}^{DS \circ ISW}$). *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation and let $l \ge 2, s \ge 2$. Let $C_{ISW}(x, r'_c)$ denote the circuit where the gates of C are replaced by the ISW gadgets and r'_c denotes the extra randomness used by these gadgets. Observe that C_{ISW} has $n_a \ell^2$ AND gates, where C has n_a AND gates. Define the scheme $S_{\ell,s}^{DS \circ ISW}(C)$ with $S_{\ell,s}^{DS \circ ISW}.\mathsf{comp}(x, r_c) : \mathbb{F}_2^{s\ell n + s n_a \ell^2} \times \mathbb{F}_2^{|r_c|} \to \mathbb{F}_2^{s\ell m}$ in the same way as $S_s^{DS}(\tilde{C}_{ISW})).\mathsf{comp}$, where an extra randomness r'_c used by C_{ISW} is included (in s copies) in r_c. Observe that $\tilde{C}_{ISW}$ has input size extended from ℓn to $\ell n + n_a \ell^2$ due to the refresh bits.*

Definition 13 (Scheme $S_{\ell,s}^{ISW \circ DS}$). *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation and let $l \ge 2, s \ge 2$. Let $C_{DS}(x)$ denote $S_s^{DS}(\tilde{C}).\mathsf{comp}$, i.e., s independent copies of the refreshed circuit $\tilde{C}$ in parallel. Define the scheme $S_{\ell,s}^{ISW \circ DS}(C)$ with $S_{\ell,s}^{ISW \circ DS}.\mathsf{comp}(x, r_c) : \mathbb{F}_2^{\ell s n} \times \mathbb{F}_2|r_c| \to \mathbb{F}_2^{\ell s m}$ being the ISW-protected version of $C_{DS}(x)$, i.e., where each input bit is replaced by ℓ shares and each gate is replaced by the corresponding gadget; the randomness r_c is used in the ISW gadgets.*

Security proofs Intuitively, algebraic security of all of the combined schemes is ensured by the fact that all of them contain dummy shuffling structure inside them, and ISW-like sharing does not compute any new intermediate function of original inputs. For examples, the ISW multiplication only algebraically "leaks" the product of the original (unshared) intermediates, but this product was already present in the original circuit.

In the following, we present these ideas more formally. The high-level idea is to partition the set of possible inputs of `comp` (for each possible fixed input of `enc`) and randomness into instances of basic dummy shuffling circuits. Then, the algebraic error τ is lower bounded by the minimum error across these instances, which is given by Theorem 2.

Proposition 2. *Let $l \geq 2, s \geq 2$. Then, for any underlying implementation C, the scheme $S5_{\ell,s}$ is τ-error-d-AS for all d, $1 \leq d \leq s-1$, with $\tau \geq 2^{-2d}(s-d)/s$.*

Proof. For each input group of shares $x_1, \ldots, x_{\ell-1}, x_{\ell,1}, \ldots, x_{\ell_s}$, let us consider $x_1, \ldots, x_{\ell-1}$ fixed. This makes the corresponding global input bit (which is also considered fixed by the model) encoded in the dummy shuffling manner, in the slotted variables $x_{\ell,1}, \ldots, x_{\ell,s}$. Furthermore, for each application of the AND gadget (Algorithm 4), let us consider all the used randomness fixed, except the shared pre-shuffled randomness $R = \mathrm{SPSR}_f$, where we fix $R_{i,j}$ for all $i \in \{1, \ldots, s\}, j \in \{2, \ldots, \ell-1\}$ (i.e., we don't fix $R_{i,1}$). Assuming by induction that both inputs' linear parts are fixed (i.e., $x_1, \ldots, x_{\ell-1}, y_1, \ldots, y_{\ell-1}$), it is easy to verify that the output linear part $(z_1, \ldots, z_{\ell-1})$ is also fixed. Furthermore, the output slotted variables $z_{\ell,i}$ would be equal to a quadratic function of $x_{\ell,i}$ and $y_{\ell,j}$, which is equal to $(x_{\ell,i}+c_x)(y_{\ell,i}+c_y)+c_z$ for some c_x, c_y, c_z, plus the unfixed SPSR_f variable $R_{i,1}$. This exactly matches the dummy shuffling structure with refreshes, up to adding some negations around the gadget (note that the XOR gadget also preserves the invariant). We thus obtain the required bound for each assignment of the fixed values and conclude that the bound holds for the full construction. □

Proposition 3. *Let $l \geq 2, s \geq 2$. Then, for any underlying implementation C, the scheme $S_{\ell,s}^{DS\circ ISW}$ is τ-error-d-AS for all d, $1 \leq d \leq s-1$, with $\tau \geq 2^{-2d}(s-d)/s$.*

Proof. This follows directly from Theorem 2, since dummy shuffling is applied on top of (the ISW-protected) implementation. □

Proposition 4. *Let $l \geq 2, s \geq 2$. Then, for any underlying implementation C, the scheme $S_{\ell,s}^{ISW\circ DS}$ is τ-error-d-AS for all d, $1 \leq d \leq s-1$, with $\tau \geq 2^{-2d}(s-d)/s$.*

Proof. The difference between the two compositions (ISW∘DS and DS∘ISW) lies only where the AND-refreshes are placed. Letting Ref denote the refresh procedure, which needs to be done before dummy shuffling (slotting), we have either ISW∘DS∘Ref or DS∘Ref∘ISW. However, the pure ISW transformation and the pure slotting procedure (making s copies with $s-1$ random inputs and shuffling) commute. Indeed, the considered $S_{\ell,s}^{ISW\circ DS}.\mathsf{comp}$ consists of s copies of the ISW-shared implementation. We can reinterpret it as dummy shuffling applied to one of the copies (without adding refreshes). In other words, ISW∘DS∘Ref is the same as DS∘ISW∘Ref. It is thus left to show that ISW∘Ref maintains the property of the refreshed circuit required to show Theorem 2.

We recall briefly that the proof of Theorem 2 in [6] requires that the copied circuit can be precomposed with a bijection (on the input and randomness) so that the resulting circuit only computes (at most) quadratic functions. This is easy to show due to the refresh procedure applied before ISW. The ISW sharing is applied to the refresh bit of each AND gate, effectively transforming it into ℓ random shares (in the case of a dummy slot), passed to the input of the slot, and added to the shared output of the corresponding AND gate (i.e., SecMult in the shared version). Therefore, the desired bijection can replace the ℓ refreshing shares with the result of the refreshing. This would make the result of the refreshing equal to the input of the composition of the circuit and the bijection. Consequently, any further SecMult gadget would only use such composition inputs as arguments and thus only quadratic functions would be computed, concluding the proof. □

Probing security We recall an informal definition of SNI from [3], for the full technical definition we refer the reader to the original paper.

Definition 14 (SNI [3]). *A gadget is t-SNI whenever any t of its wires can be simulated using only its shared inputs, and if its output encoding is uniform and $(t-d)$-wise independent even if d shares of each of its inputs are known (for all d such that $0 \leq d < t$).*

In [3], the authors stated that in the ISW scheme with ℓ-shares the SecMult (Algorithm 1) is $(\ell-1)$-SNI. In the case of $\mathrm{ISW}_\ell \circ \mathrm{DS}_s(C)$, since ISW_ℓ is applied after Dummy Shuffling, it ensures that $\mathrm{ISW}_\ell \circ \mathrm{DS}_s(C)$ is $(\ell-1)$-probing secure. However, it is not clear if $\mathrm{DS}_s \circ \mathrm{ISW}_\ell$ is $(\ell-1)$-probing secure as ISW_ℓ is applied before, although we showed in Section 4 that we should not consider it as weaker to higher-order filtering attacks.

To prove $(\ell-1)$-probing the security of $\mathrm{S5}_{\ell,s}$ gadgets, we fix the main slot to the first one, which can only lower the scheme security. By symmetry, evaluating the SNI security by fixing the first slot implies evaluating the security of the other s slots. If each of these cases do not leak, then any distribution of these cases (in particular, uniform or almost uniform) does not leak as well.

To this end, we consider the $S5_{\ell,s}$ gadget where the main slot is the first one $(x_{\ell,1})$ and the dummy slot variables $x_{\ell,2}, \ldots, x_{\ell,s}$ are refreshed using the pre-shuffled randomness (see Algorithm 4). We then implemented[5] this variation in the MaskVerif tool [2] and verified the $(\ell-1)$-SNI security of this gadget for $2 \leq s, \ell \leq 7$. By the composability of SNI gadgets, any circuit composed of such gadgets is SNI and thus is $(\ell-1)$-probing secure (when the main slot's position is fixed). Together with the distribution argument from above, this proves $(\ell - 1)$-probing security for $2 \leq s, \ell \leq 7$ of $S5_{\ell,s}$-protected circuits.

7 Conclusion

After the recent introduction of filtering attacks, no countermeasure can resist automated attacks in the white-box context. In this work, we proposed the first formal analysis of a composition of two countermeasures and showed that applying Dummy Shuffling then ISW resists state-of-the-art attacks.

We also proposed a new scheme called *Semi-Shuffled Secret-Sharing Scheme*, S5, a unique design merging a gate-wise masking scheme with an implementation-wise countermeasure. We showed it shares the same resistance against these automated attacks with a lighter implementation.

This study showed that combining countermeasures can be an efficient methodology to accumulate security properties, and raises the need of studying combinations of fault-resistant schemes or circuit-analysis-preventing countermeasures.

[5] https://github.com/S5white-box/code

References

1. Alpirez Bock, E., Brzuska, C., Michiels, W., Treff, A.: On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In: Preneel, B., Vercauteren, F. (eds.) ACNS 18. LNCS, vol. 10892, pp. 103–120. Springer, Heidelberg (Jul 2018). https://doi.org/10.1007/978-3-319-93387-0_6 2
2. Barthe, G., Belaïd, S., Cassiers, G., Fouque, P.A., Grégoire, B., Standaert, F.X.: maskVerif: Automated verification of higher-order masking in presence of physical defaults. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019, Part I. LNCS, vol. 11735, pp. 300–318. Springer, Heidelberg (Sep 2019). https://doi.org/10.1007/978-3-030-29959-0_15 3, 27
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 116–129. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978427 3, 8, 20, 27
4. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: Handschuh, H., Hasan, A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 227–240. Springer, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-30564-4_16 2
5. Biryukov, A., Udovenko, A.: Attacks and countermeasures for white-box designs. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 373–402. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_13 2, 9, 11
6. Biryukov, A., Udovenko, A.: Dummy shuffling against algebraic attacks in white-box implementations. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 219–248. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77886-6_8 2, 3, 9, 11, 12, 14, 21, 23, 24, 25, 27
7. Bogdanov, A., Rivain, M., Vejre, P.S., Wang, J.: Higher-order DCA against standard side-channel countermeasures. In: Polian, I., Stöttinger, M. (eds.) COSADE 2019. LNCS, vol. 11421, pp. 118–141. Springer, Heidelberg (Apr 2019). https://doi.org/10.1007/978-3-030-16350-1_8 2, 6, 7
8. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: Hiding your white-box designs is not enough. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 215–236. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_11 2, 5, 7
9. Castelnovi, L., Houzelot, A.: On the (im)possibility of preventing differential computation analysis with internal encodings. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(3), 452–471 (Jul 2024). https://doi.org/10.46586/tches.v2024.i3.452-471, https://tches.iacr.org/index.php/TCHES/article/view/11684 5
10. Charlès, A., Udovenko, A.: LPN-based attacks in the white-box setting. IACR TCHES **2023**(4), 318–343 (2023). https://doi.org/10.46586/tches.v2023.i4.318-343, https://doi.org/10.46586/tches.v2023.i4.318-343, https://tches.iacr.org/index.php/TCHES/article/view/11168 2, 6, 7, 11
11. Charlès, A., Udovenko, A.: White-box filtering attacks breaking SEL masking: from exponential to polynomial time. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(3), 1–24 (Jul 2024). https://doi.org/10.46586/tches.v2024.i3.1-24, https://tches.iacr.org/index.php/TCHES/article/view/11668 2, 7, 9, 11, 13

12. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: Digital Rights Management Workshop. Lecture Notes in Computer Science, vol. 2696, pp. 1–15. Springer (2002). https://doi.org/10.1007/978-3-540-44993-5_1 1, 2, 5
13. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 250–270. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/3-540-36492-7_17 1, 5
14. De Mulder, Y., Roelse, P., Preneel, B.: Cryptanalysis of the Xiao-Lai white-box AES implementation. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 34–49. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-35999-6_3 2
15. Goubin, L., Paillier, P., Rivain, M., Wang, J.: Reveal secrets in adoring poitras. a victory of reverse engineering and cryptanalysis over challenge 777. CHES 2017 Rump Session, slides (2017), https://ches.2017.rump.cr.yp.to/a905c99d1845f2cf373aad564ac7b5e4.pdf 2
16. Goubin, L., Paillier, P., Rivain, M., Wang, J.: How to reveal the secrets of an obscure white-box implementation. Cryptology ePrint Archive, Paper 2018/098 (2018), https://eprint.iacr.org/2018/098, https://eprint.iacr.org/2018/098 6, 7
17. Goubin, L., Paillier, P., Rivain, M., Wang, J.: How to reveal the secrets of an obscure white-box implementation. Journal of Cryptographic Engineering **10**(1), 49–66 (Apr 2020). https://doi.org/10.1007/s13389-019-00207-5 2, 6, 7
18. Goubin, L., Rivain, M., Wang, J.: Defeating state-of-the-art white-box countermeasures. IACR TCHES **2020**(3), 454–482 (2020). https://doi.org/10.13154/tches.v2020.i3.454-482, https://tches.iacr.org/index.php/TCHES/article/view/8597 2, 6
19. Herbst, C., Oswald, E., Mangard, S.: An AES smart card implementation resistant to power analysis attacks. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 06. LNCS, vol. 3989, pp. 239–252. Springer, Heidelberg (Jun 2006). https://doi.org/10.1007/11767480_16 9
20. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27 2, 7, 11
21. Karroumi, M.: Protecting white-box AES with dual ciphers. In: Rhee, K.H., Nyang, D. (eds.) Information Security and Cryptology - ICISC 2010. pp. 278–291. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24209-0_19 2
22. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_25 1, 2
23. Lepoint, T., Rivain, M., De Mulder, Y., Roelse, P., Preneel, B.: Two attacks on a white-box AES implementation. In: Lange, T., Lauter, K., Lisonek, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 265–285. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-43414-7_14 2
24. Rivain, M., Wang, J.: Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. IACR TCHES **2019**(2), 225–255 (2019). https://doi.org/10.13154/tches.v2019.i2.225-255, https://tches.iacr.org/index.php/TCHES/article/view/7391 2

25. Rivain, M., Wang, J.: Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 225–255 (2019). https://doi.org/10.13154/TCHES.V2019.I2.225-255, https://doi.org/10.13154/tches.v2019.i2.225-255 5
26. Sasdrich, P., Moradi, A., Güneysu, T.: White-box cryptography in the gray box - - A hardware implementation and its side channels -. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 185–203. Springer (2016). https://doi.org/10.1007/978-3-662-52993-5_10, https://doi.org/10.1007/978-3-662-52993-5_10 5
27. Seker, O., Eisenbarth, T., Liskiewicz, M.: A white-box masking scheme resisting computational and algebraic attacks. IACR TCHES **2021**(2), 61–105 (2021). https://doi.org/10.46586/tches.v2021.i2.61-105, https://tches.iacr.org/index.php/TCHES/article/view/8788 2, 9, 11, 15
28. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik **13**(4), 354–356 (Aug 1969). https://doi.org/10.1007/BF02165411, https://doi.org/10.1007/BF02165411 3
29. Tang, Y., Gong, Z., Chen, J., Xie, N.: Higher-order DCA attacks on white-box implementations with masking and shuffling countermeasures. IACR TCHES **2023**(1), 369–400 (2023). https://doi.org/10.46586/tches.v2023.i1.369-400 2, 6
30. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.X.: Shuffling against side-channel attacks: A comprehensive study with cautionary note. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 740–757. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4_44 9
31. Xiao, Y., Lai, X.: A secure implementation of white-box AES. In: 2009 2nd International Conference on Computer Science and its Applications. pp. 1–6 (2009). https://doi.org/10.1109/CSA.2009.5404239 2