IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 4, pp. 318–343. DOI:10.46586/tches.v2023.i4.318-343

LPN-based Attacks in the White-box Setting

Alex Charlès[1] and Aleksei Udovenko[2]

[1] University of Luxembourg, Esch-sur-Alzette, Luxembourg, alex.charles@uni.lu
[2] University of Luxembourg, Esch-sur-Alzette, Luxembourg, aleksei.udovenko@uni.lu

Abstract. In white-box cryptography, early protection techniques have fallen to the automated Differential Computation Analysis attack (DCA), leading to new countermeasures and attacks. A standard side-channel countermeasure, Ishai-Sahai-Wagner's masking scheme (ISW, CRYPTO 2003) prevents Differential Computation Analysis but was shown to be vulnerable in the white-box context to the Linear Decoding Analysis attack (LDA). However, recent quadratic and cubic masking schemes by Biryukov-Udovenko (ASIACRYPT 2018) and Seker-Eisenbarth-Liskiewicz (CHES 2021) prevent LDA and force to use its higher-degree generalizations with much higher complexity.

In this work, we study the relationship between the security of these and related schemes to the Learning Parity with Noise (LPN) problem and propose a new automated attack by applying an LPN-solving algorithm to white-box implementations. The attack effectively exploits strong linear approximations of the masking scheme and thus can be seen as a combination of the DCA and LDA techniques. Different from previous attacks, the complexity of this algorithm depends on the approximation error, henceforth allowing new practical attacks on masking schemes which previously resisted automated analysis. We demonstrate it theoretically and experimentally, exposing multiple cases where the LPN-based method significantly outperforms LDA and DCA methods, including their higher-order variants.

This work applies the LPN problem beyond its usual post-quantum cryptography boundary, strengthening its interest for the cryptographic community, while expanding the range of automated attacks by presenting a new direction for breaking masking schemes in the white-box model.

Keywords: White-box Cryptography · Cryptanalysis · LPN · DCA · LDA · Masking · Dummy Shuffling

1 Introduction

The seminal works of Chow, Eisen, Johnson, and van Oorschot [CEJvO02, CEJv03] presented a cryptographic model called white-box, where the attacker has knowledge of the attacked cryptographic primitive and has total access to its implementation. This model reflects the problem raised by Digital Rights Management and Mobile Payment applications, where the attacker can be an untrusted user having direct access to these implementations. The works proposed white-box implementations of the AES [AES01] and the DES [Nat79] block ciphers claiming security against key recovery attacks. Later, these and newer schemes [XL09, Kar11] were broken by a variety of attacks [BGEC04, DRP13, LRD+14].

Recently, Bos, Hubain, Michiels, and Teuwen [BHMT16] proposed a new automated attack called *Differential Computation Analysis* (DCA) that could break most previous white-box countermeasures without any knowledge of the protection scheme being employed. This attack is an application of Differential Power Analysis (DPA) from the side-channel field [KJJ99] to the white-box setting. The ISW masking scheme [ISW03] is one of the main countermeasures against the DPA attack in the side-channel studies, and it is natural

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

to apply it to the white-box implementations. However, in the white-box model, the attacker has the capacity to probe every intermediate value without any noise. This observation led to a new attack called *Linear Decoding Analysis* (LDA) [GPRW20, BU18] that breaks white-box implementations protected by the ISW masking (or, more generally, any linear masking scheme).

Biryukov and Udovenko then proposed a masking scheme employing a quadratic monomial [BU18], hence resisting LDA attacks, while if employed alone being weak to DCA. Seker, Eisenbarth, and Liskiewicz upgraded the scheme (which we call SEL-masking) by allowing it to have an arbitrarily large amount of linear shares, protecting it also from DCA attacks [SEL21]. In addition, they generalized the construction to higher degrees but only provided concrete gadgets for the cubic variant, provably secure against a quadratic generalization of the LDA attack. Furthermore, Biryukov and Udovenko showed in [BU21] that *dummy shuffling* technique also prevents LDA, but has to be combined with the ISW masking to prevent DCA. The scheme introduces copies of the function's implementation (called slots), but only one of them is processing the actual input, while the others process pseudorandom data. The right slot is chosen pseudorandomly depending on the input.

The works [GPRW20, BU21] proposed *Higher Degree Decoding Attack* (HDDA) / higher-degree algebraic attack, that, for a degree $\mathcal{O}$, can break any masking scheme with the decoding function of algebraic degree at most $\mathcal{O}$. In addition, a standard side-channel attack against the ISW masking is the *higher-order DPA*. Its adaptation to the white-box context is called *Higher-Order Differential Computation Analysis* (HODCA), and was studied in [BRVW19, GRW20, TGCX23]. An order-$\mathcal{O}$ HODCA can break any masking scheme with at most $\mathcal{O}$ linear shares, even if it is applied on top of dummy shuffling. The works [GRW20, TGCX23] also showed that when *data-dependency* information from the implementation is available, the higher-order attack variants can be significantly improved. Effectively, the data-dependency information allows to reduce the attacked window size (Subsection 2.4).

For an example, consider a sensitive binary variable s protected with quadratic SEL-masking with 3 linear shares:

$$s = x_1 \oplus x_2 \oplus x_3 \oplus x_4 x_5$$

A white-box implementation employing this scheme and exposing all the shares can be broken using HODCA of order 3 since there are 3 linear shares and their XOR-combination $x_1 \oplus x_2 \oplus x_3$ correlates to the sensitive variables s. It it also susceptible to the degree-2 HDDA, since the degree of this polynomial in $\mathbb{F}_2[x_1, x_2, x_3, x_4, x_5]$ is two. However, HODCA and HDDA have exponential complexity in their order/degree, rapidly becoming infeasible.

Previous works [BU18, BU21] suggested the possibility of applying the Learning Parity with Noise (LPN) techniques to the white-box setting. The idea is that a nonlinear countermeasure can still be approximated by a linear one. In the example above, s is equal to $x_1 \oplus x_2 \oplus x_3$ in 25% of the time (whenever $x_4 x_5 = 0$). The LDA countermeasures (including [SEL21]) already gave lower bounds on the approximation error of their gadgets/circuits by linear functions. However, the application of the LPN method to the white-box setting has not yet been studied, and these bounds so far played an abstract role as it is completely unclear how they relate to the theoretical and practical security of white-box implementations employing these protections.

Our contribution The main contribution of this work is the confirmation, development, and demonstration of the applicability of the LPN attack in practice. We denote our resulting instantiation by WBLPN to distinguish it from the general LPN problem. We analyze and benchmark the attack on several use cases, in theory and in practice, and outline directions for strengthening countermeasures. Our main motivation is to better understand the feasibility of LPN attacks in order to design new countermeasures such as higher-degree masking schemes.

The WBLPN attack can be viewed as a combination of the LDA and DCA attacks. It is most efficient in cases when the masking decoding polynomial can be well *approximated* by a linear function. The complexity of the WBPLPN attack does not directly depend on the number of linear shares (but may do so indirectly through the window size), which is its main advantage over HODCA. Our results show that LPN-based attacks are often practical and pose new threats to countermeasures previously deemed secure against automated attacks such as (HD)DA and (HO)DCA. In particular, counter-intuitively, the generalized masking scheme from [SEL21] gets weaker against the WBLPN attack as its degree grows; the dummy shuffling scheme from [BU21] may be weak against the WBLPN attack when a very small number of dummy slots is used.

1. *(White-box LPN attack)* We show and formalize how various white-box countermeasures can be modeled as an LPN problem, and how algorithms for solving LPN can be applied to break various protections. For illustration and benchmark purposes, we focus on the general SEL-masking family of schemes [SEL21], but also consider the dummy shuffling protection [BU21].

2. *(LPN attack instantiation)* We instantiate the WBLPN attack with the Pooled Gauss algorithm [EKM17], which is highly efficient for small LPN instances arising in practical white-box attacks. In particular, we show how it can be adapted to account for the specifics of the white-box setting.

3. *(Complexity analysis and comparison)* We compare the complexities of the new WBLPN attack against the (HO)DCA and (HD)DA in the cases of SEL-masking schemes and dummy shuffling, both theoretically and experimentally[1]. The comparison clearly demonstrates cases when the WBLPN attack is outperforming alternative techniques.

4. *(Countermeasures)* We discuss potential countermeasures against the white-box LPN attack and show how to amplify the approximation error of a nonlinear masking scheme by composing it with a linear masking scheme.

5. *(Higher-degree WBLPN)* Finally, we discuss a higher-degree generalization of the WBLPN attack and its potential targets.

Outline After explaining the white-box context in Section 2 and presenting the state-of-the-art automated attacks in Section 3, we show in Section 4 how an LPN-problem-solving algorithm can be used to perform an automated attack in the white-box context. We then recall and adapt in Section 5 the Pooled Gauss algorithm [EKM17] to give a concrete LPN-based white-box attack. Section 6 follows up with a detailed performance comparison of attacks. In Section 7, we discuss how to secure countermeasures against the LPN attack. Finally, in Section 8 we describe higher-degree generalization of the LPN attack.

2 Preliminaries

2.1 Notations

Given a vector or a list L of n elements, we denote by $L[i], 1 \leq i \leq n$, the i^{th} element of this list/vector. Given a list of n elements $L = (L[1], \cdots, L[n])$, we denote the concatenation of a new element to this list by $||$, such that $L||a = (L[1], \cdots, L[n], a)$. Similarly, given

[1] The source code is available at https://github.com/cryptolu/whitebox-LPN .

two lists or vectors $L_1 = (L_1[1], \cdots, L_1[n])$ and $L_2 = (L_2[1], \cdots, L_2[m])$, we denote their concatenation by the same operator: $L_1||L_2 = (L_1[1], \cdots, L_1[n], L_2[1], \cdots, L_2[m])$.

Given a pair of non-negative integers n, k, $0 \le k \le n$, we define $\binom{n}{\le k} = \sum_{i=1}^{k} \binom{n}{i}$.

The finite field of size 2 is denoted by $\mathbb{F}_2$. Addition and multiplication in $\mathbb{F}_2$ correspond respectively to the Boolean XOR and AND operations. The n-dimensional vector space over $\mathbb{F}_2$ is denoted by $\mathbb{F}_2^n$. Given two vectors $\vec{v}_1, \vec{v}_2 \in \mathbb{F}_2^n$, we denote the inner product by $\langle \vec{v}_1, \vec{v}_2 \rangle = \sum_{i=1}^{n} (\vec{v}_1[i] + \vec{v}_2[i]) \in \mathbb{F}_2$. For a vector $\vec{v} \in \mathbb{F}_2^n$, we denote its Hamming weight by $\mathrm{HW}(\vec{v}) = \sum_{i=1}^{n} \vec{v}[i] \in \mathbb{N}$. The matrix multiplication constant ω defines the complexity $O(n^\omega)$ of matrix inversion, when n is the dimension of the matrix. For practical dimensions, Strassen's algorithm is the best option and has $\omega \approx 2.8$.

Any Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be uniquely expressed in its *algebraic normal form* (ANF), which is a multivariate polynomial over $\mathbb{F}_2$:

$$f(x) = \sum_{\vec{u} \in \mathbb{F}_2^n} \lambda_{\vec{u}} \vec{x}^{\vec{u}} = \sum_{\vec{u} \in \mathbb{F}_2^n} \lambda_u \prod_{i=1}^{n} x_i^{u_i} \quad \in \quad \mathbb{F}_2[x_1, \ldots, x_n]/(x_1^2 - x_1, \ldots)$$

where $\lambda_{\vec{u}} \in \mathbb{F}_2$ are the coefficients depending on f. By the degree of f we understand the total degree of its ANF:

$$\deg f = \max_{\vec{u} : \lambda_{\vec{u}=1}} HW(\vec{u})$$

The *Walsh* transform $\hat{f} : \mathbb{F}_2^n \to \mathbb{Z}$ of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined by

$$\hat{f}(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle a, x \rangle \oplus f(x)} = |\{x \in \mathbb{F}_2^n \mid \langle a, x \rangle = f(x)\}| - |\{x \in \mathbb{F}_2^n \mid \langle a, x \rangle \neq f(x)\}|$$

It can be computed in time $\mathcal{O}(n2^n)$ using Fast Walsh-Hadamard Transform (FWHT) [FA76].

2.2 Masking schemes

White-box protected implementations come with different protection techniques such as code obfuscation, control flow obfuscation and protection, virtualization, fault countermeasures (for examples see [GPRW20, GRW20, BDG+22, BBD+22]). However, the nonlinear masking schemes [BU18, SEL21] and dummy shuffling [BU21] so far are the only known provable countermeasures against algebraic attacks.

Consider a Boolean circuit implementation of cryptographic primitive. A masking scheme represents all intermediate binary variables by multiple *shares*, and a specific function is used to combine these shares into the original value. For example, in the quadratic masking scheme from [SEL21], an intermediate Boolean variable s can be represented by four or more shares, such as $s = x_1 \oplus x_2 \oplus x_3 x_4$, with all shares except x_1 being generated (pseudo)randomly, and x_1 chosen such that it makes the expression equal to s. Since we will focus on breaking masking schemes in an automated way, we will assume that shares of sensitive values are exposed by the implementation in unknown but fixed locations of the program.

A masking scheme also comes with *gadgets*, which replace gates in the original circuit. For instance, if two sensitive bits s_1 and s_2 are masked with the masking scheme above, the gadget XOR will take as input the 4 shares of s_1 and 4 shares of s_2 and compute shares y_1, y_2, y_3, y_4, such that $y_1 \oplus y_2 \oplus y_3 y_4 = s_1 \oplus s_2$. Gadgets need to satisfy certain security requirements. However, implementations of gadgets are irrelevant to our attacks and are out of the scope of this work. Note that they are relevant in data-dependency attacks [GRW20, TGCX23].

Definition 1. For a given n-share masking scheme, its *masking scheme polynomial* is defined as the corresponding decoding function as a polynomial in $\mathbb{F}_2[x_1, \cdots, x_n]$.

Definition 2. By the *degree of a masking scheme* we call the degree $\deg(P)$ of its masking scheme polynomial $P \in \mathbb{F}_2[x_1, \cdots, x_n]$.

Definition 3. Given a masking scheme polynomial $P \in \mathbb{F}_2[x_1, \cdots, x_n]$, we denote the linear part of P by $\mathrm{lin}(P) \in \mathbb{F}_2[x_1, \cdots, x_n]$ and call it the *linear part of the masking scheme.*

Definition 4. We denote the masking scheme polynomial consisting of ℓ different linear shares and one monomial of degree d that is not divisible by any of the variables of its linear part by $P_{\ell,d}$:

$$P_{\ell,d} = x_1 + \cdots + x_\ell \ + \ x_{\ell+1}x_{\ell+2} \cdots x_{\ell+d} \ \in \ \mathbb{F}_2[x_1, \cdots, x_{\ell+d}]$$

For instance, $P_{2,3}(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3x_4x_5$ has $\deg(P_{2,3}) = 3$ and $\mathrm{lin}(P_{2,3}) = x_1 \oplus x_2$. More generally, $\deg(P_{\ell,d}) = d$ and $\mathrm{lin}(P_{\ell,d})$ contains ℓ variables. The masking scheme polynomials of the ISW scheme can be expressed by $P_{\ell,0}$ for a positive integer ℓ. Similarly, the scheme of [BU18] has masking scheme polynomial $P_{1,2}$. Finally, the generalized SEL-masking scheme includes all $P_{\ell,d}$, $\ell \geq 1, d \geq 2$, although [SEL21] provided concrete gadgets and security proofs only for $d \in \{2, 3\}$.

2.3 Dummy shuffling

Dummy shuffling [BU21] is an alternative protection against the LDA attack. In the side-channel setting (see [HOM06, VMKS12]), shuffling is used to increase measurement noise, strengthening masking schemes. In the white-box setting, shuffling offers strong protection against LDA. Dummy shuffling introduces copies of the function's implementation (called slots), but at each execution only one of them processes the actual input (the right slot), while the others process pseudorandom data (dummy slots). The right slot is chosen pseudorandomly depending on the input. The scheme of [BU21] also manipulates the slots' implementations to ensure the non-degeneracy of dummy slot computations, which is required for provable security against LDA.

2.4 Sliding window in white-box

Given a white-box implementation with employed countermeasures, an attacker generates T random plaintexts and encrypts them using the implementation, while recording all computed intermediate values for each of the T encryptions. These records are called (computational) *traces*. The i^{th} bit of the t^{th} trace, $t \in \{1, \cdots, T\}$, corresponds to the binary value of the i^{th} node of the implementation encrypting the t^{th} plaintext.

Some algorithms such as the one that we will present in Section 5, but also LDA (*c.f.* Subsection 3.3), HDDA (*c.f.* Subsection 3.4) and HODCA (*c.f.* Subsection 3.2) have high complexity in the number of nodes that they process; making them impractical in time for processing full implementations at once. These algorithms are therefore used with a *sliding window method.* For a sliding window of window size $\mathcal{W}$, an attacker considers all the values of the first $\mathcal{W}$ nodes in the T traces files, applies the analysis, then moves the window by a certain number of nodes, and repeats until all nodes are processed. For T traces, the very first window of size $\mathcal{W}$ can be expressed as a matrix as followed:

$$N = \begin{array}{r c} & \begin{array}{ccccc} \text{node } 1 & \text{node } 2 & \cdots & \text{node } \mathcal{W} \end{array} \\ \begin{array}{r} \text{trace } 1 \\ \text{trace } 2 \\ \vdots \\ \text{trace } T \end{array} & \begin{pmatrix} N_{1,1} & N_{1,2} & \dots & N_{1,\mathcal{W}} \\ N_{2,1} & N_{2,2} & \dots & N_{2,\mathcal{W}} \\ \vdots & \vdots & \vdots & \vdots \\ N_{T,1} & N_{T,2} & \dots & N_{T,\mathcal{W}} \end{pmatrix} \end{array} = \left[\overrightarrow{N_1}, \cdots, \overrightarrow{N_{\mathcal{W}}}\right]$$

$N_{i,j}$ corresponds to the bit value of the node j during encryption of the plaintext of the trace i. After the first window is shifted by $\mathcal{S}$ bits, the new window has the first column equal to $\overrightarrow{N_{1+\mathcal{S}}} = (N_{i,1+\mathcal{S}})_{i\in\{1,\cdots,T\}}$.

Definition 5. The n^{th} *node vector*, denoted by $\overrightarrow{N_n} \in \mathbb{F}_2^T$, contains all the values taken by the n^{th} node during the cipher of the T plaintexts.

When data-dependency information in the implementation is available (for example, when the implementation is represented by a Boolean circuit), the sliding window method can be replaced by non-linear and more effective methods of choosing attack windows (see [GRW20, TGCX23]). Our attack is independent of the method of choosing the attacked windows, and we expect the data-dependency-enhanced version of HODCA (*c.f.* Subsection 3.2) to compare to the data-dependency-enhanced version of WBLPN (*c.f.* Subsection 5.2) similarly to how HODCA compares to WBLPN. In this work, we focus on applications of the attacks to a single window.

2.5 Selection function

To determine the secret key hidden within a white-box implementation, the attacks require a *selection function.* This function has the same input as the studied implementation, is parameterized by a small portion of the secret key, and typically outputs an intermediate value used in the reference, unprotected implementation. For instance, in the case of AES, the most widely used selection function is (for example) the first output bit of an S-box of the first round. However, we have to consider all 256 possible values of the involved key byte. This leads to 256 different selection function candidates. Each such function can be computed by XORing of the key byte candidate to the corresponding plaintext byte, applying the S-box to the result, and returning the first output bit.

Since we assume that a masking scheme has been applied to the implementation and in particular to this sensitive bit, we are sure that for at least one *key byte guess* k, there exists a combination of nodes corresponding to the input of the masking scheme polynomial that is equal to the result of the selection function for any chosen plaintext. If there is no such combination of vectors, we can conclude that we made an incorrect key byte guess, reducing the possible key space. We will denote the set of all key byte guesses by $\mathcal{K} = \{1, \dots, |\mathcal{K}|\}$. In the case of the AES, minimally, $|\mathcal{K}| = 16 \cdot 256$, as there are 256 different key byte values per each of the 16 byte positions. This set can be made larger to increase the coverage, by considering all output bits of the S-box and/or their linear combinations.

Definition 6. We denote the selection function applied to a plaintext p, for a key guess $k \in \mathcal{K}$ by $F_k(p)$.

If we denote the list of plaintexts by P, with $P[i]$ being the plaintext used to create the i^{th} trace, then for T traces and the set $\mathcal{K}$ of all key guesses, we can write the matrix of *selection vectors*:

$$S = \begin{array}{c} \\ \text{trace } 1 \\ \text{trace } 2 \\ \vdots \\ \text{trace } T \end{array} \begin{array}{c} \begin{array}{cccc} \text{key guess } 1 & \text{key guess } 2 & \cdots & \text{key guess } |\mathcal{K}| \end{array} \\ \begin{pmatrix} F_1(P[1]) & F_2(P[1]) & \dots & F_{|\mathcal{K}|}(P[1]) \\ F_1(P[2]) & F_2(P[2]) & \dots & F_{|\mathcal{K}|}(P[2]) \\ \vdots & \vdots & \vdots & \vdots \\ F_1(P[T]) & F_2(P[T]) & \dots & F_{|\mathcal{K}|}(P[T]) \end{pmatrix} \end{array} = \left[\overrightarrow{S_1}, \overrightarrow{S_2}, \cdots, \overrightarrow{S_{|\mathcal{K}|}}\right]$$

Definition 7. *Selection vector* $\overrightarrow{S_k}$ is the vector of the output values of the selection function S_k for the T plaintexts, for a given key guess $k \in \mathcal{K}$.

3 Previous Works

In this section we present state-of-the-art of masking-scheme-breaking algorithms, though these algorithms can also break other obfuscation techniques, such as byte/nibble encodings [ABMT18, RW19].

3.1 Differential Computation Analysis (DCA)

Differential Computation Analysis [BHMT16] is an attack introduced by Bos, Hubain, Michiels, and Teuwen, derived from the gray-box Differential Power Analysis attack [KJJ99].

For the sake of the explanation, let us study the AES block cipher protected with a masking scheme polynomial $P_{1,2}$ (*c.f* Definition 4), which transforms a sensitive bit s to $s = x_1 \oplus x_2 x_3$. In the implementation, we know that there exists a node n that corresponds to the share x_1. Since $x_2 x_3 = 0$ three-quarters of the time, the node n will take the value of s also three-quarters of the time.

The problem is that s is unknown since we don't know the hidden key employed to cipher. To find a key byte, we will choose the selection function that reproduces the first round of the AES for a single byte: given a key byte guess $k \in \{0, \cdots, 255\}$, a byte position $b \in \{0, \cdots, 15\}$ and a plaintext, we compute the XOR of k with the b^{th} byte of the plaintext, then apply the AES S-box to return the first bit of the result. This bit is dependent on the key and will help us distinguish a correct key guess from an incorrect one.

In fact, for the correct key byte guess k and the correct byte b, the results of the selection function in the traces correspond to the value of a masked sensitive bit s and therefore will be three-quarters of the time matching the value of the node n corresponding the share x_1 of s. Contrariwise, if the guess is not correct, we can expect it to match the value of the node n one-half of the time on average.

To exploit this distinguisher, we can compute the correlation of all *node vectors* (*c.f.* Subsection 2.4) with all *selection vectors* (*c.f.* Subsection 2.5). For each of the 16 byte positions b, we keep the highest correlation value and its corresponding key byte guess k. With enough traces, we then deduce the key hidden in the algorithm.

This attack breaks masking schemes with masking polynomials with linear parts containing only one monomial. For instance, to apply the masking scheme polynomial $P_{\ell,0}$, the $\ell - 1$ of these shares are generated randomly, and the last one is computed such that it makes the result equal to s. Therefore, none of these shares correlate with s, making this attack ineffective against masking schemes with linear parts containing more than one monomial.

3.2 Higher Order DCA (HODCA)

To allow the DCA to break masking schemes with linear parts containing $\mathcal{O} > 1$ linear shares, a solution would be to create all m-node XOR-combinations for $m \in \{1, \cdots, \mathcal{O}\}$, and compute for all of them their correlation with every selection vector. This attack is called Higher-Order DCA (HODCA) and was studied in the white-box setting in [BRVW19].

Definition 8. The *order* $\mathcal{O}$ of a HODCA attack defines the number of node vectors that are combined together with bitwise XOR[2] before computing their correlation with the selection vectors.

[2] Other combination functions may be used in HODCA, but we restrict to XOR for the purposes of this work.

There exists $\binom{\#N}{\leq \mathcal{O}}$ $\mathcal{O}$-node XOR-combinations. The number of nodes $\#N$ of an algorithm may be quite large, making this algorithm impracticable in time for large $\#N$. To avoid this, a sliding window method with window size $\mathcal{W} \geq \mathcal{O}$ can be used to try to catch the linear part of a sensitive bit s, which limits the amount of created vectors to $\binom{\mathcal{W}}{\leq \mathcal{O}}$.

For instance, if the masking scheme polynomial $P_{2,2}$ is employed, a binary variable s is represented by four shares (x_1, x_2, x_3, x_4) such that $s = x_1 \oplus x_2 \oplus x_3 x_4$. As in the basic DCA attack, we can choose the same selection functions. Then, using a sliding window, we aim to find the shares x_1 and x_2 among other unrelated nodes within a single window. This is achieved by performing the second-order DCA by computing the $\binom{\mathcal{W}}{\leq 2}$ XOR combinations of all pairs of node vectors. One of them will be equal to $\overrightarrow{N_1} \oplus \overrightarrow{N_2}$, the XOR of the node vectors of x_1 and x_2. Since $x_3 x_4 = 0$ three-quarters of the time, the XOR of the node vectors $\overrightarrow{N_1} \oplus \overrightarrow{N_2}$ will also match three-quarters of the time (and thus correlate) with the correct selection vector. Like in the DCA attack, this allows us to recover the corresponding part of the key.

Proposition 1. *HODCA of order $\mathcal{O}$ breaks masking scheme polynomials $P_{\ell,d}$ with $\ell \leq \mathcal{O}$.*

3.3 Linear Decoding Analysis (LDA)

Linear Decoding Analysis attack was proposed in [GPRW20]. Let us consider the masking scheme polynomial $P_{3,0}$ transforming a sensitive bit s to $s = x_a \oplus x_b \oplus x_c$ applied to the AES. As in the DCA attack, we will consider the same selection function and try to recover the key bytes.

We know that for a correct key guess and a correct byte position, there exists a linear combination of 3 node vectors $\overrightarrow{N_a}$, $\overrightarrow{N_b}$, $\overrightarrow{N_c}$ corresponding to the three shares of s equal to the corresponding selection vector $\overrightarrow{S_k}$. If we denote the total number of nodes by $\#N$, to recover this combination among all other unrelated node vectors, we search for $\vec{x}$ such that:

$$N\vec{x} = \left[\overrightarrow{N_1}, \cdots, \overrightarrow{N_a}, \cdots, \overrightarrow{N_b}, \cdots, \overrightarrow{N_c}, \cdots, \overrightarrow{N_{\#N}}\right] \vec{x} = \overrightarrow{S_k}$$

Since $\#N$ can be very huge, we will use a sliding window algorithm to reduce the cost of the algorithm. For each window, we will go through all selection vectors, and try to solve the linear system. If we find a solution, either we succeeded to find one of the 16 key bytes, or it is a false positive and we should increase the total number of traces.

However, this attack only breaks masking schemes of degree at most 1. Indeed, a masking scheme polynomial having a degree superior to one makes the equations non-linear, avoiding using linear algebra to solve them.

3.4 Higher Degree Decoding Analysis (HDDA)

As well as for the HODCA attack, given a window containing $\mathcal{W}$ node vectors, to allow LDA breaking schemes of degree $\mathcal{O} > 1$, we can compute and add to the matrix all $\binom{\mathcal{W}}{\leq \mathcal{O}}$ AND-combinations of node vectors.

Definition 9. The *degree* $\mathcal{O}$ of an HDDA attack defines the number of node vectors that are combined with bitwise AND before solving the LDA matrix equation for every selection vector.

If we take the same masking scheme polynomial example for HODCA, $P_{2,2}$ transforms a bit variable s into four shares, such as $s = x_1 \oplus x_2 \oplus x_3 x_4$. Taking the same selection function as for LDA, we can catch within the same window all of its shares x_1, x_2, x_3, and x_4 among other unrelated nodes. Then, we can compute the $\binom{\mathcal{W}}{2}$ AND-combinations of all pairs of node vectors and concatenate them to the $T \times \mathcal{W}$ matrix.

One of these newly added vectors will be $\overrightarrow{N_3} \wedge \overrightarrow{N_4}$, the AND of the node vectors of x_3 and x_4. Therefore, for the selection vector $\overrightarrow{S_k}$ corresponding to the correct key guess k, we can find the solution $\overrightarrow{N_1} \oplus \overrightarrow{N_2} \oplus (\overrightarrow{N_3} \wedge \overrightarrow{N_4}) = \overrightarrow{S_k}$ using the same linear algebra as the regular LDA attack, which indicates that the chosen k may be a correct key guess.

Proposition 2. *HDDA of degree $\mathcal{O}$ breaks masking schemes of degree at most $\mathcal{O}$, including all masking scheme polynomials $P_{\ell,d}$ with $d \leq \mathcal{O}$.*

4 Security of White-box Countermeasures as an LPN Problem

In this section, we will first present in Subsection 4.1 the Learning Parity with Noise problem in the general case, as well as its expression in matrix form when asking the oracle for multiple queries. Then, after defining the *noise rate* of a masking scheme polynomial, we explain in Subsection 4.2 that searching for combinations of node vectors for a masking scheme polynomial of noise rate τ in a window of size $\mathcal{W}$ can be related to an LPN instance of dimension $\mathcal{W}$ and noise rate τ.

4.1 Learning parity with noise problem

Definition 10. The $LPN_{k,\tau}$ problem comes with two parameters: the dimension k and the noise rate τ, $0 \leq \tau < 1/2$. The goal is to retrieve a secret $\vec{x} \in \mathbb{F}_2^k$ generated uniformly at random, by having access to an oracle returning $\vec{a} \in \mathbb{F}_2^k$ and $b \in \mathbb{F}_2$, such that:

$$\langle \vec{a}, \vec{x} \rangle + e = \begin{bmatrix} a_1 \cdots a_k \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix}$$

where $\vec{a} \in \mathbb{F}_2^k$ is sampled uniformly at random on each query, and $e \in \mathbb{F}_2$ is sampled at random with $\Pr[e = 1] = \tau$ on each query.

After T queries, we can collect the sampled vectors $\vec{a}$ in a matrix A, the sampled errors e in a vector $\vec{e}$, and the results b in a vector $\vec{b}$. Since the searched solution vector $\vec{x}$ does not change, we obtain a matrix-vector formulation of the problem:

$$A\vec{x} + \vec{e} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,k} \\ \vdots & \vdots & \vdots \\ a_{T,1} & \cdots & a_{T,k} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} + \begin{bmatrix} e_1 \\ \vdots \\ e_T \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_T \end{bmatrix}$$

where (in the standard LPN problem) $\Pr[a_{i,j} = 1] = 1/2$ for all i, j, and $\Pr[e_i = 1] = \tau$ for all i.

We write $(A, \vec{b}) \leftarrow LPN_{k,\tau}^T$ to denote the matrix-vector representation of T samples.

4.2 Relation between LPN and White-box Countermeasures

Case of masking schemes Let us suppose that we have a window containing (among other unrelated nodes) all the shares of the masked sensitive bit s corresponding to the resulting bit of a selection function F (c.f. Subsection 2.5). Assume that the masking scheme polynomial $P \in \mathbb{F}_2[x_1, \cdots, x_n]$ is known. The goal is to find if there exists a combination $(\overrightarrow{N_1}, \cdots, \overrightarrow{N_n})$ of n different node vectors among the $\mathcal{W}$ available in the window and a selection vector $\overrightarrow{S_k}$ such that $P(\overrightarrow{N_1}[t], \cdots, \overrightarrow{N_n}[t]) = F_k(t) = \overrightarrow{S_k}[t]$ for all $t \in \{1, \cdots, T\}$.

The key idea to conceive the relation between this problem and LPN is to understand that we can consider the non-linear monomials of the masking scheme polynomial as noise.

Definition 11. We denote the *noise rate* of a polynomial $P \in \mathbb{F}_2[x_1, \cdots, x_n]$ by $\tau = \mathbb{P}(\bigoplus_{m \in \mathcal{M}} m(x) = 1)$, with $\mathcal{M}$ being the set of monomials in P of degree strictly greater than 1. If $\mathcal{M}$ is empty, then $\tau = 0$.

Remark 1. This definition suffices for the masking polynomials $P_{\ell,d}$. More generally, the noise rate of a polynomial P could be defined by the error of its best linear approximation, which in turn can be computed from the maximum absolute value of the Walsh transform $\hat{P}$ of P (computed in time $\mathcal{O}(n2^n)$): $\tau = 1/2 - 2^{-n-1} \cdot \max_{a \in \mathbb{F}_2^n, a \neq 0} |\hat{P}(a)|$.

Proposition 3. *For any $\ell \geq 1, d \geq 2$, the scheme polynomial $P_{\ell,d}$ (c.f Definition 4) has noise rate $\tau = \frac{1}{2^d}$.*

For a hypothetical masking scheme with the masking scheme polynomial

$$P(x_1, x_2, x_3, x_4, x_5, x_6) = x_1 \oplus x_2x_3 \oplus x_4x_5x_6$$

we would have noise rate

$$\begin{aligned} \tau &= \mathbb{P}(x_2x_3 \oplus x_4x_5x_6 = 1) \\ &= \mathbb{P}(x_2x_3 = 1)\mathbb{P}(x_4x_5x_6 = 0) + \mathbb{P}(x_2x_3 = 0)\mathbb{P}(x_4x_5x_6 = 1) = 5/16 \end{aligned}$$

In general, if all monomials of degree superior to two are uncorrelated, one can use the piling-up lemma [Mat94] to compute the noise rate of a given polynomial. For any masking scheme, it is always possible to compute the probability of the nonlinear part to be equal to zero from the full truth table, either by direct computations or by the FWHT algorithm.

So, given a n-bit input chosen uniformly at random in $\mathbb{F}_2^n$ and an n-share masking scheme polynomial P, there is a probability $1 - \tau$ that higher-degree monomials can be ignored in the computation, meaning that the computation is effectively linear. In this case, given $\text{lin}(P)$, the polynomial consisting of all monomials of P of degree 1, among all $x \in \mathbb{F}_2^n$ there is a probability $1 - \tau$ that $P(x) = \text{lin}(P)(x)$.

Case of dummy shuffling Consider an implementation protected by the linear ISW masking with ℓ shares and dummy shuffling with t dummy slots (in addition to the right slot). The linear combination of all the ℓ shares inside one slot correlates to the protected sensitive value, since it is precisely equal to the sensitive value when the slot is right (probability $\frac{1}{t+1}$) and equal to a random unrelated value otherwise (probability $\frac{t}{t+1}$). If the sum of the shares is balanced when the slot is dummy (i.e., is equal to 1 with probability 1/2), then we get the noise rate $\tau = \frac{t}{t+1} \cdot \frac{1}{2}$. However, the construction of [BU21] allows intermediate functions with weight 1/4. Such functions have an error of only 1/4 when the selection function is equal to 0. Therefore, the noise rate τ can in principle drop to $\tau = \frac{t}{t+1} \cdot \frac{1}{4}$, which is the lower bound proven in [BU21]. Thus, the actual value of τ can lie in the range $\left[\frac{t}{t+1} \cdot \frac{1}{4}, \frac{t}{t+1} \cdot \frac{1}{2}\right]$, depending on the protected circuit and details of the linear masking. From the designer's perspective, the pessimistic value $\tau = \frac{t}{t+1} \cdot \frac{1}{4}$ should be studied. From the attacker's perspective, this value might not always be reachable and so $\tau = \frac{t}{t+1} \cdot \frac{1}{2}$ has to be used to ensure the success of the attack.

Modeling the problem by LPN We have seen that in the LPN problem, one can ask for any number of noisy linear equations. This will correspond in the white-box context to creating as many traces as needed. We will denote this number of traces / noisy linear equations by T. Similarly, we have also seen that the study in the white-box context will be performed within a sliding window. This almost (*c.f.* Subsection 5.2) corresponds to the dimension of the LPN problem. We will denote the window size by $\mathcal{W}$.

Denoting the trace matrix part in the window by M, we can write the problem as follows:

$$M\,\vec{x} + \vec{e} = \begin{bmatrix} \overrightarrow{N_1}[1], & \cdots & \overrightarrow{N_{\mathcal{W}}}[1] \\ \vdots & \vdots & \vdots \\ \overrightarrow{N_1}[T], & \cdots & \overrightarrow{N_{\mathcal{W}}}[T] \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{\mathcal{W}} \end{bmatrix} + \begin{bmatrix} e_1 \\ \vdots \\ e_T \end{bmatrix} = \overrightarrow{S_k}$$
$$\text{with } \Pr[e_i = 1] = \tau \text{ for all } i \in \{1, \cdots, T\}$$

For a key byte guess $k \in \mathcal{K}$, we are searching for the best linear combination of the columns of the sliding window matrix M, represented by the vector $\vec{x}$, such that we minimize the *Hamming Weight* (HW) of $\vec{e} = M\overrightarrow{x} + \overrightarrow{S_k}$. Searching for such a linear combination is equivalent to solving LPN, as we want to find $\vec{x}$ while not knowing $\vec{x}$ and $\vec{e}$, the latter being the error vector caused by the noise of the masking scheme polynomial's approximation.

With the Law of Large Numbers, if all the linear shares are contained within this window, there exists, for enough traces and a correct key byte guess k, $\vec{x}$ such that $\mathrm{HW}(M\vec{x} + \overrightarrow{S_k}) \to \tau T$ when the number of traces $T \to +\infty$. Conversely, if at least one of the linear shares is not contained in the window or for a wrong key byte guess, we can expect $\mathrm{HW}(M\vec{x} + \overrightarrow{S_k}) \to T/2$ for any $\vec{x}$ and k when the number of traces $T \to +\infty$. For a constant $\tau < 1/2$, these two distributions can be distinguished with a sufficient number T of traces.

5 Adapting the LPN-solving Pooled Gauss Algorithm

In this section, we instantiate the LPN attack using a basic Pooled Gauss algorithm (generally considered as folklore knowledge and analyzed in [EKM17]), which can also be considered as the LPN version of the Information Set Decoding (ISD) algorithm by Prange [Pra62]. We present the modifications brought to the Pooled Gauss algorithm to make it work efficiently in the white-box context. Although more powerful advanced LPN algorithms exist, they may introduce unnecessary complexity and are unlikely to provide significant speedups at small-size instances occurring in the white-box setting. Consideration of these algorithms is left as future work.

5.1 Description of Pooled Gauss for pure LPN

Pooled Gauss is a folklore algorithm studied more formally in [EKM17]. For an LPN of dimension $\mathcal{W}$ and noise rate τ, the idea is to first ask the LPN oracle for $O(\mathcal{W}^2)$ noisy linear equations and store them in a matrix $M_{\texttt{pool}}$ and a vector $\vec{b}_{\texttt{pool}}$. From $M_{\texttt{pool}}$, we can choose randomly a *sample* of $\mathcal{W}$ equations, giving us a sub-matrix M_s of size $\mathcal{W} \times \mathcal{W}$ and a vector $\vec{b}_s$ corresponding to the chosen *sample*. If M_s is not invertible, then we sample another one until we find an invertible one. We have $(1-\tau)^{\mathcal{W}}$ chance that all of these $\mathcal{W}$ equations have the noise equal to zero. When this happens, using Gaussian elimination, we can find the solution $\vec{x}$ to the system $M_s\vec{x} = \vec{b}_p$.

We can now ask the LPN oracle for m new noisy linear equations that we store in a $m \times \mathcal{W}$ matrix M_v and a vector $\vec{b}_v$. Then we verify that $\mathrm{HW}(M_v\vec{x} + \vec{b}_v)$ is approximately equal to τm, rather than to $m/2$. If this holds, we consider that we solved the LPN instance; otherwise, at least one of our noisy equations of our selected *sample* might have a noise equal to one. In that case, we sample another couple of matrices and vectors from the same pool, and verify it with newly generated M_v and $\vec{b}_v$, until we find a solution to the LPN instance.

More precisely, rather than verifying that $\mathrm{HW}(M_v\vec{x}+\vec{b}_v)$ is approximately equal to τT, the authors of [EKM17] propose two parameters α and β, which correspond respectively to the probability of rejecting the right solution and the probability of accepting an incorrect solution. These parameters are used to determine a threshold c and a value m which are used to determine if we accept or not a solution:

$$m := \left(\frac{\sqrt{\frac{3}{2}\ln(\frac{1}{\alpha})} + \sqrt{\ln(\frac{1}{\beta})}}{\frac{1}{2}-\tau}\right)^2 \tag{1}$$

$$c := \tau m + \sqrt{3\left(\frac{1}{2}-\tau\right)\ln\left(\frac{1}{\alpha}\right)m}$$

After finding a solution $\vec{x}$ and generating M_v and $\vec{b}_v$, we will use the threshold parameter c to verify if $HW(M_v\vec{x}+\vec{b}_v) \le c$. If so, we return $\vec{x}$ as the solution, otherwise, we choose another *sample* and redo the previous steps. The procedure is summarised in Algorithm 1.

Algorithm 1 PooledGauss

```
Input: $\tau$, $\mathcal{W}$, $\alpha = \frac{1}{2^{\mathcal{W}}}$, $\beta = \left(\frac{1-\tau}{2}\right)^{\mathcal{W}}$, and an access to an $\mathrm{LPN}_{\mathcal{W},\tau}$ oracle
Output: $\vec{x}$, solution of the LPN instance
$m := \left(\frac{\sqrt{\frac{3}{2}\ln(\frac{1}{\alpha})}+\sqrt{\ln(\frac{1}{\beta})}}{\frac{1}{2}-\tau}\right)^2$
$c := \tau m + \sqrt{3\left(\frac{1}{2}-\tau\right)\ln\left(\frac{1}{\alpha}\right)m}$
$(M_{\mathtt{pool}}, \vec{b}_{\mathtt{pool}}) \leftarrow LPN_{\mathcal{W},\tau}^{\mathcal{W}^2}$
do
    do
        $M_s \leftarrow \mathcal{W}$ different randomly-chosen rows of $M_{\mathtt{pool}}$
    while $M_s \notin GL_{\mathcal{W}}(\mathbb{F}_2)$
    $\vec{b}_s \leftarrow$ the $\mathcal{W}$ elements of $\vec{b}_{\mathtt{pool}}$ corresponding to the randomly-chosen rows of $M_{\mathtt{pool}}$
    $\vec{x} := M_s^{-1}\vec{b}_s$
    $(M_v, \vec{b}_v) \leftarrow LPN_{\mathcal{W},\tau}^{m}$
while $\mathrm{HW}(M_v\vec{x}+\vec{b}_v) > c$
return $\vec{x}$
```

5.2 Adaptation of Pooled Gauss to white-box cryptography

Avoiding XOR of nodes To adapt Pooled Gauss to the white-box model, we first need to reduce the input window to the subset of the columns that span the column space and are linearly independent. The analyzed white-box implementation may contain some XOR gates, which implies that some nodes are the sum over $\mathbb{F}_2$ of two closely-located nodes. This generates a lot of windows where some of the columns are linearly dependent, which prevents finding a *sample* of rows that forms an invertible matrix. We will denote the number of linearly independent columns of a window of size $\mathcal{W}$ by $\mathcal{W}'$, which corresponds to the effective dimension of the LPN problem.

Adding a stop condition In the standard LPN problem, it is guaranteed that there exists a solution, whereas, in our white-box problem, it is likely that a given window does not contain all the linear shares of the sensitive bit of a selection function. Therefore, we need to determine the number of attempts $\mathcal{A}$ to find a solution in a *sample* having an invertible matrix.

Definition 12. We denote the probability parameter of finding at least one **n**oise-**f**ree **m**atrix after $\mathcal{A}$ iterations by p_{nfm}.

Since the probability of having a noise-free sampled matrix is $(1-\tau)^{\mathcal{W}}$, the probability of finding at least one noise-free matrix after $\mathcal{A}$ attempts is $p_{\mathrm{nfm}} = 1 - (1 - (1-\tau)^{\mathcal{W}})^{\mathcal{A}}$. Knowing p_{nfm}, we can determine

$$\mathcal{A} = \frac{\ln(1 - p_{\mathrm{nfm}})}{\ln(1 - (1-\tau)^{\mathcal{W}})} \approx \frac{-\ln(1 - p_{\mathrm{nfm}})}{(1-\tau)^{\mathcal{W}}} \tag{2}$$

(using $\ln(1+x) = x + O(x^2)$), the number of attempts of trying to find a solution $\vec{x}$ corresponding to an invertible sampled matrix M_s necessary to achieve p_{nfm} chance of having at least one of these sample matrices being noise-free.

Avoiding false-positives For the same given window, we need to solve Pooled Gauss not once, but for every key guess of a set $\mathcal{K}$, which has an impact on the threshold c. Indeed, since we will repeat Pooled Gauss on the same window $|\mathcal{K}|$ times, it increases the probability of finding a false positive. β being the probability of accepting an incorrect solution should be adapted such that after $|\mathcal{K}|$ iterations of Pooled Gauss it is equal to the value that it has in the original algorithm. Therefore, we have the same computation (c.f. Equation 1) for the threshold c, and the same probability of rejecting the correct solution α, while having β changed from $\beta = \frac{1}{\mathcal{A}} \cdot \frac{1}{2^{\mathcal{W}}} = \left(\frac{1-\tau}{2}\right)^{\mathcal{W}}$ (in [EKM17]) to $\beta = \frac{1}{\mathcal{A}\cdot|\mathcal{K}|} \cdot \frac{1}{2^{\mathcal{W}}} = \frac{1}{|\mathcal{K}|} \cdot \left(\frac{1-\tau}{2}\right)^{\mathcal{W}}$.

Determining the number of traces We denote the number of traces of the pool used for the *samples* generation by T_p and the number of traces needed to verify a solution of a sample by m; such that $T = T_p + m$. The separation is done for the theoretic independence assumption. In practice, the samples can be drawn from all the T traces; it is only important to exclude the initial sample from the verification step.

For the latter, we need to create traces to allow the verification of our found solution $\vec{x}$ following Pooled Gauss algorithm with our modified β (*c.f.* Subsection 5.2). This value is given by m, following the computation given in Equation 1, but for $\beta = \frac{1}{|\mathcal{K}|} \cdot \left(\frac{1-\tau}{2}\right)^{\mathcal{W}}$.

For the former, we need to choose several traces that avoid the collision of samples and ensure not too much noise. It is important to say that this parameter does not have any impact on the time complexity (*c.f* Subsection 5.5), but only on memory usage, which is not significant for this algorithm. Therefore, by choosing for instance $T_p = 5\mathcal{W}$, we can ensure that the number of rows that have noise would be approximately τT_p. Similarly, since we can create $\binom{T_p}{\mathcal{W}}$ different combinations of $\mathcal{W}$ rows from T_p available ones, by choosing T_p big enough to avoid having too much noise by chance, we already lower enough the probability of finding a collision.

In practice, one can choose $T = m + \mathcal{W}$, and sampling the matrix M_s from the T traces, and use the $T - \mathcal{W}$ remaining one to verify the found solution.

5.3 Optimizations with linear algebra

Since we perform Pooled Gauss for each of the key possibilities of the key guesses set $\mathcal{K}$, instead of finding $\mathcal{A}$ invertible matrices from different samples for each of the $|\mathcal{K}|$ key guesses, we can use the same invertible matrix for each of the key guesses. Since the same invertible matrix is used for the $|\mathcal{K}|$ key guesses, we can compute its inverse once and use it for all the remaining computations.

Let us denote the $\mathcal{W} \times \mathcal{W}$ sampled matrix from the $T_p \times \mathcal{W}$ array containing traces for sampling $M_{\texttt{pool}}$ by M_s, and the vector of the $\mathcal{W}$ respective entries of $S_{\texttt{pool}}$ by $\overrightarrow{S_k^s}$, for a given key guess $k \in \mathcal{K}$. Likewise, let us denote the $m \times \mathcal{W}$ verification matrix consisting of the m dedicated traces by M_v, and its corresponding selection vector by $\overrightarrow{S_k^v}$. Let us denote the matrix containing all selection vectors for sampling (resp. verifying) by S_s (resp. S_v). Finally, let $\vec{x}$ be the searched linear combination of the matrix that minimizes the Hamming weight of the error vector $\vec{e} = \vec{e_s} || \vec{e_v}$, where the Pooled Gauss algorithm aims for $\vec{e_s} = 0$. Then, we have:

$$\begin{bmatrix} \vec{e_s} \overset{?}{=} 0 \\ \text{——} \\ \vec{e_v} \end{bmatrix} = \begin{bmatrix} M_s \\ \text{—} \\ M_v \end{bmatrix} \begin{bmatrix} \vec{x} \end{bmatrix} + \begin{bmatrix} \overrightarrow{S_k^s} \\ \text{—} \\ \overrightarrow{S_k^v} \end{bmatrix} = \begin{bmatrix} M_s \\ \text{—} \\ M_v \end{bmatrix} \begin{bmatrix} M_s^{-1} \end{bmatrix} \begin{bmatrix} M_s \end{bmatrix} \begin{bmatrix} \vec{x} \end{bmatrix} + \begin{bmatrix} \overrightarrow{S_k^s} \\ \text{—} \\ \overrightarrow{S_k^v} \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{Id}_{\mathcal{W}} \\ \text{————} \\ M_v M_s^{-1} \end{bmatrix} \begin{bmatrix} \overrightarrow{M_s x} \end{bmatrix} + \begin{bmatrix} \overrightarrow{S_k^s} \\ \text{—} \\ \overrightarrow{S_k^v} \end{bmatrix}$$

We now are searching for $\overrightarrow{M_s x}$ minimizing the total weight. Because the upper part of the first matrix became the identity, then $\overrightarrow{M_s x}$ has to be equal to $\overrightarrow{S_k^s}$ (under the hypothesis $\vec{e_s} = 0$). Therefore, the total weight that we want to minimize depends only on $\vec{e_v} = \begin{bmatrix} M_v M_s^{-1} \end{bmatrix} \overrightarrow{S_k^s} + \overrightarrow{S_k^v}$, and has to be smaller than the threshold c defined in Equation 1.

Since we will have to find an $\vec{x}$ for each of the key possibilities of the set $\mathcal{K}$, we can write the problem as:

$$\begin{bmatrix} M_s \\ \text{—} \\ M_v \end{bmatrix} \begin{bmatrix} \overrightarrow{x_1} \cdots \overrightarrow{x_{|\mathcal{K}|}} \end{bmatrix} + \begin{bmatrix} \overrightarrow{S_1^s} & \cdots & \overrightarrow{S_{|\mathcal{K}|}^s} \\ \text{—} & \cdots & \text{—} \\ \overrightarrow{S_1^v} & \cdots & \overrightarrow{S_{|\mathcal{K}|}^v} \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{Id}_{\mathcal{W}} \\ \text{————} \\ M_v M_s^{-1} \end{bmatrix} \begin{bmatrix} \overrightarrow{M_s x_1} \cdots \overrightarrow{M_s x_{|\mathcal{K}|}} \end{bmatrix} + \begin{bmatrix} \overrightarrow{S_1^s} & \cdots & \overrightarrow{S_{|\mathcal{K}|}^s} \\ \text{—} & \cdots & \text{—} \\ \overrightarrow{S_1^v} & \cdots & \overrightarrow{S_{|\mathcal{K}|}^v} \end{bmatrix}$$

For the same reason, we can fix $\overrightarrow{M_s x_i}$ to $\overrightarrow{S_i^v}$, for $i \in \{1, \cdots, |\mathcal{K}|\}$, which gives us:

$$M_v M_s^{-1} \begin{bmatrix} \overrightarrow{S_1^s} \cdots \overrightarrow{S_{|\mathcal{K}|}^s} \end{bmatrix} + \begin{bmatrix} \overrightarrow{S_1^v} \cdots \overrightarrow{S_{|\mathcal{K}|}^v} \end{bmatrix} = M_v M_s^{-1} S_s + S_v = E$$

If one of the columns of the resulting $m \times |\mathcal{K}|$ matrix E has Hamming weight less than the threshold c, then its corresponding key guess becomes a good candidate. If none of the columns satisfy this inequality, then we might have chosen a noisy sample, or at least one share of the researched sensitive bit is not contained in the current window. The probability of the former should be negligible according to the chosen number of traces.

5.4 Resulting algorithm

Algorithm 2 presents WBLPN algorithm for a single window. This algorithm needs to have for input the different probabilistic parameters to ensure its success m, c, T_v and $\mathcal{A}$, as well as two arrays M and S containing respectively the traces and the selection vectors.

The algorithm begins by separating the traces and selection vector values in two groups of length T_p and m: respectively one for sampling, the other for verifying solutions found. Thereafter, it will iteratively create a $\mathcal{W} \times \mathcal{W}$ matrix M_s from the traces for sampling until M_s is invertible. Once an invertible sampled matrix M_s is found, the algorithm gets their corresponding selection vectors in the form of $\mathcal{W} \times |K|$ matrix S_s. The algorithm now computes the matrix E following Subsection 5.3 to observe if one of the resulting columns of E has a Hamming weight less than c. If so, the key guess corresponding to the same column in S might be a correct guess.

Algorithm 2 WBLPN

Inputs:

- window size $\mathcal{W}$
- expected maximum noise rate τ
- number of key guess possibilities $|\mathcal{K}|$
- number of traces used to sample m (*c.f.* Equation 1)
- threshold value c (*c.f.* Equation 1)
- number of traces used to verify a solution T_v
- required number of attempts $\mathcal{A}$ to achieve p_{nfm} chance of finding a solution
- $(m + T_p) \times \mathcal{W}$ array M containing the traces, with columns linearly independent
- $(m + T_p) \times |\mathcal{K}|$ array S containing their corresponding selection vectors

Output: A value of $k \in \{1, \cdots, |\mathcal{K}|\}$ such that the k^{th} column of S is a solution vector corresponding to a correct key guess, if it exists.

```
$M_{\texttt{pool}} \leftarrow$ the first $T_p$ traces values of $M$
$S_{\texttt{pool}} \leftarrow$ the first $T_p$ traces values of $S$
$M_v \leftarrow$ the last $m$ traces values of $M$
$S_v \leftarrow$ the last $m$ traces values of $S$
for $i \in \{1, \cdots, \mathcal{A}\}$ do
    do
        $M_s \leftarrow \mathcal{W}$ different randomly-chosen rows of $M_{\text{pool}}$
    while $M_s \notin GL_{\mathcal{W}}(\mathbb{F}_2)$
    $S_s \leftarrow$ the same subset of $\mathcal{W}$ rows constituting $M_s$ from $S_{\texttt{pool}}$
    $E \leftarrow M_v M_s^{-1} S_s + S_v$
    for $j \in \{1, \cdots, |\mathcal{K}|\}$ do
        if the Hamming Weight of the $j^{th}$ column of $E$ is $\leq c$ then
            return the $j^{th}$ element of $\mathcal{K}$
        end if
    end for
end for
```

It is very important to highlight that the array M should contain linearly independent columns to allow the algorithm to create invertible matrix samples M_s, as explained in Subsection 5.2. If this requirement is not met, the algorithm will not finish. In order to make WBLPN work, we need to remove linearly dependent columns. This implies that the dimension of the LPN problem may decrease below the expected dimension $\mathcal{W}$. Therefore, in order to save time in practice, all parameters m, c, T_v and $\mathcal{A}$ should be precomputed for all window sizes $w \leq \mathcal{W}$, allowing to reduce the computation time of the algorithm if some XOR node vectors are present inside a given window..

Remark 2. In the real case, the noise rate of a masking scheme polynomial may not be known. Fortunately, executing the algorithm for a noise rate τ ensures that it will work for any noise rate $\tau' \leq \tau$. Similarly, the attack does not require the knowledge of the actual masking scheme or countermeasure used and will succeed in all cases with the approximation error below τ.

5.5 Time complexity for a single window

First, we recall the complexity analysis of PooledGauss for LPN from [EKM17]. They set $\alpha = 1/2^{\mathcal{W}}, \beta = \frac{1}{\mathcal{A}} \cdot \frac{1}{2^{\mathcal{W}}} = \left(\frac{1-\tau}{2}\right)^{\mathcal{W}}$, so that $m = O\left(\frac{-\mathcal{W}\ln(1-\tau)}{(1/2-\tau)^2}\right)$ traces are required to ensure strong enough filtering of wrong solutions. Since we need to test a factor of $|\mathcal{K}|$ times more candidate solutions (most of which are wrong), we need to set $\beta = \frac{1}{\mathcal{A}\cdot|\mathcal{K}|} \cdot \frac{1}{2^{\mathcal{W}}}$. Note that the factor $\frac{1}{2^{\mathcal{W}}}$ was added in [EKM17] to ensure that the probability of failure decreases exponentially with $\mathcal{W}$, while it in fact suffices to fix a constant failure probability (e.g. 2^{-64}), which although does not change the asymptotic complexity.

Recall that the algorithm runs for $\mathcal{A}$ iterations of sampling a square submatrix of the trace matrix, with $\mathcal{A} = O((1-\tau)^{-\mathcal{W}})$ (the hidden constant depends on the desired success rate). In each iteration, the following main steps are performed:

1. Computation of $M_s^{-1} \in \mathbb{F}_2^{\mathcal{W}\times\mathcal{W}}$, with complexity $O(\mathcal{W}^{\omega})$, where ω is the matrix multiplication constant.

2. Computation of $M_v \times M_s^{-1}$, where M_v and the resulting matrix have sizes $m \times \mathcal{W}$. Since $m > \mathcal{W}$, this can be done in at most $\left\lceil \frac{m}{\mathcal{W}} \right\rceil$ matrix multiplications of size $\mathcal{W}\times\mathcal{W}$. The complexity is thus $O(\mathcal{W}^{\omega-1}m) = O\left(\mathcal{W}^{\omega} \cdot \frac{-\ln(1-\tau)}{(1/2-\tau)^2}\right)$.

3. Computation of $(M_v \times M_s^{-1}) \times S_s$, which is the product of $m \times \mathcal{W}$ and $\mathcal{W} \times |\mathcal{K}|$ matrices. Since $m > \mathcal{W}$ and assuming $|\mathcal{K}| > \mathcal{W}$, the multiplication can be done in $\mathcal{W}\times\mathcal{W}$ blocks in time $O\left(\mathcal{W}^{\omega} \cdot \frac{m}{\mathcal{W}} \cdot \frac{|\mathcal{K}|}{\mathcal{W}}\right) = O\left(\mathcal{W}^{\omega-1} \cdot |\mathcal{K}| \cdot \frac{-\ln(1-\tau)}{(1/2-\tau)^2}\right)$.

   Note that without the optimization from Subsection 5.3, this operation would essentially be expressed as $|\mathcal{K}|$ matrix-vector products costing $O(m \cdot \mathcal{W} \cdot |\mathcal{K}|) = O\left(\mathcal{W}^2 \cdot |\mathcal{K}| \cdot \frac{-\ln(1-\tau)}{(1/2-\tau)^2}\right)$. Therefore, the optimization saves the factor $\mathcal{W}^{3-\omega}$. For example, using practical Strassen's algorithm, the savings factor is about $\mathcal{W}^{0.2}$.

The other operations such as the addition of $m \times |\mathcal{K}|$ matrices and scanning for low-weight columns are negligible compared to the operations outlined above. Assuming $|K| > \mathcal{W}$, the complexity is dominated by the third step.

We conclude with the final time complexity $O\left(\frac{\mathcal{W}^{\omega-1}}{(1-\tau)^{\mathcal{W}}} \cdot |\mathcal{K}| \cdot \frac{-\ln(1-\tau)}{(1/2-\tau)^2}\right)$. Note that the last term is upper bounded by 4.6 for $\tau \leq 1/4$ and so can be ignored for low error rates (this covers the cases of our interest, as higher error rates are too expensive to solve in a practical time).

6 Time Comparison of WBLPN with HODCA and HDDA

In this section, we compare our implementations of Pooled Gauss for white-box cryptography, HODCA, and HDDA. We first compare WBLPN with HDDA, as they both do not depend on the linear part of the masking scheme polynomial and show that WBLPN is more interesting for breaking $P_{\ell,d}$ with $d \geq 3$. Thereafter, we compare WBLPN with HODCA. As the complexity of WBLPN is dependent on the non-linear part and the complexity of HODCA on the linear part, we cannot do the same comparison as for HDDA, but we show that WBLPN proposes competitive times compared to HODCA for reasonable window sizes.

The time measurements of Figure 1, Figure 2 and Figure 3 are recorded for $|\mathcal{K}| = 4096$, as it is the usual number of key guesses for the AES. We set $p_{\text{nfm}} = 0.999$ (*c.f.* Subsection 5.2). Each time measurement is an average of 20 iterations. For each of these iterations, we generated a fresh window uniformly at random and then recorded the time of running a given attack, which puts WBLPN in its worst-case scenario.

Indeed, as mentioned in Subsection 5.2, many nodes are the result of the XOR of two other nodes that can be contained within the same window. Since WBLPN gets rid of these node vectors, it decreases the dimension of the LPN problem, and with it its overall computation time. With randomly-generated traces, it is unlikely that a whole column of a trace is the linear combination of other columns. Therefore in these tests, WBLPN will most often work with an LPN dimension equal to the window size.

Furthermore, we are only using the guaranteed noise rate defined by the masking expression, while for defensive purposes we could pessimistically use the proven lower bounds for analysis (which may or may not happen in practice, and depend on the gadgets and their configuration), and potentially get better attacks.

All the computations are done on a 12th Gen Intel(R) Core i7-1265U 1.80 GHz CPU, with 32 GB of RAM on WSL 2 running Ubuntu 22.04 on Windows 10. All of these three implementations[3] are done in SageMath [Sag22] and are not employing any specific optimizations.

The implementation of HODCA might be slower in SageMath than HDDA and WBLPN, due to the optimized linear algebra computations of SageMath. However, this does not change their complexities, and therefore the tendency of having WBLPN faster than HODCA and HDDA for $P_{\ell,d}$ with d being chosen big enough.

6.1 Comparison of WBLPN and HDDA

The cost of WBLPN and HDDA algorithms both do not depend on the linear part of a masking scheme polynomial, facilitating their comparison. Indeed, HDDA depends on the degree of a masking scheme polynomial, while WBLPN depends on its noise rate. Both noise rate and degree are dependent only on the non-linear part of the masking scheme polynomial.

As explained in Subsection 3.4, the order $\mathcal{O}$ of the algorithm determines the maximum degree of the masking scheme that the algorithm can break. In particular, if we take the masking scheme polynomial $P_{\ell,d}$ (*c.f* Definition 4), then we would need an HDDA attack of degree at least $\mathcal{O} = d$. Its time cost increases exponentially with the degree, and polynomially with the window size (for a fixed d). Therefore, increasing d will drastically increase the cost of HDDA.

On the other hand, we have seen in Subsection 5.5 that WBLPN is exponential in the window size depending on the noise rate τ: the lower τ is, the more effective the algorithm is, while remaining exponential. Therefore, since $P_{\ell,d}$ has a noise rate $\tau = \frac{1}{2^d}$, the higher d is, the faster WBLPN is; which is the opposite case for HDDA. Thus, we can state that there exist window sizes and a degree d for a masking scheme polynomial $P_{\ell,d}$ where WBLPN would be more effective than HDDA for reasonable window sizes.

For WBLPN, the number of traces is a variable dependent on parameters T_p and m. The former does not have any impact on the time and the latter is computed as a function of $\mathcal{W}$ and τ. For HDDA, the number of traces depends only on the degree $\mathcal{O}$ and the window size $\mathcal{W}$, and is given by $\binom{\mathcal{W}}{\leq \mathcal{O}} + 30 = \sum_{i=1}^{\mathcal{O}} \binom{\mathcal{W}}{i} + 30$, which ensures probability of false positives to be below 2^{-30}. We can remark that the required number of traces for HDDA becomes problematic with the increasing of its degree and window size.

Each of the four graphs of Figure 1 shows the time comparison of HDDA and WBLPN for the average of 20 time measurements, for different window sizes and degrees d of $P_{\ell,d}$.

We can confirm the tendency of HDDA to be more time-consuming as we increase the degree; while observing the opposite for WBLPN. We can also observe that for masking scheme polynomials $P_{\ell,2}$, HDDA is outperforming WBLPN for any window size. However, for masking scheme polynomials $P_{\ell,d}, d \geq 3$ the cost of HDDA continues to increase with

[3] The source code is available at https://github.com/cryptolu/whitebox-LPN

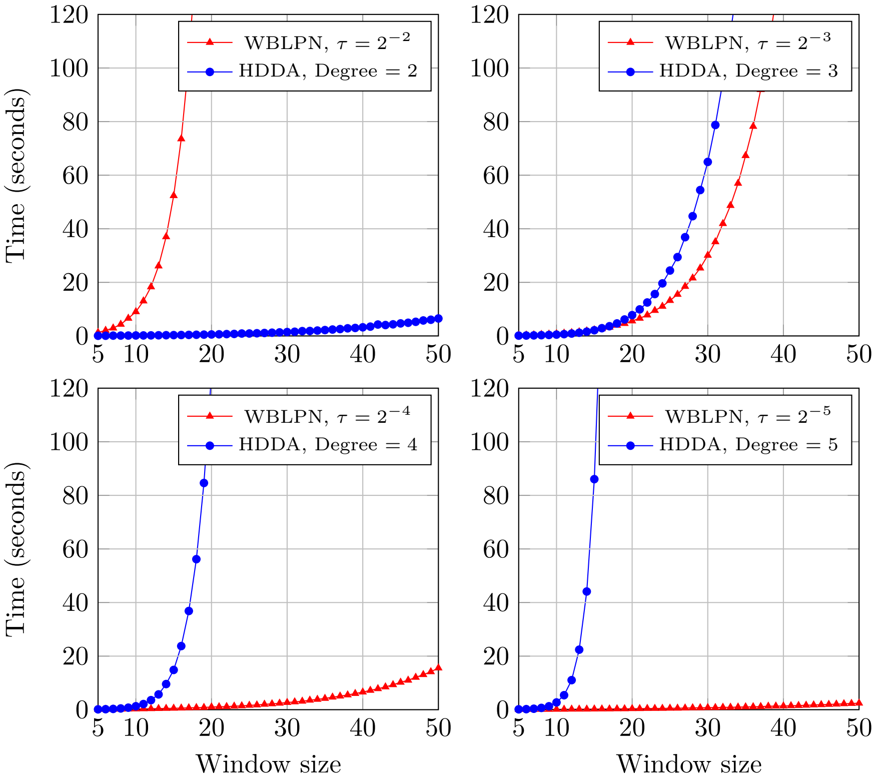

Figure 1: Comparison of the average time (in seconds) of HDDA and WBLPN as the function of the window size. Each of the graphs compares HDDA of degree d and WBLPN of noise rate $\tau = \frac{1}{2^d}$ required to break $P_{\ell,d}$. The average is taken over 20 measurements. The curves are added for visual purposes.

the degree, while the cost of WBLPN decreases and outperforms HDDA for reasonable window sizes.

Figure 1 shows that for $\tau = \frac{1}{4}$, the WBLPN algorithm is impracticable for window sizes greater than 20. Hence the masking scheme [SEL21] given by the masking scheme polynomial $P_{\ell,2}, \ell \geq 2$ should not be broken by our algorithm, but rather with HDDA of degree 2. However, in the case of $P_{\ell,3}$ (the main scheme of [SEL21]), WBLPN significantly outperforms HDDA for the observed window sizes.

Note that in the dummy shuffling model from [BU21], there exists no fixed function to decode the sensitive value (since the shuffling flags are not included in the model and can potentially be protected). Therefore, HDDA of any order $\mathcal{O}$ can not attack dummy shuffling, while WBLPN can. We give more details in comparison of WBLPN and HODCA in the following.

Experimental attack on [SEL21]: For the publicly available SEL implementation[4] of nonlinear degree $d = 3$ and with 4 linear shares with window size 30 we successfully recover the keys with WBLPN using $\tau = 2^{-3}$. For such parameters, for our non-optimized implementations HDDA of degree 3 takes 13.16s per window and $\binom{30}{\leq 3} + 30 = 4555$ traces, HODCA of order 4 takes already 12 minutes for 100 traces per window, which is not even enough to avoid false-positives. Finally, WBLPN with $\tau = \frac{1}{8}$ takes 8.4 seconds and 915 traces (WBLPN and HDDA are faster than in our general benchmarks, due to the removal

[4] Available at https://github.com/UzL-ITS/white-box-masking

of the linearly dependent node vectors to reduce the dimension). If SEL of degree 4 was available, for a single window of size 25 HDDA of degree 4 would take at most about 10 minutes (and 22 GB of RAM), whereas WBLPN of noise rate $\tau = \frac{1}{2^4}$ would take at most 1.73 seconds (and negligible memory usage).

6.2 Comparison of WBLPN and HODCA

Case of SEL-masking As for HDDA attack, the time of HODCA is polynomial in the window size and exponential in its order $\mathcal{O}$. However, contrarily to HDDA, we cannot compare HODCA with WBLPN in their order/noise rate, as the noise rate of WBLPN depends only on the non-linear part of the masking scheme polynomial, while the order of HODCA depends only on the linear part of it.

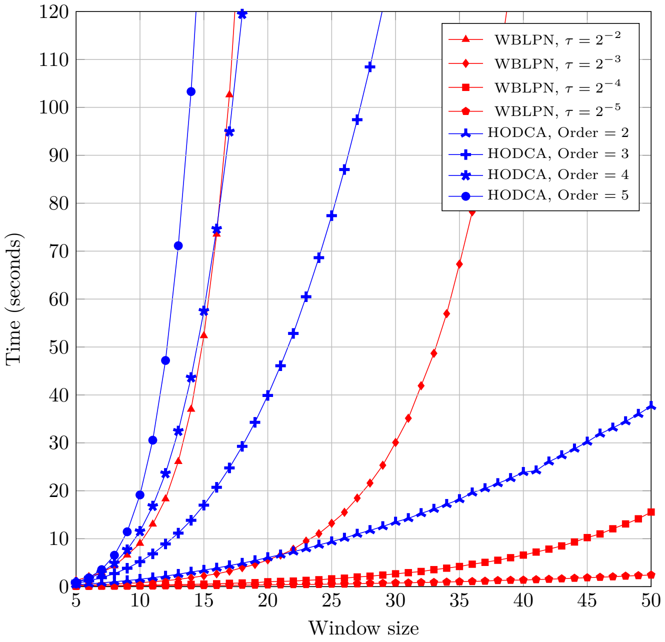

Figure 2: Comparison of the average time (in seconds) of HODCA of different degrees and WBLPN for different noise rates as a function of the window size. The average is taken over 20 measurements. The curves are added for visual purposes.

We benchmarked our implementations of WBLPN and HODCA. The results are given in Figure 2. This comparison shows that for $\tau = \frac{1}{2^d}, d \geq 3$, the WBLPN algorithm is outperforming HODCA of order three or greater for reasonable window sizes.

For the HODCA attack, we performed the time comparison for a constant number of traces equal to 100. Indeed, the number of created node vectors increases with the order, increasing with it the probability of finding false positives. However, the noise rate of $P_{\ell,d}$ will diminish with d, decreasing the number of traces required to ensure observe the correlation.

Case of dummy shuffling Consider an implementation protected by linear ISW masking with ℓ shares and dummy shuffling with t dummy slots. The HODCA needs order ℓ to capture all the shares inside one slot, which when combined would correlate to the shared sensitive value. The work [BU21] proves a lower bound on the noise rate $\tau \geq \frac{1}{4} \cdot \frac{t}{t+1}$. In particular, for $t = 1$ dummy slot, we have $\tau \geq \frac{1}{8}$. Assuming the lower bound is reached, the WBLPN can attack the scheme given that all the ℓ linear shares are contained in a small window, for example, the window of size 40 requires 2 minutes of processing (see Figure 2), while HODCA would quickly become infeasible with the growth of ℓ. On the other hand, when the number of dummy slots increases, the noise rate bound τ raises to $\tau \geq \frac{1}{4}$. This significantly slows down WBLPN where only the window of size 18 can be covered in 2 minutes (using our implementation of the PooledGauss algorithm). Although an optimized low-level implementation could increase the feasible range of window sizes for the attacks, the steep exponential growth clearly shows that the range is very small. We conclude that dummy shuffling has to be used with a sufficient number of dummy slots and linear shares at the same time, to prevent both HODCA and WBLPN attacks.

7 Countermeasures against White-box LPN

Two main directions of securing implementations against WBLPN are increasing the window size required for the attack and the error rate of the protections, since the complexity is proportional to $(1-\tau)^{-\mathcal{W}} = 2^{-\mathcal{W}\cdot\log(1-\tau)}$: it is exponential in the window size and in $-\log(1-\tau)$. Bringing τ closer to $1/2$ brings this complexity factor closer to $2^{\mathcal{W}}$.

From the designer's perspective, enforcing a large window size in attacks is a challenging problem, and can partly be achieved by code/circuit obfuscation techniques, dummy operations, randomization of gadgets, etc., as well as using linear masking of high orders. This direction is out of the scope of this work.

Designing nonlinear countermeasures is a difficult problem as well, and so far there are only a few studied candidates [BU18, SEL21, BU21]. As our work shows, the resistance of the SEL-masking to WBLPN decreases with the growth of the scheme's degree. Therefore, dummy shuffling is the only current potential candidate for a high-degree countermeasure resisting WBLPN, as discussed in the previous section. However, we also propose a general technique for raising the noise rate of a masking scheme.

7.1 Amplifying noise rate of a masking scheme

In [BU18], it was suggested to use a combination of nonlinear and linear masking schemes. However, the composition details were partially discussed only later in [GRW20]. We will now observe that the composition order affects crucially the resistance to the WBLPN attack.

Nonlinear-then-linear composition The first option is to apply linear masking on top of a nonlinear masking scheme. Since the linear shares can be recombined by the WBLPN attack, the original nodes of the nonlinear masking scheme's gadgets are accessible. Therefore, this composition order preserves the masking scheme polynomial's noise rate τ.

Linear-then-nonlinear composition In this option, the implementation is first protected by linear masking, with nonlinear masking applied on top of it. Assuming each circuit's gate is protected by independent shares, we can show that the noise rate of the nonlinear masking scheme can be amplified.

Proposition 4. *Let $P \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a masking scheme polynomial with noise rate τ. For an integer $t \geq 1$, define the masking scheme polynomial $P_t \in \mathbb{F}_2[x_1, \ldots, x_{tn}]$ by*

$$P_t = P(x_1, \ldots, x_t) \oplus P(x_{t+1}, \ldots, x_{2t}) \oplus \ldots P(x_{tn-t+1}, \ldots, x_{tn})$$

Then, the noise rate τ_t of P_t is given by $\tau_t = \frac{1}{2}\left(1-(1-2\tau)^t\right)$.

Proof. Since P_t consists of the addition of t polynomials of independent sets of variables, the best linear approximation of P_t is composed of the best linear approximation of each application of P. Thus, the noise rate can be computed using Matsui's piling-up lemma [Mat94] (note that the bias is equal to $1/2 - \tau$). □

Example 1. Let us consider the application of the BU-masking scheme $P_{1,2}$ on top of the ISW linear masking $P_{t,0}$. That is, an original intermediate value s is shared as $s = x_1 \oplus \ldots \oplus x_t$, which is then shared as

$$s = (x_{1,1} \oplus x_{1,2}x_{1,3}) \oplus (x_{2,1} \oplus x_{2,2}x_{2,3}) \oplus \cdots \oplus (x_{t,1} \oplus x_{t,2}x_{t,3})$$

By using $\tau = 1/4$ (for $P_{1,2}$) in the proposition, we obtain that the resulting masking scheme polynomial has the noise rate $\tau_t = \frac{1}{2} - \frac{1}{2^{t+1}}$, which gets exponentially close to $1/2$ with increasing t. This countermeasure already reaches the noise rate $\tau = 0.4375$ for $P_{1,2}$ applied to only 3 linear shares with $P_{3,0}$. Observing in Figure 2 that WBLPN cannot be performed in practice for a noise rate $\tau = 1/4$ and window size $\mathcal{W} = 20$, we can infer that WBLPN would also not be doable in practical time for such noise rate and for a window size that would ensure catching all 9 shares of this masking scheme at once in a practical context.

Remark 3. Proposition 4 only proves the noise rate of the masking scheme polynomial, not of the gadgets or their compositions. Designing masking schemes with high-error full algebraic security is yet an open problem.

Remark 4. Unfortunately, this approach does not work for dummy shuffling, since the assumption of Proposition 4 does not hold: the nonlinear shares of each of the linear shares are not independent: inside each slot, they are either all correct, or random. Therefore, the noise rate stays the same.

8 Higher-Order White-Box LPN

In this section, we present a higher-order generalization of WBLPN, which is similar to how HDDA generalizes LDA.

We start by describing a higher-order White-Box Pooled Gauss. Similarly to HODCA (*c.f* Subsection 3.2) and HDDA (*c.f.* Subsection 3.4), given a window of size $\mathcal{W}$, for an order $\mathcal{O}$ we increase the window by adding $\binom{\mathcal{W}}{\leq \mathcal{O}}$ AND combinations of up to $\mathcal{O}$ node vectors in the window. This increases the algorithm's time cost as the time complexity of WBLPN is increasing exponentially with the window size (*c.f.* Subsection 5.5). However, all degree-$\mathcal{O}$ monomials of the masking scheme polynomial will be included in the window, and thus will not be considered as noise anymore. This higher-order version can be considered as a trade-off between increasing the window size and reducing the noise rate. Observing in Figure 2 how the noise rate has a huge impact on the time cost, this solution remains realistic.

We will extend the previous Definition 11 of the noise rate of a polynomial, to allow it to define higher-order noise rates:

Definition 13. We denote the *noise rate of order* $\mathcal{O}$ of a polynomial $P \in \mathbb{F}_2[x_1, \cdots, x_n]$ by: $\tau_{\mathcal{O}} = \mathbb{P}_{x \in \mathbb{F}_2^n}(\bigoplus_{m \in \mathcal{M}_{\mathcal{O}}} m(x) = 1)$, with $\mathcal{M}_{\mathcal{O}}$ the set of monomials of degree strictly greater than $\mathcal{O}$ that are contained in P. If $\mathcal{M}_{\mathcal{O}}$ is empty, then $\tau_{\mathcal{O}} = 0$.

Since we are increasing the dimension of the LPN problem from $\mathcal{W}$ to $\binom{\mathcal{W}}{\leq \mathcal{O}}$, we should compute the probabilistic parameters m, c, T_v and $\mathcal{A}$ accordingly to this extended window size. This is reflected in Algorithm 3, which we call HO-WBLPN.

It is also important for this algorithm to work to redo the step of reducing the columns of the window presented in Subsection 5.2, as it is likely that one of the nodes of the window is the AND of some two nodes also contained in the window.

Algorithm 3 HO-WBLPN

Inputs:

- order $\mathcal{O}$
- window size $\mathcal{W}$
- expected maximum $\mathcal{O}$-order noise rate $\tau_{\mathcal{O}}$
- number of key guess possibilities $|\mathcal{K}|$
- number of traces used to sample m for order $\mathcal{O}$
- threshold value c for order $\mathcal{O}$
- number of traces used to verify a solution T_v for order $\mathcal{O}$
- number of attempts $\mathcal{A}$ to achieve chance p_{nfm} of finding a solution for order $\mathcal{O}$
- $(m + T_p) \times \mathcal{W}$ array M containing the traces, with columns linearly independent
- $(m + T_p) \times |\mathcal{K}|$ array S containing their corresponding selection vectors

Output: A value of $k \in \{1, \cdots, |\mathcal{K}|\}$ such that the k^{th} column of S is a solution vector corresponding to a correct key guess if it exists.

```
$M_{\texttt{expanded}} \leftarrow M$
for all $\binom{\mathcal{W}}{\leq \mathcal{O}}$ combinations $C$ of columns of $M$ do
    $M_{\texttt{expanded}} \leftarrow M_{\texttt{expanded}}$ || the vector resulting of the AND of the columns $C$
end for
$M_{\texttt{expanded}} \leftarrow$ the basis of the column space of $M_{\texttt{expanded}}$
return WBLPN(Number of columns of $M_{\texttt{expanded}}, \tau_{\mathcal{O}}, |\mathcal{K}|, m, c, T_v, \mathcal{A}, M_{\texttt{expanded}}, S$)
```

Potential applications The masking scheme from Section 7 (Example 1) is quadratic, and thus has the second-order noise rate $\tau_2 = 0$, as it does not contain any monomial of degree greater than 2. If a noise rate of order $\mathcal{O}$ of a masking scheme polynomial is equal to zero, it is sufficient to use plain HDDA of degree $\mathcal{O}$ (*c.f* Subsection 3.4) to break it.

However, consider a hypothetical masking scheme with the polynomial

$$\tilde{P} = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5x_6 \oplus x_7x_8 \oplus x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}$$

Since $\tilde{P}$ contains 4 linear monomials, to break this scheme, HODCA needs to be of order 4. Likewise, $\tilde{P}$ has algebraic degree equal to 7, so HDDA also needs to be of order 7 to break it, which is not practical. Finally, the noise rate of this polynomial $\tau \approx 0.254$, which, as we saw in Section 6 would be very heavy for WBLPN for a window size $\mathcal{W} \geq 20$. However, since $\tau_2 = \frac{1}{2^7}$, the second order HO-WBLPN is very effective. This is illustrated in the left graph of Figure 3, where we can observe HO-WBLPN being faster than order-4 HODCA and noise rate 0.25 WBLPN, for reasonable window sizes.

Similarly, [SEL21] only proved $\tau \geq \frac{1}{16}$ for their cubic scheme's gadgets, and the second-degree noise-rate $\tau_2 \geq \frac{1}{2^{12}}$. If we compare third degree HDDA, WBLPN with $\tau = \frac{1}{16}$, and second order HO-WBLPN with $\tau_2 = \frac{1}{2^{12}}$ that break it, we can observe in the right graph Figure 3 that both LPN-based methods outperform HDDA for reasonable window sizes, while having similar time, which shows the in-practice applicability of HO-WBLPN.

The used noise rates $\tau = \frac{1}{16}, \tau = \frac{1}{2^{12}}$ correspond to a hypothetical attack based on the proven bound of [SEL21]. The main goal of this thought experiment is to illustrate the utility of HO-WBLPN for designers in evaluating provable security of a countermeasure.

Time Complexity of HO-WBLPN We showed in Subsection 5.5 that WBLPN has complexity $O\left(\frac{\mathcal{W}^{\omega-1}}{(1-\tau)^{\mathcal{W}}} \cdot |\mathcal{K}| \cdot \frac{-\ln(1-\tau)}{(1/2-\tau)^2}\right)$, and we can use this estimation to determine the complexity of HO-WBLPN of order $\mathcal{O}$. Indeed, instead of having the first-order noise rate τ, it would rather have the order-$\mathcal{O}$ noise rate $\tau_{\mathcal{O}}$ (the minimum approximation error of the masking scheme polynomial by a degree-$\mathcal{O}$ function). Likewise, for a window size $\mathcal{W}$,

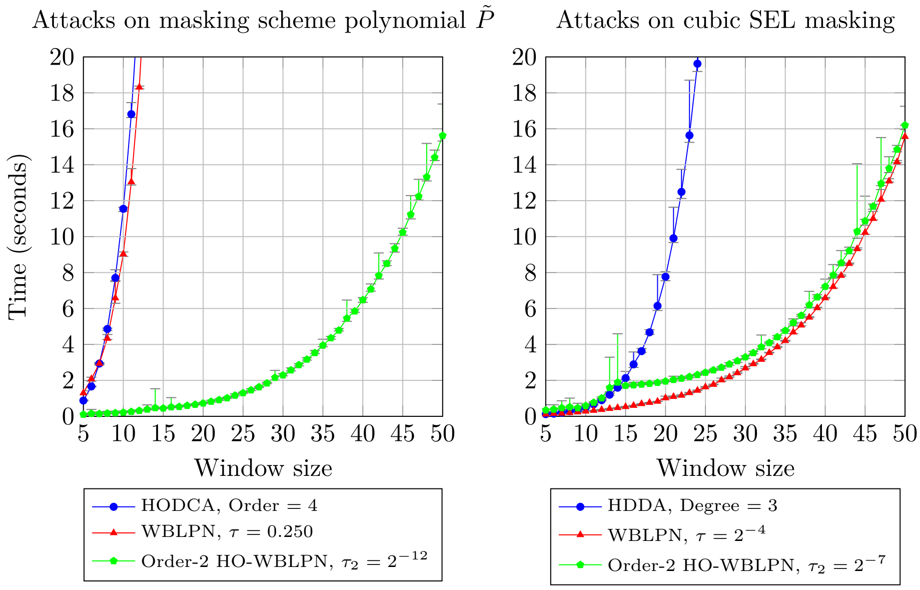

Figure 3: Comparison of the average time (in seconds) of the three possible attacks on the example masking scheme polynomial $\tilde{P}$ on the left graph; and of the three possible attacks on the cubic version of SEL masking scheme. The average is taken over 20 measurements. The error bars represent the minimum and the maximum measured times.

it will increase it to $\binom{\mathcal{W}}{\leq\mathcal{O}} = \sum_{d=0}^{\mathcal{O}}\binom{\mathcal{W}}{d} = O(\mathcal{W}^{\mathcal{O}})$ (ignoring small d-dependent constants) before applying the WBLPN algorithm.

This extension of the window also costs $\binom{\mathcal{W}}{\leq\mathcal{O}}m = O(\mathcal{W}^{\mathcal{O}}m)$ XOR bit-wise operations, which is dominated by the cost of performing WBLPN on this increased window. Therefore, we can deduce that the complexity of HO-WBLPN of order $\mathcal{O}$ is:

$$O\left(\frac{\mathcal{W}^{(\omega-1)\cdot\mathcal{O}}}{(1-\tau_{\mathcal{O}})^{\mathcal{W}^{\mathcal{O}}}}\cdot|\mathcal{K}|\cdot\frac{-\ln(1-\tau_{\mathcal{O}})}{(1/2-\tau_{\mathcal{O}})^2}\right)$$

9 Conclusion

With the effectiveness of automated masking-scheme-breaking algorithms in white-box cryptography, the need for new masking schemes resistant to such attacks rose. We showed that our new algorithm WBLPN has a competitive time cost compared to such attacking schemes while having a complexity dependent on a different parameter: the noise rate. Therefore, we expect our result to force future studies of white-box masking schemes showing their resistance to this attack by studying their noise rate with respect to the efficiency of WBLPN.

We decided to adapt Pooled Gauss algorithm to the white-box context, but an interesting way of pushing forward this study would be to compare other LPN-solving algorithms for small dimensions. Furthermore, to relate the LPN problem to the white-box context, we supposed that the noise is generated randomly, whereas it is not the case with the white-box setting, and therefore we are still not using all the information at our disposal.

Acknowledgments We thank the anonymous reviewers and the shepherd Estuardo Alpírez Bock for their insightful comments and suggestions. This work was supported by the Luxembourg National Research Fund's (FNR) and the German Research Foundation's (DFG) joint project APLICA (C19/IS/13641232).

References

[ABMT18] 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, *ACNS 18*, volume 10892 of *LNCS*, pages 103–120. Springer, Heidelberg, July 2018. 324

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001. 318

[BBD$^+$22] Guillaume Barbu, Ward Beullens, Emmanuelle Dottax, Christophe Giraud, Agathe Houzelot, Chaoyun Li, Mohammad Mahzoun, Adrián Ranea, and Jianrui Xie. ECDSA white-box implementations: Attacks and designs from CHES 2021 challenge. *IACR TCHES*, 2022(4):527–552, 2022. 321

[BDG$^+$22] Sven Bauer, Hermann Drexler, Max Gebhardt, Dominik Klein, Friederike Laus, and Johannes Mittmann. Attacks against white-box ECDSA and discussion of countermeasures: A report on the WhibOx contest 2021. *IACR TCHES*, 2022(4):25–55, 2022. 321

[BGEC04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 227–240. Springer, Heidelberg, August 2004. 318

[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, *CHES 2016*, volume 9813 of *LNCS*, pages 215–236. Springer, Heidelberg, August 2016. 318, 324

[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, *COSADE 2019*, volume 11421 of *LNCS*, pages 118–141. Springer, Heidelberg, April 2019. 319, 324

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 373–402. Springer, Heidelberg, December 2018. 319, 321, 322, 337

[BU21] Alex Biryukov and Aleksei Udovenko. Dummy shuffling against algebraic attacks in white-box implementations. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 219–248. Springer, Heidelberg, October 2021. 319, 320, 321, 322, 327, 335, 336, 337

[CEJv03] 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, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, Heidelberg, August 2003. 318

[CEJvO02] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In *Digital Rights Management Workshop*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002. 318

[DRP13] 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, *SAC 2012*, volume 7707 of *LNCS*, pages 34–49. Springer, Heidelberg, August 2013. 318

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 486–514. Springer, Heidelberg, August 2017. 320, 328, 329, 330, 333

[FA76] Fino and Algazi. Unified matrix treatment of the fast Walsh-Hadamard transform. *IEEE Transactions on Computers*, C-25(11):1142–1146, 1976. 321

[GPRW20] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *Journal of Cryptographic Engineering*, 10(1):49–66, April 2020. 319, 321, 325

[GRW20] Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures. *IACR TCHES*, 2020(3):454–482, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8597. 319, 321, 323, 337

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06*, volume 3989 of *LNCS*, pages 239–252. Springer, Heidelberg, June 2006. 322

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003. 318

[Kar11] Mohamed Karroumi. Protecting white-box AES with dual ciphers. In Kyung-Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology - ICISC 2010*, pages 278–291, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 318

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999. 318, 324

[LRD$^+$14] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box AES implementation. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 265–285. Springer, Heidelberg, August 2014. 318

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397. Springer, Heidelberg, May 1994. 327, 338

[Nat79] National Institute of Standards and Technology. FIPS-46: Data Encryption Standard (DES), 1979. Revised as FIPS 46-1:1988, FIPS 46-2:1993, FIPS 46-3:1999. 318

[Pra62] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962. 328

[RW19] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR TCHES*, 2019(2):225–255, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7391. 324

[Sag22] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.7, Release Date: 2022-09-19)*, 2022. https://www.sagemath.org. 334

[SEL21] Okan Seker, Thomas Eisenbarth, and Maciej Liskiewicz. A white-box masking scheme resisting computational and algebraic attacks. *IACR TCHES*, 2021(2):61–105, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8788. 319, 320, 321, 322, 335, 337, 339

[TGCX23] Yufeng Tang, Zheng Gong, Jinhai Chen, and Nanjiang Xie. Higher-order DCA attacks on white-box implementations with masking and shuffling countermeasures. *IACR TCHES*, 2023(1):369–400, 2023. 319, 321, 323

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 740–757. Springer, Heidelberg, December 2012. 322

[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, 2009. 318