A White-Box Masking Scheme Resisting Computational and Algebraic Attacks

Okan Seker, Thomas Eisenbarth, and Maciej Liskiewicz

University of Lübeck, Germany
{okan.seker,thomas.eisenbarth}@uni-luebeck.de,
liskiewi@tcs.uni-luebeck.de

Abstract. White-box cryptography attempts to protect cryptographic secrets in pure software implementations. Due to their high utility, white-box cryptosystems (WBC) are deployed by the industry even though the security of these constructions is not well defined. A major breakthrough in generic cryptanalysis of WBC was Differential Computation Analysis (DCA), which requires minimal knowledge of the underlying white-box protection and also thwarts many obfuscation methods. To avert DCA, classic masking countermeasures originally intended to protect against highly related side-channel attacks have been proposed for use in WBC. However, due to the controlled environment of WBCs, new algebraic attacks against classic masking schemes have quickly been found. These algebraic DCA attacks break all classic masking countermeasures efficiently, as they are independent of the masking order.
In this work, we propose a novel generic masking scheme that can resist both DCA and algebraic DCA attacks. The proposed scheme extends the seminal work by Ishai et al. which is probing secure and thus resists DCA, to also resist algebraic attacks. To prove the security of our scheme, we demonstrate the connection between two main security notions in white-box cryptography: *probing security* and *prediction security*. Resistance of our masking scheme to DCA is proven for an arbitrary order of protection, using the well-known strong non-interference notion by Barthe et al. Our masking scheme also resists algebraic attacks, which we show concretely for first and second order algebraic protection. Moreover, we present an extensive performance analysis and quantify the overhead of our scheme, for a proof-of-concept protection of an AES implementation.

Keywords: White-box Cryptography · Boolean Masking · Non-linear Masking · Probing Security · Prediction Security · Differential Computation Analysis · Algebraic Attacks

1 Introduction

Protecting secrets purely in software is a great challenge, especially if a full system compromise is not simply declared out-of-scope of the security model. With fully homomorphic encryption still complex and computationally expensive [42] and secure enclaves being notoriously buggy at this time [15, 41, 52], industry may opt for white-box cryptosystems (WBC) or even be required to do so by industry standards like EMVCo [8,44]. White-box cryptography promises implementation security of cryptographic services in pure software solutions, mainly by protecting keys and intermediate cipher states through layers of obfuscation. While white-box cryptography is successfully sold by several companies as one ingredient of secure software solutions (e.g. [27]), analysis of deployed solutions is lacking, as is a sound framework to analyze white-box implementations. The white-box model assumes the cryptographic primitive to run in an untrusted environment where the white-box adversary has complete control over the implementation. The adversary can read and modify every memory access or intermediate state and can interrupt the implementation at will. White-box cryptography was introduced in 2002 by Chow et al. [18, 19]. The main idea of their scheme is to represent a cryptographic algorithm as a network of look-up tables and key-dependent tables. In order to protect the key dependent tables, Chow et al. proposed to use *input and output encodings*. Although the method provides security guarantees for individual tables, the combinations of protected tables still leaks information [5]. In fact, all published academic proposals for WBC [14, 35, 39, 55] have been practically broken [5, 24, 38, 54].

Cryptanalysis of WBCs usually requires a time-consuming reverse engineering step to surpass included obfuscation layers [31]. To overcome this, *computational analysis* of white-box cryptosystems has been proposed. Computational analysis is inspired by physical grey box attacks, mainly *side-channel attacks* (SCA). Computational analysis attacks, like side-channel attacks, perform a statistical analysis of observable intermediate states of a cryptographic implementation, e.g. via a physical side-channel [26, 28, 36]; if the implementation is not protected against this kind of attack, the side-channel may reveal critical information, usually the secret key material used. At CHES 2016, Bos et al. [12] proposed *Differential Computation Analysis* (DCA) and showed that DCA can extract keys from a wide range of different white-box implementations very efficiently, without requiring a detailed reverse engineering of the implementation. Following this work, further generic computational analysis techniques have been proposed for white-box implementations, such as Zero Difference Enumeration [1], Collision Attacks, and Mutual Information Analysis [47]. Alpirez Bock et al. [9] analyzed the ineffectiveness of internal encodings and explained why DCA works so well in the white-box setting. Even fault attacks [2, 11] have been shown to be an effective method for state and key recovery attacks on white-box implementations [7, 12]. Biryukov et al. [6] introduced two new types of fault attacks to reveal the structure of a white-box implementation, an important step of overcoming obfuscation in WBC.

To meet the threat of DCA and other computational analysis, *masking schemes* provide a natural protection mechanism. Masking splits a sensitive variable x into n shares, such that x can be recovered from $d + 1$ ($n \geq d + 1$) shares, while no information can be recovered from fewer than $d+1$ shares [17]. It is a popular and effective countermeasure in the SCA literature. Most important examples are *Boolean masking* introduced by Ishai et al. [34] which has been generalized by

Rivain and Prouff [46], *Threshold Implementations* defined by Nikova et al. [43], and *polynomial masking* as defined in [48] based on Shamir's secret sharing [51]. Recently the idea of *combined countermeasures* to resist both side-channel and fault attacks were introduced in the literature [45, 49, 50].

As an example for this methodology, we can consider the dedicated masked white-box implementation introduced in [37]. However, the implementation was broken in [47]. In addition, for secure WBC, other countermeasures such as fault protection and obfuscation layers need to be added [6] and additional randomness should be included in the input [10], as internal randomness generators could be disabled by the white-box adversary. Furthermore, higher order variants of DCA have been shown to be effective when applied to masked white-box implementations due to the adversary's ability to observe shares without noise [10]. Although the noise-free environment makes the attack easier, techniques like control flow obfuscation, input/output encodings and shuffling [53] create artificial noise in white-box environments [1, 10], effectively increasing the complexity of higher order DCA significantly. More *devastatingly*, a new class of generic *algebraic DCA* (or in short *algebraic attacks*) has been proposed recently [6, 31]. Algebraic DCA are able to break masked WBC independently of *the masking orders if the masking is linear*. Yet all current masking proposals are vulnerable to algebraic DCA. Mark that the scheme defined by [6] indeed resists first order algebraic attacks due to its *non-linear* structure.

To sum up, although there exist informal ideas on how to create a secure white-box design that can resist both computational and algebraic DCA, formal and generic constructions with a security analysis are missing.

Our contribution: In this paper, we provide the first generic and combined masking scheme that resists state-of-the-art white-box attacks: DCA and algebraic attacks. Classic masking schemes can be applied to WBC, however none of them can *individually* achieve security against both attacks. To fill this gap, we examine the `ISW` transformation introduced by Ishai et al. [34] and extend it to the white-box context.

We improve the `ISW` transformation by adding a multiplicatively shared nonlinear share. This additional nonlinear share provides security against algebraic attacks. The secret sharing of our masking scheme then consists of two components: linear and non-linear shares: Linear shares to resist DCA attacks (or *computational attacks*) and non-linear shares to increase the degree of the decoding function and therefore to prevent algebraic attacks. We present the structure of generic masking that resists an arbitrary order computational and first or second order algebraic attacks in Section 3. To analyze the security of our construction in Section 4, we focus on two security notions in cryptography: *probing security* and *prediction security* that cover security against computational attacks and security against algebraic attacks respectively. *Probing model* was introduced by Ishai et al. [34]. Later it was revised by Rivain et al. [46] to a new model called SCA security was emerged. The model states that every tuple of n or less intermediate variables must be independent of any sensitive variable. It was shown that an n^{th}-order Boolean masking scheme provides security against n^{th}-order

SCA. The complexity of computational attacks grows with the masking order. However, the notion is not sufficient to secure a complete block cipher, as stated in [22], thus a new and stronger notion called t-strong non-interference (t-SNI) was defined by [4]. The stronger t-SNI notion enables the composability of the small secure gadgets to generate a complete constructions. As stated in [10], an n^{th}-order masking provides security against n^{th}-order probing attacks and n^{th}-order DCA attacks with additional obfuscation layers.

To cover algebraic attacks a new security notion called *Prediction Security* was defined in [6]. The prediction security of a circuit C (with an encoding function E), is based on the probability of an adversary ($\mathcal{A}$) to accurately predict values of any single function (of d^{th} order) over intermediate values computed in the circuit C (composed with encoding E). The aim of such a prediction is to distinguish two sequences of plaintexts (chosen by the adversary) by analyzing the corresponding software trace. For example, an n^{th}-order Boolean masking that is inherently protected against DCA is vulnerable against first order algebraic attacks, since the adversary can utilize a linear function (i.e. a first order function) and combine a subset of intermediate variables to recover the secret value.

In this work, we further show that the probing security and prediction security notions *are incomparable*. First, we prove that our masking scheme is indeed secure against computational attacks by showing that it is secure in the probing model with the given order using the non-interference notions by Barthe et al. [4]. We give a concrete construction for first and second order prediction security and prove their security. We extend the security definitions given in [6] and give a novel composability proof for the second order prediction secure constructions. Besides the formal proofs, we verify the probing security of our masking scheme using the tool MaskVerif [3] for specific orders. Furthermore, we update and use the tool produced by [6] to experimentally verify the first order prediction security of our scheme. The implementation that can be used with MaskVerif and the updated version of the tool produced by [6] is available as open source[1].

In the Section 5 we introduce a proof-of-concept AES implementation to analyze the overhead and experimentally verify the security properties of our scheme using a simple leakage test. The analysis includes the number of needed gates and number of required randomness for different orders of protection. We show that our combined approach outperforms the previous approaches which required to combine two different masking schemes to resist both attacks.

2 Preliminaries

In this section, we provide the notation and definitions used in this paper. We also identify the challenges that need to be addressed for secure white-box designs.

First, we summarize the notation used throughout the paper. In the following, we use some finite ring $(\mathbb{K}, \oplus, \otimes)$ with an *addition* operation $\oplus$ and a

[1] `https://github.com/UzL-ITS/white-box-masking`

multiplication operation $\otimes$. We often omit the multiplication symbol $\otimes$ and thus write xy instead of $x \otimes y$. Although we introduce the notations using $\mathbb{K}$ we fix $\mathbb{K} = \mathrm{GF}(2)$ through the paper. A vector space over $\mathbb{K}$ of dimension ℓ is denoted by $\mathbb{K}^\ell$. For $a, b \in \mathbb{Z}$ with $a < b$, we define $[a, b] := \{a, a+1, \dots, b-1, b\}$. The letters $x, y, z, \dots$ represent the sensitive variables. Random variables are represented by the letter r, with an index as r_i or r^i. To denote a random selection of a variable r from the field $\mathbb{K}$, we use $r \in_R \mathbb{K}$.

A variable x is split into $n+1$ linear shares $x_0, \dots, x_n$ such that $x = \bigoplus_{i=0}^{n} x_i$ and a single share (e.g. x_0) split into $d+1$ non-linear shares $\tilde{x}_0, \dots, \tilde{x}_d$ such that $x_0 = \prod_{j=0}^{d} \tilde{x}_j$. A vector of shares $(\tilde{x}_0, \dots, \tilde{x}_d, x_1, \dots, x_n)$ is denoted by $\overline{x}$. For a subset $I \subseteq [0, n]$ of indices, we denote by $x_{|I} = (x_i)_{i \in I}$ the sub-vector of shares indexed by I. A *gadget* G for a function $f\colon \mathbb{K}^a \to \mathbb{K}^b$ (with regard to a masking order) is an arithmetic circuit with $a \cdot (n+d+1)$ inputs and $b \cdot (n+d+1)$ outputs grouped into a vectors of shares $\overline{x}^{(1)}, \dots, \overline{x}^{(a)}$, resp. b vectors of shares $\overline{y}^{(1)}, \dots, \overline{y}^{(b)}$. The gadget needs to be correct, i.e. $G(\overline{x}^{(1)}, \dots, \overline{x}^{(a)}) = (\overline{y}^{(1)}, \dots, \overline{y}^{(b)})$ iff $f(x^{(1)}, \dots, x^{(a)}) = (y^{(1)}, \dots, y^{(b)})$ for all possible inputs and for all values generated by the random gates. The values assigned to wires that are not output wires are called *intermediate variables.* Bold numbers **0** and **1** are used to denote constant functions.

As usual, we model the white-box implementations as Boolean circuits represented by directed acyclic graphs. Each node in a circuit C, with $k > 0$ inputs, corresponds to a k-ary Boolean function. Nodes with the indegree equal to zero are called inputs of C and nodes with the outdegree equal to zero are called outputs of C.

Let $\mathsf{x} = (\mathsf{x}_1, \dots, \mathsf{x}_N)$ (resp. $\mathsf{y} = (\mathsf{y}_1, \dots, \mathsf{y}_M)$) be a vector of input (resp. output) nodes in some fixed order. For each node v in C, we say that it computes a Boolean function $f_v : \mathbb{F}_2^N \to \mathbb{F}_2$ defined as follows:

- for all $1 \le i \le N$ set $f_{\mathsf{x}_i}(z) = z_i$,
- for all non-input nodes v in C set $f_v(z) = g_v(f_{c_1}(z), \dots, f_{c_k}(z))$, where $c_1, \dots, c_k$ are nodes having an outgoing edge to v and $g_v : \mathbb{F}_2^k \to \mathbb{F}_2$.

The set of f_v for all nodes v in C is denoted $\mathcal{F}(C)$, the set of f_{x_i} for all input nodes x_i is denoted $\mathcal{X}(C)$, and the set of f_v for all non-input nodes v in C is denoted $\mathcal{F}(C \setminus \mathcal{X})$.

Recall, that any Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ has unique representation of the form $f(x) = \bigoplus_{b \in \mathbb{F}^n} a_b \, x_1^{b_1} \dots x_n^{b_n}$, with $a_b \in \mathbb{F}_2$. The (algebraic) degree of f, denoted $\deg(f)$, is the maximum degree of a monomial $x_1^{b_1} \dots x_n^{b_n}$, with $a_b = 1$.

If $\mathcal{V} = \{g_1, \dots, g_{|\mathcal{V}|}\}$ is a set of Boolean functions with the same domain $\mathbb{F}_2^n$ then by the d-th order closure of $\mathcal{V}$ (denoted $\mathcal{V}^{(d)}$) we call the vector space of all functions obtained by composing any function of degree at most d with functions from $\mathcal{V}$, i.e., $\mathcal{V}^{(d)}$ contains functions of the form $f \circ (g_1(x), \dots, g_{|\mathcal{V}|}(x))$ for all $f : \mathbb{F}_2^{|\mathcal{V}|} \to \mathbb{F}_2$, with $\deg(f) \le d$. For example, $\mathcal{F}^{(1)}(C)$ is spanned by $\{\mathbf{1}\} \cup \mathcal{F}(C)$ and $\mathcal{F}^{(2)}(C)$ is spanned by $\{\mathbf{1}\} \cup \{g_i g_j \mid g_i, g_j \in \mathcal{F}(C)\}$.

Differential Computational Analysis: The idea of using *side-channel attacks* to recover critical secrets in WBC has been introduced by Bos et al. [12]. Differential computational analysis utilizes internal states of the software execution (such as memory accesses) to generate software traces. DCA is regarded as one of the most efficient attacks against white-box implementations, since it does not require full knowledge of the white-box design and thus avoids the time-consuming reverse engineering process. The first part of DCA consists of collecting software traces using memory addresses, intermediate values or written/read values by the implementation. In the second part a statistical analysis is performed using the software traces collected in the first part.

To resist against DCA, a natural approach is to use the well-known side-channel analysis countermeasure *masking* [17]. The masking is carried out in two steps as defined in the seminal work by Ishai, Sahai, and Wagner in 2003 [34]. First, input data is transformed by representing each input x by $n+1$ shares in such a way that

$$x = x_0 \oplus \cdots \oplus x_n,$$

where $x \in \mathbb{F}_2$ and n of the shares are distributed uniformly and independently. Additionally, the circuit is adapted by replacing all AND and XOR gates with gadgets processing the shares of the inputs. Throughout the paper, the two stages of masking will be defined as `ISW` transformation.

Masking schemes rely on the availability of good randomness, which is usually provided by secure RNGs, e.g. in the form of a secure and efficient Pseudorandom Generator [23, 33]. Similarly, randomness generation for white-box implementations has been analyzed in the literature. Due to the adversarial ability to control the execution environment in the white-box model, the attacker can simply disable any external randomness sources. Therefore, white-box implementations have to rely on internal randomness sources in combination with additional obfuscation countermeasures [1, 6, 10]. Remark that the effectiveness of DCA comes from its universality and its ability to avoid reverse-engineering, which can be extremely costly [31]. By combining masking with an obfuscation layer, the adversary is thus again forced to do a time-consuming reverse engineering step to bypass the obfuscation, which cannot be done by an automated tool, while the masking prevents obfuscation-oblivious attacks such as DCA.

Algebraic Attacks: Algebraic attacks have been introduced during the WhibOx contest of CHES2017 [20]. Although the majority of the implementations in the contest were broken in less than one day, even the strongest design (by means of the surviving time: 28 days) was broken by algebraic analysis [6, 31]. Algebraic attacks try to find a set of circuit nodes whose d^{th}-order of combination equals to a predictable vector. Observe that if an implementation is protected by a linear masking, there exists a set of circuit nodes (corresponding to the secret shares) such that a linear combination (i.e. the first order combination) is always equal to a predictable secret value. This means that linear masking is inherently vulnerable to first-order algebraic attacks *independently of the masking order* [6, 31]. Like DCA, algebraic attacks do not require complex reverse engineering and are thus a generic threat that any white-box implementation needs to address.

Another challenge for secure white-box implementation is the adversaries' ability to collect noise-free measurements. The security of masking schemes against side-channel attacks or DCA requires noisy observations [16]. To deal with this problem, artificial noise sources such as control flow obfuscation [1], shuffling [10], and input and output encodings [9] have been analyzed in the literature. The artificial noise introduced by these methods increases the complexity of higher order DCA dramatically. It has been shown in [10] that the complexity of attacks increases with the order of the masking and the order of the obfuscation layers. Therefore, the probing model is a valid approach to analyze the security of masking schemes of white-box implementations against DCA. Due to the artificial noise sources, it becomes infeasible for an attacker to combine the required number of shares to recover the sensitive information. Throughout the paper we assume a reliable randomness source is provided as part of the implementation, in other words, randomness can be provided via pseudorandom values derived from the input and protected by obfuscation layers, as done in [1, 6, 47]. Therefore, the attacks on randomness sources and the adversaries' ability to disable randomness is out-of-scope in this work. For a full white-box implementation, other techniques (fault protection, randomness generation, obscurity layers) need to be added [6,10] in addition to a secure masking scheme, which we introduce throughout this work.

In the next section, we introduce our masking scheme, which resists both computational and algebraic attacks by using an adapted version of the `ISW` transformation.

3 Secure Masking Construction

The proposed masking scheme is based on two ideas: an `ISW`-like masking to increase the number of shares required to eliminate computation attacks and using a multiplicative sharing to increase the degree of the decoding function. We call the first part linear sharing of order n and the second part non-linear sharing of degree d. And the resulting construction is named (n, d)-masking. We start with the data transformation and define our masking function:

$$\texttt{Encode}(x, \tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_{n-1}) = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)\,,$$

where $\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_{n-1} \in_R \mathbb{F}_2$ are chosen randomly and independently from $\mathbb{F}_2$, and

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

Observe that our masking scheme is obtained from the `ISW` transformation by replacing the first share x_0 in `ISW` by a non-linear sharing $x_0 = \prod_{j=0}^{d} \tilde{x}_j$. The unmasking function is defined as follows:

$$\texttt{Decode}(\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n) = \prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i.$$

The data transformation is followed by the transformations of each AND and XOR gate. Throughout the paper, we define the transformed gates as `And` and `Xor` (or `And`$[n, d]$ and `Xor`$[n, d]$) gadgets respectively.

3.1 Gate Transformations

In this section the generic constructions for `Xor`, `And` are presented. Additionally, we provide definition of the `RefreshMask` gadget, which is needed to protect against algebraic attacks. The scheme can be used for an arbitrary order n of linear masking and any degree d of the non-linear component. Though the constructions are general, the algebraic security depends on the variable structure (the details can be found in Section 4). The intermediate variables (which become the bottlenecks in the design) need a special structure depending on the non-linear degree d are the following:

– The intermediate variable $\mathcal{U}$ used in `Xor` and specified in Equation (1),
– The intermediate variables $r_{j,0}$ in Equation (2), used in `And`, outputs the variables $\mathcal{V}$,
– The intermediate variables $\mathcal{W}$ and $\mathcal{R}$ used in `RefreshMask`, Equation (3).

In the following descriptions we first introduce the *functionalities* of these variables which can be defined for arbitrary orders of n and d. Afterwards, we will show the *computational structure* of these variables for $d = 1$ and $d = 2$.

Let x and y be two bits and consider an (n, d)-masking scheme, i.e. x and y have been split into $(n + d + 1)$ shares such that $\prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i = x$ and $\prod_{j=0}^{d} \tilde{y}_j \oplus \bigoplus_{i=1}^{n} y_i = y$.

Xor$[n, d]$ *Gadget:* A masked representation of $z = x \oplus y$ with $n + d + 1$ shares such that $\prod_{j=0}^{d} \tilde{z}_j \oplus \bigoplus_{i=1}^{n} z_i = z$ can be calculated as follows:

Step-0: The input shares are processed by `RefreshMask` gadgets;

$$\overline{x} \leftarrow \texttt{RefreshMask}(x) \text{ and } \overline{y} \leftarrow \texttt{RefreshMask}(y).$$

Step-1: The values of the non-linear shares are processed:

$$\tilde{z}_i = \tilde{x}_i \oplus \tilde{y}_i \text{ for } 0 \leq i \leq d.$$

Step-2: Computation of linear shares:

$$z_i = \begin{cases} x_i \oplus y_i, & \text{for } 1 \leq i < n \\ x_i \oplus y_i \oplus \mathcal{U}, & \text{for } i = n. \end{cases}$$

where the functionality of $\mathcal{U}$ is defined as follows:

$$\mathcal{U} = \bigoplus_{\substack{I \subsetneq \{0,\ldots,d\} \\ I \neq \emptyset}} \prod_{i \in I} \tilde{x}_i \prod_{j \notin I} \tilde{y}_j \tag{1}$$

Moreover, we can introduce the computational structure of $\mathcal{U}$ for a secure masking scheme as follows:
– `Xor`$[n, 1]$: $\mathcal{U} = \tilde{x}_0\tilde{y}_1 \oplus \tilde{x}_1\tilde{y}_0$
– `Xor`$[n, 2]$: $\mathcal{U} = \tilde{x}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{y}_2(\tilde{x}_0 \oplus \tilde{y}_0)) \oplus \tilde{y}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{x}_0(\tilde{x}_2 \oplus \tilde{y}_2))$
– `Xor`$[n, d]$ for $d \geq 3$, the functionality of $\mathcal{U}$ can be defined as in Equation (1). However the computational structure should be described carefully in order not to create vulnerabilities in algebraic security.

Algorithm 1 $\texttt{Xor}(\overline{x}, \overline{y})$

```
Input: The shares $\overline{x} = ((\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]})$ and $\overline{y} = ((\tilde{y}_j)_{j\in[0,d]}, (y_i)_{i\in[1,n]})$.
Output: The shares of $x \oplus y$ as $\overline{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$.
1: $\overline{x} \leftarrow \texttt{RefreshMask}(\overline{x})$
2: $\overline{y} \leftarrow \texttt{RefreshMask}(\overline{y})$
3: for $0 \le j \le d$ do
4:     $\tilde{z}_j \leftarrow \tilde{x}_j \oplus \tilde{y}_j$
5: for $1 \le i < n$ do
6:     $z_i \leftarrow x_i \oplus y_i$
7: $z_n \leftarrow x_n \oplus y_n \oplus \mathcal{U}$
8: return $\bar{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$
```

And$[n,d]$ *Gadget:* A masked representation of $z = xy$ with $n+d+1$ shares such that $\prod_{j=0}^{d} \tilde{z}_j \oplus \bigoplus_{i=1}^{n} z_i = z$ can be calculated as follows:

Step-0: The input shares are processed by `RefreshMask` gadgets;

$$\overline{x} \leftarrow \texttt{RefreshMask}(x) \text{ and } \overline{y} \leftarrow \texttt{RefreshMask}(y).$$

Step-1: The calculations of the values with multiplicative representation are processed. Additional random bits $r^{i,j}$ are generated in order to attain algebraic security in the second step.

$$\tilde{z}_i = \tilde{x}_i \tilde{y}_{i'} \oplus r^{i,1} \oplus \cdots \oplus r^{i,n} \text{ for } 0 \le i \le d \text{ where } i' = i+1 \bmod (d+1).$$

Step-2: The variables $r_{j,i}$ for $0 \le i < j \le n$ are generated as follows:

$$r_{j,i} = \begin{cases} (r_{i,j} \oplus (\tilde{x}_0 \cdots \tilde{x}_d) y_j) \oplus x_j (\tilde{y}_0 \cdots \tilde{y}_d), & \text{for } i = 0 \quad \textbf{(a)} \\ (r_{i,j} \oplus x_i y_j) \oplus x_j y_i, & \text{for } 1 \le i \le n \text{ where } r_{i,j} \in_R \mathbb{F}_2 \quad \textbf{(b)} \end{cases},$$

The calculations for $1 \le i \le n$ are processed as identical to the `ISW-And` gadget. However, for $i = 0$ the calculations require a special computational structure:

$$r_{j,0} = [r_{0,j} \oplus (\tilde{x}_0 \cdots \tilde{x}_d) y_j] \oplus x_j (\tilde{y}_0 \cdots \tilde{y}_d) \text{ for } 1 \le j \le n. \qquad (2)$$

Observe that $r_{i,j}$ for $1 \le i < j \le n$ is assigned a uniformly random value. However, $r_{0,j}$ cannot be assigned as random. Instead, $r_{0,j}$ should be defined in such a way that the following equation holds:

$$\bigoplus_{j=1}^{n} r_{0,j} = \bigoplus_{\substack{I \subset \{0,\ldots,d\} \\ I \neq \emptyset}} \prod_{i\in I} \tilde{x}_i \tilde{y}_{i'} \prod_{j \notin I} (r^{j,1} \oplus \cdots \oplus r^{j,n}) \text{ where } i' = i+1 \bmod (d+1).$$

Throughout the paper we denote the right-hand side of the above equation as $\mathcal{V}$. Note that the above functionality for $r_{j,0}$ (given on the right-hand side

Algorithm 2 $\texttt{And}(\overline{x}, \overline{y})$

Input: The shares $\overline{x} = ((\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]})$ and $\overline{y} = ((\tilde{y}_j)_{j\in[0,d]}, (y_i)_{i\in[1,n]})$.
Output: The vector of shares of xy as $\overline{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$.

```
 1: $\overline{x} \leftarrow \texttt{RefreshMask}(\overline{x})$
 2: $\overline{y} \leftarrow \texttt{RefreshMask}(\overline{y})$
 3: for $0 \le i \le d$ do
 4:     $\tilde{z}_i = \tilde{x}_i \tilde{y}_{i'}$                          ▷ $i' = i + 1 \bmod (d+1)$
 5:     for $1 \le j \le n$ do
 6:         $r^{i,j} \leftarrow \texttt{rand}(0,1)$
 7:         $\tilde{z}_i = \tilde{z}_i \oplus r^{i,j}$
 8: for $0 \le i \le n$ do
 9:     for $i < j \le n$ do
10:         if $i = 0$ then
11:             $r_{j,0} \leftarrow$ as described in the text.
12:         else
13:             $r_{i,j} \leftarrow \texttt{rand}(0,1)$
14:             $r_{j,i} \leftarrow (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$
15: for $1 \le i \le n$ do
16:     $z_i \leftarrow x_i y_i$
17:     for $0 \le j \le n$ and $j \ne i$ do
18:         $z_i \leftarrow z_i \oplus r_{i,j}$                          ▷ Denoted by $z_{i,j}$
19: return $\overline{z} = ((\tilde{z}_j)_{j\in[0,d]}, (z_i)_{i\in[1,n]})$
```

of Equation (2)) is not secure against an algebraic attack, even if it is only a first order one. Below we provide a secure computational structure for the case of an $(n, 1)$ and $(n, 2)$-masking.

– $\texttt{And}[n,1] : r_{j,0} = \tilde{x}_1(\tilde{x}_0 y_j \oplus r^{0,j}\tilde{y}_0) \oplus \tilde{y}_1(\tilde{y}_0 x_j \oplus r^{1,j}\tilde{x}_0) \oplus r^{1,j}(r^{0,1} \oplus \ldots \oplus r^{0,n})$.

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

– $\texttt{And}[n,d]$ for $d \ge 3$ the circuit nodes that calculates $r_{j,0}$ should be structured in such a way that algebraic security properties are satisfied.

Step-3: The final step can be performed identical to an `ISW-And` gadget: For every $1 \le i \le n$, compute $z_i = x_i y_i \oplus \bigoplus_{i \ne j} r_{i,j}$.

`RefreshMask`$[n,d]$ *Gadget:* This operation has a crucial importance for generating an algebraically secure implementation. In fact, it has to be combined with each `Xor` and `And` gadget in order to obtain a fully secure masking scheme. The security details can be found in Section 4.

Step-1: For $0 \le i \le d$, calculate $\tilde{x}'_i = \tilde{x}_i \oplus \tilde{r}$ where $\tilde{r} \in_R \mathbb{F}_2$.

Step-2: For $1 \leq i < j < n$, calculate $x'_i = x_i \oplus r$ and $x_j = x_j \oplus r$ where $r \in_R \mathbb{F}_2$.
Step-3: In the last step we need to define two intermediate variables as follows:

$$\mathcal{W}' = \bigoplus_{I \subsetneq \{0,\ldots,d\}} \prod_{i\in I} \tilde{x}_i \prod_{j \notin I} \tilde{r}_j \quad \text{and } \mathcal{W} = \bigoplus_{\substack{I \subsetneq \{0,\ldots,d\} \\ I \neq \emptyset}} \prod_{i \in I} (\tilde{x}_i \oplus r_0) \prod_{j \notin I} \tilde{r}_j,$$

Here, as usual, a product over the empty set I is evaluated as 1. Using the above equations we define the variable $\mathcal{R} = \mathcal{W} \oplus \mathcal{W}'$. Now, we can introduce the variables that need to be added to the final share x_n as:

$$x'_n \leftarrow x_n \oplus \mathcal{W} \oplus \mathcal{R} \text{ where } \mathcal{R} = \mathcal{W}' \oplus \mathcal{W}. \tag{3}$$

Remark that we cannot directly add $\mathcal{W}'$ to the final share x_n due to algebraic security properties. Therefore, the variables $\mathcal{W}$ and $\mathcal{R}$ should be added to the final share in order to define an algebraically secure mask refreshing gadget. The computational structure of the circuit nodes to calculate $\mathcal{W}$ and $\mathcal{R}$ for `RefreshMask`$[n,1]$ and `RefreshMask`$[n,2]$ can be found below.

– `RefreshMask`$[n,1]$: $\mathcal{W} = \tilde{r}_0(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0)$ and $\mathcal{R} = (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0) \oplus r_0$.
– `RefreshMask`$[n,2]$:
$$\begin{aligned}\mathcal{W} = {} & \tilde{r}_1\tilde{r}_2(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_0\tilde{r}_2(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_0\tilde{r}_1(\tilde{x}_2 \oplus r_0) \oplus \\ & \tilde{r}_2(\tilde{x}_0 \oplus r_0)(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0)(\tilde{x}_2 \oplus r_0) \oplus \\ & \tilde{r}_0(\tilde{x}_1 \oplus r_0)(\tilde{x}_2 \oplus r_0), \\ \mathcal{R} = {} & (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0)(\tilde{r}_2 \oplus r_0) \oplus \\ & r_0\left[\tilde{r}_2(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_0(\tilde{x}_1 \oplus r_0)\right] \oplus \\ & r_0\left[\tilde{r}_2(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_2 \oplus r_0) \oplus \tilde{r}_0(\tilde{x}_2 \oplus r_0)\right].\end{aligned}$$
– `RefreshMask`$[n,d]$ for $d \geq 3$ the circuit nodes that calculate $\mathcal{W}$ and $\mathcal{R}$ should be constructed in such a way that algebraic security properties are satisfied.

3.2 Correctness and Performance Analysis

Next, we introduce the transformation $\mathtt{T}^{(n,d)}$ to generate a Boolean circuit that is protected by an (n,d)-masking scheme by using the gadgets described in Section 3.1. The following lemma summarizes the correctness of the transformation $\mathtt{T}^{(n,d)}$.

Lemma 1. *Let us denote the Boolean circuit C initialized with data D by $C[D]$. The transformation $\mathtt{T}^{(n,d)} : C[D] \mapsto C'[D']$ where C' uses* `And`*,* `Xor`*,* `RefreshMask` *gadgets and* `Encoding`*,* `Decoding` *functions described in Section 3 with randomness gates is a functionality preserving transformation, i.e. $C[D]$ and $C'[D']$ have the same input-output behavior.*

Algorithm 3 `RefreshMask`($\overline{x}$)

```
Input: The shares $\overline{x} = ((\tilde{x}_j)_{j\in[0,d]}, (x_i)_{i\in[1,n]})$
Output: The shares $\overline{x} = ((\tilde{x}'_j)_{j\in[0,d]}, (x'_i)_{i\in[1,n]})$
 1: for $0 \le j \le d$ do
 2:     $\tilde{r}_j \leftarrow \texttt{rand}(0,1)$
 3:     $\tilde{x}'_j \leftarrow \tilde{x}_j \oplus \tilde{r}_j$
 4: for $1 \le i \le n$ do $x'_i \leftarrow x_i$
 5: for $1 \le i \le n$ do
 6:     for $i+1 \le j \le n$ do
 7:         $r_{i,j} \leftarrow \texttt{rand}(0,1)$
 8:         $x'_i \leftarrow x'_i \oplus r_{i,j}$                ▷ Denoted by $a_{i,j}$
 9:         $x'_j \leftarrow x'_j \oplus r_{i,j}$                ▷ Denoted by $b_{j,i}$
10: $r_0 \leftarrow \texttt{rand}(0,1)$           ▷ $r_0$ is used to compute $\mathcal{W}$ and $\mathcal{R}$
11: $x'_n \leftarrow x_n \oplus \mathcal{W} \oplus \mathcal{R}$
12: return $(((\tilde{x}'_j)_{j\in[0,d]}, (x'_i)_{i\in[1,n]})$
```

The proof for this lemma can be found in Appendix A. In conclusion, the transformation $\mathtt{T}^{(n,d)}$ can be used to transform any circuit to an (n, d)-masked circuit in a functionality preserving manner. Although we are using an n^{th} order linear masking, the scheme only provides an $(n-1)^{th}$ probing security. Due to the non-linear sharing, the masking loses one share to increase the decoding order. Also the algebraic security depends on the structure of the Equations (1), (2), and (3) in each gadget as underlined above. The details can be found in Section 4.2.

Performance Analysis: In order to compare our construction with the previous schemes we analyze the performance of our scheme in terms of bitwise operations and randomness requirements. An analytical comparison of different orders and a comparison between the `ISW` transformation and (n, d)-masking scheme can be found in Table 1.

In the following analysis, for simplicity, we use the symbol vertical bar ($|$) to separate the number of Xor, And operations respectively. We exclude the `RefreshMask` gadgets inside the `Xor` and `And` gadgets to analyze the constructions straightforwardly. Since the structure of the bottleneck variables depends on the non-linear degree d, we use a symbolic approach to analyze the performance numbers for the higher orders (i.e. for $d \ge 3$). We use subscripts to denote the number of operations within $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{R}$, e.g., $\mathcal{U}_x$ and $\mathcal{U}_a$ represent the number of bitwise Xor, And operations within $\mathcal{U}$ respectively.

As seen in Table 1, the `Xor` gadget can be transformed efficiently. The cost of the gadget in the `ISW` transformation is $n+1$ bitwise Xor operations while an (n, d)-masking requires $n+d+2$ bitwise Xor operations and the additional cost of the variables $\mathcal{U}$. Therefore, the cost of the `Xor` gadget can be calculated as; $(n+d+2+\mathcal{U}_x)|\mathcal{U}_a$.

The cost of an `And` gadget can be analyzed easily by comparing it step by step with the `ISW` transformation. As seen in the construction in Section 3, the gadget can be divided into three stages.

Table 1. The number of bitwise operations in a masked `Xor`, `And` and `RefreshMask` (or `RefM` in short) gadget. Remark that $(n, 0)$-masking scheme corresponds to `ISW` gadgets. The last part of the table corresponds to the overhead of (n, d)-masking scheme compared to the `ISW` transformation.

	Xor	And	Randomness
$\texttt{Xor}[n,0]$	$n+1$	-	-
$\texttt{And}[n,0]$	$2n(n+1)$	$(n+1)^2$	$n(n+1)/2$
$\texttt{RefM}[n,0]$	$n(n-1)$	-	$n(n-1)/2$
$\texttt{Xor}[n,1]$	$n+4$	2	-
$\texttt{And}[n,1]$	$2n^2+5n-1$	n^2+7n+2	$n(n+3)/2$
$\texttt{RefM}[n,1]$	$n(n-1)+8$	3	$(n(n-1)/2)+2$
$\texttt{Xor}[n,2]$	$n+9$	6	-
$\texttt{And}[n,2]$	$2n^2+15n-2$	$n^2+27n+3$	$n(n+5)/2$
$\texttt{RefM}[n,2]$	$n(n-1)+30$	22	$(n(n-1)/2)+3$
$\texttt{Xor}[n,d]$	$n+d+2+\mathcal{U}_x$	$\mathcal{U}_a$	-
$\texttt{And}[n,d]$	$n(2n+d-1)+\mathcal{V}_x$	$n^2+d+1+\mathcal{V}_a$	$n(n+2d+1)/2$
$\texttt{RefM}[n,d]$	$n(n-1)+d+1+\mathcal{W}_x+\mathcal{R}_x$	$\mathcal{W}_a+\mathcal{R}_a$	$(n(n-1)/2)+d+1$
	Overhead		
$\texttt{Xor}[n,d]$	$d+1+\mathcal{U}_x$	$\mathcal{U}_a$	-
$\texttt{And}[n,d]$	$n(2n+d-3)+\mathcal{V}_x-1$	$d+\mathcal{V}_a-n$	nd
$\texttt{RefM}[n,d]$	$d+1+\mathcal{W}_x+\mathcal{R}_x$	$\mathcal{W}_a+\mathcal{R}_a$	$d+1$

- **Step-1** requires $n(d+1)$ random bits and the cost of processing these values can be calculated as $n(d+1)|d+1$.
- **Step-2(a)** includes the calculations of $r_{j,0}$ for $1 \leq j \leq n$. For the $(n, 1)$ masking, $\mathcal{V}_x = 4n$ and $\mathcal{V}_a = 7n$. Additionally, the calculations of $r^{0,1} \oplus \ldots \oplus r^{0,n}$ require $n-1$ Xor. Similarly, $(n, 2)$ masking $\mathcal{V}_x = 12n$ and $\mathcal{V}_a = 27n$. Also the intermediate variables u, v, and uv are calculated only once and they require $2(n-1)|1$.
- **Step-2(b) & Step-3** involve the calculations of $r_{j,i}$ for $1 \leq i < j \leq n$, $i \neq 0$ and Step-3. These parts can be processed identical to the `ISW` transformation and cost $2n(n-1)|n^2$ gates, while the required number of random bits is $n(n-1)/2$. Observe that the cost of these parts are exactly the cost of an `ISW-AND` gadget with n shares.

To sum up, we express the cost of $\texttt{And}[n,d]$ gadget as $(n(2n+d-1)+\mathcal{V}_x)|(n^2+d+1+\mathcal{V}_a)$ gates, and the required randomness as $n(n+2d+1)/2$.

We analyze the performance of the `RefreshMask` gadget using a similar methodology. The total number of required randomness and the number of required bitwise Xor operations can be calculated as $(n(n-1)/2)+d+1$ and $n(n-1)+d+1$ respectively. As in the previous gadgets, the calculations of $\mathcal{W}$ and $\mathcal{R}$ add more calculations to the structure. The numbers for $\texttt{RefreshMask}[n,1]$ and $\texttt{RefreshMask}[n,2]$ can be seen in Table 1.

Using the performance analysis, we show the exact overhead of our scheme. The numbers in the overhead section of Table 1 can be calculated by comparing the cost of the n^{th}-order `ISW` transformation with an (n, d)-masking scheme. As seen in the table, the cost principally depends on the calculation of the values $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{R}$ while the randomness is affected by the masking degrees n and d.

4 Security Against Computational and Algebraic Attacks

In this section, we use the definition of non-interference as defined by [4] which guarantees security against t-probes for $t \leq n$ as proposed by Ishai et al. [34] and security against algebraic attacks of degree d as proposed in [6]. First, we recall briefly both security notions and then we prove that our (n, d) construction is secure against probing up to order $n - 1$ and against algebraic attacks for $d = 1$ and $d = 2$. Remark that security against probing follows from security against computational attacks of the same order, since the underlying idea of computational attacks corresponds to side-channel attacks in the probing model.

4.1 Security Notions

In this section, we cover the security notions that we used to prove our security properties starting with the probing model. Roughly speaking, in the setting of the probing model, an adversary may invoke the (randomized) construction multiple times and adaptively choose the inputs. Prior to each invocation, the adversary may fix an arbitrary set of $t \leq n$ wires of the circuit values which can be observed during that invocation. In this paper we use a more refined security model defined as t-non-interference(t-NI) and t-strong non-interference(t-SNI) defined in [4]. We use the restatements of the definitions by Coron et al. [21].

Definition 1 (t-NI Security). *Let G be a gadget which takes as input $n+1$ shares $(x_i)_{0 \leq i \leq n}$ and outputs $n+1$ shares $(y_i)_{0 \leq i \leq n}$. The gadget G is said to be t-NI secure if for any set of t_1 probed intermediate variables and any subset $\mathcal{O} \subset [0, n]$ of output indices, such that $t_1 + |\mathcal{O}| \leq t$, there exists a subset $I \subset [0, n]$ of input indices which satisfies $|I| \leq t_1 + |\mathcal{O}|$, such that the t_1 intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Definition 2 (t-SNI Security). *Let G be a gadget which takes as input $n+1$ shares $(x_i)_{0 \leq i \leq n}$ and outputs $n+1$ shares $(y_i)_{0 \leq i \leq n}$. The gadget G is said to be t-SNI secure if for any set of t_1 probed intermediate variables and any subset $\mathcal{O} \subset [0, n]$ of output indices, such that $t_1 + |\mathcal{O}| \leq t$, there exists a subset $I \subset [0, n]$ of input indices which satisfies $|I| \leq t_1$, such that the t_1 intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

The main difference between the t-NI and t-SNI security notions is that in the latter notion the size of the input subsets I does not depend on the size of the set of probed output shares $\mathcal{O}$. Thus, t-SNI security notion ensures the input-output separation which is an essential component for composability of the gadgets.

Note that security in the probing model is a necessary but not a *sufficient* condition for a secure white-box implementation. E.g., a white-box adversary can implement an algebraic attack to recover the secret key from a masked white-box implementation.

Definition 3 (Prediction Security (d-PS), [6]). *Let $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit, $E : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ an arbitrary function, $d \geq 1$ an integer, and $\mathcal{A}$ an adversary. Consider the following security experiment:*

Algorithm 4 $\mathrm{PS}^{C,E,d}(\mathcal{A}, b)$

```
1: $(\tilde{f}, x^{[0]}, x^{[1]}, \tilde{y}) \leftarrow \mathcal{A}(C, E, d)$ where
       $\tilde{f} \in \mathcal{F}^{(d)}(C), x^{[l]} = (x_1^{[l]}, \ldots, x_Q^{[l]}), x_i^{[l]} \in \mathbb{F}_2^N, \tilde{y} \in \mathbb{F}_2^Q$
2: $(r_1, \ldots, r_Q) \xleftarrow{\$} (\mathbb{F}_2^{R_E})^Q$
3: $(\tilde{r}_1, \ldots, \tilde{r}_Q) \xleftarrow{\$} (\mathbb{F}_2^{R_C})^Q$
4: for $f \in \mathcal{F}^{(d)}(C)$ do
5:     $y(f) = (f(E(x_1^{[b]}, r_1), \tilde{r}_1), \ldots, f(E(x_Q^{[b]}, r_Q), \tilde{r}_Q))$
6: $F \leftarrow \{f \in \mathcal{F}^{(d)}(C) \mid y(f) = \tilde{y}\}$
7: if $F = \{\tilde{f}\}$ then return 1 else return 0
```

In the above experiment, $\xleftarrow{\$}$ means sampling uniformly at random. Finally, we define the advantage of an adversary $\mathcal{A}$ as

$$Adv_{C,E,d}^{PS}[\mathcal{A}] \;=\; \left|\Pr[PS^{C,E,d}(\mathcal{A}, 0) = 1] - \Pr[PS^{C,E,d}(\mathcal{A}, 1) = 1]\right|$$

The pair (C, E) is said to be d^{th} order prediction-secure (d-PS) if for any adversary $\mathcal{A}$ the advantage is negligible.

In summary, prediction security analyzes the behaviour of functions from $\mathcal{F}^{(d)}(C)$ composed with an encoding function E. Consider two elements $x, x' \in \mathbb{F}_2^N$, if an adversary is able to find a function $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ such that $f(E(x, \cdot), \cdot)$ is constant (or high-bias) but $f(E(x', \cdot), \cdot)$ is non-constant (or low-bias) then the adversary can distinguish these inputs and therefore the pair (C, E) is considered insecure. Thus, prediction security requires every function from the set $\mathcal{F}^{(d)}(C)$ to have low-bias.

The outline of proofs can be summarized as follows. First our gadgets and encoding function have been proven to satisfy ϵ-1-AS, Definition 7 and Definition 6 (resp. ϵ-2-AS Definition 9 and Definition 8). Next, the composability of our gadgets is proven using 1-AS composability result in [6] and using 2-AS composability results given in Proposition 10 and 11 respectively. Third, we prove the composability of the encoding function with an arbitrary boolean circuit in

Proposition 7 and 12. In Section 4.5 we give the quantitative bounds for the prediction security using the Proposition 1.

In [6], it is shown that a circuit C achieves d-PS, if there are enough random bits used in the circuit depending on the bias bound. Before giving the relation between random bits and d-PS, we first remark the set of functions that is needed for the analysis.

Definition 4 ([6]). *Let $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^{M}$ be a Boolean circuit, $E : \mathbb{F}_2^{N} \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ an arbitrary function and $d \geq 1$ an integer. For any function $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ and for any $x \in \mathbb{F}_2^N$ define $f_x : \mathbb{F}_2^{R_E} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2$ given by $f_x(r_e, r_c) = f(E(x, \cdot), \cdot)$ and denote the set of all such functions as $\mathcal{R}_d$:*

$$\mathcal{R}_d = \{f_x(r_e, r_c) \mid f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}, x \in \mathbb{F}_2^N\}.$$

Using the above definition, a bound for the d-PS is given as in the following proposition which indicates the required number of random bits.

Proposition 1 (Corollary 1 in [6]). *Let ϵ be the maximum bias among all functions from $\mathcal{R}_d$, i.e., $\epsilon = \max_{f_x \in \mathcal{R}_d} \ \mathcal{E}(f_x)$. Let $e = -\log_2(1/2 + \epsilon)$. Then for any adversary $\mathcal{A}$ choosing vector size of Q:*

$$Adv^{PS}_{C,E,d}[\mathcal{A}] \leq 2^{-k}$$

if $e > 0$ and $R_C \geq k \cdot (1 + 1/e)$.

Although it may seem that prediction security covers probing security (or vice versa), both notions are in fact *incomparable*. Therefore, both notions are needed to analyze a secure white-box implementation. To illustrate the incomparability of the two notions, let us consider two examples; a white-box implementation protected with an n^{th}*-order Boolean masking* and *minimalist quadratic masking* defined in [6].

Example 1 (Probing Secure Masking Vulnerable to Algebraic Attacks). Applying an `ISW` transformation to the circuit and the data results in an n^{th}-order probing secure implementation. However, a first-order algebraic attack can exploit a first-order (linear) combination of intermediate values which is equal to a predictable value. Therefore, an n^{th}-order Boolean masking is secure in the probing model, but not secure in prediction security, as shown in [6].

Example 2 (Algebraically secure masking vulnerable to probing). As the second example, we use the encoding function $\texttt{Encode}(x, x_0, x_1) = (x_0, x_1, x_0x_1 \oplus x)$. As given in [6] the masking scheme satisfies first order algebraic security. However, it is not probing secure, not even first order probing secure. A leakage is caused by the *unbalanced* sharing where the third share $x_0x_1 \oplus x$ statistically depends on the sensitive variable x. For any value x we have $\Pr_{x_0,x_1 \in_R \mathbb{F}_2}[(x_0x_1 \oplus x) = x] = 3/4$. Thus, there exists no first order function that is equal to a predictable vector, but there exists one node (the last share) that is highly correlated with a predictable vector.

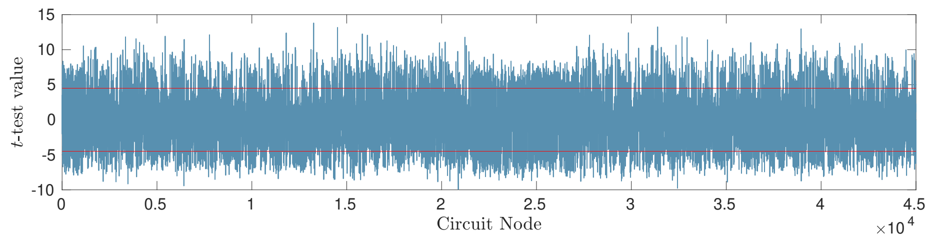

Fig. 1. A first-order leakage detection on a circuit that simulates AES-128 with the masking defined in [6]. Clearly, the t-test value exceeds the threshold values shown by red lines.

To practically verify this leakage, we implement a basic bitwise AES-128 circuit using the Sbox designed by Boyar and Peralta [13] and implement a basic leakage detection test using 500 traces with 45000 nodes ($N = 500$ and $M = 45000$). As seen in Figure 1, the test shows the intense leakage. The details of the experimental setup regarding the leakage detection, trace collection and the variable selection can be found Section 5.1.

As illustrated in Example 1, prediction security is based on finding a degree-d function whose output equals to a predictable value. However, in probing we only need to find a set of variables which depends on a predictable value as seen in Example 2. As a main result, we prove the security of our scheme in two steps:

1. Prove probing security using Definitions 1 and 2 for an arbitrary (n, d) scheme
2. Prove prediction security using Definition 3 for $(n, 1)$ and $(n, 2)$ schemes

4.2 Security Against Computational Attacks in the Probing model

We start with providing some auxiliary definitions that form the basis of our security proofs.

Definition 5 ((n, d)-family of shares). *A vector $\overline{x} = (\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ of $n + d + 1$ intermediate variables is called an* (n, d)-family of shares *if every tuple of the form $((\tilde{x}_i)_{i \in \tilde{I}}, (x_i)_{i \in I})$ such that $|\tilde{I}| \leq d + 1$ and $|I| \leq n - 1$ of $\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n$ is uniformly distributed and independent of any sensitive variable where $x = \prod_{j=0}^{d} \tilde{x}_j \oplus \bigoplus_{i=1}^{n} x_i$ is a sensitive variable.*

We can extend the definition as: two (n, d)-families of shares $\overline{x} = (\tilde{x}_0 \ldots, \tilde{x}_d, x_1, \ldots, x_n)$ and $\overline{y} = (\tilde{y}_0 \ldots, \tilde{x}_d, y_1, \ldots, x_n)$ are called to be $(n-1)$-*independent* of one another if every tuple composed of $((\tilde{x}_i)_{i \in \tilde{I}}, (x_i)_{i \in I})$ and $((\tilde{y}_j)_{j \in \tilde{J}}, (y_j)_{j \in J})$ with $|\tilde{I}|, |\tilde{J}| \leq d + 1$ and $|I|, |J| \leq n - 1$ is uniformly distributed and independent of any sensitive variable. Two (n, d)-families are $(n-1)$-*dependent* of one another if they are not $(n-1)$-independent.

To prove security of our scheme in the t-SNI notion, we decompose C into basic components, which we call *randomized elementary transformations*. Such a component gets as input two $(n-1)$-independent (n,d)-families of shares, resp. one (n,d)-family of shares, and it returns a (n,d)-family of shares.

In this section, we first prove that the randomized elementary transformations (gadgets) specified as in Algorithm 1, 2, and 3 satisfy non-interference notions. One challenge for proving t-SNI security results from the fact that in the proposed sharing only *a subset of shares* is uniformly distributed. The product of non-linear shares x_0 (as expressed by $x_0 = \prod_{j=0}^{d} \tilde{x}_j$) is non-uniformly distributed or *biased*. Hence, x_0 can be predicted correctly by an adversary with high probability. Thus, the non-linear shares do not contribute to security against probing attacks. To address this fact, we consider the non-linear shares as public values accessible by the adversary or as *free probes*, due to the bias of their product. We use the following fact in our proofs.

Fact 1. *Let G be a masked operation that operates on an (n,d) family of shares $\overline{x} = (\tilde{x}_0, \dots, \tilde{x}_d, x_1, \dots, x_n)$ (or two (n,d) family of shares $\overline{x} = (\tilde{x}_0, \dots, \tilde{x}_d, x_1, \dots, x_n)$ and $\overline{y} = (\tilde{y}_0, \dots, \tilde{y}_d, y_1, \dots, y_n)$) as defined in Definition 5. A simulator can access non-linear shares $(\tilde{x}_0, \dots, \tilde{x}_d)$ (or $(\tilde{x}_0, \dots, \tilde{x}_d)$ and $(\tilde{y}_0, \dots, \tilde{y}_d)$) as free probes or public inputs to the gadgets.*

Now we are ready to prove the security of our basic constructions. We start with the t-SNI property of the `RefreshMask` and `And` gadgets. Then, we continue with the t-NI property of the `Xor` gadget. The proofs can be found in Appendix A.

Proposition 2. *(t-SNI of `RefreshMask`) Let $\overline{x} = (\tilde{x}_0, \dots, \tilde{x}_d, x_1, \dots, x_n)$ be an (n,d)-family of shares, with $n \geq 2$, as input of Algorithm 3 to refresh masking. Then every tuple of t_1 intermediate variables and t_2 output variables in Algorithm 3 such that $t_1 + t_2 \leq t$ can be simulated by at most t_1 linear shares taken from $\overline{x}$.*

Proposition 3. *(t-SNI of `And`) Let $\overline{x} = (\tilde{x}_0, \dots, \tilde{x}_d, x_1, \dots, x_n)$ and $\overline{y} = (\tilde{y}_0, \dots, \tilde{y}_d, y_1, \dots, y_n)$ be two $(n-1)$-independent (n,d)-families of shares, with $n \geq 2$, inputs of Algorithm 2 for `And`. Then every tuple of t_1 intermediate variables and t_2 output variables such that $t_1 + t_2 \leq t$ can be simulated by at most t_1 linear shares taken from $\overline{x}$ and $\overline{y}$.*

Proposition 4. *(t-NI of `Xor`) Let $\overline{x} = (\tilde{x}_0, \dots, \tilde{x}_d, x_1, \dots, x_n)$ and $\overline{y} = (\tilde{y}_0, \dots, \tilde{y}_d, y_1, \dots, x_n)$ be two $(n-1)$-independent (n,d)-families of shares, with $n \geq 2$, as input of Algorithm 1 to compute `Xor`. Then every tuple of t_1 intermediate variables and t_2 output variables such that $t_1 + t_2 < t$ can be simulated by at most $t_1 + t_2$ linear shares taken from $\overline{x}$ and $\overline{y}$.*

In conclusion, we prove the security against t-probes of our individual gadgets such that $t < n$. Next, we analyze an arbitrary circuit C as a combination of our gadgets. As stated in [4], an algorithm is said to be t-NI if all gadgets are t-NI and every non-linear usage of a secret state is guarded by t-SNI refreshing gadgets. Moreover, it is sufficient to make the algorithm t-SNI, if every input or the output of a t-NI gadget is processed by a t-SNI gadget. Since the `RefreshMask` operation is proven to be secure in the t-SNI notion, we can use the operation defined in Section 3.1 to generate an arbitrary circuit that is secure against $(n-1)^{th}$-order probing attacks and therefore secure against $(n-1)^{th}$-order computational attacks.

Experimental Verification: To support the results, we provide an experimental verification of the gadgets `And`$[n,d]$, `Xor`$[n,d]$ and `RefreshMask`$[n,d]$ for $d=1$ and 2 and for $n=1,2,3,4$ and 5 using the tool MaskVerif [3]. We implement our masking scheme (with the given orders n and d) inside the tool and experimentally verify the security features of our gadgets. The implementation of our scheme that can be used with MaskVerif is available as open source [2]. The experiments are run on on Intel Core i5-6400 CPU@ 2.70GHz and a summary of the experimental results can be found in Table 2.

4.3 Algebraic Security of the $(n, 1)$-Masking Scheme

In this section, we analyze the prediction security (Def. 3) of our $(n,1)$-masking scheme using the gadgets in Section 3.1. The security proofs use two auxiliary notions: algebraic circuit security which deals which he security of any Boolean function C and algebraic encoding security which deals with the security of the encoding function E:

Definition 6 (Algebraic Encoding Security (ϵ-1-AS)). *Let $E(x,r):\mathbb{F}_2^N\times\mathbb{F}_2^{R_E}\to\mathbb{F}_2^{N'}$ be an arbitrary encoding function. Let $\mathcal{Y}$ be the set of functions given by the output bits of E. The function E is called 1^{st}-order algebraically ϵ-secure (ϵ-1-AS) if for any $f\in\mathcal{Y}^{(1)}\setminus\{\mathbf{0},\mathbf{1}\}$ and for any $x\in\mathbb{F}_2^N$ the bias of the function $f(x,\cdot):\mathbb{F}_2^{R_E}\to\mathbb{F}_2$ is not greater than ϵ:*

$$\max_{f\in\mathcal{Y}^{(1)}\setminus\{\mathbf{0},\mathbf{1}\},x\in\mathbb{F}_2^N}\mathcal{E}(f(x,\cdot))\le\epsilon$$

Definition 7 (Algebraic Circuit Security (ϵ-1-AS)). *Let $C(x,r):\mathbb{F}_2^{N'}\times\mathbb{F}_2^{R_C}\to\mathbb{F}_2^M$ be a Boolean circuit and and let ϵ be a real number, with $0\le\epsilon<1/2$. Then C is called first-order algebraically ϵ-secure (ϵ-1-AS) if for any $f\in\mathcal{F}^{(1)}(C)\setminus\{\mathbf{0},\mathbf{1}\}$ one of the following conditions holds:*

(a) f is an affine function of x,
(b) for any $x\in\mathbb{F}_2^N$, $\mathcal{E}(f(x,\cdot))\le\epsilon$ where $f(x,\cdot):\mathbb{F}_2^{R_C}\to\mathbb{F}_2$,

where $\mathcal{E}(\cdot)$ represents the bias of a Boolean function $g:\mathbb{F}_2^\ell\to\mathbb{F}_2$ i.e., $\mathcal{E}(g)=|1/2-wt(g)/2^\ell|$ and $wt(g)$ is the weight of g, i.e., the number of nonzero entries of its truth table.

[2] `https://github.com/UzL-ITS/white-box-masking`

Table 2. A summary of SNI/NI/Probing security verification of our gadgets. The inputs of an (n,d) gadget are two $(n-1)$-independent family of shares $\bar{x}$ and $\bar{y}$ (resp. one $(n-1)$-independent families of shares $\bar{x}$ for `RefreshMask` or `RefM` in short). The number of observations (#Obs.) represents the total number of (intermediate and output) variables within the specified gadget. The timing corresponds to the total time for MaskVerif to verify the SNI, NI and probing security notions, respectively.

	Free Probes	# Obs.	SNI	NI	Probing
`RefM`$[2,1]$	$[\tilde{x}_0,\tilde{x}_1]$	23	0.01s	0.01s	0.01s
`Xor`$[2,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	16	-	0.01s	0.05s
`And`$[2,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	52	0.01s	0.01s	¡0.01s
`RefM`$[3,1]$	$[\tilde{x}_0,\tilde{x}_1]$	30	0.01s	0.01s	0.01s
`Xor`$[3,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	19	-	0.01s	0.01s
`And`$[3,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	86	0.02s	0.02s	0.02s
`RefM`$[4,1]$	$[\tilde{x}_0,\tilde{x}_1]$	40	0.02s	0.01s	¡0.01s
`Xor`$[4,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	22	-	0.01s	0.01s
`And`$[4,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	123	0.06s	0.05s	0.05s
`RefM`$[5,1]$	$[\tilde{x}_0,\tilde{x}_1]$	53	0.05s	0.01s	0.01s
`Xor`$[5,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	48	-	0.01s	0.01s
`And`$[5,1]$	$[\tilde{x}_0,\tilde{x}_1]$, $[\tilde{y}_0,\tilde{y}_1]$	170	2.25s	0.59s	0.45s
`RefM`$[2,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$	55	0.01s	0.01s	0.01s
`Xor`$[2,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	24	-	¡0.01s	0.01s
`And`$[2,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	98	0.02s	0.01s	0.01s
`RefM`$[3,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$	62	0.01s	0.01s	0.01s
`Xor`$[3,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	27	-	0.01s	0.01s
`And`$[3,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	154	0.03s	0.01s	0.02s
`RefM`$[4,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$	72	0.02s	0.01s	0.01s
`Xor`$[4,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	30	-	0.01s	0.01s
`And`$[4,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	215	0.61s	0.08s	0.07s
`RefM`$[5,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$	85	0.04s	0.01s	0.01s
`Xor`$[5,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	33	-	0.01s	0.01s
`And`$[5,2]$	$[\tilde{x}_0,\tilde{x}_1,\tilde{x}_2]$, $[\tilde{y}_0,\tilde{y}_1,\tilde{y}_2]$	284	10.47s	1.06s	1.11s

ϵ-1-AS of the Gadgets Using the above definitions, we employ the following methodology to prove the 1-PS of our scheme. We first divide the circuit into smaller circuits (namely `And`, `Xor` and `RefreshMask` gadgets as defined in Section 3.1) and show that the gadgets satisfy Definition 6. This gives us a bias bound for the individual gadgets. Using the 1-AS composability result in [6] we make sure that any composition of our gadgets (C) is also 1-AS. Finally, we combine C with encoding function E and using Proposition 1 we complete the 1-PS security proof of our scheme.

While proving algebraic encoding security is quite straightforward, proving algebraic circuit security needs significant attention. The methodology to prove algebraic circuit security in [6] can be divided into two steps. The first step consists of showing $\mathcal{E}(f(x,r)) \neq 1/2$ for all $f \in \mathcal{F}^{(1)}(C)$ and for all $x \in \mathbb{F}_2^N$ except for constant functions and affine functions of x. A verification algorithm is provided in [6]. The provided algorithm generates a truth table by evaluating the circuit on all possible inputs and records each node in the circuit. Another truth table is formed by selecting the values where the input is fixed $x = c$. That is, the second truth table corresponds to the values of the circuit nodes where the input x is fixed to a value c while r takes all possible values. Observe that the latter truth table is a subset of the former one. Finally, the algorithm compares the dimensions of the basis of the truth tables for each restriction, to check if there is a constant function f when the input is fixed to a value c.

The second step is to find the maximum degree term (i.e. node in the circuit) and calculate the corresponding bias bound. As proven in [40], the degree of a Boolean function gives us a boundary for the weight of the function such that $wt(g) \leq 2^{N-\deg(g)}$, where N is the number of inputs of the function g. Using this bound we can analyze the bias bound of a function $f \in \mathcal{F}^{(1)}(C)$. Observe that the maximum degree of f is equal to the maximum degree node in C, since f contains *only* linear combinations of the nodes. That is, for all $f \in \mathcal{F}^{(1)}(C)$, $\deg(f) \leq \max(\deg(c_i)_{c_i \in C})$ and thus $wt(f) \geq 2^{N-\max(\deg(c_i)_{c_i \in C})}$. Using this minimum weight value, the linear-bias bound of the gadget can be calculated as:

$$\epsilon = \left|\frac{1}{2} - \frac{wt(f)}{2^N}\right| \leq \left|\frac{1}{2} - \frac{2^{N-\deg(f)}}{2^N}\right| = \left|\frac{1}{2} - \frac{1}{2^{\deg(f)}}\right|. \tag{4}$$

Due to the first part of the proof, we know that there are no constant functions and therefore the bias cannot grow.

Using the discussion above, we will prove the security of our gadgets by showing that there exists no constant function $f(x, \cdot) \in \mathcal{F}^{(1)}(C)$ for all $x \in \mathbb{F}_2^N$ and by calculating the corresponding bias boundary of the gadgets. We start with the first order algebraic security proof for a `RefreshMask`$[n, 1]$ gadget that uses the construction given in Section 3.1.

Proposition 5. *Let C be the circuit representation of the* `RefreshMask` *gadget using a masking scheme with an arbitrary order n and a fixed degree $d = 1$. C takes as input $n + 2$ shares $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \leq i \leq n})$ and outputs $n + 2$ shares $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \leq i \leq n})$. The gadget* `RefreshMask`$[n, 1]$ *is ϵ-1-AS with $\epsilon := 1/4$.*

The proof of Proposition 5 can be found in Appendix A. We proceed with first order algebraic security proof for an `And`$[n, 1]$ gadget that uses the construction given in Section 3.1.

Proposition 6. *Let C be the circuit representation of the* `And` *gadget using a masking scheme with an arbitrary order n and a fixed degree $d = 1$. C takes as input $n + 2$ shares $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \leq i \leq n})$ and $(\tilde{y}_0, \tilde{y}_1, (y_i)_{1 \leq i \leq n})$ and outputs $n + 2$ shares $(\tilde{z}_0, \tilde{z}_1, (z_i)_{1 \leq i \leq n})$. The gadget* `And`$[n, 1]$ *is ϵ-1-AS with $\epsilon := 7/16$.*

Proof. In the first part of the proof, we show that there exists no function $f \in \mathcal{F}^{(1)}(C)$ such that f is constant when inputs are fixed.

First, let us the reformulate the circuit C as follows:

$$C \colon ((\mathbb{F}_2^{n+2} \times \mathbb{F}_2^{n+2}), \mathbb{F}_2^{R_C}) \to \mathbb{F}_2^{n+2}$$
$$((\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \leq i \leq n}), (\tilde{y}_0, \tilde{y}_1, (y_i)_{1 \leq i \leq n}), \bar{r}) \mapsto (\tilde{z}_0, \tilde{z}_1, (z_i)_{1 \leq i \leq n}).$$

where $\bar{r}$ denotes the set of randomness that is used in the circuit. Next, we define three classes of edges within the circuit:

- R: The set of random bits,
- B: The set of linear shares i.e. x_i and y_j for all $1 \leq i, j \leq n$,
- M: The set of non-linear shares i.e. $\tilde{x}_0$, $\tilde{x}_1$, $\tilde{y}_0$ and $\tilde{y}_1$.

Using the above classification we can analyze the nodes $c_i \in C$ with respect to their input edges. We define the nodes as $c_i : (u_i^1, u_i^2) \mapsto v_i$ where $u_i^1,\ u_i^2 \in \mathbb{F}_2$ represent the input bits of the node and $v_i \in \mathbb{F}_2$ represents the output bit of the node. The classification of the nodes can be listed as follows;

1. $u_i^1 \in \mathsf{R}$ or $u_i^2 \in \mathsf{R}$,
2. $u_i^1 \in \mathsf{B}$ or $u_i^2 \in \mathsf{B}$,
3. $u_i^1 \in \mathsf{M}$ and $u_i^2 \in \mathsf{M}$.

Assume that there exists a function $f \in \mathcal{F}^{(1)}(C)$ such that f is constant when the inputs $\overline{x}$ and $\overline{y}$ are fixed. We can represent the function as $f = \bigoplus_{i \in I} v_i$ where $I \subseteq C$. Remark that the input shares are randomized, since they are first processed by `RefreshMask` gadgets. Therefore f should include a reconstructed combination of the shares i.e., f should include a combination of nodes such that $\tilde{x}_0\tilde{x}_1 \oplus x_1 \oplus \cdots \oplus x_n$ (resp. $\tilde{y}_0\tilde{y}_1 \oplus y_1 \oplus \cdots \oplus y_n$) is formed.

Any linear combination of the nodes of 1 and 2 cannot be constant due to `RefreshMask` gadgets, since either a node is random (non-fixed by definition) or the node corresponds to linear masking (randomized by `RefreshMask`). Therefore, f should include at least one node from the 3^{rd} class to form the reconstructed multiplicative representation: x_0 or y_0. Clearly, the nodes from the 3^{rd} class can be found in **Step-1** and **Step-2(a)** where the following computations are processed:

- $\tilde{z}_0$ and $\tilde{z}_1$,
- $\tilde{x}_1(\tilde{x}_0 y_j \oplus r^{0,j}\tilde{y}_0) = \tilde{x}_1\tilde{x}_0 y_j \oplus r^{0,j}\tilde{x}_1\tilde{y}_0$ for $1 \leq j \leq n$,
- $\tilde{y}_1(\tilde{y}_0 x_j \oplus r^{1,j}\tilde{x}_0) = \tilde{y}_1\tilde{y}_0 x_j \oplus r^{1,j}\tilde{y}_1\tilde{x}_0$ for $1 \leq j \leq n$.

The use of parenthesis indicates the order in which the nodes are used in the above equations. The resulting order eliminates the generation of an affine function of x_0 or y_0 (the shares represented by $\tilde{x}_0, \tilde{x}_1$ and $\tilde{y}_0, \tilde{y}_1$ respectively), although these nodes calculate the correct function ($r_{j,0}$ as seen in Equation (2)). Any linear combination of these nodes cannot be constant and thus there exists no constant function $f \in \mathcal{F}^{(1)}(C)$ when the inputs are fixed.

In the second part, we examine the highest degree term in the gadget and find the corresponding bias. For `And`$[n, 1]$ the maximum degree term can be found in line 16 of Algorithm 2. Specifically, $x_n y_n$ which contains a node of the form $\tilde{r}_0^x\tilde{r}_1^x\tilde{r}_0^y\tilde{r}_1^y$ where $\tilde{r}_0^x, \tilde{r}_1^x$ (resp. $\tilde{r}_0^y, \tilde{r}_1^y$) are the randomness used in `RefreshMask`$(\overline{x})$ (resp. `RefreshMask`$(\overline{y})$). Clearly the corresponding bias and the bias bound of the gadget can be calculated as 2^{-4} and $\epsilon \leq \left|1/2 - 1/2^4\right| = 7/16$ respectively. Thus the `And` gadget is ϵ-1-AS with $\epsilon := 7/16$.

Table 3. First-order algebraic security verification of individual gadgets. Input corresponds to the number of shares for both inputs (i.e. $2(n+2)$). Random states the number of random values (R_C) within the circuit and it is calculated by the randomness requirement of two `RefreshMask` gadgets and additional randomness in the gadget. The number of intermediate variables represents the number of nodes in the gadget.

	Max degree	Bias Bound	Input	Random	Intermediate	Time
`Xor`$[1,1]$	2	1/4	6	6	8	3.5s
`And`$[1,1]$	4	7/16	6	6	12	4s.
`Xor`$[2,1]$	2	1/4	8	8	8	45.7s
`And`$[2,1]$	4	7/16	8	13	24	$\approx$ 114min
`Xor`$[3,1]$	2	1/4	10	10	8	$\approx$ 17min
`And`$[3,1]$	4	7/16	10	19	36	$\approx$ 5 days

Although we are not giving a proof for the `Xor` gadget, the same discussion can be carried out and it can be shown that the `Xor`$[n,1]$ gadget is ϵ-1-AS with $\epsilon := 1/4$. We provide experimental verification of the first order gadgets, including the `Xor` gadget next.

Experimental Verification: To support the results, we provide experimental verification of the first order gadgets `And`$[n,1]$ and `Xor`$[n,1]$ (and inherently `RefreshMask`$[n,1]$) for $n = 1, 2$ and 3 using the tool given in [6][3]. First we adapt our scheme to work with the tool, i.e. we implement our masking scheme (with the given orders n and d) as a class inside the tool. We then run the verification algorithm as explained above. The updated version of the tool including our scheme is available as open source [4].

We confirm the first order algebraic security of our scheme for different orders. Details are shown in Table 3. The algorithm is run on an Intel Xeon Silver 4114 CPU@2.20GHz and, as shown in the table, the time that algorithm takes increases exponentially with the increasing number of nodes within the gadgets. The bias bound does not depend on the linear degree n, since the maximum degree term is found within the terms that depend on the non-linear degree d.

Encoding-Circuit Composability In the last part of the security analysis, we use the composability result of ϵ-1-AS given in [6]. Since the gadgets (`Xor`$[n,1]$, `And`$[n,1]$, `RefreshMask`$[n,1]$, as defined in Section 3.1) are ϵ-1-AS, an arbitrary combination of these gadgets is also ϵ-1-AS by Proposition 4 in [6]. Moreover, we observe that `Encode`$[n,1]$ is ϵ-1-AS with $\epsilon := 1/2^2$ according to Definition 6. Recall that for an $(n,1)$ scheme the highest degree term is found in the last share: $x_n = x \oplus \tilde{x}_0\tilde{x}_1 \oplus \bigoplus_{i=1}^{n-1} x_i$ and clearly no linear combination of $(\tilde{x}_0, \tilde{x}_1, x_1, \ldots, x_n)$ is constant. Thus, for `Encode`$[n,1]$ $\forall f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and $\forall x \in \mathbb{F}_2^N$ the bias of

[3] `https://github.com/cryptolu/whitebox`
[4] `https://github.com/UzL-ITS/white-box-masking`

$f(x,\cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ':

$$\max_{f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}, x \in \mathbb{F}_2^N} \mathcal{E}(f(x,\cdot)) \leq 1/4.$$

Finally, we can combine our construction with the encoding-circuit linear-composability result from [6].

Proposition 7. *Let $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit, and let $E : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be a function. If E is encoding ϵ-1-AS and C is circuit ϵ-1-AS then, for $d = 1$, it is true:*

$$\max_{f_x \in \mathcal{R}_d} \mathcal{E}(f_x) \ \leq \ \epsilon,$$

where $\mathcal{R}_d$ is defined in Definition 4.

4.4 Algebraic Security of the $(n, 2)$-Masking Scheme

To prove the second-order prediction security (2-PS) of the $(n, 2)$-masking scheme we proceed as follows. For our encoding function E and any Boolean circuit C constructed from gadgets as defined in Section 3.1 we will analyze, for $d = 2$, the maximum bias over all functions in the set $\mathcal{R}_d$ defined in Definition 4. The main result of this section is that the maximum bias over all such functions is small. To obtain a bound on the prediction security of the $(n, 2)$-masking, we combine this bias with the upper bound on 2-PS from Proposition 1:

$$\mathrm{Adv}^{\mathrm{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{-k},$$

assuming $e = -\log_2(1/2 + \epsilon) > 0$ and the number of random bits of the circuit C is $\geq k \cdot (1 + 1/e)$, where, recall, $\epsilon = \max_{f_x \in \mathcal{R}_d} \ \mathcal{E}(f_x)$. To prove that for all f_x in $\mathcal{R}_2$ the bias $\mathcal{E}(f_x)$ is small, we use auxiliary definitions which extend the notion of the first-order algebraic security defined in [6] in a non-trivial way.

Definition 8 (Algebraic Encoding Security (ϵ-2-AS)). *Let $E(x, r) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary encoding function. Let $\mathcal{Y}$ be the set of functions given by the output bits of E and let ϵ be a real number, with $1/4 \leq \epsilon < 1/2$. The function E is called second-order algebraically ϵ-secure (ϵ-2-AS) if, for $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$, it is true that:*

1. *$\forall f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}$ and $\forall x \in \mathbb{F}_2^N$ the bias of $f(x,\cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ':*
$$\max_{f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}, x \in \mathbb{F}_2^N} \mathcal{E}(f(x,\cdot)) \leq \epsilon'.$$
2. *$\forall f \in \mathcal{Y}^{(2)} \setminus \{\mathbf{0},\mathbf{1}\}$ and $\forall x \in \mathbb{F}_2^N$ the bias of $f(x,\cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ:*
$$\max_{f \in \mathcal{Y}^{(2)} \setminus \{\mathbf{0},\mathbf{1}\}, x \in \mathbb{F}_2^N} \mathcal{E}(f(x,\cdot)) \leq \epsilon.$$

Definition 9 (Algebraic Circuit Security (ϵ-2-AS)). *Let $C(x,r) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit and let ϵ be a real number, with $1/4 \leq \epsilon < 1/2$. Then C is called second-order algebraically ϵ-secure (ϵ-2-AS) if*

1. *C is ϵ'-1-AS, with $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$, and*
2. *for any function $f \in \mathcal{F}^{(2)}(C \setminus \mathcal{X}) \setminus \{\mathbf{0}, \mathbf{1}\}$ and for all $x \in \mathbb{F}_2^N$ it holds: $\mathcal{E}(f(x,\cdot)) \leq \epsilon$, where $f(x,\cdot) : \mathbb{F}_2^{R_C} \to \mathbb{F}_2$.*

The rest of this section is organised as follows. First, we estimate the values ϵ for ϵ-2-AS of the basic gadgets `RefreshMask`, `And`, and `Xor`. Then we prove the composability result, i.e., that combining ϵ-2-AS circuits leads to the ϵ-2-AS composed circuit. Finally, we show that our encoding scheme E is ϵ-2-AS, with $\epsilon := 7/16$, and that from the ϵ-2-AS security of C we get an estimation on $\max_{f_x \in \mathcal{R}_d} \mathcal{E}(f_x)$, for $d = 2$.

ϵ-2-AS of the Gadgets Using the new definition we will prove the second order prediction security of our $(n, 2)$ basic gadgets. We first show that there exists no constant function $f(c,\cdot) \in \mathcal{F}^{(2)}(C)$ for all $c \in \mathbb{F}_2^N$. In the second step, we calculate the corresponding *first-order* and *second-order* bias bounds. We start with the ϵ-2-AS of the `RefreshMask`$[n, 2]$ gadget.

Proposition 8. *Let C be the circuit representation of the `RefreshMask` gadget using a masking scheme with an arbitrary order n and a fixed degree $d = 2$. C takes as input $n+3$ shares $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1\leq i\leq n})$ and outputs $n+3$ shares $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1\leq i\leq n})$. The gadget `RefreshMask`$[n, 2]$ is ϵ-2-AS with $\epsilon := 31/64$.*

The proof of Proposition 8 can be found in Appendix A. Next, we prove the second order algebraic security of `And`$[n, 2]$ gadget.

Proposition 9. *Let C be the circuit representation of the `And` gadget using a masking scheme with an arbitrary order n and a fixed degree $d = 2$. C takes as input $n+3$ shares $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1\leq i\leq n})$, $(\tilde{y}_0, \tilde{y}_1, \tilde{y}_2, (y_i)_{1\leq i\leq n})$ and outputs $n+3$ shares $(\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, (z_i)_{1\leq i\leq n})$. The gadget `And`$[n, 2]$ is ϵ-2-AS with $\epsilon := (1/2 - 1/2^{12})$.*

Proof. Similar to the proof of Proposition 6, we reformulate the circuit C as follows:

$$C\colon ((\mathbb{F}_2^{n+3} \times \mathbb{F}_2^{n+3}), \mathbb{F}_2^{R_C}) \to \mathbb{F}_2^{n+3}$$
$$((\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, (x_i)_{1\leq i\leq n}), (\tilde{y}_0, \tilde{y}_1, \tilde{y}_2, (y_i)_{1\leq i\leq n}), \bar{r}) \mapsto (\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, (z_i)_{1\leq i\leq n}).$$

Next we use the classification of the nodes that we used in the proof of Proposition 6:

- R: The set of random bits,
- B: The set of linear shares i.e. x_i and y_j for all $1 \leq i, j \leq n$,
- M: The set of non-linear shares i.e. $\tilde{x}_0$, $\tilde{x}_1$, $\tilde{x}_2$, $\tilde{y}_0$, $\tilde{y}_1$ and $\tilde{y}_2$.

Using the above classification we can analyze the nodes $c_i \in C \setminus \mathcal{X}$ with respect to its input edges. We define the nodes as $c_i : (u_i^1, u_i^2) \mapsto v_i$ where $u_i^1,\ u_i^2 \in \mathbb{F}_2$ represent the input bits of the node and $v_i \in \mathbb{F}_2$ represents the output bit of the node. The classification of the depending nodes is as follows, (1) $u_i^1 \in \mathsf{R}$ or $u_i^2 \in \mathsf{R}$, (2) $u_i^1 \in \mathsf{B}$ or $u_i^2 \in \mathsf{B}$, (3) $u_i^1 \in \mathsf{M}$ and $u_i^2 \in \mathsf{M}$.

Assume that there exists a function $f \in \mathcal{F}^{(2)}(C \setminus \mathcal{X})$ such that $f(x, y, \cdot)$ is constant for a pair of *fixed* inputs $x, y \in \mathbb{F}_2^{n+3}$. As in Proposition 6, input shares are randomized, due to the initial `RefreshMask` gadgets. Therefore, f should include a reconstructed combination of the shares i.e., f should include a combination of nodes such that $\tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus x_1 \oplus \cdots \oplus x_n$ (resp. $\tilde{y}_0\tilde{y}_1\tilde{y}_2 \oplus y_1 \oplus \cdots \oplus y_n$) is formed. This observation indicates that f should include nodes from the third class which can be found in **Step-1** and **Step-2(a)**. However, the nodes $c_i \in C \setminus \mathcal{X}$ contain at most one value from each multiplicative representation i.e., each node contains only one non-linear share. Thus, any first or second order combination cannot contain all three non-linear shares and $f(x, y, \cdot)$ cannot be fixed for all $x, y \in \mathbb{F}_2^{n+3}$.

In the second part of the proof we examine the highest degree term in the circuit. The maximum degree term can be found in line 16 of Algorithm 2 for $\texttt{And}[n, 2]$. We can see that the maximum degree term for this case is 6. Thus the linear bias bound of the gadget can be seen as follows:

$$\mathcal{E}(f') < \epsilon' := \frac{1}{2} - \frac{1}{2^6} \text{ , where } f' \in \mathcal{F}^{(1)}(C)$$

Thus, $\texttt{And}[n, 2]$ is ϵ'-1-AS. Using the same argument, we can see that the maximum degree term $f \in \mathcal{F}^{(2)}(C)$ is less than or equal to 12. This result is followed by:

$$\mathcal{E}(f) \leq \epsilon := \frac{1}{2} - \frac{1}{2^{12}} \text{ , where } f \in \mathcal{F}^{(2)}(C).$$

Observe that in the first part of the proof we showed that there exists no function $f \in \mathcal{F}^{(2)}(C)$ such that $f(c, \cdot)$ is constant, which implies both linear and second order biases cannot grow. Thus $\texttt{And}[n, 2]$ gadget is ϵ-2-AS circuit where $\epsilon := 1/2 - 1/2^{12}$.

Using the same idea we can prove that the $\texttt{Xor}[n, 2]$ gadget is a ϵ-2-AS circuit with $\epsilon := 1/2 - 1/2^6$.

Circuit Composability In the previous subsection, we have shown that our basic gadgets `RefreshMask`, `And`, and `Xor` are ϵ-2-AS, for some specific values ϵ. Now, we prove that any circuit obtained by the composition of such gadgets remains ϵ-2-AS. To cover all cases, we consider separately the *parallel* (Proposition 10) and the *sequential* (Proposition 11) composability of two circuits. In particular, from Proposition 10 we can follow, e.g., that if one applies, e.g., $\texttt{And}(\overline{x}, \overline{y})$ for inputs $\overline{x} = \texttt{And}(\overline{x}_1, \overline{y}_1)$ and $\overline{y} = \texttt{Xor}(\overline{x}_2, \overline{y}_2)$ then, from Proposition 10, the parallel composition:

$$\texttt{And}(\overline{x}_1, \overline{y}_1); \texttt{Xor}(\overline{x}_2, \overline{y}_2)$$

with input $\overline{x}_1, \overline{y}_1, \overline{x}_2, \overline{y}_2$ and output $\overline{x}, \overline{y}$ is an ϵ-2-AS circuit. Moreover, due to Proposition 11, we get that

$$\texttt{And}(\texttt{And}(\overline{x}_1, \overline{y}_1), \texttt{Xor}(\overline{x}_2, \overline{y}_2))$$

remains ϵ-2-AS.

Proposition 10 (ϵ-2-AS-Circuit-Parallel-Composability). *Assume $C_1(x_1, r_1)$ and $C_2(x_2, r_2)$ are two (disjoint) ϵ-2-AS circuits. Let C be the circuit obtained by parallel composition of C_1 and C_2, i.e. by considering the input of C_1 and the input of C_2 as the input of C and, analogously, the output of C_1 and the output of C_2 as the output of C. Moreover let r_1 and r_2 be the extra random input of C:*

$$C(x_1, x_2, (r_1, r_2)) = (C_1(x_1, r_1), C(x_2, r_2)).$$

Then $C(x_1, x_2, (r_1, r_2))$ is also an ϵ-2-AS circuit.

Proof. Assume C_1 and C_2 are ϵ-2-AS and let $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$. From [6] we know that the composition C is ϵ'-1-AS. Thus, C satisfies the condition 1 in Definition 9. To see that the condition 2 is true as well, let us consider a function $f(x_1, x_2, (r_1, r_2)) \in \mathcal{F}^{(2)}(C \setminus \mathcal{X}) \setminus \{\mathbf{0}, \mathbf{1}\}$, where $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$ denotes all input nodes of C and $\mathcal{X}_i$ all input nodes of C_i, $i = 1, 2$. We will show that for any x_1, x_2, the bias is bounded as follows

$$\mathcal{E}(f(x_1, x_2, \cdot, \cdot)) \leq \epsilon.$$

Assume x_1 and x_2 are arbitrary, but fixed inputs. To simplify the notation, let $\tilde{f}$ denote the function $\tilde{f}(r_1, r_2) = f(x_1, x_2, (r_1, r_2))$. In the most general case, $\tilde{f}$ has the form

$$\tilde{f}(r_1, r_2) \;=\; c \;\oplus\; u_2(r_1) \;\oplus\; v_2(r_2) \;\oplus\; \sum_{i=1}^{\tau} u_1^i(r_1)\; v_1^i(r_2), \tag{5}$$

where $c \in \{\mathbf{0}, \mathbf{1}\}$ is a constant and u_2, v_2, u_1^i, and v_1^i are functions such that:

$$u_2 \in \mathcal{F}^{(2)}(C_1 \setminus \mathcal{X}_1) \setminus \{\mathbf{0}, \mathbf{1}\}, \quad v_2 \in \mathcal{F}^{(2)}(C_2 \setminus \mathcal{X}_2) \setminus \{\mathbf{0}, \mathbf{1}\}, \text{ and}$$
$$u_1^i \in \mathcal{F}^{(1)}(C_1 \setminus \mathcal{X}_1) \setminus \{\mathbf{0}, \mathbf{1}\}, \quad v_1^i \in \mathcal{F}^{(1)}(C_2 \setminus \mathcal{X}_2) \setminus \{\mathbf{0}, \mathbf{1}\}, \text{ for } i = 1, \ldots \tau,$$

To prove that the bias of $\tilde{f}$ is bounded by ϵ, we consider two cases.

Case 1: u_2 or v_2 in Eq. (5) **are non-trivial.** Let w.l.o.g. v_2 be non-trivial, i.e., that it contains at least one term. For every fixed (but arbitrary) r_1, function (5) can be represented as a function in $\mathcal{F}^{(2)}(C_2 \setminus \mathcal{X}_2)$ as follows:

$$\tilde{f}_{r_1}(r_2) \;=\; c \;\oplus\; c_0 \;\oplus\; v_2(r_2) \;\oplus\; \sum_{i=1}^{\tau} c_i\; v_1^i(r_2),$$

where $c_0 = v_2(r_1)$ and, in case $\tau \geq 1$, $c_i = v_1^i(r_1)$, for $i = 1, \ldots, \tau$. By assumption that C_2 is ϵ-2-AS we get that $\mathcal{E}(\tilde{f}_{r_1}(r_2)) \leq \epsilon$ and since this bound is true for every r_1, we can conclude that the function $\tilde{f}$, as defined in (5), has the bias bounded by ϵ, too.

Case 2: $\tilde{f}$ in Eq. (5) **has the form $\tilde{f}(r_1, r_2) = c \oplus \sum_{i=1}^{\tau} u_1^i(r_1)\ v_1^i(r_2)$, with $\tau \geq 1$.** Now, for every fixed r_1, function (5) can be represented as a function in $\mathcal{F}^{(1)}(C_2 \setminus \mathcal{X}_2)$ as follows:

$$\tilde{f}_{r_1}(r_2) \ = \ c \ \oplus \ \sum_{i=1}^{\tau} c_i \ v_1^i(r_2),$$

where $c_i = v_1^i(r_1)$, for $i = 1, \dots, \tau$. If for every r_1 it would be true that some $c_i \neq 0$, then we could deduce immediately that the bias of $\tilde{f}$ is bounded by $\epsilon' \leq \epsilon$. Unfortunately, it can happen that for some strings r_1, all coefficients c_i vanish implying that $\tilde{f}_{r_1}(\cdot)$ has bias 1/2. Below, we argue that there are sufficiently many values for r_1 such that at least one coefficient c_i is nonzero. In consequence we will be able to bound the bias of $\tilde{f}$ in this case.

Consider the coefficient c_1. Recall that it is defined as a function $c_1 = u_1^1(r_1)$ in $\mathcal{F}^{(1)}(C_1 \setminus \mathcal{X}_1)$. From the assumption, the bias of $u_1^1(\cdot)$ is bounded as follows

$$\mathcal{E}(u_1^1(r_1)) = |1/2 - wt(u_1^1)/2^{|r_1|}| \leq \epsilon'.$$

From this inequality, one can deduce that the number $wt(u_1^1)$ of values for r_1, for which $c_1 = u_1^1(r_1) = 1$, is at least $wt(u_1^1) \geq 2^{|r_1|}\,(1/2 - \epsilon')$. Let, for short, $R_1 := \{r_1 \mid c_1 = u_1^1(r_1) = 1\}$ denote the set of all such strings r_1. Its cardinality is

$$|R_1| \geq 2^{|r_1|}\,(1/2 - \epsilon')\,.$$

Now, we consider the functions $\tilde{f}_{r_1}(\cdot)$ restricting r_1 to random strings from R_1, i.e. we consider

$$\tilde{f}_{r_1}(r_2) \ = \ c \ \oplus \ \sum_{i=1}^{\tau} c_i \ v_1^i(r_2), \text{ with } r_1 \in R_1.$$

From the assumptions we know that every such $\tilde{f}_{r_1}$ has bias bounded by ϵ':

$$\mathcal{E}(\tilde{f}_{r_1}) = |1/2 - wt(\tilde{f}_{r_1})/2^{|r_2|}| \leq \epsilon', \quad \text{for all } r_1 \in R_1.$$

This means that for every $r_1 \in R_1$ it is true:

$$2^{|r_2|}\,(1/2 - \epsilon') \ \leq \ wt(\tilde{f}_{r_1}) \ \leq \ 2^{|r_2|}\,(1/2 + \epsilon')\,.$$

Combining this inequality with the bound on $|R_1|$, one can conclude that

$$2^{|r_1|+|r_2|}\,(1/2 - \epsilon')^2 \ \leq \ |\{(r_1, r_2) \mid \tilde{f}(r_1, r_2) = 1\}| \ \leq \ 2^{|r_1|+|r_2|}\,(1/2 + \epsilon')\,.$$

Now, using our definition for $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$, the left-hand side can be written as $2^{|r_1|+|r_2|}\,(1/2 - \epsilon)$ and $(1/2 + \epsilon')$ on the right-hand side can be bounded by $(1/2 + \epsilon)$. Thus we get

$$2^{|r_1|+|r_2|}\,(1/2 - \epsilon) \ \leq \ |\{(r_1, r_2) \mid \tilde{f}(r_1, r_2) = 1\}| \ \leq \ 2^{|r_1|+|r_2|}\,(1/2 + \epsilon)\,.$$

This completes the proof that in Case 2 the bias bound $\mathcal{E}(\tilde{f}) \leq \epsilon$ holds.

Proposition 11 (ϵ-2-AS-Circuit-Sequential-Composability). *Consider ϵ-2-AS circuits $C_1(x_1, r_1)$ and $C_2(x_2, r_2)$. Let C be the circuit obtained by connecting the output of C_1 to the input x_2 of C_2 and letting the input r_2 of C_2 be the extra input of C:*

$$C(x_1, (r_1, r_2)) = C_2(C_1(x_1, r_1), r_2).$$

Then $C(x_1, (r_1, r_2))$ is also an ϵ-2-AS circuit.

Proof. We will proceed analogously to the proof of Proposition 10. Assume C_1 and C_2 are ϵ-2-AS and let $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$. From [6] we know that the composition C is ϵ'-1-AS and thus, the first condition in Definition 9 is satisfied. To see that also the second condition is true, let us consider a function $f(x_1, r_1, r_2) \in \mathcal{F}^{(2)}(C \setminus \mathcal{X}_1) \setminus \{\mathbf{0}, \mathbf{1}\}$, where $\mathcal{X}_1$ denotes all input nodes of C_1. In the proof we will denote by $\mathcal{X}_2$ all input nodes of C_2. Note, that in our construction $\mathcal{X}_2$ coincide with the output nodes of C_1.

Our task is to show that for any x_1 the bias

$$\mathcal{E}(f(x_1, \cdot, \cdot)) \leq \epsilon.$$

Assume x_1 is an arbitrary, but fixed input and let $\tilde{f}$ denote the function $\tilde{f}(r_1, r_2) = f(x_1, r_1, r_2)$. In the most general case it has the form

$$\tilde{f}(r_1, r_2) \;=\; c \;\oplus\; u_2(r_1) \;\oplus\; v_2(y(r_1), r_2) \;\oplus\; \sum_{i=1}^{\tau} u_1^i(r_1)\; v_1^i(y(r_1), r_2), \tag{6}$$

where $c \in \{\mathbf{0}, \mathbf{1}\}$ is a constant, $y(r_1) = C_1(x_1, r_1)$ denotes the output of C_1 on (x_1, r_1), and u_2, v_2, u_1^i, and v_1^i are functions such that:

$$u_2 \in \mathcal{F}^{(2)}(C_1 \setminus \mathcal{X}_1) \setminus \{\mathbf{0}, \mathbf{1}\}, \quad v_2 \in \mathcal{F}^{(2)}(C_2 \setminus \mathcal{X}_2) \setminus \{\mathbf{0}, \mathbf{1}\}, \text{ and}$$
$$u_1^i \in \mathcal{F}^{(1)}(C_1 \setminus \mathcal{X}_1) \setminus \{\mathbf{0}, \mathbf{1}\}, \quad v_1^i \in \mathcal{F}^{(1)}(C_2 \setminus \mathcal{X}_2) \setminus \{\mathbf{0}, \mathbf{1}\}, \text{ for } i = 1, \ldots \tau,$$

To prove that the bias of $\tilde{f}$ is bounded by ϵ, we consider three cases.

Case 1: v_2 in Eq. (6) is non-trivial. We assume that v_2 has at least one term. For every fixed (but arbitrary) r_1, function $\tilde{f}$ can be expressed as a function in $\mathcal{F}^{(2)}(C_2 \setminus \mathcal{X}_2)$ as follows:

$$\tilde{f}_{r_1}(r_2) \;=\; c \;\oplus\; c_0 \;\oplus\; v_2(\hat{y}, r_2) \;\oplus\; \sum_{i=1}^{\tau} c_i\; v_1^i(\hat{y}, r_2),$$

where $\hat{y} := y(r_1) = C_1(x_1, r_1)$, $c_0 := u_2(r_1)$, and $c_i := u_1^i(r_1)$, for $i = 1, \ldots, \tau$ (note, that in this case τ can be 0 or for $\tau \geq 1$, all coefficient values c_i can be 0). By the assumption that C_2 is ϵ-2-AS, we get that $\mathcal{E}(\tilde{f}_{r_1}(r_2)) \leq \epsilon$ and since this bound is true for every r_1, we can conclude that the function $\tilde{f}$, as defined in (6), has the bias bounded by ϵ, too.

Case 2: $\tilde{f}$ in Eq. (6) has the form $\tilde{f}(r_1, r_2) = c \oplus u_2(r_1) \oplus \sum_{i=1}^{\tau} u_1^i(r_1)\, v_1^i(y(r_1), r_2)$, with $\tau \geq 1$. In this case, for every fixed r_1, we can represent $\tilde{f}$ a function in

$\mathcal{F}^{(1)}(C_2 \setminus \mathcal{X}_2)$ as follows:

$$\tilde{f}_{r_1}(r_2) \;=\; c \;\oplus\; c_0 \;\oplus\; \sum_{i=1}^{\tau} c_i \; v_1^i(\hat{y}, r_2),$$

where, as in Case 1, $\hat{y} := C_1(x_1, r_1)$, $c_0 := u_2(r_1)$, and $c_i := u_1^i(r_1)$, for $i = 1, \ldots, \tau \geq 1$. Note, that for some strings r_1 it can happen that all coefficients $c_i = 0$, what means that for such r_1 function $\tilde{f}_{r_1}$ has bias $1/2$. Below, we argue that there are sufficiently many r_1 such that at least one coefficient c_i is non-zero. This will suffice to bound the bias of $\tilde{f}$.

Next, we consider the coefficient c_1 that, recall, is defined as $c_1 := u_1^1(r_1)$ for the function $u_1^1(\cdot)$ in $\mathcal{F}^{(1)}(C_1 \setminus \mathcal{X}_1)$. From the assumption, its bias is bounded as follows

$$\mathcal{E}(u_1^1(r_1)) = |1/2 - wt(u_1^1)/2^{|r_1|}| \leq \epsilon'.$$

From this, we get that $wt(u_1^1) \geq 2^{|r_1|}\,(1/2 - \epsilon')$. Let $R_1 := \{r_1 \mid c_1 = u_1^1(r_1) = 1\}$ denote the set of all such strings r_1. Its cardinality is at least $2^{|r_1|}\,(1/2 - \epsilon')$. Consider $\tilde{f}_{r_1}(\cdot)$ restricting r_1 to strings from R_1 only:

$$\tilde{f}_{r_1}(r_2) \;=\; c \;\oplus\; \sum_{i=1}^{\tau} c_i \; v_1^i(r_2), \text{ with } r_1 \in R_1.$$

From the assumptions, we know that every such $\tilde{f}_{r_1}$ has bias bounded by ϵ':

$$\mathcal{E}(\tilde{f}_{r_1}) = |1/2 - wt(\tilde{f}_{r_1})/2^{|r_2|}| \leq \epsilon', \quad \text{for all } r_1 \in R_1.$$

This means that: $2^{|r_2|}\,(1/2 - \epsilon') \;\leq\; wt(\tilde{f}_{r_1}) \;\leq\; 2^{|r_2|}\,(1/2 + \epsilon')$, for all $r_1 \in R_1$, and combining this with the bound on $|R_1|$, we get

$$2^{|r_1|+|r_2|}\,(1/2 - \epsilon')^2 \;\leq\; |\{(r_1, r_2) \mid \tilde{f}(r_1, r_2) = 1\}| \;\leq\; 2^{|r_1|+|r_2|}\,(1/2 + \epsilon').$$

Using our definition for $\epsilon' = \frac{1}{2} - \sqrt{\frac{1}{2} - \epsilon}$ we can conclude that

$$2^{|r_1|+|r_2|}\,(1/2 - \epsilon) \;\leq\; |\{(r_1, r_2) \mid \tilde{f}(r_1, r_2) = 1\}| \;\leq\; 2^{|r_1|+|r_2|}\,(1/2 + \epsilon).$$

This completes the proof that in Case 2 the bias $\mathcal{E}(\tilde{f}) \leq \epsilon$, too.

Case 3: $\tilde{f}$ in Eq. (6) has the form $\tilde{f}(r_1, r_2) \;=\; c \;\oplus\; u_2(r_1)$. In this case the bound on the bias of $\tilde{f}$ follows directly from the assumption that C_1 is ϵ-2-AS.

Encoding-Circuit Composability Finally, we prove that a composition of an ϵ-2-AS encoding function E with an ϵ-2-AS circuit C leads to a construction for which the second order closure of $\mathcal{F}(C(E))$ contains functions of bias $\leq \epsilon$. Similarly to the security analysis of `Encode`$[n, 1]$, the highest degree term of `Encode`$[n, 2]$ is found in the last share : $x_n = x \oplus \tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus \bigoplus_{i=1}^{n-1} x_i$ and clearly no first or second order combination of $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, x_1, \ldots, x_n)$ is constant. Thus the following holds for `Encode`$[n, 2]$:

Table 4. Summary of the ϵ-1-AS and ϵ-2-AS bounds for the `And`, `Xor` and `RefreshMask` gadgets and encoding function. The variable $e = -\log_2(1/2 + \epsilon)$ where $\epsilon = \max_{f_x \in \mathcal{R}_d} \mathcal{E}(f_x)$ as in Proposition 1 and R_c denotes the minimum number of randomness to achieve 128-bit security.

	`And`	`Xor`	`RefreshMask`	`Encode`	$e \approx$	$R_c \geq$
ϵ-1-AS	7/16	1/4	1/4	1/4	9.3×10^{-2}	1.503
ϵ-2-AS	2047/4096	31/64	31/64	7/16	3.5×10^{-4}	3.6×10^5

1. $\forall f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and $\forall x \in \mathbb{F}_2^N$ the bias of $f(x, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ':
$$\max_{f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}, x \in \mathbb{F}_2^N} \mathcal{E}(f(x, \cdot)) \leq 3/8.$$
2. $\forall f \in \mathcal{Y}^{(2)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and $\forall x \in \mathbb{F}_2^N$ the bias of $f(x, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ϵ:
$$\max_{f \in \mathcal{Y}^{(2)} \setminus \{\mathbf{0},\mathbf{1}\}, x \in \mathbb{F}_2^N} \mathcal{E}(f(x, \cdot)) \leq 7/16.$$

This follows from the proposition below, which can be proven analogously as Proposition 11.

Proposition 12. *Let $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit, and let $E : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be a function. If E is encoding ϵ-2-AS and C is circuit ϵ-2-AS then, for $d = 2$, it is true:*
$$\max_{f_x \in \mathcal{R}_d} \mathcal{E}(f_x) \ \leq \ \epsilon,$$
where $\mathcal{R}_d$ is defined in Definition 4.

4.5 Prediction Security – a Summary

In the previous sections, we have shown the algebraic circuit security of our gadgets and algebraic encoding security of our encoding functions. In this section, we give the quantitative bounds on prediction security.

Let us start with the first order prediction security bound of an $(n, 1)$ scheme. According to Proposition 1, $e = -\log_2(1/2 + \epsilon) = -\log_2(1/2 + 7/16) \approx 0.093$ and the number of required random bits to achieve 128-bit security can be calculated as: $R_c \leq k \cdot (1 + 1/e) = 128 \cdot (1 + 1/0.093) \approx 1503$. In conclusion, a circuit C composed with $(n, 1)$ gadgets and $\texttt{Encode}[n, 1]$ is 1-PS ($Adv^{PS}_{C,E,1} \leq 2^{-128}$) if the circuit contains $R_c \leq 1503$ random bits.

Next, we give the second order prediction bound for an $(n, 2)$ scheme. According to Proposition 1, $e = -\log_2(1/2 + \epsilon) = -\log_2(1/2 + 2047/4096) \approx 3.5 \times 10^{-4}$. As a result the number of required random bits to achieve 128-bit security is drastically increased and is calculated as: $R_c \leq k \cdot (1 + 1/e) = 128 \cdot (1 + 1/3.5 \times 10^{-4}) \approx 3.6 \times 10^5$. A summary of first and second order algebraic security properties can be found in Table 4.

d \ n	0	1	2	n
0	○	◐ `ISW Transformation` [34]		
1	◑	◑ [6]	● Ex. 3	●$[n, 1]$
2	◑	◑	● Ex. 4	●$[n, 2]$
d	◑	◑	●	● Sec. 3.1

Table 5. The security properties of masking schemes. The mark ○ (resp. ●) means the scheme is vulnerable (resp. resistant against) both to computational and algebraic attacks. Mark ◑ (resp. ◐) stands for vulnerability to computational but resistant against algebraic attacks (resp. resistant against computational but vulnerability to algebraic attacks). Remark that a masking scheme with $(n, 0)$ is the `ISW` transformation [34] while a masking scheme with $(1, 1)$ is the scheme in [6]. The example structures for the masking schemes with $(2, 1)$ and $(3, 1)$ can be found in Appendix B.

Comparison with other masking schemes: An (n, d) scheme as defined in Section 3 is a combination of linear and multiplicative components. The allocation of these components gives us different orders of protections and thus the scheme has two corner cases: **(1)** $d = 0$ with $n \geq 1$ and **(2)** $d \geq 1$ with $n = 0$. The first case **(1)** acts as an additive masking. Such schemes are widely used in the literature e.g. Boolean masking [46], Threshold Implementations [43], polynomial masking [48] and domain oriented masking [32]. The common point of these schemes is that the degree of their encoding function is one, thus they are vulnerable to algebraic attacks i.e. not prediction secure. On the other hand, the latter case **(2)** corresponds to a multiplicative masking scheme which is vulnerable to side-channel attacks [25,30] i.e. not probing secure. Therefore the masking schemes in the literature need to be combined with other masking schemes to accomplish both *prediction* and *probing* security notions.

A straightforward approach is to employ both linear and multiplicative components, as done in *affine masking* introduced by Fumaroli et al. [25]. The scheme processes a sensitive value x in the form of $r_1 x \oplus r_0$ such that $r_1, r_0 \in_R \mathbb{F}_2^n$ and *fixed* for each execution of the algorithm. As stated by the authors, affine masking is not *perfectly* secure against higher-order SCA but provides *practical* security. Indeed some pairs of intermediate variables of the scheme depend on sensitive variables. A second order side-channel attack can break the affine masking. Also it is not clear how to generalize the scheme. Another approach to combine linear and multiplicative components is given in [6]. However, the scheme alone does not provide security against computation attacks as described in Example 2. As a result, our scheme can be seen as a generalization of affine masking and the scheme by [6] in the sense of employing both linear and multiplicative components while providing *provable* security in both *prediction* and *probing* security notions. A summary of the security properties of the our scheme with different security orders is presented in Table 5.

Table 6. The number of gadgets in one round of AES.

	`SubBytes`	`MixColumns`	`AddRoundKey`	`ShiftRows`
`And`	16×32	-	-	-
`Xor`	16×83	27	128	-

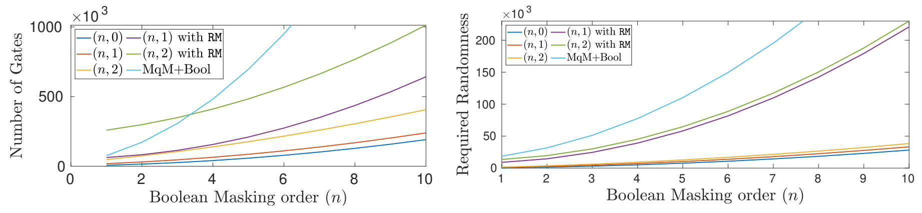

Fig. 2. Total number of bitwise operations and required randomness for one round of AES-128 with different $(n, 0)$, $(n, 1)$ and $(n, 2)$ masking schemes with and without initial `RefreshMask` gadgets

5 A Proof-of-Concept AES Implementation

In this section we introduce a white-box AES design based on the masking scheme defined in Section 3. The AES block cipher consists of multiple rounds of operations on its state. The operations include three linear layers: `MixColumns`, `ShiftRows`, and `AddRoundKey` and one non-linear layer `SubBytes`. The bitwise implementation for the linear operations can be defined straightforwardly. In our construction we use the bitwise AES-Sbox design by Boyar and Peralta [13] and the exact number of `And` and `Xor` gadgets within one round of AES-128 can be seen in Table 6. The total number of bitwise operations[5] can be calculated using Table 6 and the performance analysis in Table 1. A visual representation of the AES-128 implementations with $(n, 0)$ (i.e. `ISW`-transformation), $(n, 1)$-masking scheme and $(n, 2)$-masking scheme is shown in Figure 2. Moreover, the analysis contains the algebraically secure gadgets where each input is associated with a `RefreshMask` gadget, and the idea of using two different masking schemes (first Minimalist quadratic Masking and second Boolean masking as in [6]).

As seen in Figure 2, our hybrid constriction outperforms the idea of using a first order linear masking on top of a non-linear masking. As stated in [6], using a combination of two masks even with the first order protections requires roughly 200.000 gates per AES round. Since the foundation of our scheme is the `ISW` transformation, we can increase the probing security aspect of our scheme efficiently. However, increasing the non-linear order is the bottleneck of our scheme. When we compare the smallest possible implementations, we see that one round of AES-128 with $(2, 0)$, $(2, 1)$ and $(2, 2)$-masking schemes re-

[5] The bitwise `SubBytes` design by Boyar and Peralta [13] also requires Not gates. Although we didn't give the explicit description of a `Not` gadget in our masking scheme, it can be easily defined as identical to the `Not` gadget in the `ISW` transformation i.e. by flipping the n^{th} share.

quires 15201, 30808(82678) and 74875(298315) gates respectively. The values in the parenthesis correspond to the gadgets where the inputs are *first* processed by RefreshMask gadgets. Clearly, RefreshMask gadgets impose a heavy overhead on our scheme. Therefore a significant performance advantage can be achieved by further optimizing the RefreshMask gadget. While the first order algebraically secure implementation requires a small overhead over an unprotected implementation, the second-order algebraically secure implementation comes with a substantial cost. One round of AES-128 with $(2, 1)$, $(3, 1)$ and $(4, 1)$-masking schemes requires 30808(82678), 46115(113945) and 64494(156264) gates respectively. Therefore, we can conclude that one can increase the security against computational attacks with small overhead compared with the overhead of increasing the security against algebraic attacks. Furthermore, the randomness requirements of our scheme increases similarly to the ISW-transformation as seen in Figure 2.

5.1 Experimental Setup

To experimentally verify the security properties of our scheme we used the proof-of-concept AES-128 implementation. The implementations using $(n, 0)$, $(n, 1)$ and $(n, 2)$ masking schemes including the analysis are available as open source[6].

Software traces are simulated by encrypting N random plaintext and collecting the output of each node. We denote the i^{th} trace (corresponding to the encryption of i^{th} plaintext) by $t_i = \{v_1^i, \ldots, v_M^i\}$ where v_j^i denotes the output of j^{th} node and M denotes the number of the nodes in the circuit. Using the software traces we demonstrate a simple leakage detection test by the test vector leakage assessment (TVLA) as proposed by Goodwill et al. [29]. In the first part of the test, two different sets of side-channel traces are collected by processing either a fixed input or a random input under the same conditions in a random pattern. After collecting the traces, we calculate the means (μ_f, μ_r) and standard deviations (σ_f, σ_r) for the two sets. Welch's t-test is executed as in Equation (7) where n_f and n_r denote the number of traces for fixed and random sets respectively.

$$t = \frac{\mu_f - \mu_r}{\sqrt{(\sigma_f^2/n_f) + (\sigma_r^2/n_r)}}. \tag{7}$$

Using the experimental setup we implement a first order leakage detection test using 10000 traces (i.e. $n_f + n_r = 10000$) and $M = 80000$ (corresponds to the two round of AES-128). As expected the test shows no observable leakage. The illustration of the test can be seen in Figure 3.

6 Conclusion

White-box cryptography has become a popular method to protect cryptographic keys in an insecure software realm potentially controlled by the adversary. All

[6] https://github.com/UzL-ITS/white-box-masking

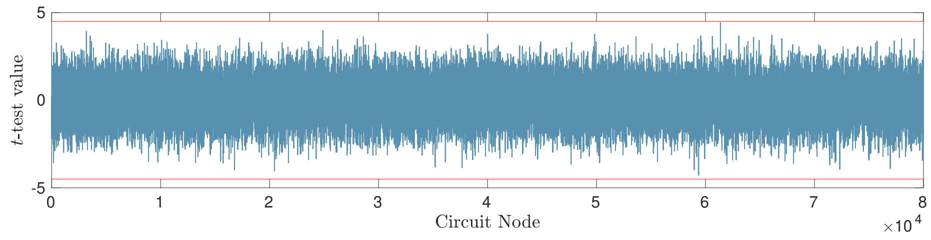

Fig. 3. A first-order leakage test on a circuit that simulates the AES-128 with $(2, 1)$-masking defined in Section 3.1. Clearly, t-test value lie in threshold values as drawn by red lines ($[-4.5, 4.5]$).

white-box cryptosystems in the literature have been practically broken due to differential computation analysis. Algebraic attacks have shown the inefficacy of classic side-channel countermeasures when they are applied in the white-box setting. Therefore, the need for a secure and reliable method to protect white-box implementations against both attacks has become evident.

We have proposed the first masking scheme that combines linear and non-linear components to achieve resistance against computational and algebraic attacks. The new scheme extends the `ISW` transformation to resist algebraic attacks by increasing the order of the decoding function. We have analyzed the two prevalent security notions in the white-box model, *probing security* and *prediction security*, and underlined the incompatibility of the notions, which reveals that a scheme should satisfy both notions. We have used the well-known SNI security notion to prove the $(n-1)^{th}$ order probing security of an (n, d)-masking scheme and thus we showed that our scheme can resist $(n-1)^{th}$-order computation attacks. We proved first and second order prediction security for the concrete construction of the $(n, 1)$ and $(n, 2)$ masking scheme, respectively. the scheme has been defined generically and can be applied to any orders of n and d, as long as the computational structure satisfies the algebraic properties. We have examined the implementation cost of our scheme for arbitrary orders of protection and compare it with the `ISW` transformation. We have extended the algebraic verification tool to support our scheme and to validate our results. The updated code has been made publicly available. Finally, a proof-of-concept AES-128 bit-wise implementation was provided to perform leakage detection and extensive performance analysis. The analysis showed that the new combined masking scheme outperforms the previous approaches which require to combine two different masking schemes to resist both attacks.

References

1. Banik, S., Bogdanov, A., Isobe, T., Jepsen, M.B.: Analysis of Software Countermeasures for Whitebox Encryption. IACR Trans. Symmetric Cryptol. **2017**(1), 307–328 (2017)

2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE **94**(2), 370–382 (2006)
3. Barthe, G., Belaïd, S., Cassiers, G., Fouque, P., Grégoire, B., Standaert, F.: maskverif: Automated verification of higher-order masking in presence of physical defaults. In: Sako, K., Schneider, S.A., Ryan, P.Y.A. (eds.) Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11735, pp. 300–318. Springer (2019). https://doi.org/10.1007/978-3-030-29959-0_15, `https://doi.org/10.1007/978-3-030-29959-0\_15`
4. 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: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 116–129. CCS '16, ACM, New York, NY, USA (2016)
5. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a White Box AES Implementation. In: Handschuh, H., Hasan, M.A. (eds.) Selected Areas in Cryptography. pp. 227–240. Springer (2005)
6. Biryukov, A., Udovenko, A.: Attacks and Countermeasures for White-box Designs. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018. pp. 373–402. Springer International Publishing, Cham (2018)
7. Bock, E.A., Bos, J.W., Brzuska, C., Hubain, C., Michiels, W., Mune, C., Gonzalez, E.S., Teuwen, P., Treff, A.: White-box cryptography: don't forget about grey-box attacks. Journal of Cryptology **32**(4), 1095–1143 (2019)
8. Bock, E.A., Brzuska, C., Fischlin, M., Janson, C., Michiels, W.: Security reductions for white-box key-storage in mobile payments. Cryptology ePrint Archive, Report 2019/1014 (2019), `https://eprint.iacr.org/2019/1014`
9. Bock, E.A., Brzuska, C., Michiels, W., Treff, A.: On the Ineffectiveness of Internal Encodings-Revisiting the DCA Attack on White-Box Cryptography. In: International Conference on Applied Cryptography and Network Security. pp. 103–120. Springer (2018)
10. Bogdanov, A., Rivain, M., Vejre, P.S., Wang, J.: Higher-order DCA against standard side-channel countermeasures. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 118–141. Springer (2019)
11. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults. In: Fumy, W. (ed.) Advances in Cryptology EUROCRYPT'97, Lecture Notes in Computer Science, vol. 1233, pp. 37–51. Springer (1997)
12. 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.) Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. pp. 215–236. Springer (2016)
13. Boyar, J., Peralta, R.: New logic minimization techniques with applications to cryptology. Cryptology ePrint Archive, Report 2009/191 (2009)
14. Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: Another attempt. IACR Cryptology ePrint Archive **2006**, 468 (2006)
15. Bulck, J.V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In: 27th USENIX Security Symposium (USENIX Security 18). p. 991–1008 (Aug 2018)

16. Chari, S., Jutla, C., Rao, J.R., Rohatgi, P.: A cautionary note regarding evaluation of AES candidates on smart-cards. In: Second Advanced Encryption Standard Candidate Conference. pp. 133–147. Citeseer (1999)
17. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.) Advances in Cryptology – CRYPTO 99, Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999)
18. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: A White-Box DES Implementation for DRM Applications. In: Feigenbaum, J. (ed.) Digital Rights Management. pp. 1–15. Springer (2003)
19. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: White-Box Cryptography and an AES Implementation. In: Nyberg, K., Heys, H. (eds.) Selected Areas in Cryptography. pp. 250–270. Springer (2003)
20. Contest, T.W.: CHES 2017 Capture the Flag Challenge The WhibOx Contest, An ECRYPT White-Box Cryptography Competition. `https://whibox-contest.github.io/`
21. Coron, J.S., Greuet, A., Prouff, E., Zeitoun, R.: Faster Evaluation of SBoxes via Common Shares. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. pp. 498–514. Springer (2016)
22. Coron, J.S., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) Fast Software Encryption. pp. 410–424. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
23. Coron, J.S., Greuet, A., Zeitoun, R.: Side-channel Masking with Pseudo-Random Generator. Cryptology ePrint Archive, Report 2019/1106 (2019), `https://eprint.iacr.org/2019/1106`
24. De Mulder, Y., Wyseur, B., Preneel, B.: Cryptanalysis of a Perturbated White-Box AES Implementation. In: Gong, G., Gupta, K.C. (eds.) Progress in Cryptology - INDOCRYPT 2010. pp. 292–310. Springer (2010)
25. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine Masking against Higher-Order Side Channel Analysis, pp. 262–280. Springer (2011)
26. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Cryptographic Hardware and Embedded Systems CHES 2001. pp. 251–261. Springer (2001)
27. Gemalto: Sentinel® LDK product brief. `https://sentinel.gemalto.com/resources/software/sentinel-ldk-feature-brief/`
28. Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: Advances in Cryptology–CRYPTO 2014, pp. 444–461. Springer (2014)
29. Gilbert Goodwill, B.J., Jaffe, J., Rohatgi, P., et al.: A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop (2011)
30. Golić, J.D., Tymen, C.: Multiplicative masking and power analysis of aes. In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 198–212. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
31. Goubin, L., Paillier, P., Rivain, M., Wang, J.: How to reveal the secrets of an obscure white-box implementation. Journal of Cryptographic Engineering (Apr 2019)

32. Gross, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In: Proceedings of the 2016 ACM Workshop on Theory of Implementation Security. p. 3. TIS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2996366.2996426, `https://doi.org/10.1145/2996366.2996426`
33. Ishai, Y., Kushilevitz, E., Li, X., Ostrovsky, R., Prabhakaran, M., Sahai, A., Zuckerman, D.: Robust Pseudorandom Generators. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) Automata, Languages, and Programming. pp. 576–588. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
34. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings. pp. 463–481. Springer (2003)
35. 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 (2010)
36. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer (1999)
37. Lee, S., Kim, T., Kang, Y.: A Masked White-Box Cryptographic Implementation for Protecting Against Differential Computation Analysis. IEEE Transactions on Information Forensics and Security **13**(10), 2602–2615 (2018)
38. Lepoint, T., Rivain, M., De Mulder, Y., Roelse, P., Preneel, B.: Two Attacks on a White-Box AES Implementation. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) Selected Areas in Cryptography – SAC 2013. pp. 265–285. Springer (2014)
39. Link, H.E., Neumann, W.D.: Clarifying obfuscation: improving the security of white-box DES. International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II **1**, 679–684 Vol. 1 (2005)
40. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes, vol. 16. Elsevier (1977)
41. Moghimi, A., Irazoqui, G., Eisenbarth, T.: CacheZoom: How SGX Amplifies the Power of Cache Attacks. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 69–90. Springer (2017)
42. Moore, C., O'Neill, M., O'Sullivan, E., Doröz, Y., Sunar, B.: Practical homomorphic encryption: A survey. In: 2014 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 2792–2795. IEEE (2014)
43. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of non-linear functions in the presence of glitches. In: Information Security and Cryptology–ICISC 2008, pp. 218–234. Springer (2009)
44. Payment, E.M.: Software-based Mobile Payment Security Requirements. `https://www.emvco.com/terms-of-use/?u=wp-content/uploads/documents/EMVCo-SBMP-16-G01-V1.4_SBMP_Security_Requirements.pdf`
45. Reparaz, O., Meyer, L.D., Bilgin, B., Arribas, V., Nikova, S., Nikov, V., Smart, N.: CAPA: The Spirit of Beaver against Physical Attacks. Cryptology ePrint Archive, Report 2017/1195 (2017)
46. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010: 12th International Workshop, Santa Barbara, USA, August 17-20, 2010. Proceedings. pp. 413–427. Springer (2010)

47. Rivain, M., Wang, J.: Analysis and Improvement of Differential Computation Attacks against Internally-Encoded White-Box Implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(2), 225–255 (Feb 2019)
48. Roche, T., Prouff, E.: Higher-order glitch free implementation of the AES using secure multi-party computation protocols. Journal of Cryptographic Engineering **2**(2), 111–127 (2012)
49. Schneider, T., Moradi, A., Güneysu, T.: ParTI – Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II, pp. 302–332. Springer (2016)
50. Seker, O., Fernandez-Rubio, A., Eisenbarth, T., Steinwandt, R.: Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 394–430 (Aug 2018)
51. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
52. Van Bulck, J., Piessens, F., Strackx, R.: SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In: Proceedings of the 2nd Workshop on System Software for Trusted Execution. SysTEX'17, Association for Computing Machinery, New York, NY, USA (2017)
53. 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.) Advances in Cryptology – ASIACRYPT 2012. Springer (2012)
54. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In: Adams, C., Miri, A., Wiener, M. (eds.) Selected Areas in Cryptography. pp. 264–277. Springer (2007)
55. 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)

A Additional Proofs

In this Appendix, we give the proofs for Lemma 1, the correctness of our scheme and for the Propositions that concern the security features of the gadgets whose proof is not given in the paper.

Lemma 1: Correctness of Circuit Transformation $T^{(n,d)}$

Proof. For simplicity, let us denote `Encode` as:

$$\texttt{Encode}(x, \tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_{n-1}) = \texttt{Encode}(x).$$

Next we prove the functionality preserving property of each gadget.

- $x = \texttt{Decode}(\texttt{RefreshMask}(\texttt{Encode}(x))$
  $= \texttt{Decode}(\texttt{RefreshMask}((\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n))$
  $= \texttt{Decode}((\tilde{x}_0 \oplus \tilde{r}_0), \ldots, (\tilde{x}_d \oplus \tilde{r}_d), (x_1 \oplus \bigoplus_{j=2}^{n} r_{1,j}), (r_{1,2} \oplus x_2 \bigoplus_{j=3}^{n} r_{2,j}),$
  $\ldots, (\bigoplus_{i=1}^{n-1} r_{i,n} \oplus x_n \oplus \mathcal{W} \oplus \mathcal{R}))$
  $= (\tilde{x}_0 \oplus \tilde{r}_0) \cdots (\tilde{x}_d \oplus \tilde{r}_d) \oplus x_1 \oplus \cdots \oplus x_n \oplus \mathcal{W} \oplus \mathcal{R}$
  $= \tilde{x}_0 \cdots \tilde{x}_d \oplus \mathcal{W}' \oplus x_1 \oplus \cdots \oplus x_n \oplus \mathcal{W} \oplus \mathcal{R}$
  $= \tilde{x}_0 \cdots \tilde{x}_d \oplus x_1 \oplus \cdots \oplus x_n$
  $= x$
- $\texttt{Decode}(\texttt{Xor}(\texttt{Encode}(x), \texttt{Encode}(y)))$
  $= \texttt{Decode}(\tilde{x}_0 \oplus \tilde{y}_0, \ldots, \tilde{x}_d \oplus \tilde{y}_d, x_1 \oplus y_1, \ldots, x_{n-1} \oplus y_{n-1}, x_n \oplus y_n \oplus \mathcal{U})$
  where $(\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n) = \texttt{RefreshMask}(\texttt{Encode}(x))$ and
  $(\tilde{y}_0, \ldots, \tilde{y}_d, y_1, \ldots, y_n) = \texttt{RefreshMask}(\texttt{Encode}(y))$.
  $= [(\tilde{x}_0 \oplus \tilde{y}_0) \cdots (\tilde{x}_d \oplus \tilde{y}_d)] \oplus [(x_1 \oplus y_1) \oplus \cdots \oplus (x_{n-1} \oplus y_{n-1}) \oplus (x_n \oplus y_n \oplus \mathcal{U})]$
  $= [\tilde{x}_0 \cdots \tilde{x}_d \oplus \mathcal{U} \oplus \tilde{y}_0 \cdots \tilde{y}_d] \oplus [(x_1 \oplus y_1) \oplus \cdots \oplus (x_{n-1} \oplus y_{n-1}) \oplus (x_n \oplus y_n \oplus \mathcal{U})]$
  $= (\tilde{x}_0 \cdots \tilde{x}_d \oplus x_1 \oplus \cdots x_n) \oplus (\tilde{y}_0 \cdots \tilde{y}_d \oplus y_1 \oplus \cdots y_n)$
  $= x \oplus y.$
- $xy = \texttt{Decode}(\texttt{And}(\texttt{Encode}(x), \texttt{Encode(y)}))$
  $= \texttt{Decode}(\texttt{And}((\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n), (\tilde{y}_0, \ldots \tilde{y}_d, y_1, \ldots, y_n)))$
  where $(\tilde{x}_0, \ldots, \tilde{x}_d, x_1, \ldots, x_n) = \texttt{RefreshMask}(\texttt{Encode}(x))$ and
  $(\tilde{y}_0, \ldots, \tilde{y}_d, y_1, \ldots, y_n) = \texttt{RefreshMask}(\texttt{Encode}(y))$.
  $= \texttt{Decode}(\tilde{z}_0, \ldots, \tilde{z}_d, z_1, \ldots, z_n)$

  where the output shares can be listed as follows:

$$\tilde{z}_i = \tilde{x}_i \tilde{y}_{i'} \oplus r^{i,1} \oplus \cdots \oplus r^{i,n} \text{ for } 0 \le i \le d,$$
$$z_i = x_i y_i \oplus \bigoplus_{\substack{j=1 \\ j \ne i}}^{n} r_{i,j} \text{ for } 1 \le i \le n.$$

  Also, the values $r_{i,j}$ can be listed as:

$$r_{0,j} = \mathcal{F}(x_j, y_j) = [r_{0,j} \oplus (\tilde{x}_0 \ldots \tilde{x}_d) y_j] \oplus x_j (\tilde{y}_0 \ldots \tilde{y}_d) \text{ for } 1 \le j \le n,$$
$$r_{i,j} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i \text{ for } 1 \le i < j \le n.$$

Therefore,

$$\begin{aligned}
\texttt{Decode}(\bar{z}) &= \tilde{z}_0 \cdots \tilde{z}_d \oplus z_1 \oplus \ldots \oplus z_n \\
&= \prod_{i=0}^{d} \left[\tilde{x}_i \tilde{y}_{i'} \oplus r^{i,1} \oplus \cdots \oplus r^{i,n}\right] \oplus \bigoplus_{i=1}^{n} \left[x_i y_i \oplus \bigoplus_{\substack{j=0 \\ j \neq i}}^{n} r_{i,j} \right] \\
&= \left[(\tilde{x}_0 \cdots \tilde{x}_d)(\tilde{y}_0 \cdots \tilde{y}_d) \oplus \mathcal{V}\right] \oplus \left[\bigoplus_{i=1}^{n} (x_i y_i \oplus \bigoplus_{\substack{j=1 \\ j \neq i}}^{n} r_{i,j}) \right] \oplus \\
&\left[\bigoplus_{j=1}^{n} ((r_{0,j} \oplus (\tilde{x}_0 \ldots \tilde{x}_d) y_i) \oplus x_i (\tilde{y}_0 \ldots \tilde{y}_d)) \right] \\
&= \left[(\tilde{x}_0 \cdots \tilde{x}_d)(\tilde{y}_0 \cdots \tilde{y}_d) \oplus \mathcal{V}\right] \oplus \left[\bigoplus_{1 \leq i,j \leq n} x_i y_j \right] \oplus \\
&\left[\mathcal{V} \oplus \bigoplus_{i=1}^{n} (\tilde{x}_0 \ldots \tilde{x}_d) y_i \oplus x_i (\tilde{y}_0 \ldots \tilde{y}_d) \right] \\
&= (\tilde{x}_0 \cdots \tilde{x}_d)(\tilde{y}_0 \cdots \tilde{y}_d) \oplus \bigoplus_{1 \leq i,j \leq n} x_i y_j \oplus \bigoplus_{i=1}^{n} ((\tilde{x}_0 \ldots \tilde{x}_d) y_i \oplus x_i (\tilde{y}_0 \ldots \tilde{y}_d)) \\
&= (\tilde{x}_0 \cdots \tilde{x}_d \oplus x_1 \oplus \cdots x_n)(\tilde{y}_0 \cdots \tilde{y}_d \oplus y_1 \oplus \cdots y_n) \\
&= xy.
\end{aligned}$$

Hence we showed that the gadgets introduced in Section 3 are functionally preserving gadgets. Therefore, the transformation that generates an (n, d)-masked circuit is a functionally preserving transformation.

Proposition 2: t-SNI of $\texttt{RefreshMask}[n, d]$ *gadget*

Proof. In order to prove the proposition, we first assume that the simulator can access the values $(\tilde{x}_i)_{i \in [0,d]}$ by Fact 1 and we show that every set of intermediate variables with t_1 elements and every set of output variables with t_2 such that $t_1 + t_2 \leq t$ can be simulated from a set of input shares $U = ((\tilde{x}_i)_{i \in [0,d]}, (x_i)_{i \in I})$ such that $|I| \leq t_1$.

Let us first classify the variables. The intermediate variables are x_i, $r_{i,j}$, $\tilde{x}_i$, $\tilde{r}_j$, $a_{i,j}$, $b_{j,i}$ (where $a_{i,j}$, $b_{j,i}$ as defined in Algorithm 3) and the intermediate variables within $\mathcal{W}$, $\mathcal{R}$ and the outputs are $\tilde{x}'_i$, x'_i.

Next, we can define I as follows:

- For each selected variable x_i, $r_{i,j}$ and $a_{i,j}$ add i to I and $b_{j,i}$ add j to I.
- For each selected $\tilde{r}_j$, $\tilde{x}_j$ and $\tilde{x}_j \oplus \tilde{r}_j$, we don't need to add any value since $\tilde{x}_j$ is accessible by the simulator.

– **Line 11, $\mathcal{W}$ and $\mathcal{R}$**: If one of the variables of form $\prod_{i\in J}(\tilde{x}_i \oplus r_0)\prod_{i\notin J}\tilde{r}_i$ where $J \subsetneq \{0,\dots,d\}$ is selected, no values need to be added due to Fact 1. If one of the variables inside $\mathcal{R}$ is selected, no values need to be added since in the expression only shares $\tilde{x}_i$ and random variables are used.

It is clear that I contains at most t_1 elements since each selected value adds at most one index to I.

Now we can define the simulator. For all $i \in I$ the simulator can sample all $r_{i,j}$ for $j \in [i+1,n]$ and compute all partials sums $a_{i,j}$ and $b_{j,i}$ and thus the output x'_i. For all $i \in [0,d]$ the simulator can sample $\tilde{r}_i$ and compute the output $\tilde{x}'_i$. Moreover the simulator can compute $\mathcal{W}$ and $\mathcal{R}$ by sampling random variables and computing $(\tilde{x}_i)_{i\in[0,d]}$.

Finally, we need to consider the simulation of the output shares x'_i such that $i \notin I$. Observe that $i \notin I$ means that any random value in the partial sum of x'_i is not probed and is not involved in a partial sum of it. Hence we can simulate x'_i by an uniformly random value. Also, by Fact 1 any output variable $\tilde{x}'_i$ can be simulated. As a result any set of t_1 selected intermediate variables and any set of t_2 output variables can simulated by $U = ((\tilde{x}_i)_{i\in[0,d]}, (x_i)_{i\in I})$ such that $|I| \le t_1$.

Proposition 3: t-SNI of $\texttt{And}[n,d]$ *gadget*

Proof. In order to prove the proposition, we first assume that the simulator can access the values $(\tilde{x}_i)_{i\in[0,d]}$ and $(\tilde{y}_i)_{i\in[0,d]}$ by Fact 1. Then we show that every set of t_1 intermediate variables and every set of t_2 output variables such that $t_1 + t_2 \le t$ can be simulated by two sets of input shares $(\tilde{x}_i)_{i\in[0,d]}$ and $(x_i)_{i\in I}$ such that $|I| \le t_1$, resp. $(\tilde{y}_j)_{j\in[0,d]}$ and $(y_j)_{j\in J}$ such that $|J| \le t_1$.

We first need to construct the sets of indices I and J corresponding to the shares of x and y. The following two cases cover every variable in **Step-2(b)** and **Step-3**:

Group 1: For all x_i, y_i, x_iy_i, add i to I and J.
Group 2: For all $r_{i,j}$ or $z_{i,j}$ add, i to I and J where $z_{i,j}$ denotes the j^{th} partial sum of z_i.

Note that after thees steps, we have $I = J$ and we denote this common set as U.

Group 3: For all $x_iy_j \oplus r_{i,j}$, if $i \in U$ or $j \in U$, add both i, j to I and J.
Group 4: For all x_iy_j add, i to I and j to J.

To cover **Step-1** and **Step-2(a)** we need to use the following classification:

Group 5: : For all $\tilde{x}_i$, $\tilde{y}_i$, $r^{i,j}$ and combination of these, no values are needed to be added due to Fact 1.
Group 6: For all $\tilde{x}_iy_j$ (resp. $\tilde{y}_jx_i$), add j to J (resp. i to I).
Group 7: For all values of the form $\prod_{i\in K}\tilde{x}_i\prod_{j\in L}\tilde{y}_j$ where $K, L \subsetneq \{0,\dots,d\}$, no values are needed to be added due to Fact 1.

Clearly, I and J have at most one index per selected variable, and therefore $|I| \leq t_1$ and $|J| \leq t_1$.

We now define the simulator for the intermediate variables. The simulation of the variables in Group 1 and Group 4 can be performed easily.

Group 1: To simulate x_i, y_i, or $x_i y_i$, we can simply use the input variables, as both x_i and y_i are known from I and J.

Group 4: To simulate $x_i y_j$ we can simply use the input variables, as both x_i and y_j are known from I and J.

For the remaining groups **Group 1** and **Group 4** (i. e. probed variables $r_{j,i}$, $z_{i,j}$ or $x_i y_j \oplus r_{i,j}$), we use the following claim.

Claim 1 *If $i \notin U$, then $r_{i,j}$ is not selected and does not enter in the computation of any probed $z_{i,k}$. Similarly, if $j \notin U$, then $r_{j,i}$ is not selected and does not enter in the computation of any selected $c_{j,k}$.*

Proof. For $i < j$, the variable $r_{i,j}$ is used in all partial sums $c_{i,k}$ for $k > j$. Also $r_{i,j}$ is used in $r_{i,j} \oplus x_i y_j$, which is a part of $r_{j,i}$. Note that $r_{j,i}$ is used in all partial sums $c_{i,k}$ for $k > i$.

For $1 < i < j$ let us consider the following cases:

Case 1: $\{i, j\} \in U$ means that all the variables $r_{i,j}$, $x_i y_j$, $x_i y_j \oplus r_{i,j}$, $x_j y_i$ and $r_{j,i}$ can be perfectly simulated while simulating $r_{i,j}$ by a uniformly random value.

Case 2: $i \in U$ and $j \notin U$ implies that we can simulate $r_{i,j}$ as a uniformly random value and if $x_i y_j \oplus r_{i,j}$ is also selected we can perfectly simulate it since $i \in U$ and $j \in J$ by Claim 1.

Case 3: $i \notin U$ and $j \in U$ indicates that any variable of the form $r_{i,j}$ and $z_{i,j}$ is not selected by Claim 1. More importantly $r_{i,j}$ is not used in any other selected value. Thus, we can simulate $r_{j,i}$ with a uniformly random value. Also we can simulate $x_i y_j \oplus r_{i,j}$ (observe that $x_i y_j \oplus r_{i,j} = x_j y_i \oplus r_{j,i}$) since $j \in U$ and $i \in J$.

Case 4: $i \notin U$ and $j \notin U$ means that if $x_i y_j \oplus r_{i,j}$ is selected we can simply simulate it with a uniformly random value, since $r_{i,j}$ is not selected and does not enter any calculation.

From the above analysis we can see that any variable $r_{i,j}$ can be simulated if $i \in U$ including all partial sums $z_{i,k}$ and z_i. Now, we need to consider the variables from **Step-1** and **Step-2(a)**.

- Every variable $\tilde{x}_i$, $\tilde{y}_i$, $\tilde{x}_i \tilde{y}_{i'}$, $\tilde{x}_i y_j$, $\tilde{y}_j x_i$ $r^{i,j}$ or xor of these values can be simulated according to Fact 1.
- Every variable in **Step-2(a)** can be simulated since the simulator accesses $\tilde{x}_{i \in [0,d]}$ and $\tilde{y}_{j \in [0,d]}$.

Hence, we show that any set of intermediate variables, with t_1 elements can be simulated by the sets $((\tilde{x}_i)_{i\in[0,d]},(x_i)_{i\in I})$ and $((\tilde{y}_j)_{j\in[0,d]},(y_j)_{j\in J})$ which are uniformly random and independent of any sensitive variable.

In the last part of the proof, we focus on the simulation of an arbitrary set of output variables $((\tilde{z}_i)_{i\in\tilde{\mathcal{O}}},(z_i)_{i\in\mathcal{O}})$ where $\tilde{\mathcal{O}}\subset[0,d]$ and $\mathcal{O}\subset[1,n]$ with t_2 elements such that $t_1+t_2\le t$. Let us first analyze the non-linear output shares $(\tilde{z}_i)_{i\in\tilde{\mathcal{O}}}$. Observe that we can simulate $r^{i,j}$ as uniformly random values and perfectly simulate $\tilde{z}_i$ by Fact 1.

Next, we focus on the output shares $(z_i)_{i\in\mathcal{O}}$. From the discussion above, we can see that we can simulate outputs z_i with $i\in U$ perfectly. Now, consider z_i with $i\notin U$. A set of indices V is constructed as follows: For each variable $x_iy_j\oplus r_{i,j}$ in Group 3 with $i\notin U$ and $j\notin U$ (corresponding to Case 4 described above), we add j to V if $i\in\mathcal{O}$ or i to V if $i\notin\mathcal{O}$. Note that we only considered variables in Group 3, where we increased I and J by two elements. As V was only increased by one element, we have $|U|+|V|\le t_1$ and thus $|U|+|V|+|\tilde{\mathcal{O}}|+|\mathcal{O}|<n$. Hence, there is an index $j^*\in[0,n]$ such that $j^*\notin(U\cup V\cup\mathcal{O})$. By definition, we have

$$z_i = x_iy_i \oplus \bigoplus_{j=0;j\neq i}^{n} r_{i,j} = r_{i,j^*} \oplus \left(x_iy_i \oplus \bigoplus_{j=0;j\neq i;j\neq j^*}^{n} r_{i,j} \right).$$

We will now show that r_{i,j^*} and $r_{j^*,i}$ are not processed in the computation of any selected intermediate variable or another output variable $z_{i'}$ with $i'\in\mathcal{O}$. Observe that, if $i\notin U$ (resp. $j^*\notin U$) neither r_{i,j^*} (resp. $r_{j^*,i}$) nor any partial sum $z_{i,k}$ (resp. $z_{j^*,k}$) was selected. Therefore, $j^*\notin\mathcal{O}$ and z_{j^*} were also not selected. Hence, r_{i,j^*} and $r_{j^*,i}$ are not used in the computation of a selected intermediate variable.

In the last part of the proof, we need to show that r_{i,j^*} and $r_{j^*,i}$ are not needed for other output variables $z_{i'}$.

If $i<j^*$, then $x_iy_{j'}\oplus r_{i,j^*}$ was not selected (since $j^*\notin V$ and $i\in\mathcal{O}$). If $j^*<i$, then $x_{j*}y_i\oplus r_{j^*,i}$ was not selected (since $j^*\notin(V\cup\mathcal{O})$. Hence, r_{i,j^*} and $r_{j^*,i}$ are not used in the computation of any output variable $z_{i'}$ and we simulate z_i by sampling a random value.

Proposition 4: t-NI of $\texttt{Xor}[n,d]$ *gadget*

Proof. In order to prove the proposition, we first assume that the simulator can access the values $(\tilde{x}_i)_{i\in[0,d]}$ and $(\tilde{y}_i)_{i\in[0,d]}$ and show that every set of intermediate variables including the output shares with $\le t$ elements can be simulated by two sets of input shares $(\tilde{x}_i)_{i\in[0,d]}$ and $(x_i)_{i\in I}$ such that $|I|\le t$ (resp. $(\tilde{y}_j)_{j\in[0,d]}$ and $(y_j)_{j\in J}$ such that $|J|\le t$). We denote the concatenations of these tuples by $U=((\tilde{x}_i)_{i\in[0,d]},(x_i)_{i\in I})$ and $V=((\tilde{y}_j)_{j\in[0,d]},(y_j)_{j\in J})$.

We can define I and J as follows: for each selected x_i, y_i, $x_i\oplus y_i$ add i to I and J. Due to Fact 1, all selected variables $\tilde{x}_i$, $\tilde{y}_i$, $\tilde{x}_i\oplus\tilde{y}_i$, $\tilde{x}_i\tilde{y}_j$ and $\prod_{i\in K}\tilde{x}_i\prod_{j\notin K}\tilde{y}_j$, doe not increase the size of the sets I and J.

It is clear that I and J contains at most t elements since each selected variable adds at most one index to I and/or J. Remark that the above classification also covers the output shares.

Now we can define the simulator. Every variable of the form x_i, y_i, $x_i \oplus y_i$ (resp. $\tilde{x}_i$, $\tilde{y}_i$, $\tilde{x}_i \oplus \tilde{y}_i$) can be simulated by the sets U and V. Moreover every variable of the form $\prod_{i \in K} \tilde{x}_i \prod_{j \notin K} \tilde{y}_j$ where $K \subsetneq \{0, \ldots, d\}$ can be simulated by Fact 1.

Proposition 5: ϵ-1-AS of `RefreshMask`$[n, 1]$ *Gadget*

Proof. In the first part of the proof, we show that there exists no function $f \in \mathcal{F}^{(1)}(C)$ such that f is constant when inputs are fixed. Assume that there exists a function $f \in \mathcal{F}^{(1)}(C)$ such that f is constant when the inputs $(\tilde{x}_0, \tilde{x}_1, (x_i)_{1 \leq i \leq n})$ are fixed. As seen in Algorithm 3, the only nodes that do not contain a random variable(i.e. not fixed) can be found in line 11 where the values $\mathcal{W}$ and $\mathcal{R}$ are processed. By the definition of $\mathcal{W}$ each input is accompanied by a random value. And $\mathcal{R}$ contains only random values. Therefore each node is accompanied by a random node and any linear combination of these nodes cannot be constant. Hence there exists no constant function $f \in \mathcal{F}^{(1)}(C)$ such that inputs are fixed.

In the second part, we examine the highest degree term in the gadget. The maximum degree term can be found in $\mathcal{R}$ with degree 2. Therefore the corresponding bias and the bias bound of the gadget can be calculated as 2^{-2} and $\epsilon \leq \left|1/2 - 1/2^2\right| = 1/4$ respectively. Thus the `RefreshMask` gadget is ϵ-1-AS with $\epsilon := 1/4$.

Proposition 8: ϵ-2-AS of `RefreshMask`$[n, 2]$ *Gadget*

Proof. First let us consider a function $f \in \mathcal{F}^{(2)}(C \setminus \mathcal{X})$. Assume that there exists a $c \in \mathbb{F}_2^{n+3}$ such that $f(c, \cdot)$ is constant. As seen in Algorithm 3, the nodes that do not contain a random variable (i.e. not fixed) are found in line 11 where the values $\mathcal{W}$ and $\mathcal{R}$ are processed. By the computational structure of $\mathcal{W}$ and $\mathcal{R}$ given in `RefreshMask`$[n, 2]$, the input nodes are accompanied by a random value. Moreover each node contains only one non-linear share, thus any first or second order combination cannot contain all three non-linear shares such that the variable $\tilde{x}_0\tilde{x}_1\tilde{x}_2$ is formed. Hence $f(c, \cdot)$ cannot be a constant for all $c \in \mathbb{F}_2^{n+3}$.

In the second part of the proof we analyze the m such that $m = \max(\deg(c_i)_{c_i \in C \setminus \mathcal{X}})$. Observe that the highest degree term in the gadget can be found in $\mathcal{R}$ with degree 3. Thus the linear bias bound of the gadget can be seen as follows:

$$\frac{1}{2} - \frac{1}{2^2} \leq \epsilon' = \mathcal{E}(f') < \frac{1}{2} - \frac{1}{2^3} \text{ where } f' \in \mathcal{F}^{(1)}(C \setminus \mathcal{X}).$$

This result implies that `RefreshMask`$[n, 1]$ is ϵ'-1-AS gadget. Moreover, the highest degree term of $f \in \mathcal{F}^{(2)}(C \setminus \mathcal{X})$ is less than or equal to 6 which implies:

$$\frac{1}{2} - \frac{1}{2^5} \leq \epsilon = \mathcal{E}(f) < \frac{1}{2} - \frac{1}{2^6} \text{ where } f \in \mathcal{F}^{(2)}(C \setminus \mathcal{X}).$$

Observe that, in the first part of the proof we showed that there exists no function $f \in \mathcal{F}^{(2)}(C \setminus \mathcal{X})$ such that $f(c, \cdot)$ is constant, which implies both linear and second order biases cannot grow. Thus the RefreshMask gadget is ϵ-2-AS with $\epsilon := 31/64$.

B Example Constructions

Example 3. $n = 2$, $d = 1$

Here is an example construction for the $(2, 1)$-masking scheme:

- $\texttt{Encode}(x, x_1, \tilde{x}_0, \tilde{x}_1) = (\tilde{x}_0, \tilde{x}_1, x_1, x_2)$ where $x_2 = \tilde{x}_0\tilde{x}_1 \oplus x_1 \oplus x$.
- $\texttt{Decode}(\overline{x}) = \tilde{x}_0\tilde{x}_1 \oplus x_1 \oplus x_2$.
- $\texttt{Xor}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, z_1, z_2)$ such that $z = x \oplus y$:
  - $\tilde{z}_0 = \tilde{x}_0 \oplus \tilde{y}_0$,
  - $\tilde{z}_1 = \tilde{x}_1 \oplus \tilde{y}_1$,
  - $z_1 = x_1 \oplus y_1$,
  - $z_2 = x_2 \oplus y_2 \oplus \tilde{x}_1\tilde{y}_0 \oplus \tilde{x}_0\tilde{y}_1$.
- $\texttt{And}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, z_1, z_2)$ such that $z = xy$;

  Step-1: First, calculate the multiplicative representations of the output share z_0:
  - $\tilde{z}_0 = \tilde{x}_0\tilde{y}_1 \oplus r^{0,1} \oplus r^{0,2}$,
  - $\tilde{z}_1 = \tilde{x}_1\tilde{y}_0 \oplus r^{1,1} \oplus r^{1,2}$ where $(r^{0,1}, r^{0,2}, r^{1,1}, r^{1,2}) \leftarrow \texttt{rand}(0, 1)$

  Step-2(a): Calculate the intermediate values $r_{j,0}$ which include the reconstruction of the values x_0 and y_0:
  - $r_{1,0} = \tilde{x}_1(\tilde{x}_0 y_1 \oplus r^{0,1}\tilde{y}_0) \oplus \tilde{y}_1(\tilde{y}_0 x_1 \oplus r^{1,1}\tilde{x}_0) \oplus r^{1,1}(r^{0,1} \oplus r^{0,2})$,
  - $r_{2,0} = \tilde{x}_1(\tilde{x}_0 y_2 \oplus r^{0,2}\tilde{y}_0) \oplus \tilde{y}_1(\tilde{y}_0 x_2 \oplus r^{1,2}\tilde{x}_0) \oplus r^{1,2}(r^{0,1} \oplus r^{0,2})$.

  Step-2(b): Calculate the intermediate values $r_{j,0}$ which do not include the reconstruction of the values x_0 and y_0:
  - $r_{1,2} \leftarrow \texttt{rand}(0, 1)$,
  - $r_{2,1} = (r_{1,2} \oplus x_1 y_2) \oplus x_2 y_1$.

  Step-3: Finally, calculate the rest of the shares:
  - $z_1 = x_1 y_1 \oplus r_{1,0} \oplus r_{1,2}$,
  - $z_2 = x_2 y_2 \oplus r_{2,0} \oplus r_{2,1}$.
- $\texttt{RefreshMask}(\overline{x}) = (\tilde{x}_0, \tilde{x}_1, x_1, x_2)$
  1. First, calculate the non-linear components of the output share x_0:
     - $\tilde{x}_0 = \tilde{x}_0 \oplus \tilde{r}_0$,
     - $\tilde{x}_1 = \tilde{x}_1 \oplus \tilde{r}_1$ where $(\tilde{r}_0, \tilde{r}_1) \leftarrow \texttt{rand}(0, 1)$
  2. Calculate the rest the linear masks:
     - $x_1 = x_1 \oplus r_1$,
     - $x_2 = x_2 \oplus r_1$ where $r_1 \leftarrow \texttt{rand}(0, 1)$
  3. Select a random bit $r_0 \leftarrow \texttt{rand}(0, 1)$ and calculate the intermediate variable with $\mathcal{W}$ and $\mathcal{R}$:
     - $\mathcal{W} = \tilde{r}_0(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0)$ and $\mathcal{R} = (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0) \oplus r_0$ where
     - $x_2 = x_2 \oplus \mathcal{W} \oplus \mathcal{R}$

Example 4. Example: $n = 2$, $d = 2$

Here is an example construction for the $(2,2)$-masking scheme:

- $\texttt{Encode}(x, x_1, \tilde{x}_0, \tilde{x}_1, \tilde{x}_2) = (\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, x_1, x_2)$ where $x_2 = \tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus x_1 \oplus x$.
- $\texttt{Decode}(\overline{x}) = \tilde{x}_0\tilde{x}_1\tilde{x}_2 \oplus x_1 \oplus x_2$.
- $\texttt{Xor}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, z_1, z_2)$ such that $z = x \oplus y$
  - $\tilde{z}_i = \tilde{x}_i \oplus \tilde{y}_i$ for $i = \{0,1,2\}$
  - $z_1 = x_1 \oplus y_1$
  - $z_2 = x_2 \oplus y_2 \oplus \tilde{x}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{y}_2(\tilde{x}_0 \oplus \tilde{y}_0)) \oplus \tilde{y}_1(\tilde{x}_2\tilde{y}_0 \oplus \tilde{x}_0(\tilde{x}_2 \oplus \tilde{y}_2))$
- $\texttt{And}(\overline{x}, \overline{y}) = (\tilde{z}_0, \tilde{z}_1, \tilde{z}_2, z_1, z_2)$ such that $z = xy$

  Step-1: First, calculate the multiplicative representations of the output share z_0:
  - $\tilde{z}_0 = \tilde{x}_0\tilde{y}_1 \oplus r^{0,1} \oplus r^{0,2}$,
  - $\tilde{z}_1 = \tilde{x}_1\tilde{y}_2 \oplus r^{1,1} \oplus r^{1,2}$,
  - $\tilde{z}_2 = \tilde{x}_2\tilde{y}_0 \oplus r^{2,1} \oplus r^{2,2}$ where $r^{i,j} \leftarrow \texttt{rand}(0,1)$ for $i = \{0,1,2\}$, $j = \{1,2\}$.

  Step-2(a): Calculate the intermediate values $r_{j,0}$ where the combination of random nodes are defined as; $u = (r^{1,1} \oplus r^{1,2})$ and $v = (r^{2,1} \oplus r^{2,2})$.
  - $r_{1,0} = \mathcal{F}(x_1, y_1) = \tilde{x}_0\left[\tilde{x}_2(\tilde{x}_1 y_1 \oplus r^{0,1}\tilde{y}_0) \oplus r^{1,1}v\tilde{y}_1\right] \oplus$
    $\tilde{y}_0\left[\tilde{y}_1(\tilde{y}_2 x_1 \oplus r^{1,1}\tilde{x}_2) \oplus r^{0,1}u\tilde{x}_2\right] \oplus$
    $\tilde{x}_0\tilde{y}_1(r^{1,1}\tilde{x}_2\tilde{y}_0 \oplus r^{2,1}\tilde{x}_1\tilde{y}_2) \oplus r^{0,1}\tilde{x}_1\tilde{y}_2(v \oplus \tilde{x}_2\tilde{y}_0) \oplus$
    $\tilde{x}_2\tilde{y}_0(r^{0,1}\tilde{x}_0 \oplus r^{1,1}\tilde{y}_1) \oplus uvr^{0,1}$.
  - $r_{2,0} = \mathcal{F}(x_2, y_2) = \tilde{x}_0\left[\tilde{x}_2(\tilde{x}_1 y_2 \oplus r^{0,2}\tilde{y}_0) \oplus r^{1,2}v\tilde{y}_1\right] \oplus$
    $\tilde{y}_0\left[\tilde{y}_1(\tilde{y}_2 x_2 \oplus r^{1,2}\tilde{x}_2) \oplus r^{0,2}u\tilde{x}_2\right] \oplus$
    $\tilde{x}_0\tilde{y}_1(r^{1,2}\tilde{x}_2\tilde{y}_0 \oplus r^{2,2}\tilde{x}_1\tilde{y}_2) \oplus r^{0,2}\tilde{x}_1\tilde{y}_2(v \oplus \tilde{x}_2\tilde{y}_0) \oplus$
    $\tilde{x}_2\tilde{y}_0(r^{0,2}\tilde{x}_0 \oplus r^{1,2}\tilde{y}_1) \oplus uvr^{0,2}$.

  Step-2(b): Calculate the intermediate values $r_{j,0}$ which do not include the reconstruction of the values x_0 and y_0:
  - $r_{1,2} \leftarrow \texttt{rand}(0,1)$,
  - $r_{2,1} = (r_{1,2} \oplus x_1 y_2) \oplus x_2 y_1$.

  Step-3: Finally, calculate the rest of the shares:
  - $z_1 = x_1 y_1 \oplus r_{1,0} \oplus r_{1,2}$,
  - $z_2 = x_2 y_2 \oplus r_{2,0} \oplus r_{2,1}$.
- $\texttt{RefreshMask}(\overline{x}) = (\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, x_1, x_2)$
  1. First, calculate the multiplicative representations of the output share x_0:
     - $\tilde{x}_0 = \tilde{x}_0 \oplus \tilde{r}_0$,
     - $\tilde{x}_1 = \tilde{x}_1 \oplus \tilde{r}_1$,
     - $\tilde{x}_2 = \tilde{x}_2 \oplus \tilde{r}_2$, where $(\tilde{r}_0, \tilde{r}_1, \tilde{r}_2) \leftarrow \texttt{rand}(0,1)$
  2. Calculate the rest the linear shares:
     - $x_1 = x_1 \oplus r_1$,
     - $x_2 = x_2 \oplus r_1$ where $r_1 \leftarrow \texttt{rand}(0,1)$
  3. Select a random bit $r_0 \leftarrow \texttt{rand}(0,1)$ and calculate the intermediate variable with $\mathcal{W}$ and $\mathcal{R}$:

- $\mathcal{W} = \tilde{r}_1\tilde{r}_2(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_0\tilde{r}_2(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_0\tilde{r}_1(\tilde{x}_2 \oplus r_0) \oplus$
  $\tilde{r}_2(\tilde{x}_0 \oplus r_0)(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0)(\tilde{x}_2 \oplus r_0) \oplus \tilde{r}_0(\tilde{x}_1 \oplus r_0)(\tilde{x}_2 \oplus r_0),$
  $\mathcal{R} = (\tilde{r}_0 \oplus r_0)(\tilde{r}_1 \oplus r_0)(\tilde{r}_2 \oplus r_0) \oplus$
  $r_0 \left[\tilde{r}_2(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_0 \oplus r_0) \oplus \tilde{r}_0(\tilde{x}_1 \oplus r_0)\right] \oplus$
  $r_0 \left[\tilde{r}_2(\tilde{x}_1 \oplus r_0) \oplus \tilde{r}_1(\tilde{x}_2 \oplus r_0) \oplus \tilde{r}_0(\tilde{x}_2 \oplus r_0)\right].$
- $x_2 = x_2 \oplus \mathcal{W} \oplus \mathcal{R}$