Probing Security through Input-Output Separation and Revisited Quasilinear Masking

Dahmun Goudarzi[1], Thomas Prest[2], Matthieu Rivain[3] and Damien Vergnaud[4,5]

[1] Independent researcher
dahmun.goudarzi@gmail.com

[2] PQShield, Oxford, United Kingdom
thomas.prest@pqshield.com

[3] CryptoExperts, Paris, France
matthieu.rivain@cryptoexperts.com

[4] Sorbonne Université, CNRS, LIP6, Paris, France
[5] Institut Universitaire de France, Paris, France

Abstract. The probing security model is widely used to formally prove the security of masking schemes. Whenever a masked implementation can be proven secure in this model with a reasonable *leakage rate*, it is also provably secure in a realistic leakage model known as the *noisy leakage model*. This paper introduces a new framework for the composition of probing-secure circuits. We introduce the security notion of *input-output separation* (IOS) for a refresh gadget. From this notion, one can easily compose gadgets satisfying the classical probing security notion –which does not ensure composability on its own– to obtain a *region probing secure* circuit. Such a circuit is secure against an adversary placing up to t probes in each gadget composing the circuit, which ensures a tight reduction to the more realistic noisy leakage model. After introducing the notion and proving our composition theorem, we compare our approach to the composition approaches obtained with the (Strong) Non-Interference (S/NI) notions as well as the Probe-Isolating Non-Interference (PINI) notion. We further show that any uniform SNI gadget achieves the IOS security notion, while the converse is not true. We further describe a refresh gadget achieving the IOS property for any linear sharing with a quasilinear complexity $\Theta(n \log n)$ and a $O(1/\log n)$ leakage rate (for an n-size sharing). This refresh gadget is a simplified version of the quasilinear SNI refresh gadget proposed by Battistello, Coron, Prouff, and Zeitoun (ePrint 2016). As an application of our composition framework, we revisit the quasilinear-complexity masking scheme of Goudarzi, Joux and Rivain (Asiacrypt 2018). We improve this scheme by generalizing it to any base field (whereas the original proposal only applies to field with nth powers of unity) and by taking advantage of our composition approach. We further patch a flaw in the original security proof and extend it from the random probing model to the stronger region probing model. Finally, we present some application of this extended quasilinear masking scheme to AES and MiMC and compare the obtained performances.

Keywords: Probing Security · Composition · Quasilinear Masking · IOS Notion

1 Introduction

In cryptography, side-channel attacks are all attacks based on extracting information from a physical implementation of a cryptosystem. Rather than exploiting some weakness in the underlying cryptographic algorithm, the leakage information is exploited by attackers to extract the secret key from a specific implementation.

Probing security is a notion put forward by Ishai, Sahai and Wagner in [31] to evaluate the security of a circuit against a class of physical attacks. Specifically, they consider *t-probing attacks* in which the adversary has the ability to place some probes on t wires of a circuit processing some secrets. The circuit is said to be *t-probing secure* if no information leaks from the values of the t probed wires. More formally, one should be able to perfectly simulate the distribution of the probed wires without any knowledge on the secrets. In their paper, Ishai *et al.* propose a scheme, the so-called ISW scheme, to compile a circuit into a new randomized circuit (*i.e.* a circuit featuring random generation gates) which is resistant to t-probing attacks. Their scheme used some additive secret sharing (a.k.a. Boolean masking) of the processed variables. Specifically, each variable x is split into $n \geq 2$ variables $x_1, x_2, \ldots, x_n$, called the *shares*, which are uniformly distributed among n-tuples satisfying $x = x_1 + x_2 + \cdots + x_n$ (where $+$ is the addition on $\mathbb{F}_2$ in the original scheme).

Using such an additive sharing to protect a cryptographic computation was already proposed in 1999 as a protection against side-channel attacks [20, 28]. Many *masking schemes* describing efficient implementations of ciphers protected at some given (low) orders were published in the early 2000's, see *e.g.* [34, 3, 35]. In this context, the probing security notion is analogous to the security against so-called *higher-order* side-channel attacks. In such an attack, an adversary uses t leakage points from a power consumption trace (or electromagnetic trace) to extract information on the secret. If properly implemented, a t-probing secure scheme achieves provable security against this kind of attacks. The ISW scheme had hence a strong impact on the side-channel research community and it was used as a building block in many popular masking schemes, see *e.g.* [40, 33, 18, 24, 22, 39, 43, 30, 11, 12, 32].

Although an ISW-based masking scheme can achieve some level of resistance against side-channel attacks, the probing security notion is not fully satisfactory in this context. In practice a side-channel adversary gets some leakage on the full computation and has no reason to limit herself to t leakage points. Nevertheless, the side-channel leakage is often (or can be made) *noisy* and the noise is known to be amplified by the masking order [20]. This was the motivation behind the formal *noisy leakage model* introduced by Prouff and Rivain in [38]. In this model, every variable (or wire) x in the computation leaks a noisy function $f(x)$. The noisy property is captured by assuming that the bias introduced in the distribution of x by an observation of $f(x)$ is smaller than some bound δ.

Subsequently, Duc, Dziembowski and Faust showed that the security in the noisy leakage model could be obtained for a probing-secure scheme through a security reduction [25]. In a nutshell, the so-called DDF reduction considers an intermediate leakage model called the *random-probing model*, which was already considered by Ishai *et al.* in [31] and formalized by Ajtai in [2], in which each variable (or wire) is leaked to the adversary with a given probability p. By applying the Chernoff bound, one gets that a t-probing secure circuit $\widehat{C}$ is also p-random probing secure with $p = O(t/|\widehat{C}|)$ (where $|\widehat{C}|$ denotes the number of wires of $\widehat{C}$). Duc *et al.* could then show a transition from the p-random probing security to the δ-noisy leakage security with $\delta = O(p/|\mathbb{K}|)$ where $|\mathbb{K}|$ is the base field of the computation. It was recently shown that the impact of the field size can be relaxed by refining the granularity of the computation [29] or considering alternative definitions of the noisy leakage model [37].

The DFF reduction and the obtained security in the noisy leakage model is thus mainly impacted by the *leakage rate* (or *probing rate*) which is the ratio between the number of tolerated probes and the size of the circuit [6]. In order to tolerate a significant leakage parameter $\delta = O(t/|\widehat{C}|)$, the leakage rate should be as close as possible to 1. In particular, one should be able to tolerate a number of probes that grows linearly with the circuit.

To this aim, the circuit should achieve the stronger notion of *region probing security* formalized by Andrychowicz, Dziembowski, and Faust in [6], namely it should be separable into regions that each tolerate some amount of probes independently of the total size of the circuit. This notion was already considered in the work of Ishai *et al.* and their scheme was shown to be region probing secure. Specifically, it can tolerate up to $t < n/2$ probes per protected gate, or *gadget*, for a masking order n. Since the ISW gadgets require $O(n^2)$ operations, the obtained leakage rate is of $O(1/n)$. Such a leakage rate is not fully satisfactory since it implies that the leakage noise should decrease linearly with the number of shares. In particular, no security can be obtained for the ISW gadgets in the context of a constant leakage rate (*i.e.* on a given target device) and some practical attacks were exhibited to underline this issue [9].

Fortunately, some schemes are known that achieve constant (or quasi-constant) leakage rates. Such a scheme was first proposed by Ajtai in [2] which achieves random probing security with leakage rate $O(1)$. Another scheme, partly based on Ajtai's work, was proposed by Andrychowicz, Dziembowski, and Faust in [6] which achieves probing security with leakage rate $O(1/\log n)$, and random-probing security with leakage rate $O(1)$. More recently, Ananth, Ishai and Sahai [5] have proposed a conceptually simpler approach to achieve random-probing security with leakage rate $O(1)$. This approach has been further improved by Belaïd, Coron, Prouff, Rivain and Taleb in [13]. In terms of complexity, all these proposals imply a size of the protected circuit of $O(|C|n^2)$ or larger, where $|C|$ is the size of the original circuit. This was recently improved by Goudarzi, Joux and Rivain who proposed a scheme making use of a Fast Fourier Transform-based (FFT) polynomial multiplication to obtain the first construction achieving a $O(|C|n \log n)$ complexity with a $O(1/\log n)$ leakage rate. Unfortunately, their security proof has a flaw that we exhibit in this paper. Moreover, their scheme is restricted to working on base fields including the nth powers of unity, which notably excludes fields of characteristic 2 that are yet essential in some cryptographic primitives (such as the AES block cipher).

In [8], Barthe, Belaïd, Dupressoir, Fouque, Grégoire, Strub and Zucchini formalized the notion of *composable gadgets* which notably allows to prove region probing security. More precisely, they introduced the notion of *Strong Non-Interference* (SNI) which refines the notion of probing security, by separating between external and internal probes in the circuits. SNI security allows composing masked gadgets since the notion implies that gadgets stop the propagation of dependencies. However, compared to classical probing-secure gadgets, SNI gadgets are usually less efficient than probing-secure ones and require more randomness. Another approach consists in composing SNI gadgets and NI gadgets (a relaxation of the SNI notion) in a careful way to achieve security with better performances (see [14] and references therein). In [19], Cassiers and Standaert introduced the notion of *Probe Isolating Non-Interference* (PINI) that allows secure composition and efficient implementations. It relies on the position of probes in a target implementation. Thanks to this notion, linear functions are directly composable and do not require to be refreshed and non-linear operations remain efficient. A circuit achieves PINI-security (and is consequently probing secure) if all its gadgets are PINI but the notion is not sufficient to achieve region-probing security.

In this paper, we introduce a new composition framework to construct circuits (or masked implementations) satisfying the region probing security notion. For this purpose, we formalize the property of *input-output separation* (IOS) for a refresh gadget and we show that it allows to simply construct region probing circuits from (weaker) probing secure gadgets and in particular more efficient gadgets which are only proven probing-secure but not SNI, e.g. [29, 11]. We show that this notion can be obtained from uniform SNI or PINI refresh gadgets but also with a simpler design, namely a variant of the refreshing algorithm due to Battistello, Coron, Prouff and Zeitoun [10]. It is worth mentioning that the original refreshing gadget from [10] was proven SNI but for our purposes, we simplify and extend it and show that it achieves our new IOS security notion. The proposed variant can be used to refresh any kind of linear sharing with a quasilinear complexity $\Theta(n \log n)$ and a $O(1/\log n)$ leakage rate (for an n-size sharing).

We then revisit the quasilinear masking scheme of Goudarzi, Joux and Rivain [29] (which we shall call the GJR scheme hereafter). This scheme is based on a polynomial sharing of the form $a = \sum_i a_i \omega^i$, where a is the plain variable and the a_i's are the corresponding shares, and it uses an FFT-based polynomial multiplication to achieve a quasilinear complexity. We describe an improved version of the GJR scheme which works on any base field, including binary fields, and which relies on our composition framework. We further patch a flaw in the original security proof and extend it from the random probing model to the stronger region probing model. Specifically, our improved GJR scheme is secure in the region probing model provided that the underlying FFT algorithm is probing secure. From this ground, we obtain a probing-secure FFT using the approach of [29], that is by relying on a large field $|\mathbb{F}| = \Theta(2^\lambda)$ and taking ω at random. We hence get a region-probing-secure scheme for large fields. For smaller fields, our result is essentially a security reduction from the region probing security of the full scheme to the probing security of the FFT. Finally, we present an application of our extended GJR scheme and compare it with a more standard scheme based on SNI gadgets for two different ciphers: the Advanced Encryption Standard (AES) [1] and MiMC [4]: a cipher with efficient arithmetic representation on a large field. We show that this masking scheme significantly improves the efficiency of the masked cipher for a masking order $n \geq 64$ for MiMC and $n \geq 512$ for the AES. For the AES instantiation, we present a variant of Gao-Mateer additive FFT [27] with improved efficiency and which may be of independent interest.

2 Background on Probing Secure Circuits

2.1 Notations

In this paper, $\mathbb{K}$ shall denote a finite field. Vectors shall be denoted with bold letters, *e.g.* $\boldsymbol{x}$. For any two random variables (or random vectors) X and Y, we shall write $X \overset{\text{id}}{=} Y$ whenever X and Y are identically distributed. For some positive integer $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \ldots, n\}$. For any two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{K}^n$, $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ denotes their inner product. For any finite set I, we denote by $|I|$ the cardinality of I. Let $I \subseteq [n]$ and $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{K}^n$, we denote by $\boldsymbol{v}_{|I}$ the $|I|$-tuple $(v_i)_{i \in I}$. We shall denote $x \leftarrow \mathcal{X}$ the action of picking x uniformly at random in some set $\mathcal{X}$, and $y \leftarrow \mathcal{A}(x)$ the action of defining y as the output of an algorithm $\mathcal{A}$ on input x. If $\mathcal{A}$ is a probabilistic algorithm, then $y \leftarrow \mathcal{A}(x)$ is a random assignment of y on input x and for a uniform random tape.

2.2 Basic Definitions

Arithmetic circuits. Given a finite field $\mathbb{K}$, an *arithmetic circuit* is a circuit processing elements of $\mathbb{K}$ through simple arithmetic operations. Formally, it is modeled as a directed acyclic graph whose vertices are *gates* that belong to the following types:

- input gate (fan-in 0, fan-out 1) which holds an input value of the circuit,
- output gate (fan-in 1, fan-out 0) which receives an output value of the circuit,
- constant gate (fan-in 0, fan-out 1) which outputs a constant value of $\mathbb{K}$,
- addition gate (fan-in 2, fan-out 1) which outputs the sum (on $\mathbb{K}$) of the two input values,
- subtraction gate (fan-in 2, fan-out 1) which outputs the difference (on $\mathbb{K}$) of the two input values,
- multiplication gate (fan-in 2, fan-out 1) which outputs the product (on $\mathbb{K}$) of the two input values,

– copy gate (fan-in 1, fan-out 2) which outputs two copies of the input.

The addition, subtraction and multiplication gates are further called *operation gates*. The edges of an arithmetic circuit are called the *wires*. A *randomized arithmetic circuit* is a arithmetic circuit augmented with a

– random gate (fan-in 0, fan-out 1) which outputs a fresh uniform random value of $\mathbb{K}$.

Given some assignment of the input gates, all the wires of a circuit can be assigned subsequently following the input-output behavior of the gates, which finally leads to an assignment of the output gates. For an arithmetic circuit C with n input gates and m output gates, we denote $\boldsymbol{y} = C(\boldsymbol{x}) \in \mathbb{K}^m$ the output of C (*i.e.* the assignment of the output gates of C) on input $\boldsymbol{x} \in \mathbb{K}^n$ (*i.e.* when the input gates are assigned to $\boldsymbol{x}$). For a randomized arithmetic circuit C with q random gates, we denote $\boldsymbol{y} = C^{\boldsymbol{\rho}}(\boldsymbol{x})$ the output of C on input $\boldsymbol{x}$ and such that each random gate outputs a coordinate of $\boldsymbol{\rho} \in \mathbb{K}^q$. The parameter $\boldsymbol{\rho}$ is then called the *random tape* of C. Whenever $\boldsymbol{\rho}$ is omitted, $\boldsymbol{y} = C(\boldsymbol{x})$ denotes the random vector obtained for a uniform distribution of $\boldsymbol{\rho}$.

Let C be a randomized arithmetic circuit with n input gates, m output gates, q random gates, and let consider that the wires of C are labeled from 1 to s (where s is the total number of wires in C). Then for any set $\mathcal{W} \subseteq [s]$ with $|\mathcal{W}| = t$, we shall denote by $C^{\boldsymbol{\rho}}_{\mathcal{W}}(\boldsymbol{x}) \in \mathbb{K}^t$ the tuple composed of the assignments of the wires with labels in $\mathcal{W}$ on input $\boldsymbol{x} \in \mathbb{K}^n$ and random tape $\boldsymbol{\rho} \in \mathbb{K}^q$. In particular, each coordinate of $C^{\boldsymbol{\rho}}_{\mathcal{W}}(\boldsymbol{x})$ is a deterministic function of $\boldsymbol{x}$ and $\boldsymbol{\rho}$. Here again, whenever $\boldsymbol{\rho}$ is omitted, $C_{\mathcal{W}}(\boldsymbol{x})$ denotes the random vector obtained for a uniform distribution of $\boldsymbol{\rho}$ on $\mathbb{K}^q$.

Circuit compilers. We now recall the definition of *circuit compilers* as formalized in [5] (but adapted to arithmetic circuits). We shall call a $\mathbb{K}$-*string* any tuple of elements from the base field $\mathbb{K}$.

Definition 1 (Circuit Compiler)**.** A circuit compiler is a triplet of algorithms $(\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ defined as follows:

- $\mathsf{Compile}$ (circuit compilation) is a deterministic algorithm that takes as input an arithmetic circuit C and outputs a randomized arithmetic circuit $\widehat{C}$.
- Encode (input encoding) is a probabilistic algorithm that takes as input a $\mathbb{K}$-string $\boldsymbol{x}$ and outputs a $\mathbb{K}$-string $\widehat{\boldsymbol{x}}$.
- Decode (output decoding) is a deterministic algorithm that takes as input a $\mathbb{K}$-string $\widehat{\boldsymbol{y}}$ and outputs a $\mathbb{K}$-string $\boldsymbol{y}$.

These three algorithms satisfy the following properties:

- **Correctness:** For every arithmetic circuit C of input length ℓ, and for every $x \in \mathbb{K}^\ell$, we have
$$\Pr\big(\mathsf{Decode}\big(\widehat{C}(\widehat{\boldsymbol{x}})\big) = C(\boldsymbol{x}) \;\big|\; \widehat{\boldsymbol{x}} \leftarrow \mathsf{Encode}(\boldsymbol{x})\big) = 1 \ ,$$
where $\widehat{C} = \mathsf{Compile}(C)$.
- **Efficiency:** For some parameter called the *encoding order* $n \in \mathbb{N}$, the running time of $\mathsf{Compile}(C)$ is $\mathrm{poly}(n, |C|)$, the running time of $\mathsf{Encode}(\boldsymbol{x})$ is $\mathrm{poly}(n, |\boldsymbol{x}|)$ and the running time of $\mathsf{Decode}\big(\widehat{\boldsymbol{y}}\big)$ is $\mathrm{poly}(n, |\widehat{\boldsymbol{y}}|)$, where $\mathrm{poly}(n, q) = O(n^{k_1} q^{k_2})$ for some constants k_1, k_2.

Sharings and gadgets. Let $n \in \mathbb{N}$ and let $\boldsymbol{v} \in (\mathbb{K}^*)^n$. A $\boldsymbol{v}$*-linear sharing* of $x \in \mathbb{K}$ is a vector $\boldsymbol{x} \in \mathbb{K}^n$ such that $\langle \boldsymbol{v}, \boldsymbol{x} \rangle = x$. The coordinates of a linear sharing $\boldsymbol{x} \in \mathbb{K}^n$ are called the *shares* of $\boldsymbol{x}$. A random vector $\boldsymbol{x}$ is a *uniform* $\boldsymbol{v}$*-linear sharing* of x if $\langle \boldsymbol{v}, \boldsymbol{x} \rangle = x$ and $\boldsymbol{x}_{|I}$ is uniformly distributed over $\mathbb{K}^t$ for any $I \subset [n]$ with $|I| < n$.

Let $\boldsymbol{v}$-Enc denote a probabilistic algorithm that on input x outputs a uniform $\boldsymbol{v}$-linear sharing of x. For instance $\boldsymbol{v}$-$\mathsf{Enc}(x)$ performs the following:

$$x_1 \leftarrow \mathbb{K};\ x_2 \leftarrow \mathbb{K};\ \cdots\ x_{n-1} \leftarrow \mathbb{K};$$
$$x_n \leftarrow v_n^{-1}\big(x - \langle \boldsymbol{v}, (x_1, \ldots, x_{n-1}, 0)\rangle\big)$$

and returns the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$. We further denote $\boldsymbol{v}$-Dec the deterministic algorithm that on input of a $\boldsymbol{v}$-linear sharing of x outputs x. This algorithm simply computes the inner product $\boldsymbol{v}$-$\mathsf{Dec}(\boldsymbol{x}) = \langle \boldsymbol{v}, \boldsymbol{x} \rangle$.

For any operation $g : (x, y) \in \mathbb{K}^2 \mapsto z \in \mathbb{K}$ and for any vector $\boldsymbol{v} \in \mathbb{K}^n$, a *$\boldsymbol{v}$-gadget* of g is a randomized arithmetic circuit with $2n$ input gates and n output gates, which, on input of a $\boldsymbol{v}$-linear sharing of x and a $\boldsymbol{v}$-linear sharing of y, outputs a $\boldsymbol{v}$-linear sharing of $z = g(x, y)$, for any $x, y \in \mathbb{K}$. In particular, G is a *$\boldsymbol{v}$-gadget* of g if and only if for every random tape $\boldsymbol{\rho}$, $\boldsymbol{v}$-$\mathsf{Dec}(G^{\boldsymbol{\rho}}(\boldsymbol{x}, \boldsymbol{y})) = g(x, y)$. A *$\boldsymbol{v}$-refresh gadget* is a randomized arithmetic circuit with n input gates and n output gates, which, on input of a $\boldsymbol{v}$-linear sharing of x outputs a $\boldsymbol{v}$-linear sharing of x, for any $x \in \mathbb{K}$.

Standard circuit compilers. Consider a family of vectors $\mathcal{V} = \{\boldsymbol{v}_n \in \mathbb{K}^n\}_{n \in \mathbb{N}}$ and three families of gadgets $\mathcal{G}^{\oplus} = \{G_n^{\oplus}\}_{n \in \mathbb{N}}$, $\mathcal{G}^{\otimes} = \{G_n^{\otimes}\}_{n \in \mathbb{N}}$ and $\mathcal{G}^{\mathsf{R}} = \{G_n^{\mathsf{R}}\}_{n \in \mathbb{N}}$ such that for every $n \in \mathbb{N}$, $G_n^{\oplus}$ is a $\boldsymbol{v}_n$-gadget for the addition on $\mathbb{K}$, $G_n^{\otimes}$ is a $\boldsymbol{v}_n$-gadget for the multiplication on $\mathbb{K}$, and G_n^{R} is a $\boldsymbol{v}_n$-refresh gadget.

The *standard circuit compiler* for $(\mathcal{V}, \mathcal{G}^{\oplus}, \mathcal{G}^{\otimes}, \mathcal{G}^{\mathsf{R}})$ with encoding order n is the circuit compiler for which

- Encode applies $\boldsymbol{v}_n$-Enc to each coordinate of the input $\mathbb{K}$-string;
- Decode applies $\boldsymbol{v}_n$-Dec to each coordinate of the input $\mathbb{K}$-string;
- $\mathsf{Compile}$ takes an arithmetic circuit C and outputs the randomized arithmetic circuit $\widehat{C}$ such that each addition gate is replaced by an addition gadget $G_n^{\oplus}$ followed by a refresh gadget G_n^{R}, each multiplication gate is replaced by a multiplication gadget $G_n^{\otimes}$ followed by a refresh gadget G_n^{R}, each constant gate outputting α is replaced by n constant gates with constants $(\alpha \cdot v_1^{-1}, 0, \ldots, 0)$ followed by a refresh gadget G_n^{R} and each copy gate is replaced by a copy of the input sharing (through n copy gates) followed by a refresh gadget G_n^{R} per output sharing.

It is not hard to see that such a circuit compiler achieves correctness and efficiency, provided, for the latter, that the sizes of the gadgets $G_n^{\oplus}$, $G_n^{\otimes}$ and G_n^{R} are polynomial in n.

To ease the presentation, we restrict the notion of standard circuit compiler to three types of gadgets (addition, multiplication and refresh) but in practice we consider compilers for which the addition gadget is replaced by a broader class of *sharewise* gadgets. These gadgets apply a linear operation (addition, subtraction, multiplication by a constant, or any $\mathbb{F}_0$-linear operation if $\mathbb{F}$ is an $\mathbb{F}_0$-module) sharewisely to the input linear sharing(s).

2.3 Probing Security

Throughout the paper, the notion of *simulator* will refer to a polynomial-time probabilistic algorithm. We will say that a random vector $\boldsymbol{w}$ can be *perfectly simulated* (possibly given some input $\boldsymbol{in}$) if there exists a simulator $\mathcal{S}$ that (given $\boldsymbol{in}$) outputs a vector which is identically distributed as $\boldsymbol{w}$ (over the internal randomness of the simulator), which shall be denoted $\mathcal{S}(\boldsymbol{in}) \overset{\text{id}}{=} \boldsymbol{w}$.

Informally speaking, a randomized arithmetic circuit achieves *t-probing security*, if leaking the value of t arbitrary wires (*i.e.* allowing *t probes* on the circuit) does not reveal any information about the input (provided that the latter has been properly encoded). This is formally define hereafter.

Definition 2 (Probing Security)**.** A randomized arithmetic circuit $\widehat{C}$ is *t-probing secure* w.r.t. an encoding algorithm Encode if for every plain input $\boldsymbol{x}$ and for every set $\mathcal{W} \subseteq \big[|\widehat{C}|\big]$, with $|\mathcal{W}| \leq t$, there exists a simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$ such that

$$\mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) \overset{\text{id}}{=} \widehat{C}_{\mathcal{W}}(\mathsf{Encode}(\boldsymbol{x})) \ .$$

A circuit compiler (Compile, Encode, Decode) is said to achieve t-probing security if for every arithmetic circuit C, the randomized arithmetic circuit $\widehat{C} = \mathsf{Compile}(C)$ is t-probing secure w.r.t. Encode. Note that factually, the parameter t is a function of the encoding order n. For instance, the first probing-secure scheme due to Ishai, Sahai and Wagner achieves t-probing security with $t \leq (n-1)/2$ and an efficiency $|\widehat{C}| = \Theta(n^2|C|)$.

Most probing-secure circuit compilers are based on the composition of gadgets. These gadgets are themselves probing-secure w.r.t. the underlying encoding scheme but they must also satisfy *composition* properties so that the overall compiled circuit is probing secure. In particular, the notions of *(strong) non-interference*, or (S)NI and *probe isolating non-interference*, or PINI have been proposed and studied in [8, 14, 7, 19]. In this paper, we introduce another notion called *input-output separation* (see Section 3) which is aimed to enable the composition for a stronger notion of probing security, namely the *region probing security.* In a nutshell, a circuit is *region probing secure* if it is composed of several sub-circuits (*e.g.* several gadgets) that can each tolerate some constant amount of probes (irrespective of the total number of sub-circuits). We shall then consider the *probing rate* (or *leakage rate*) of such a circuit as the maximum ratio between the number of tolerable probes over the size of a sub-circuit. Region probing security is formalized hereafter.

Let us first introduce the notion of circuit partition. For any (randomized) arithmetic circuit C, we call $C \equiv (C_1, C_2, \ldots, C_m)$ a *circuit partition* where each C_i is a sub-circuit of C such that the gates of the C_i's form a partition of the gates of C. We further denote by $\mathcal{W}_{C_i}$ the set of wires with source gate in C_i, so that $\mathcal{W}_{C_1}, \ldots, \mathcal{W}_{C_m}$ is a partition of $[|C|]$.

Definition 3 (Region Probing Security)**.** A randomized circuit $\widehat{C}$ is *r-region probing secure* (*i.e.* with probing rate r) w.r.t. an encoding algorithm Encode if there exists a circuit partition $\widehat{C} \equiv (C_1, C_2, \ldots, C_m)$ such that for every plain input $\boldsymbol{x}$ and for every set $\mathcal{W}_1 \subseteq \mathcal{W}_{C_1}$, $\mathcal{W}_2 \subseteq \mathcal{W}_{C_2}$, $\ldots$, $\mathcal{W}_m \subseteq \mathcal{W}_{C_m}$, with $|\mathcal{W}_i| \leq \lceil r|C_i|\rceil$, there exists a simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$ such that

$$\mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) \overset{\text{id}}{=} \widehat{C}_{\mathcal{W}}(\mathsf{Encode}(\boldsymbol{x})) \ ,$$

where $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2 \cup \ldots \cup \mathcal{W}_m$. A circuit compiler (Compile, Encode, Decode) is *r-region probing secure* if for every circuit C the compiled circuit $\widehat{C} = \mathsf{Compile}(C)$ is r-region probing secure w.r.t. Encode (where r might be a function of the encoding order and the circuit size).

We shall further say that a circuit $\widehat{C}$ is *(r,ε)-region probing secure* (*i.e.* with probing rate r and simulation failure ε), if the simulator fails (*i.e.* returns $\perp$) with probability

$$\Pr\left(\mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) = \perp\right) \leq m \cdot \varepsilon \ ,$$

(m being the number of regions) and returns a perfect simulation otherwise:

$$\left(\mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) \mid \mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) \neq \perp\right) \overset{\text{id}}{=} \widehat{C}_{\mathcal{W}}(\mathsf{Encode}(\boldsymbol{x})) \ .$$

The region probing security is a relevant security property for a cryptographic implementation while considering side-channel attacks. Indeed, security in the so-called *noisy leakage model* which captures the physical reality of power and electromagnetic side-channel leakages can be reduced to region probing security. These notions and reductions are recalled in Appendix B.

3 Composability from Input-Output Separation

3.1 Input-Output Separation

We introduce hereafter the *input-output separation* security notion for a refresh gadget. Such a property has originally been used in the GJR scheme to achieve composition in the random probing model [29]. We formalize this notion hereafter as general composition property to achieve region probing security. For the sake of simplicity, the definition given in this section only considers refresh gadgets but it can be generalized to any kind of gadgets (see Appendix A for a general definition).

We first introduce the notion of uniformity for a gadget which will be a requirement for our new security notion.

Definition 4 (Uniformity)**.** Let $\boldsymbol{v} \in (\mathbb{K}^*)^n$. A $\boldsymbol{v}$-refresh gadget G is *uniform*, if for every $\boldsymbol{x} \in \mathbb{K}^n$, the output $G(\boldsymbol{x})$ is a uniform $\boldsymbol{v}$-linear sharing of $\langle \boldsymbol{v}, \boldsymbol{x} \rangle$.

In the following, we shall say that a pair of vector $(\boldsymbol{x}, \boldsymbol{y}) \in (\mathbb{K}^n)^2$ is *admissible* for a gadget G if there exists a random tape $\boldsymbol{\rho}$ such that $\boldsymbol{y} = G^{\boldsymbol{\rho}}(\boldsymbol{x})$. For an admissible pair $(\boldsymbol{x}, \boldsymbol{y})$ and a set $\mathcal{W} \subseteq [|G|]$, the wire distribution of G in $\mathcal{W}$ *induced* by $(\boldsymbol{x}, \boldsymbol{y})$, denoted $G_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y})$, is the random vector $G^{\boldsymbol{\rho}}_{\mathcal{W}}(\boldsymbol{x})$, *i.e.* the tuple of wire values for the wire indexes in $\mathcal{W}$, obtained for a uniform drawing of $\boldsymbol{\rho}$ among the set $\{\boldsymbol{\rho} \in \mathbb{K}^q \,;\, G^{\boldsymbol{\rho}}_{\mathcal{W}}(\boldsymbol{x}) = \boldsymbol{y}\}$.

Definition 5 (IOS)**.** Let $\boldsymbol{v} \in (\mathbb{K}^*)^n$ and let G be a $\boldsymbol{v}$-refresh gadget with s wires. G is said *t-input-output separative (t-IOS)*, if it is uniform and if for every admissible pair $(\boldsymbol{x}, \boldsymbol{y})$ and every set of wires $\mathcal{W} \subseteq [s]$ with $|\mathcal{W}| \leq t$, there exists a (two-stage) simulator $\mathcal{S}_{G,\mathcal{W}} = \big(\mathcal{S}^{(1)}_{G,\mathcal{W}}, \mathcal{S}^{(2)}_{G,\mathcal{W}}\big)$ such that

1. $\mathcal{S}^{(1)}_{G,\mathcal{W}}(\perp) = (\mathcal{I}, \mathcal{J})$ where $\mathcal{I}, \mathcal{J} \subseteq [n]$, with $|\mathcal{I}| \leq |\mathcal{W}|$ and $|\mathcal{J}| \leq |\mathcal{W}|$;
2. $\mathcal{S}^{(2)}_{G,\mathcal{W}}(\boldsymbol{x}_{|\mathcal{I}}, \boldsymbol{y}_{|\mathcal{J}}) \overset{\text{id}}{=} G_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y})$.

A $\boldsymbol{v}$-refresh gadget is simply said to be *IOS* if it is *n-IOS*.

The above definition generalizes the notion of *input-output linear separability* used in the GJR scheme [29]. Our definition has two differences with the GJR notion:

- the GJR notion requires a deterministic (functional) relation between the probed wires and the input/output shares whereas we only require the ability of simulating the probed wires from some input/output shares;
- the GJR notion requires the knowledge of arbitrary linear combinations of the input/output shares whereas we require the knowledge of some input/output shares.

The first difference makes our definition easier to achieve without impacting the composability. Indeed, in any probing security context, the ability of achieving a perfect simulation is sufficient to prove the security. The second difference makes our definition harder to achieve[1] but more useful to different composition contexts (where the probing security might not rely on linear algebra). Moreover, we describe in Section 4 a refresh gadget achieving our version of input-output separation.

The intuition behind the IOS notion can be understood as follows. Any probing leakage from an IOS refresh gadget can be simulated given a subset of its input shares and output shares. We can therefore reduce the standard region probing security game to a game in which the refresh gadget does not leak anything but its surrounding gadgets leak more. The uniformity property then implies that the leakages from two gadgets separated by a refresh gadget are mutually independent. One can then achieve a perfect simulation of the full leakage through independent simulations of the *separated* leakages from the two gadgets.

This is illustrated on Figure 1. The full probing leakage $(\boldsymbol{w}_1, \boldsymbol{w}_{\mathsf{R}}, \boldsymbol{w}_2)$ can be simulated from $(\boldsymbol{w}_1, \boldsymbol{y}_{|\mathcal{I}}, \boldsymbol{y}'_{|\mathcal{J}}, \boldsymbol{w}_2)$. Moreover, the refresh uniformity implies that, given x, the separated leakages $(\boldsymbol{w}_1, \boldsymbol{y}_{|\mathcal{I}})$ and $(\boldsymbol{w}_2, \boldsymbol{y}'_{|\mathcal{J}})$ are mutually independent. Therefore, if one can simulate $(\boldsymbol{w}_1, \boldsymbol{y}_{|\mathcal{I}})$ on the one hand and $(\boldsymbol{w}_2, \boldsymbol{y}'_{|\mathcal{J}})$ on the other hand, then one can simulate the full leakage.

[1]A set of shares being a particular case of a set of linear combinations of shares.

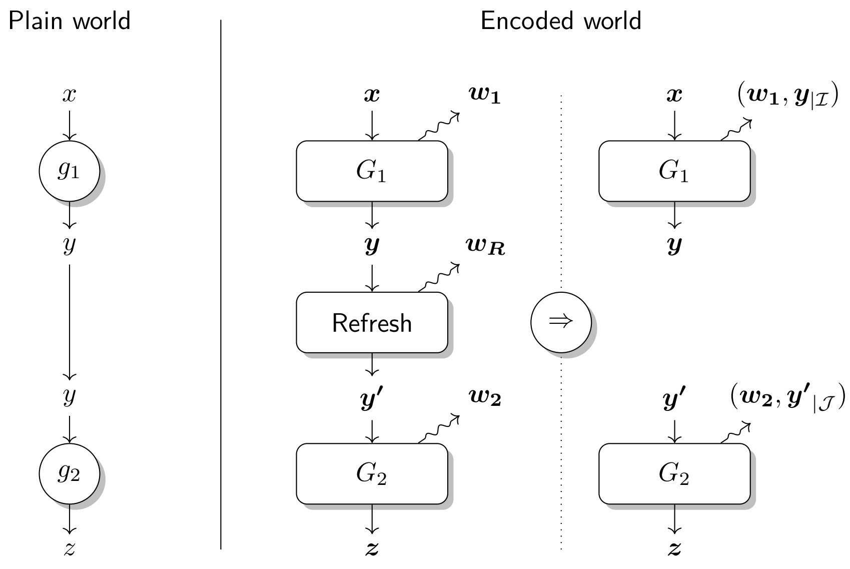

Figure 1: Illustration of the IOS property.

3.2 Composition Theorem

We now provide a formal proof of composition based on the IOS property defined above. Specifically, we show that a standard circuit compiler interleaving operation gadgets and refresh gadgets is region probing secure provided that its operation gadgets are probing secure, and its refresh gadgets are IOS.

As introduced in Section 2, we consider hereafter a family of vectors $\mathcal{V} = \{\boldsymbol{v}_n \in \mathbb{K}^n\}_{n\in\mathbb{N}}$ and three families of gadgets $\mathcal{G}^\oplus = \{G_n^\oplus\}_{n\in\mathbb{N}}$, $\mathcal{G}^\otimes = \{G_n^\otimes\}_{n\in\mathbb{N}}$ and $\mathcal{G}^{\mathsf{R}} = \{G_n^{\mathsf{R}}\}_{n\in\mathbb{N}}$ such that for every $n \in \mathbb{N}$, $G_n^\oplus$ is a $\boldsymbol{v}_n$-gadget for the addition on $\mathbb{K}$, $G_n^\otimes$ is a $\boldsymbol{v}_n$-gadget for the multiplication on $\mathbb{K}$, and G_n^{R} is a $\boldsymbol{v}_n$-refresh gadget. The following theorem gives our composition result for the standard circuit compiler for $(\mathcal{V}, \mathcal{G}^\oplus, \mathcal{G}^\otimes, \mathcal{G}^{\mathsf{R}})$.

Theorem 1. *If for every $n \in \mathbb{N}$,*

- *$G_n^\oplus$ is $t_n^\oplus$-probing secure (w.r.t. $\boldsymbol{v}_n$-Enc),*
- *$G_n^\otimes$ is $t_n^\otimes$-probing secure (w.r.t. $\boldsymbol{v}_n$-Enc),*
- *G^{R_n} is t_n^{R}-IOS,*

then the standard circuit compiler for $(\mathcal{V}, \mathcal{G}^\oplus, \mathcal{G}^\otimes, \mathcal{G}^{\mathsf{R}})$ is r_n-region probing secure with

$$r_n = \max_{t\le t_n^{\mathsf{R}}} \min\Big(\frac{t_n^\oplus - 3t}{|G_n^\oplus|}, \frac{t_n^\otimes - 3t}{|G_n^\otimes|}, \frac{t}{|G_n^{\mathsf{R}}|}\Big) . \tag{1}$$

Proof. Let $n \in \mathbb{N}$ and let $t \le t_n^{\mathsf{R}}$. Let C be an arithmetic circuit composed of m operation gates, and let $\widehat{C}$ be the randomized arithmetic circuit obtained by calling the standard circuit compiler for $(\mathcal{V}, \mathcal{G}^\oplus, \mathcal{G}^\otimes, \mathcal{G}^{\mathsf{R}})$ on C. We shall denote by $G_1, G_2, \ldots, G_m$ the operation gadgets of $\widehat{C}$ and by $G_1^{\mathsf{R}}, G_2^{\mathsf{R}}, \ldots, G_m^{\mathsf{R}}$ the refresh gadgets of $\widehat{C}$ where G_i^{R} is placed in output of G_i for every i. We further denote by $\mathcal{W}_{G_i}$ and $\mathcal{W}_{G_i^{\mathsf{R}}}$ the set of wires with source gate in G_i and G_i^{R} respectively. Finally, we denote t_i the integer such that $t_i = t_n^\oplus - 3t$ if $G_i = G_n^\oplus$ and $t_i = t_n^\otimes - 3t$ otherwise (*i.e.* if $G_i = G_n^\otimes$) for $i \in \{1, \ldots, m\}$.

Let

$$\mathcal{W} = \bigcup_{i=1}^{m} \mathcal{W}_i \cup \bigcup_{i=1}^{m} \mathcal{W}_i^{\mathsf{R}} \subseteq [|\widehat{C}|]$$

where $\mathcal{W}_i \subseteq \mathcal{W}_{G_i}$, with $\mathcal{W}_i \leq t_i$, and $\mathcal{W}_i^{\mathsf{R}} \subseteq \mathcal{W}_{G_i^{\mathsf{R}}}$, with $\mathcal{W}_i^{\mathsf{R}} \leq t$ for every $i \in [m]$. We will show that for any input $\boldsymbol{in}$ of C, there exists a simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$ such that

$$\mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) \overset{\text{id}}{=} \widehat{C}_{\mathcal{W}}(\mathsf{Encode}(\boldsymbol{in})) \ ,$$

which directly implies the r_n-region probing security of the standard circuit compiler with

$$r_n = \min\Big(\frac{t_n^{\oplus} - 3t}{|G_n^{\oplus}|} \, , \, \frac{t_n^{\otimes} - 3t}{|G_n^{\otimes}|} \, , \, \frac{t}{|G_n^{\mathsf{R}}|} \Big) \ .$$

The above shall hold for every $t \leq t_n^{\mathsf{R}}$ which yields the maximum in (1).

The simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$ is simply obtained by running the simulators inherited from the probing security of the G_i's and the IOS property of the G_i^{R}'s. Specifically:

- The IOS property of the G_i^{R}'s implies that, for every $i \in [m]$, there exists a (two-stage) simulator $\mathcal{S}_{G_i^{\mathsf{R}},\mathcal{W}_i^{\mathsf{R}}} = \big(\mathcal{S}^{(1)}_{G_i^{\mathsf{R}},\mathcal{W}_i^{\mathsf{R}}}, \mathcal{S}^{(2)}_{G_i^{\mathsf{R}},\mathcal{W}_i^{\mathsf{R}}}\big)$ such that for every $(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{K}^n \times \mathbb{K}^n$ admissible for G_i^{R}:

  1. $\mathcal{S}^{(1)}_{G_i^{\mathsf{R}},\mathcal{W}_i^{\mathsf{R}}}(\perp) = (\mathcal{I},\mathcal{J})$ where $\mathcal{I},\mathcal{J} \subseteq [n]$, with $|\mathcal{I}| \leq |\mathcal{W}|$ and $|\mathcal{J}| \leq |\mathcal{W}|$;

  2. $\mathcal{S}^{(2)}_{G_i^{\mathsf{R}},\mathcal{W}_i^{\mathsf{R}}}(\boldsymbol{x}_{|\mathcal{I}}, \boldsymbol{y}_{|\mathcal{J}})$ outputs a perfect simulation of $\widehat{C}_{\mathcal{W}_i^{\mathsf{R}}}(\mathsf{Encode}(\boldsymbol{in}))$ given that the pair of input/output sharings of G_i^{R} equals $(\boldsymbol{x},\boldsymbol{y})$.

  Here $\boldsymbol{x}_{|\mathcal{I}}$ corresponds to $|\mathcal{I}| \leq t$ (output) wires of the gadget G_i and $\boldsymbol{y}_{|\mathcal{J}}$ corresponds to $|\mathcal{J}| \leq t$ (input) wires of the gadget G_j subsequent to the refresh G_i^{R}. In particular, there exist two sets $\mathcal{I}_i \subseteq \mathcal{W}_{G_i}$ and $\mathcal{J}_i \subseteq \mathcal{W}_{G_j}$ such that

  $$\boldsymbol{x}_{|\mathcal{I}} = \widehat{C}_{\mathcal{I}_i}(\mathsf{Encode}(\boldsymbol{in})) \ \text{ and } \ \boldsymbol{y}_{|\mathcal{J}} = \widehat{C}_{\mathcal{J}_i}(\mathsf{Encode}(\boldsymbol{in})) \ .$$

- Let ϕ and ψ be the index-mapping functions such that the two input sharings of gadget G_i are output sharings of refresh gadgets $G_{\phi(i)}^{\mathsf{R}}$ and $G_{\psi(i)}^{\mathsf{R}}$. By defining $\overline{\mathcal{W}}_i := \mathcal{W}_i \cup \mathcal{I}_i \cup \mathcal{I}_{\phi(i)} \cup \mathcal{I}_{\psi(i)}$, we get

  $$\bigcup_{i=1}^{m} \overline{\mathcal{W}}_i = \bigcup_{i=1}^{m} \mathcal{W}_i \cup \bigcup_{i=1}^{m} \mathcal{I}_i \cup \bigcup_{i=1}^{m} \mathcal{J}_i \ \text{ with } \ \overline{\mathcal{W}}_i \subseteq \mathcal{W}_{G_i} \ \text{ and } \ |\overline{\mathcal{W}}_i| \leq t_i + 3t \ .$$

- The probing security of the G_i's implies that, for every $i \in [m]$, there exists a simulator $\mathcal{S}_{G_i,\overline{\mathcal{W}}_i}$ such that

  $$\mathcal{S}_{G_i,\overline{\mathcal{W}}_i}(\perp) \overset{\text{id}}{=} \widehat{C}_{\overline{\mathcal{W}}_i}(\mathsf{Encode}(\boldsymbol{in})) \ .$$

We now have all the ingredients to describe the simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$. It proceeds as follows:

1. $\mathcal{S}_{\widehat{C},\mathcal{W}}$ first call the simulators $\mathcal{S}^{(1)}_{G_i^{\mathsf{R}},\mathcal{W}_i^{\mathsf{R}}}(\perp)$ to get the sets $\mathcal{I}_i$'s and $\mathcal{J}_i$'s.
2. $\mathcal{S}_{\widehat{C},\mathcal{W}}$ then calls the simulators $\mathcal{S}_{G_i,\overline{\mathcal{W}}_i}(\perp)$ to get tuples

$$\boldsymbol{z}_i = (\boldsymbol{w}_i, \boldsymbol{x}_i, \boldsymbol{y}_{\phi(i)}, \boldsymbol{y}_{\psi(i)}) \overset{\text{id}}{=} \widehat{C}_{\overline{\mathcal{W}}_i}(\mathsf{Encode}(\boldsymbol{in})) \ ,$$

where $\boldsymbol{w}_i$, $\boldsymbol{x}_i$, $\boldsymbol{y}_{\phi(i)}$, $\boldsymbol{y}_{\psi(i)}$ corresponds to the indexes $\mathcal{W}_i$, $\mathcal{I}_i$, $\mathcal{I}_{\phi(i)}$ and $\mathcal{I}_{\psi(i)}$ respectively. By the uniformity property of the refresh the distributions $\widehat{C}_{\mathcal{W}_{G_i}}(\mathsf{Encode}(\boldsymbol{in}))$, given $\boldsymbol{in}$, are mutually independent which implies

$$(\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_m) \overset{\text{id}}{=} \widehat{C}_{\bigcup_i \overline{\mathcal{W}}_i}(\mathsf{Encode}(\boldsymbol{in})) \ .$$

3. $\mathcal{S}_{\widehat{C},\mathcal{W}}$ finally calls the simulators $\mathcal{S}^{(2)}_{G_i^{\mathsf{R}},\mathcal{W}_i^{\mathsf{R}}}$ on inputs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ to get tuples $\boldsymbol{w}_i^{\mathsf{R}}$ such that
$$\boldsymbol{w}_i^{\mathsf{R}} \overset{\text{id}}{=} \widehat{C}_{\mathcal{W}_i^{\mathsf{R}}}(\mathsf{Encode}(\boldsymbol{in})) \ ,$$
and outputs $(\boldsymbol{w}_1, \boldsymbol{w}_1^{\mathsf{R}}, \ldots, \boldsymbol{w}_m, \boldsymbol{w}_m^{\mathsf{R}})$ as simulation.

The IOS property finally implies

$$\mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) = (\boldsymbol{w}_1, \boldsymbol{w}_1^{\mathsf{R}}, \ldots, \boldsymbol{w}_m, \boldsymbol{w}_m^{\mathsf{R}}) \overset{\text{id}}{=} \widehat{C}_{\mathcal{W}}(\mathsf{Encode}(\boldsymbol{in})) \ ,$$

which concludes the proof.

□

3.3 Comparison with Non-Interference Security Notions

It is well-known that composition of probing secure gadgets is not always probing secure [23]. Stronger security definitions were previously proposed to analyse the security of large circuits viewed as the composition of simple gadgets. The first such notion, (strong) non-interference, or (S)NI, was proposed in [8]. The notion of *Probe Isolating Non-Interference* (PINI) was also recently introduced in [19]. In this section, we compare our composition approach with the ones underlying the (S)NI and PINI notions and then show some implications between these notions and our IOS notion.

We first recall the (S)NI and PINI definitions while extending them from standard Boolean sharing to the general case of $\boldsymbol{v}$-sharings. For a $\boldsymbol{v}$-refresh gadget, the (S)NI notion is defined as follows:

Definition 6 (NI and SNI)**.** Let $\boldsymbol{v} \in (\mathbb{K}^*)^n$ and let G be a $\boldsymbol{v}$-refresh gadget with s wires. G is said t-Non-Interferent (t-NI) (*resp.* t- Strong Non-Interferent (t-SNI)), if for every $\boldsymbol{x}$ and every set of internal wires $\mathcal{W} \subseteq [s]$ with $|\mathcal{W}| \leq t_1$ and every set of output wires $\mathcal{O} \subseteq [s]$ with $|\mathcal{O}| \leq t_2$ and $t_1 + t_2 \leq t$, there exists a (two-stage) simulator $\mathcal{S}_{G,\mathcal{W},\mathcal{O}} = \big(\mathcal{S}^{(1)}_{G,\mathcal{W},\mathcal{O}}, \mathcal{S}^{(2)}_{G,\mathcal{W},\mathcal{O}}\big)$ such that

1. $\mathcal{S}^{(1)}_{G,\mathcal{W},\mathcal{O}}(\perp) = \mathcal{I}$ where $\mathcal{I} \subseteq [n]$, with $|\mathcal{I}| \leq t_1 + t_2$ (*resp.* with $|\mathcal{I}| \leq t_1$);
2. $\mathcal{S}^{(2)}_{G,\mathcal{W},\mathcal{O}}(\boldsymbol{x}_{|\mathcal{I}}) \overset{\text{id}}{=} G_{\mathcal{W}\cup\mathcal{O}}(\boldsymbol{x})$.

A $\boldsymbol{v}$-refresh gadget is simply said to be *NI* (*resp. SNI*) if it is $(n-1)$-*NI* (*resp.* $(n-1)$-*SNI*).

If a gadget achieves NI-security, then a probe of an internal wire or an output wire can be simulated using one probe on each of the input sharings of the gadget. If it achieves the stronger SNI-security notion then only probes of internal wires are propagated to inputs (and it thus guarantees independence between the inputs and outputs even with access to the internal wires).

For a $\boldsymbol{v}$-refresh gadget, the PINI notion is defined as follows:

Definition 7 (PINI)**.** Let $\boldsymbol{v} \in (\mathbb{K}^*)^n$ and let G be a $\boldsymbol{v}$-refresh gadget with s wires. G is said t-Probe Isolating Non-Interferent (t-PINI), if for every $\boldsymbol{x}$ and every set of internal wires $\mathcal{W} \subseteq [s]$ with $|\mathcal{W}| \leq t_1$ and every set of output wires $\mathcal{O} \subseteq [s]$ with $|\mathcal{O}| \leq t_2$ and $t_1 + t_2 \leq t$, there exists a (two-stage) simulator $\mathcal{S}_{G,\mathcal{W},\mathcal{O}} = \big(\mathcal{S}^{(1)}_{G,\mathcal{W},\mathcal{O}}, \mathcal{S}^{(2)}_{G,\mathcal{W},\mathcal{O}}\big)$ such that

1. $\mathcal{S}^{(1)}_{G,\mathcal{W},\mathcal{O}}(\perp) = \mathcal{I}$ where $\mathcal{I} \subseteq [n]$, with $|\mathcal{I}| \leq t_1$;
2. $\mathcal{S}^{(2)}_{G,\mathcal{W},\mathcal{O}}(\boldsymbol{x}_{|\mathcal{I}\cup\mathcal{J}}) \overset{\text{id}}{=} G_{\mathcal{W}\cup\mathcal{O}}(\boldsymbol{x})$;

where $\mathcal{J} \subseteq [n]$ is the set of indices of output shares in $\mathcal{O}$. A $\boldsymbol{v}$-refresh gadget is simply said to be *PINI* if it is $(n-1)$-*PINI.*

Comparison of the composition approaches. We discuss hereafter the composition approaches related to the (S)NI notion, the PINI notion and our new IOS notion.

(S)NI composition approach. The NI and SNI notions were proposed in [8] as composition notions for probing-secure gadgets. The authors show how to compose t-NI and t-SNI gadgets to achieve t-probing security, which was further generalized in [14]. Theses results can actually be extended to region probing security. Let us consider the standard circuit compiler as defined in Section 2. If the underlying refresh gadget is SNI and the underlying addition and multiplication gadgets are NI, then it can be checked that the compiled circuit can tolerate up to $t/2$ probes per gadget. In other words, from an SNI refresh gadget, one simply needs NI operation gadgets to obtain a region probing-secure composition.

PINI composition approach. The PINI notion was introduced to allow trivial composition of probing-secure gadgets [19]. Specifically composing any number of PINI gadgets in any way results in a circuit achieving PINI security which further implies probing security. Another advantage of the PINI notion is that it is satisfied by any sharewise gadget (*i.e.* a gadget which simply applies an operation sharewisely) without requiring any refreshing or randomness. Although the PINI notion enables simpler composition, it is limited to probing security (or PINI security) and cannot be extended to region probing security. To illustrate this impossibility, let us consider the following simple example. Suppose that some circuit compiler applies a single-input sharewise gadget G (for instance squaring on $\mathbb{F}_{256}$) successively many times to an input n-sharing $\boldsymbol{x}$. After N gadgets each leaking t probes, all the shares can be recovered whenever $N > n/t$.

IOS composition approach. Our composition approach consists in interleaving an IOS refresh gadget between any pair of successive operation gadgets of the compiled circuit (as in the definition of the standard circuit compiler). Doing so, we can lower the requirement on the operation gadgets: they simply needs to achieve the weaker notion of probing security (see Theorem 1 above).

Comparison. We compare the three composition approaches for the standard circuit compiler as introduced in Section 2. This compiler basically replaces each gate by the corresponding operation gadget and it interleaves a refresh gadget in each connection between two operation gadgets. Assuming that the refresh gadget satisfies a given notion in $\{\text{SNI}, \text{PINI}, \text{IOS}\}$, we look at *(i)* what is the security notion required for the operation gadgets? *(ii)* what is the obtained security notion for the composed circuit?

- **SNI:**
  - *(i)* The **NI** notion is sufficient for the operation gadgets.
  - *(ii)* The composition of NI operation gadgets and SNI refresh gadgets implies the **region probing security** of the composed circuit.
- **PINI:**
  - *(i)* The **PINI** notion is sufficient for the operation gadgets.
  - *(ii)* The composition of PINI gadgets implies the **probing security** of the composed circuit. Let us stress that with PINI operation gadgets, PINI refresh gadgets are actually useless.

- **IOS:**

  (i) The **probing security** is sufficient for the operation gadgets.

  (ii) The composition of probing-secure operation gadgets and IOS refresh gadgets implies the **region probing security** of the composed circuit.

Our composition approach, hence achieves the stronger notion of region probing security from the weaker notion of probing security for operation gadgets based on the IOS security of the refresh gadget.

Relations between (S)NI, PINI and IOS. Besides the differences in terms of composition approach, it seems that non-interference notions and IOS are separated notions (*i.e.* PI/SNI do not imply IOS and IOS does not imply PI/SNI). First, since the (S)NI and PINI definitions do not require uniformity of the gadget, these notions do not imply IOS security. But even while considering uniform gadgets, these notions are different in nature. On the one hand, the necessity for the IOS simulator to be given output shares $\boldsymbol{y}|_J$ prevents IOS from implying (S)NI or PINI whose simulators only rely on input shares. On the other hand, the IOS simulation is constrained to match a certain pre-sampled output $\boldsymbol{y}$ which is not (entirely) revealed to the simulator. Such a constraint seems incompatible with (S)NI or PINI, hence discarding an implication from these notions to IOS. We stress that a formal proof of this separation (*e.g.* by exhibiting example of gadgets satisfying a notion and not the other) is still an open issue.

4 An Input-Output Separative Refresh Gadget

Battistello, Coron, Prouff and Zeitoun propose in [9] the so-called (template) horizontal side-channel attacks against the ISW [31] and the Rivain-Prouff [40] secure multiplication schemes. These attacks exploit the fact that, for those schemes, the leaking information on each share increases with the number of shares in the presence of a constant leakage rate. Battistello *et al.* describe a variant of the ISW multiplication with probing-security that is heuristically secure against this kind of attacks. In the full version of their paper [10], they further propose a new refreshing gadget with complexity $O(n \log n)$. In this section, we simplify and extend their gadget for any $\boldsymbol{v}$-linear sharing and we prove that it achieves the IOS security notion.

4.1 Refresh Gadget Description

The modified refresh gadget RefreshGadget is described in Algorithm 1. It is defined recursively: for $n = 2$, given a $\boldsymbol{v}$-linear sharing $\boldsymbol{x} = (x_1, x_2)$ it outputs $\boldsymbol{y} = (y_1, y_2) = (x_1 + r, x_2 - r \cdot v_1 \cdot v_2^{-1})$ such that $\langle \boldsymbol{y}, \boldsymbol{v} \rangle = \langle \boldsymbol{x}, \boldsymbol{v} \rangle$ and for $n \geq 4$ a power of 2, the RefreshGadget gadget is applied recursively on the two halves of the share (Steps 4-5) and a post-processing layer is applied to the whole sharing (Steps 6-9). Note that the original refresh gadget proposed in [9] makes use of an additional and similar pre-processing layer before the two recursive calls to the RefreshGadget gadget, but this layer is not necessary to achieve the IOS property. It results that our variant is twice more efficient in terms of computation and randomness generation.

Algorithm 1 RefreshGadget

```
Require: $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{v} = (v_1, \ldots, v_n)$,
Ensure: $\boldsymbol{y} = (y_1, \ldots, y_n)$ such that $\langle \boldsymbol{y}, \boldsymbol{v} \rangle = \langle \boldsymbol{x}, \boldsymbol{v} \rangle$
1: if $n = 2$ then
2:     $r \xleftarrow{R} \mathbb{F}$
3:     return $(x_1 + r, x_2 - r \cdot v_1 \cdot v_2^{-1})$
4: $(s_1, \ldots, s_{n/2}) \leftarrow \mathsf{RefreshGadget}(a_1, \ldots, a_{n/2}; v_1, \ldots, v_{n/2})$        ▷ Recursive call
5: $(s_{n/2+1}, \ldots, s_n) \leftarrow \mathsf{RefreshGadget}(a_{n/2+1}, \ldots, a_n; v_{n/2+1}, \ldots, v_n)$        ▷ Recursive call
6: for $i = 1, \ldots, n/2$ do
7:     $r_i \xleftarrow{R} \mathbb{F}$
8:     $y_i \leftarrow s_i + r_i$
9:     $y_{i+n/2} \leftarrow s_{i+n/2} - r_i \cdot v_i \cdot v_{i+n/2}^{-1}$
```

Let us denote $R(n)$, $A(n)$ and $M(n)$ the randomness complexity, the number of additions and the number of scalar multiplications of the RefreshGadget algorithm for length-n linear sharing. We have $R(2) = 1$, $A(2) = 2$ and $M(2) = 1$ and $R(2n) = 2R(n)+n$, $A(2n) = 2A(n) + 2n$ and $M(2n) = 2M(n) + n$ for all $n \geq 2$. By induction, we thus have for any $n \geq 2$, a power of 2, we obtain

$$R(n) = M(n) = n\log(n)/2 \quad \text{and} \quad A(n) = n\log(n) \ . \tag{2}$$

4.2 Proof of Input-Output Separation

Theorem 2. *The refresh gadget from Algorithm 1 is input-output separative.*

Proof. Throughout the proof, we denote by $L = \left[\frac{n}{2}\right]$ and $H = [n] \setminus L$.

Uniformity. Let $\boldsymbol{v} \in (\mathbb{K}^*)^n$. We show that the $\boldsymbol{v}$-refresh gadget RefreshGadget is *uniform*, namely that if for every $\boldsymbol{x} \in \mathbb{K}^n$, the output RefreshGadget$(\boldsymbol{x})$ is a uniform $\boldsymbol{v}$-linear sharing of $\langle \boldsymbol{v}, \boldsymbol{x} \rangle$. The proof is by induction on n.

For $n = 2$, given $\boldsymbol{x} = (x_1, x_2) \in \mathbb{K}^2$, the gadget outputs $\boldsymbol{y} = (y_1, y_2)$ defined as $(x_1 + r, x_2 - r \cdot v_1 \cdot v_2^{-1})$ where r is picked uniformly at random in $\mathbb{K}$ and one can see readily that $\boldsymbol{y}$ is a uniformly distributed $\boldsymbol{v}$-linear sharing of $\langle \boldsymbol{v}, \boldsymbol{x} \rangle$.

For $n \geq 4$, given $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{K}^n$, the gadget first computes $(s_1, \ldots, s_{n/2})$ and $(s_{n/2+1}, \ldots, s_n)$ as outputs of RefreshGadget$(x_1, \ldots, x_{n/2})$ and RefreshGadget$(x_{n/2+1}, \ldots, x_n)$. By the induction hypothesis, $\boldsymbol{s}_{|L} = (s_1, \ldots, s_{n/2})$ and $\boldsymbol{s}_{|H} = (s_{n/2+1}, \ldots, s_n)$ are uniform and independent $\boldsymbol{v}_{|L} = (v_1, \ldots, v_{n/2})$-linear sharing and $\boldsymbol{v}_{|H} = (v_{n/2+1}, \ldots, v_n)$-linear sharing of $\langle \boldsymbol{v}_{|L}, \boldsymbol{x}_{|L} \rangle$ and $\langle \boldsymbol{v}_{|H}, \boldsymbol{x}_{|H} \rangle$ (respectively) where $\boldsymbol{x}_{|L} = (x_1, \ldots, x_{n/2})$ and $\boldsymbol{x}_{|H} = (x_{n/2+1}, \ldots, x_n)$. The gadget RefreshGadget then picks uniformly at random r_i in $\mathbb{K}$ for $i \in L$ and sets $y_i = s_i + r_i$ and $y_{i+n/2} = s_{i+n/2} - r_i \cdot v_i \cdot v_{i+n/2}^{-1}$ for $i \in L$. Denoting $r'_i = s_i + r_i (= y_i)$ for $i \in L$ the vector $(r'_1, \ldots, r'_{n/2})$ is uniformly distributed in $\mathbb{K}^{n/2}$ and independent from $\boldsymbol{c}$ and we have $y_{i+n/2} = s_{i+n/2} - (r'_i - s_i) \cdot v_i \cdot v_{i+n/2}^{-1}$ for $i \in L$ where $\boldsymbol{s}_{|L}$ and $\boldsymbol{s}_{|H}$ are uniform and independent $\boldsymbol{v}_{|L}$-linear sharing and $\boldsymbol{v}_{|H}$-linear sharing of $\langle \boldsymbol{v}_{|L}, \boldsymbol{x}_{|L} \rangle$ and $\langle \boldsymbol{v}_{|H}, \boldsymbol{x}_{|H} \rangle$. The vector $\boldsymbol{d}$ is therefore a uniformly distributed $\boldsymbol{v}$-linear sharing of $\langle \boldsymbol{v}, \boldsymbol{x} \rangle$.

IOS. We now show the IOS property. In the following we shall denote by $\boldsymbol{w} = G_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y})$ the wire distribution for $\mathcal{W}$ induced by $(\boldsymbol{x}, \boldsymbol{y})$. Without loss of generality, we assume that if the attacker probes a product of a random value r_i by some publicly known constant (such as the value $r_i \cdot v_i \cdot v_{i+n/2}^{-1}$ which appear in Step 9), then it can be replaced by an attacker which probes directly the random value r_i. We show how to achieve a perfect simulation of $\boldsymbol{w}$ from $\boldsymbol{x}_{|I}$ and $\boldsymbol{y}_{|J}$ for two sets I and J such that $|I| \leq |\mathcal{W}|$ and $|J| \leq |\mathcal{W}|$.

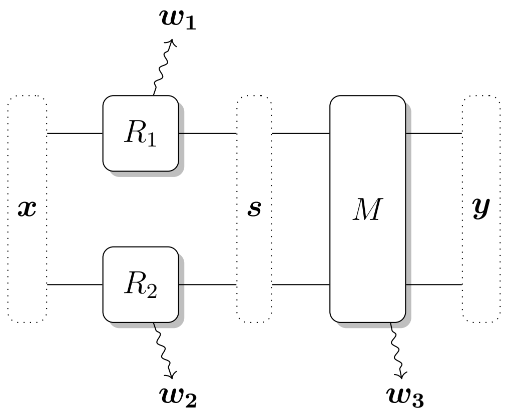

Figure 2: IOS refresh gadget with pobes $(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3)$.

The case $|\mathcal{W}| \geq n$ is straightforward: we let $I = J = [n]$ and we can simulate $\boldsymbol{w}$ (and more generally all the wires of G) directly from the full sharings $(\boldsymbol{x}, \boldsymbol{y})$ and by picking a random tape uniformly for the set $\{\boldsymbol{\rho} \in \mathbb{K}^q \,;\, G^{\boldsymbol{\rho}}_{\mathcal{W}}(\boldsymbol{x}) = \boldsymbol{y}\}$. We therefore consider $|\mathcal{W}| < n$ in the following. The proof also works by induction on n.

For $n = 2$, we have $|\mathcal{W}| \leq 1$. The case $|\mathcal{W}| = 0$ is straightforward. For $|\mathcal{W}| = 1$, the tuple $\boldsymbol{w}$ consists in a single variable among x_1, x_2, y_1, y_2, and r, the output of the single random gate. If the variable is either x_1, y_1 or r, we let $I = J = \{1\}$, otherwise we let $I = J = \{2\}$. The simulation is straightforward for any x_i or y_i. For r, we simply let $r = y_1 - x_1$.

For $n \geq 2$, we denote R_1 the gadget corresponding to the first recursive call to RefreshGadget (Step 4), R_2 the gadget corresponding to the second recursive call to RefreshGadget (Step 5) and M the gadget corresponding to the post-processing layer (Steps 6-9). We denote by $\mathcal{W}_1$, $\mathcal{W}_2$, and $\mathcal{W}_3$, the subset of $\mathcal{W}$ corresponding to wire indexes from R_1, R_2, and M respectively, so that $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2 \cup \mathcal{W}_3$. Without loss of generality, all the outputs of R_1 and R_2, which are also inputs of M, are included to $\mathcal{W}_1$ and $\mathcal{W}_2$, but not to $\mathcal{W}_3$. We further denote $t_1 = |\mathcal{W}_1|$, $t_2 = |\mathcal{W}_2|$, and $t_3 = |\mathcal{W}_3|$, so that

$$t_1 + t_2 + t_3 = |\mathcal{W}| < n \ ,$$

as well as $\boldsymbol{w}_1 = G_{\mathcal{W}_1}(\boldsymbol{x}, \boldsymbol{y})$, $\boldsymbol{w}_2 = G_{\mathcal{W}_2}(\boldsymbol{x}, \boldsymbol{y})$, and $\boldsymbol{w}_3 = G_{\mathcal{W}_3}(\boldsymbol{x}, \boldsymbol{y})$, so that

$$(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3) = \boldsymbol{w} \ .$$

Let $\boldsymbol{s} = (s_1, \ldots, s_n)$ denote the linear sharing in output of the block $(R_1 \parallel R_2)$, and $\boldsymbol{s}_{|L} = (s_1, \ldots, s_{\frac{n}{2}})$ and $\boldsymbol{s}_{|H} = (s_{\frac{n}{2}+1}, \ldots, s_n)$ be the respective output of R_1 and R_2. By definition of the layer M, we have $\langle \boldsymbol{v}_{|L}, \boldsymbol{s}_{|L} \rangle + \langle \boldsymbol{v}_{|H}, \boldsymbol{s}_{|H} \rangle = \langle \boldsymbol{v}_{|L}, \boldsymbol{y}_{|L} \rangle + \langle \boldsymbol{v}_{|H}, \boldsymbol{y}_{|H} \rangle$. On the other hand, the gadgets R_1 and R_2 are uniform, which implies that the tuples $\boldsymbol{s}_{|L}$ and $\boldsymbol{s}_{|H}$ are uniformly distributed among the tuples of $\mathbb{K}^{\frac{n}{2}}$ satisfying

$$\begin{aligned} \langle \boldsymbol{v}_{|L}, \boldsymbol{s}_{|L} \rangle &= \langle \boldsymbol{v}_{|L}, \boldsymbol{x}_{|L} \rangle \ , & (3) \\ \langle \boldsymbol{v}_{|H}, \boldsymbol{s}_{|H} \rangle &= \langle \boldsymbol{v}_{|H}, \boldsymbol{x}_{|H} \rangle) \ , & (4) \\ \langle \boldsymbol{v}_{|L}, \boldsymbol{s}_{|L} \rangle + \langle \boldsymbol{v}_{|H}, \boldsymbol{s}_{|H} \rangle &= \langle \boldsymbol{v}_{|L}, \boldsymbol{y}_{|L} \rangle + \langle \boldsymbol{v}_{|H}, \boldsymbol{y}_{|H} \rangle \ , & (5) \end{aligned}$$

By induction R_1 and R_2 are IOS, which implies that we can perfectly simulate $(\boldsymbol{w}_1, \boldsymbol{w}_2)$ from $\boldsymbol{x}_{|I_1}$, $\boldsymbol{x}_{|I_2}$, $\boldsymbol{s}_{|J_1}$, and $\boldsymbol{s}_{|J_2}$ for some sets $I_1, J_1 \subseteq L$, $I_2, J_2 \subseteq H$, such that $|I_1|, |J_1| \leq t_1$, and $|I_2|, |J_2| \leq t_2$. Without loss of generality, if $t_1 \geq \frac{n}{2}$, we let $I_1 = J_1 = L$, and if $t_2 \geq \frac{n}{2}$, we let $I_2 = J_2 = H$.

The set I is simply constructed as $I = I_1 \cup I_2$. We show hereafter how to achieve a perfect simulation of $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$ from $\boldsymbol{x}_{|I}$ and $\boldsymbol{y}_{|J}$ for some set J such that $|J| \leq |\mathcal{W}|$. As depicted in Figure 3, we will hence get a perfect simulation of $(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3) = \boldsymbol{w}$ from $\boldsymbol{x}_{|I}$ and $\boldsymbol{y}_{|J}$ which shall complete the proof.

$$\begin{array}{ccc}
\boldsymbol{x}_{|I}, \quad \boldsymbol{y}_{|J} & \Rightarrow & (\underbrace{\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2},} \quad \boldsymbol{w}_3) \\
 & & \Downarrow \\
\boldsymbol{x}_{|I} & \Rightarrow & (\boldsymbol{w}_1, \boldsymbol{w}_2)
\end{array}$$

Figure 3: Overview of the simulation process.

We now show how to achieve a perfect simulation of $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$ from $\boldsymbol{x}_{|I}$ and $\boldsymbol{y}_{|J}$ for some set J such that $|J| \leq |\mathcal{W}|$. Let $\mathcal{V}_i$ denote the set of variables $\{s_i, r_i, y_i, s_{i+\frac{n}{2}}, y_{i+\frac{n}{2}}\}$. All the variables in $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$ are included in $\bigcup_{i \in L} \mathcal{V}_i$, where

- $\boldsymbol{s}_{|J_1}$ contains some of the $(s_i)_{i \in L}$;
- $\boldsymbol{s}_{|J_2}$ contains some of the $(s_{i+\frac{n}{2}})_{i \in L}$;
- $\boldsymbol{w}_3$ contains some of the $(r_i)_{i \in L}$, $(y_i)_{i \in L}$, and $(y_{i+\frac{n}{2}})_{i \in L}$.

We construct J as follows. For every $i \in L$:

- if two or more variables from $\mathcal{V}_i$ appear in $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$, we include i and $i + \frac{n}{2}$ to J;
- if one single variable from $\mathcal{V}_i$ appears in $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$,
  - if the variable is s_i, r_i, or y_i, we include i to J;
  - if the variable is $s_{i+\frac{n}{2}}$, or $y_{i+\frac{n}{2}}$, we include $i + \frac{n}{2}$ to J;
- if no variables from $\mathcal{V}_i$ appear in $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$, we do not include i nor $i + \frac{n}{2}$ to J.

Clearly the number of indices added to J is at most the number of variables in $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$, that is

$$|J| \leq |J_1| + |J_2| + t_3 \leq t_1 + t_2 + t_3 = |\mathcal{W}| \ .$$

We now explain how to simulate $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$ from $\boldsymbol{x}_{|I}$ and $\boldsymbol{y}_{|J}$ while satisfying the constraints (3), (4) and (5).

Let us first assume that we have $t_1 < \frac{n}{2}$ and $t_2 < \frac{n}{2}$. This implies that not all the $(s_i)_{i \in L}$ (resp. the $(s_{i+\frac{n}{2}})_{i \in L}$) have to be simulated, which releases the constraints (3) and (4). By construction of J, the $(y_i)_{i \in L}$ and $(y_{i+\frac{n}{2}})_{i \in L}$ that appear in $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$ are also included in $\boldsymbol{y}_{|J}$ and can thus be straightly and perfectly simulated. Then, for every $i \in L$:

- if $\{i, i + \frac{n}{2}\} \subseteq J$, we can perfectly simulate s_i, r_i, $s_{i+\frac{n}{2}}$ from y_i and $y_{i+\frac{n}{2}}$ as

$$\begin{aligned}
&r_i \leftarrow \mathbb{K} \\
&s_i \leftarrow y_i + r_i \\
&s_{i+\frac{n}{2}} \leftarrow y_{i+\frac{n}{2}} - r_i \cdot v_i \cdot v_{i+n/2}^{-1}
\end{aligned}$$

- if $i \in J$ and $i + \frac{n}{2} \notin J$, we can perfectly simulate s_i and r_i from y_i as

$$\begin{aligned}
&r_i \leftarrow \mathbb{K} \\
&s_i \leftarrow y_i + r_i
\end{aligned}$$

- if $i \notin J$ and $i + \frac{n}{2} \in J$, we can perfectly simulate $s_{i+\frac{n}{2}}$ from $y_{i+\frac{n}{2}}$ as

$$\begin{aligned} &r_i \leftarrow \mathbb{K} \\ &s_{i+\frac{n}{2}} \leftarrow y_{i+\frac{n}{2}} - r_i \cdot v_i \cdot v_{i+n/2}^{-1} \end{aligned}$$

This way we have completed a full simulation of all the variables appearing in $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$ while satisfying the constraint (5).

Let us now relax the assumption on t_1 and t_2. Since $t_1 + t_2 < n$ we have $t_1 < \frac{n}{2}$ or $t_2 < \frac{n}{2}$. Without loss of generality, we assume $t_2 < \frac{n}{2}$ (the case $t_1 < \frac{n}{2}$ would be handled similarly). Therefore we might have $t_1 \geq \frac{n}{2}$ implying $I_1 = J_1 = L$, $L \subseteq I$ and $L \subseteq J$. In that case, we must simulate all the $(s_i)_{i \in L}$ while satisfying the constraint (3). The simulation works as above, but we keep one index $i^* \in L$ for which $i^* + \frac{n}{2} \notin J$ for the end of the simulation (such an index exists otherwise we would have $|J| = n > |\mathcal{W}|$). For this index i^*, we can perfectly simulate s_{i^*} and r_{i^*} from $\boldsymbol{x}_{|I_1} = \boldsymbol{x}_{|L}$, $(s_i)_{i \in L \setminus \{i^*\}}$ and y_{i^*} as

$$\begin{aligned} s_{i^*} &\leftarrow \langle \boldsymbol{v}_{|L}, \boldsymbol{x}_{|L} \rangle - \sum_{i \in L \setminus \{i^*\}} s_i \cdot v_i \\ r_{i^*} &\leftarrow y_{i^*} - s_{i^*} \end{aligned}$$

We thus achieve a perfect simulation of $(\boldsymbol{s}_{|J_1}, \boldsymbol{s}_{|J_2}, \boldsymbol{w}_3)$ from $\boldsymbol{x}_{|I}$ and $\boldsymbol{y}_{|J}$ while satisfying the constraints (3), (4) and (5), which concludes the proof. □

5 Revisiting the GJR Masking Scheme

In this section, we revisit the quasilinear-complexity Goudarzi-Joux-Rivain (GJR) masking scheme [29]. We first describe a variant of this scheme making use of the IOS refresh gadget described above and which is more general than the original scheme in the sense that it works on any base field $\mathbb{K}$ equipped with an Fast Fourier Transform (FFT) for multiple-point polynomial evaluation. We then show that the use of our refresh allows to patch a flaw in the security proof of the original scheme. We shall refer to the improved GJR scheme as the GJR$^+$ scheme hereafter.

5.1 The GJR$^+$ Scheme

As in the original scheme, the GJR$^+$ scheme is based on so called ω-encodings which are $\boldsymbol{v}_\omega$-linear sharings with

$$\boldsymbol{v}_\omega = (1, \omega, \dots, \omega^{n-1}) \ . \tag{6}$$

For such a vector, a sharing $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ of a plain value $x \in \mathbb{K}$ can be seen as the coefficients of a polynomials $P_{\boldsymbol{x}} = \sum_{i=1}^{n} x_i \cdot \alpha^{i-1} \in \mathbb{K}[\alpha]$ such that $P_{\boldsymbol{x}}(\omega) = x$. The quasilinear complexity can then be achieved by using efficient FFT-based multiplication for the multiplication gadget. Note that such encoding is close to but different from *Shamir's secret sharing* [41]. In the latter the shares are defined as evaluations of a polynomial in fixed points and for which the plain value is the degree-0 coefficient.

We assume the existence of a Fast Fourier Transform (FFT) algorithm that, given any polynomial $P \in \mathbb{K}[\alpha]$ of degree $< 2n$, maps the coefficients of P to the evaluations of P over $2n$ points of $\mathbb{K}$, with a complexity of $\widetilde{O}(n)$ operations. That is:

$$\mathsf{FFT}_{\boldsymbol{\alpha}} : (x_1, x_2, \dots, x_{2n}) \mapsto (u_1, u_2, \dots, u_{2n}) \quad \text{with} \quad u_j = \sum_{i=1}^{2n} x_i \cdot \alpha_j^{i-1}$$

for every $j \in [2n]$, for some $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_{2n}) \in \mathbb{K}^{2n}$. We further assume that this FFT algorithm can be written as an arithmetic circuit on $\mathbb{K}$ solely composed of additions, subtractions and multiplication by constants in $\mathbb{K}$, and that it features an inverse FFT algorithm with the same properties (in terms of type and number of operations).

The GJR$^+$ scheme is a standard circuit compiler for $(\mathcal{V}, \mathcal{G}^{\oplus}, \mathcal{G}^{\otimes}, \mathcal{G}^{\mathsf{R}})$ (see definition in Section 2), with $\mathcal{V} = \{\boldsymbol{v}_\omega^{(n)}\}_{n\in\mathbb{N}}$ where $\boldsymbol{v}_\omega^{(n)} \in \mathbb{K}^n$ is the vector defined in (6). As in the original scheme, we assume in the following that the order n is a power of two. The scheme could be easily extended to deal with non-power of two at the cost of a small constant efficiency factor.

We now give the description of the associated $\boldsymbol{v}_\omega^{(n)}$-gadgets. For the sake of clarity we shall omit the superscript and simply note $\boldsymbol{v}_\omega$ in what follows.

Refresh Gadget. We use the refresh gadget of Section 4 (see Algorithm 1) for $\boldsymbol{v}_\omega$-sharings *i.e.* with encoding vector $\boldsymbol{v}$ assigned to $\boldsymbol{v}_\omega$. This refresh gadget is applied in output of each operation gadget (in accordance to the definition of the standard circuit compiler). We recall that this gadget achieves the uniformity and IOS properties defined in Subsection 3.1.

Addition Gadget. Given two $\boldsymbol{v}_\omega$-sharings $\boldsymbol{x} = (x_1, \dots, x_n)$ and $\boldsymbol{y} = (y_1, \dots, y_n)$, the addition gadget outputs

$$\boldsymbol{x} + \boldsymbol{y} = (x_1 + y_1, \dots, x_n + y_n) \ .$$

This is done *via* n addition gates processing each share separately. Hence this addition gadget achieves $(n-1)$-probing security.

Subtraction Gadget. Given two $\boldsymbol{v}_\omega$-sharings $\boldsymbol{x} = (x_1, \dots, x_n)$ and $\boldsymbol{y} = (y_1, \dots, y_n)$, the subtraction gadget outputs

$$\boldsymbol{x} - \boldsymbol{y} = (x_1 - y_1, \dots, x_n - y_n) \ .$$

This is done *via* n subtraction gates processing each share separately. Hence this subtraction gadget achieves $(n-1)$-probing security.

Multiplication Gadget. Let $\boldsymbol{v}'_\omega \in \mathbb{K}^{2n}$ be the vector defined as

$$\boldsymbol{v}'_\omega = \mathsf{FFT}_{\boldsymbol{\alpha}}^{-1}(1, \omega, \omega^2, \dots, \omega^{2n-1}) \ .$$

Let $\mathsf{Compress}$ be the $\mathbb{K} \times \mathbb{K}^{2n} \to \mathbb{K}^n$ mapping defined as

$$\mathsf{Compress}(\omega; t_1, t_2, \dots, t_{2n}) = (t_1 + \omega^n \cdot t_{n+1},\ t_2 + \omega^n \cdot t_{n+2},\ \dots,\ t_n + \omega^n \cdot t_{2n})$$

Let $\mathbf{0}$ denotes the n-dimensional all-0 vector and $\|$ denote the concatenation operator. Given two $\boldsymbol{v}_\omega$-sharings $\boldsymbol{x}$ and $\boldsymbol{y}$, the multiplication gadget proceeds as

1. $\boldsymbol{r} \leftarrow \mathsf{FFT}_{\boldsymbol{\alpha}}(\boldsymbol{x} \parallel \mathbf{0})$
2. $\boldsymbol{s} \leftarrow \mathsf{FFT}_{\boldsymbol{\alpha}}(\boldsymbol{y} \parallel \mathbf{0})$
3. $\boldsymbol{u} \leftarrow \boldsymbol{r} \cdot \boldsymbol{s}$
4. $\boldsymbol{u}' \leftarrow \mathsf{Refresh}(\boldsymbol{v}'_\omega; \boldsymbol{u})$
5. $\boldsymbol{t} \leftarrow \mathsf{FFT}_{\boldsymbol{\alpha}}^{-1}(\boldsymbol{u}')$
6. $\boldsymbol{z} \leftarrow \mathsf{Compress}(\omega; \boldsymbol{t})$

and outputs $\boldsymbol{z}$. Note that $\boldsymbol{r}$, $\boldsymbol{s}$, $\boldsymbol{u}$, $\boldsymbol{u}'$, $\boldsymbol{t}$ are $(2n)$-dimensional vectors. Only the input/output sharings $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are n-dimensional vectors. The procedure is depicted on Figure 4 for illustration.

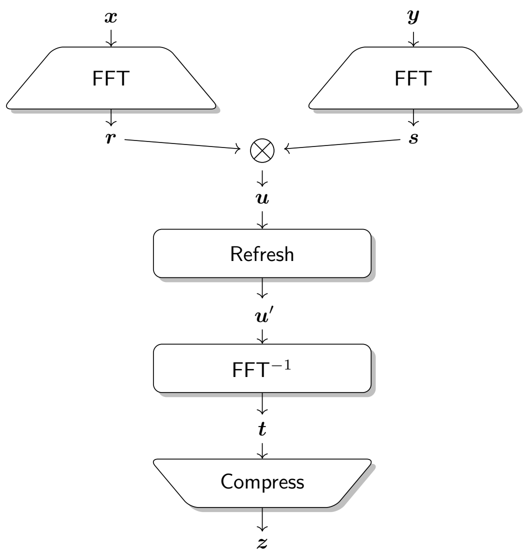

Figure 4: Multiplication gadget.

Remark. This multiplication gadget is similar to the GJR multiplication gadget but we introduce a refreshing in Step 4. This refreshing is done using Algorithm 1 (see Section 4) where the encoding vector $\boldsymbol{v}'_\omega$ and the input sharing are of size $2n$.

Correctness. Let x and y be the values encoded by $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively and let $P_{\boldsymbol{x}} \in \mathbb{K}[\alpha]$ and $P_{\boldsymbol{y}} \in \mathbb{K}[\alpha]$ be the degree-$(n-1)$ polynomials whose coefficients are the coordinates of $\boldsymbol{x}$ and $\boldsymbol{y}$, so that we have $P_{\boldsymbol{x}}(\omega) = x$ and $P_{\boldsymbol{y}}(\omega) = y$.

Let us first assume that Step 4 applies an identity mapping, *i.e.* $\boldsymbol{u}' = \boldsymbol{u}$. Then Steps 1–5 perform a classical FFT-based polynomial multiplication. Namely, the coordinates of $\boldsymbol{t}$ are the coefficients of the polynomial $P_{\boldsymbol{t}} \in \mathbb{K}[\alpha]$ such that $P_{\boldsymbol{t}}(\alpha) = P_{\boldsymbol{x}}(\alpha) \cdot P_{\boldsymbol{y}}(\alpha)$, and in particular $P_{\boldsymbol{t}}(\omega) = x \cdot y$. Then Step 6 outputs a vector $\boldsymbol{z}$ such that $\langle \boldsymbol{v}_\omega, \boldsymbol{z} \rangle = P_{\boldsymbol{t}}(\omega) = x \cdot y$, *i.e.* a $\boldsymbol{v}_\omega$-sharing of $x \cdot y$.

Let $\boldsymbol{v}''_\omega = (1, \omega, \omega^2, \ldots, \omega^{2n-1})$, then we have

$$P_{\boldsymbol{t}}(\omega) = \langle \boldsymbol{v}''_\omega, \boldsymbol{t} \rangle = x \cdot y \quad \Leftrightarrow \quad \langle \boldsymbol{v}'_\omega, \mathsf{FFT}_{\boldsymbol{\alpha}}(\boldsymbol{t}) \rangle = x \cdot y \ .$$

By correctness of the FFT-based polynomial multiplication, we hence have that $\boldsymbol{u} = \mathsf{FFT}_{\boldsymbol{\alpha}}(\boldsymbol{t})$ is a $\boldsymbol{v}'_\omega$-sharing of $x \cdot y$. Let us now consider the actual multiplication gadget with refreshing at Step 4. By correctness of the refresh algorithm, $\boldsymbol{u}'$ is also a $\boldsymbol{v}'_\omega$-sharing of $x \cdot y$, and by the above relation we have that $\langle \boldsymbol{v}'_\omega, \boldsymbol{u}' \rangle = x \cdot y$ implies $\langle \boldsymbol{v}''_\omega, \mathsf{FFT}^{-1}_{\boldsymbol{\alpha}}(\boldsymbol{u}') \rangle = x \cdot y$, which is $\langle \boldsymbol{v}''_\omega, \boldsymbol{t} \rangle = x \cdot y$. We hence get the correctness of the multiplication gadget.

Scalar Multiplication Gadget. For the particular case of a multiplication by a constant, a dedicated scalar multiplication gadget can be used which is much more efficient than a regular multiplication gadget. Given a $\boldsymbol{v}_\omega$-sharing $\boldsymbol{x} = (x_1, \ldots, x_n)$ and a constant $\alpha \in \mathbb{K}$, the scalar multiplication gadget outputs

$$\alpha \cdot \boldsymbol{x} = (\alpha \cdot x_1, \ldots, \alpha \cdot x_n) \ .$$

This is done *via* n multiplication gates processing each share separately. Hence this scalar multiplication gadget achieves $(n-1)$-probing security.

Square Gadget. For the particular case of a field $\mathbb{K}$ of characteristic 2, a square can be computed through a dedicated gadget much more efficiently than with a regular multiplication gadget. Given a $\boldsymbol{v}_\omega$-sharing $\boldsymbol{x} = (x_1, \ldots, x_n)$ of x, the square gadget outputs

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_n) \quad \text{with} \quad y_i = x_i^2 \cdot \omega^{i-1}$$

for every $i \in [n]$. We then have $\langle \boldsymbol{v}_\omega, \boldsymbol{y} \rangle = \langle \boldsymbol{v}_\omega, \boldsymbol{y} \rangle^2$ by linearity of the squaring on a field of characteristic 2, which implies that $\boldsymbol{y}$ is indeed a $\boldsymbol{v}_\omega$-sharing of x^2. The square gadget involves $2n$ multiplication gates processing each share separately. Hence this square gadget achieves $(n-1)$-probing security. More generally, a sharewise gadget can compute any q^k-th power on a field of characteristic q (*i.e.* compute the k-th Frobenius map).

Note that, extending the standard circuit compiler to include such gadget is straightforward but it would make the formalism heavier so we skip this extension from our presentation.

5.2 Field Extension and FFT Algorithm

In order to instantiate the GJR$^+$ scheme, it is necessary to consider an implementation of secure multiplication at order n over a finite field $\mathbb{K}$ and an element ω such that there exists an FFT algorithm which allows quasilinear multiplication of polynomials of degree at most n and coefficients in $\mathbb{K}$ and which can be written as an arithmetic circuit on $\mathbb{K}$ solely composed of additions, subtractions and multiplication by constants.

A possible approach (which was used in [29]), is to consider finite fields $\mathbb{K} = \mathbb{F}_q$ that contain a $(2n)$-th root of unity ω (*i.e.* such that $2n \mid q-1$). However, most of the time, we cannot choose the underlying algebraic structure and we have to consider a specific cryptographic primitive with a given structure and to implement it securely. In order to extend the original scheme to any finite field $\mathbb{F}_p^m$ for some prime number p (with $m \geq 1$), we can use the general *additive* FFT proposed by Cantor in [44, 17]. In this case we can instantiate it at order n over $\mathbb{F}_p^\ell$ where ℓ is the minimum even value greater than m such that $p^\ell \geq 2n$.

In particular, for most symmetric cryptographic schemes, the underlying structure is a finite field of characteristic 2 and over such a binary field, the approach from [29] does not apply at all. For this case of utmost practical importance, we can use the Gao-Mateer additive FFT [27] for secure implementation of multiplications at order n over binary fields $\mathbb{F}_{2^m}$ for $m \geq 2$. The Gao-Mateer additive FFT is a variant of Cantor additive FFT that works over finite fields of characteristic 2. Using this transform, if m is even with $2^m \geq 2n$, then we can use directly our technique over $\mathbb{K} = \mathbb{F}_{2^m}$ and otherwise we can simply instantiate it over $\mathbb{K} = \mathbb{F}_{2^\ell}$ where ℓ is the smallest even integer for which $2^\ell \geq 2n$ and $m \mid \ell$.

5.3 Security Reduction

This section provides a security reduction for the GJR$^+$ scheme. We show that under the probing security of the FFT, the scheme achieves region probing security. More formally, the reduction is based on the following hypothesis on the FFT algorithm.

Hypothesis 1 (FFT Probing Security)**.** *The circuits processing*

$$\mathsf{FFT}_{\boldsymbol{\alpha}} : (\boldsymbol{x} \parallel \boldsymbol{0}) \mapsto \boldsymbol{r} \quad \textit{and} \quad \mathsf{FFT}_{\boldsymbol{\alpha}}^{-1} : \boldsymbol{u}' \mapsto \boldsymbol{t}$$

are t_n^{FFT}-probing secure w.r.t. the $\boldsymbol{v}_\omega$-encoding and the $\boldsymbol{v}'_\omega$-encoding respectively.

We can then state our reduction theorem. A discussion of the practical meaning of Hypothesis 1 is given after the theorem proof.

Theorem 3. *Under the FFT Probing Security hypothesis and the t_n^{R}-IOS property of the refresh gadget, the GJR$^+$ compiler is r_n-region probing secure with*

$$r_n = \max_{t \le t_n^{\mathsf{R}}} \min\left(\frac{t_n^{\mathsf{FFT}} - 6t}{2 \cdot |\mathsf{FFT}_n|} , \frac{t}{|G_n^{\mathsf{R}}|} \right) \tag{7}$$

where $|\mathsf{FFT}_n|$ denotes the (maximum) number of wires in the FFT circuits for $2n$ input sharings.

Note that the refresh gadget described in Section 4 satisfies $t_n^{\mathsf{R}} = n - 1$ and $|G_n^{\mathsf{R}}| = 3n \log n$. Assuming the FFT algorithm is quasilinear and that it can tolerate a linear number of probes (in the encoding order n) and denoting

$$\begin{aligned} |\mathsf{FFT}_n| &= \alpha \cdot n \log n \\ |G_n^{\mathsf{R}}| &= \beta \cdot n \log n \\ t_n^{\mathsf{FFT}} &= \gamma \cdot n \end{aligned}$$

for some constants α, β and γ (with $\gamma < 1$), one can check that the minimum in Equation 16 is reached for

$$t = \left(\frac{\beta\gamma}{2(\alpha + 3\beta)} \right) \cdot n \quad \Longrightarrow \quad r_n = \left(\frac{\gamma}{2(\alpha + 3\beta)} \right) \cdot \frac{1}{\log n} \; . \tag{8}$$

In particular, we obtain a probing rate $r_n = \Theta(1/\log n)$.

The proof of Theorem 3 is based on the two following lemmas.

Lemma 1. *Under the FFT Probing Security hypothesis the circuit processing*

$$(\boldsymbol{x}, \boldsymbol{y}) \mapsto \boldsymbol{u} = \mathsf{FFT}_{\boldsymbol{\alpha}}(\boldsymbol{x} \parallel \boldsymbol{0}) \otimes \mathsf{FFT}_{\boldsymbol{\alpha}}(\boldsymbol{y} \parallel \boldsymbol{0})$$

is t_n^{FFT}-probing secure w.r.t. the $\boldsymbol{v}_\omega$-encoding.

Proof. Let us denote by $\widehat{C}$ the considered circuit and $\mathcal{W}$ the set of probed wires from $\widehat{C}$ such that $|\mathcal{W}| \le t_n^{\mathsf{FFT}}$. We show how to construct the simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$ that outputs a perfect distribution of $\widehat{C}_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{x}$ and $\boldsymbol{y}$ are uniform $\boldsymbol{v}_\omega$-linear sharings. The simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$ first call the simulators $\mathcal{S}_{\mathsf{FFT},\mathcal{W}_1}$ and $\mathcal{S}_{\mathsf{FFT},\mathcal{W}_2}$ by constructing $\mathcal{W}$ as follows: for every $w \in \mathcal{W}$, w is added to $\mathcal{W}_1$ if it corresponds to a wire in the first FFT (*i.e.* applying to $\boldsymbol{x}$) and w is added to $\mathcal{W}_1$ if it corresponds to a wire in the second FFT (*i.e.* applying to $\boldsymbol{y}$). Whenever w corresponds to a product $u_i = r_i \cdot s_i$, then the wire corresponding to r_i is added to $\mathcal{W}_1$ and the wire corresponding to s_i is added to $\mathcal{W}_2$. By construction, we have $|\mathcal{W}_1|, |\mathcal{W}_2| \le |\mathcal{W}| \le t_n^{\mathsf{FFT}}$ which ensures that $\mathcal{S}_{\mathsf{FFT},\mathcal{W}_1}$ and $\mathcal{S}_{\mathsf{FFT},\mathcal{W}_2}$ output perfect simulations of all the wires in $\mathcal{W}$ pertaining to the two FFT circuits (by FFT Probing Security hypothesis). Moreover, by construction, they also output the pairs (r_i, s_i) for all the wires in $\mathcal{W}$ corresponding to a product $u_i = r_i \cdot s_i$ which can then be perfectly simulated as well. $\square$

Lemma 2. *Under the FFT Probing Security hypothesis the circuit processing*

$$\boldsymbol{u}' \mapsto \boldsymbol{z} = \mathsf{Compress}(\omega; \mathsf{FFT}_{\boldsymbol{\alpha}}^{-1}(\boldsymbol{u}'))$$

is $(t_n^{\mathsf{FFT}}/2)$-probing secure w.r.t. the $\boldsymbol{v}'_\omega$-encoding.

Proof. The proof follows the same lines as the proof of Lemma 1. Let $\widehat{C}$ denote the considered circuit and $\mathcal{W}$ the set of probed wires from $\widehat{C}$ such that $|\mathcal{W}| \le t_n^{\mathsf{FFT}}/2$. The simulator $\mathcal{S}_{\widehat{C},\mathcal{W}}$ essentially relies on the simulator $\mathcal{S}_{\mathsf{FFT},\mathcal{W}'}$ where $\mathcal{W}'$ is constructed as follows: for every $w \in \mathcal{W}$, w is added to $\mathcal{W}'$ if it corresponds to a wire in the FFT. Otherwise, w correspond to a wire in the computation $z_i = t_i + \omega^n \cdot t_{n+i}$ for some i, in which case we add the wires corresponding to t_i and t_{n+i} to $\mathcal{W}'$. By construction, we have $|\mathcal{W}'| \leq 2 \cdot |\mathcal{W}| \leq t_n^{\mathsf{FFT}}$ which ensures that $\mathcal{S}_{\mathsf{FFT},\mathcal{W}'}$ outputs a perfect simulation of all the wires in $\mathcal{W}'$, *i.e.* of all the wires in $\mathcal{W}$ pertaining to the FFT plus the pairs (t_i, t_{n_i}) for every i such that a wire in the computation $z_i = t_i + \omega^n \cdot t_{n+i}$ appears in $\mathcal{W}$. The latter wires can then also be perfectly simulated from the pairs (t_i, t_{n_i}), which concludes the proof. □

Proof. (Theorem 3) The proof simply holds from Lemma 1 and Lemma 2 by applying the composition theorem (Theorem 1). We further note that the term depending on the addition gadget can be removed from the expression of the probing rate since the latter satisfies $t_n^{\oplus} = n - 1$ and $|G_n^{\oplus}| = 2n$ which clearly makes it greater than the term depending on the FFT. □

Theorem 3 formally shows that if probing security can be demonstrated for the FFT algorithm, then we obtain region probing security for the GJR$^+$ scheme. Unfortunately, it is not clear whether the classical FFT algorithms are probing secure or not. To some extent, this open issue is related to the choice of ω: some choices lead to probing insecurity[2] while it is not clear whether some choices can provide probing security. Nevertheless, following the approach of [29] it is possible to obtain random probing security by picking ω randomly on a large enough field $\mathbb{K}$. This is formally stated in Subsection 5.4.

Discussion on Hypothesis 1. We now provide some insights about whether Hypothesis 1 is verified in practice. Given input values $\mathbb{K}, \alpha$ and ω, there exists an effectively computable function that checks whether Hypothesis 1 is verified. Indeed, given a $\boldsymbol{v}_\omega$-encoding $\boldsymbol{x}$ of the value x, there exists a circuit C of size $\Theta(n \log n)$ that takes $\boldsymbol{x}$ as input and computes $\mathsf{FFT}_{\boldsymbol{\alpha}}(\boldsymbol{x} \| \boldsymbol{0})$. Probing a node i of the circuit reveals $\langle \boldsymbol{u}_i, \boldsymbol{x} \rangle$, where the value of the vector $\boldsymbol{u}_i \in \mathbb{K}^n$ depends only of $\boldsymbol{\alpha}$ and i. The adversary can recover $x = \langle \boldsymbol{v}_\omega, \boldsymbol{x} \rangle$ if and only if he can probe a subset S such that $\boldsymbol{v}_\omega$ is in the span of $(\boldsymbol{u}_i)_{i \in S}$. Indeed, according to Lemma 1 of [29] (see Appendix C):

$$\left(\exists (a_i)_{i \in S} \in \mathbb{K}^{|S|} \text{ s.t. } \boldsymbol{v}_\omega = \sum_i a_i \boldsymbol{u}_i \right) \Leftrightarrow \left(x = \sum_i a_i \cdot \langle \boldsymbol{u}_i, \boldsymbol{x} \rangle \right). \tag{9}$$

Therefore Hypothesis 1 can be verified by checking, for all $\binom{\Theta(n \log n)}{|S|}$ choices of $|S|$ probes in the NTT circuit, whether the left part of Equation 9 holds, a subtask that can be done via Gaussian elimination. When $|S| = \Theta(n)$, the number of subsets to check is superexponential in n but is still tractable via exhaustive search for small values of n.

As an illustration, the circuit C in Figure 5 computes the degree-8 NTT over $\mathbb{F}_{257}$ for input $(x_0, x_1, x_2, x_3, 0, 0, 0, 0)$. Each node i is labelled by a vector $\boldsymbol{u}_i$ such that probing i reveals $\langle \boldsymbol{u}_i, \boldsymbol{x} \rangle$, with $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$. Note that since half the input coefficients are 0, the circuit description is somewhat simpler than a full NTT. By exhaustive search, we can see that C is 3-probing secure for $\omega = 138$, whereas it is only 2-probing secure for $\omega = 209$. Indeed, in the latter case $\boldsymbol{v}_\omega = (1, 209, 248, 175)$, and one can check that:

$$\boldsymbol{v}_\omega = 51 \cdot (1, 64, 241, 4) + 243 \cdot (0, 0, 1, 0) + 207 \cdot (1, 241, 256, 16). \tag{10}$$

This illustrates the importance of the choice of ω in Hypothesis 1.

Distinct finite fields may yield distinct values for t_n^{NTT}. Indeed, applying the same methodology to $\mathbb{F}_{97}$, we found values of ω for which $t_4^{\mathsf{NTT}} = 2$ and $t_8^{\mathsf{NTT}} = 5$, where each time the NTT has degree $2n$.

[2]For instance, taking ω to 0 or to some nth power of unity when the FFT algorithm is the NTT (as considered in [29]) can be shown to lead to some obvious probing security flaw.

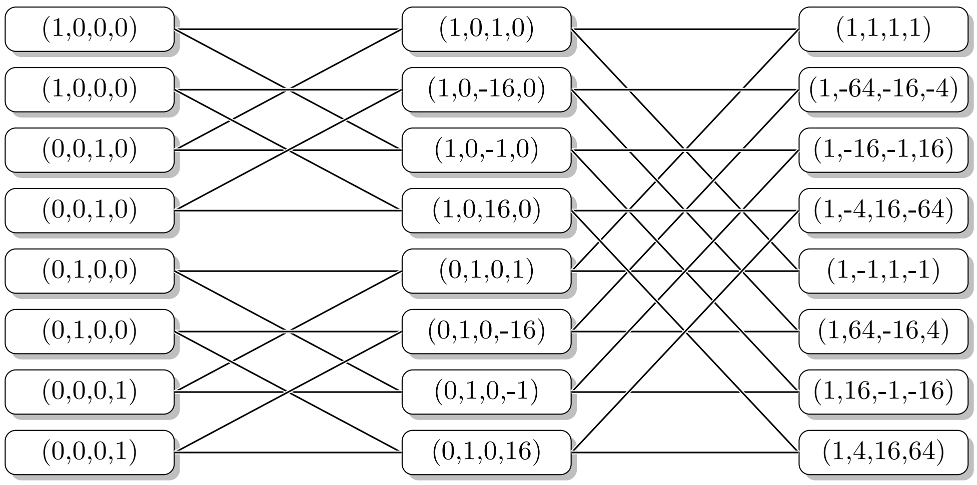

Figure 5: Circuit of the degree-8 NTT applied on an order-4 encoding $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ for the prime field $\mathbb{F}_{257}$. Probing a node i reveals $\langle \boldsymbol{u}_i, \boldsymbol{x} \rangle$ to an attacker for a given $\boldsymbol{u}_i$. Each butterfly (a subcircuit of the form $\bowtie$) computes $(\boldsymbol{c}, \boldsymbol{d}) \leftarrow (\boldsymbol{a} + \lambda \boldsymbol{b}, \boldsymbol{a} - \lambda \boldsymbol{b})$ given an input $(\boldsymbol{a}, \boldsymbol{b})$ and a fixed λ.

For prime fields and a power-of-two NTT, we were able in practice to determine t_n^{FFT} only for $n \leq 8$, due to combinatorial explosion. This raises the question of proposing algorithms more efficient than exhaustive search for compute t_n^{FFT}. Note that $t_n^{\mathsf{FFT}} + 1$ is the minimum weight w for which we can find a vector $\boldsymbol{a}$ with at most w non-zero coefficients such that $\boldsymbol{a} \cdot \boldsymbol{U} = \boldsymbol{v}_\omega$, where $\boldsymbol{U} = (\boldsymbol{u}_i)_i \in \mathbb{K}^{\Theta(n \log n) \times n}$. This can be cast as a specific instance of the information-set decoding (ISD) problem, which is common in code-based cryptography. However, unlike ISD instances that are usually studied in code-based cryptanalysis, the matrix $\boldsymbol{U}$ we consider has more lines than columns, is not random but fixed, and the underlying field may be non-binary. We see the question of computing t_n^{FFT} more efficiently as an interesting open problem.

5.4 Security Proof for Large Fields

The security proof given hereafter follows the same lines as the original proof from [29] but it is more general as it applies to any instance of the GJR$^+$ scheme (with any field and FFT algorithm) and it holds in the stronger region probing model. Moreover, our security proof corrects a flaw in the original proof which we exhibit hereafter.

Flaw in the original proof. The original GJR scheme is based on a different refresh gadget for the composition. In a nutshell, their gadget follows the classical approach of adding a random ω-encoding of 0. The latter is generated based on a fixed ω-encoding of 0 denoted $\boldsymbol{e}$ in [29], which is randomly generated at the beginning of the computation and which is considered to be fully leaked to the adversary. When a fresh ω-encoding of 0 must be generated, one draws a random vector $\boldsymbol{u}$ and multiplies it with $\boldsymbol{e}$ through the multiplication gadget, which gives a fresh and uniform ω-encoding of 0. Such a refresh procedure satisfies a slightly weaker version of the IOS property (see Subsection 3.1 for comparison). Specifically, when a sharing $\boldsymbol{x}$ is refreshed into a new sharing $\boldsymbol{x}'$, the leakage from the refresh procedure can be simulated by linear combinations of $\boldsymbol{x}$ and linear combinations of $\boldsymbol{x}'$. These leaking linear combinations can in turn be perfectly simulated with overwhelming probability over the random distribution of ω, provided that ω is defined on a large enough field $\mathbb{K}$ (see Lemmas 1 & 2 of [29] which we recall in

Appendix C). The flaw in the proof is that it implicitly assumes that the aforementioned leaking linear combinations have constant coefficients with respect to ω. However, by definition of the refresh procedure, these coefficients depend on $\boldsymbol{e}$, the initial ω-encoding of 0, which cannot be considered as constant with respect to ω. This prevents the application of Lemmas 1 & 2 of [29]. This bug invalidates the composition security proof of the original GJR scheme although it does not lead to an obvious security flaw: it is not clear whether the linear combinations coming from the refresh imply an exploitable information leakage (or equivalently a simulation failure).

New proof. The region-probing security of the GJR$^+$ scheme simply holds from the IOS property of the refresh gadget and assuming that the underlying FFT algorithm is somehow linear. This is captured by the following definition.

Definition 8 (Linear FFT Circuit)**.** An FFT circuit is said *linear* if the circuits processing

$$\mathsf{FFT}_{\boldsymbol{\alpha}} : (\boldsymbol{x} \parallel \boldsymbol{0}) \mapsto \boldsymbol{r} \quad \text{and} \quad \mathsf{FFT}_{\boldsymbol{\alpha}}^{-1} : \boldsymbol{u}' \mapsto \boldsymbol{t}$$

are composed of additions and multiplications by constants (on $\mathbb{K}$).

The above definition implies that the value carried by each wire in the FFT circuit can be expressed as a linear combination of the coordinates of the input sharing. This property is necessary to apply the security argument of the original GJR scheme. Note that this requirement is relatively weak since it is satisfied by classical FFT algorithms such as the NTT (used in [29]) and the Gao-Mateer additive FFT [27].

Corollary 1. *If the FFT circuit is linear and made of $|\mathsf{FFT}_n| = \alpha\, n \log n$ wires, the GJR$^+$ compiler is (r_n, ε_n)-region probing secure with*

$$r_n = \left(\frac{1}{2\alpha + 18}\right) \frac{1}{\log n} \ , \tag{11}$$

and

$$\varepsilon_n = \frac{n}{|\mathbb{K}|} \ . \tag{12}$$

Proof. Using Lemmas 1 & 2 of [29] (which are recalled in Appendix C for the sake of completeness) and thanks to the linearity of the FFT circuit, we have that for any choice of $n-1$ leaking wires from the FFT circuit, the probability that the leaking wires cannot be perfectly simulated is lower than $n/|\mathbb{K}|$. Besides the linearity of the FFT circuit, the only requirement for this upper bound to apply is that the choice of the leaking wires is made independently of ω, which occurs in the region probing model since the placement of the probes by the adversary is done independently of the random generation of ω. We can then directly apply Theorem 3 and obtain the probing rate r_n from Equation 8 with $\gamma = \frac{n-1}{n} \approx 1$ and $\beta = 3$. □

In Appendix B, we further detail the security proof of GJR$^+$ in the random probing model which holds from its security in the region probing model by applying the Chernoff's bound. This further implies the security of GJR$^+$ in the noisy leakage model by the reduction from [25].

6 Application

In this section, we present an application of our extended GJR scheme and compare it with a more standard scheme based on SNI gadgets. We investigate the masked computation of two different ciphers:

- The Advanced Encryption Standard (AES) [1]: a very common application scenario which favors efficient masking schemes on the field $\mathbb{F}_{256}$;
- MiMC [4]: a cipher with efficient arithmetic representation on a large field. MiMC has been designed with the aim to minimize the number of multiplications (which makes it particularly amenable to masked computation). We focus on the prime-field variant of MiMC (the base field is a prime field $\mathbb{F}_p$).

For these two application contexts, we described masked computations based on the two following masking schemes:

- The GJR$^+$ scheme described in Section 5 of this paper, in two modes of application:
  - on a binary field with the Gao-Mateer FFT algorithm (to mask AES),
  - on a prime field with NTT algorithm (to mask MiMC).
- An extended ISW scheme, that we shall refer to as ISW$^+$, and which is based on
  - the ISW multiplication gadget [31] over the base field $\mathbb{K}$ (either $\mathbb{F}_{256}$ for AES or $\mathbb{F}_p$ for MiMC),
  - share-wise linear gadgets (for additions, subtractions, $\mathbb{F}_{256}$-squares, multiplications by constants),
  - the BCPZ quasilinear refresh gadget [10].

We first address implementation aspects of the two above masking schemes, then describes masking of AES and MiMC with these schemes and finally provide comparison of performances in terms of operation counts and randomness consumption.

Cautionary note: To ease the presentation, we consider that each gadget includes a refreshing of its output sharing, except the ISW multiplication gadget which achieves the SNI notion without further refreshing. In a masked computation, a sharing might be input of several gadgets which would be an issue with respect to region probing security (*e.g.* one could accumulate t probes on this sharing per gadget). We therefore impose that such a sharing is refreshed before each new usage.

6.1 Implementation of GJR$^+$

6.1.1 Multiplication gadget on $\mathbb{F}_p$ based on the NTT

It is well-known that polynomials can be multiplied in quasi-linear time in finite fields using the *Number Theoretic Transform* (NTT), a Fast Fourier Transform (FFT) which requires that the coefficient ring contain certain roots of unity. More precisely, it is possible to multiply two polynomials of degree $\leq N$ in a finite field $\mathbb{F}_q$ in $O(N \log N)$ arithmetic operations in $\mathbb{F}_q$ if $\mathbb{F}_q$ contains a primitive $2N$-th root of unity (which occurs if and only if $2N$ divides $q-1$). The number theoretic transform was introduced by Pollard [36] and we refer the reader to [42, Section 8.2] for an exhaustive description. As stated in [42, Theorem 8.18], if N is a power of S ($N = 2^m$), the multiplication of two polynomials of degree $< N$ in a finite field $\mathbb{F}_q$ which contains a $2N$-th root of unity requires $6N\log(N) + 6N$ additions in $\mathbb{F}_q$, $3N\log(N) + 4N - 2$ multiplications by constants in $\mathbb{F}_q$, $2N$ (bilinear) multiplications in $\mathbb{F}_q$ and $2N$ divisions by $2N$ in $\mathbb{F}_q$.

Our multiplication gadget (for an encoding order n) described in Subsection 5.1 over such a finite field has thus a total complexity of $8n\log(n)+11n$ additions in $\mathbb{F}_q$, $5n\log(n)+7n-2$ multiplications by constants in $\mathbb{F}_q$ and $2n$ (bilinear) multiplications in $\mathbb{F}_q$. It requires $2n\log(n) + 2n$ random elements from $\mathbb{F}_q$.

6.1.2 Multiplication gadget on $\mathbb{F}_{2^k}$ based on the Gao-Mateer FFT

The classical NTT cannot be applied when the underlying field does not have the desired roots of unity. We describe the *additive FFT* algorithm proposed by Gao and Mateer in 2010 [27], which works over fields of characteristic two. The idea of this class of FFTs is to evaluate polynomials of degree m over a linear (additive) subspace of $\mathbb{K}[x]$ rather than a group and it comes in two flavors: generic algorithms for an arbitrary m, or specialized ones for m a power of two. The specialized algorithms are faster than the generic ones, but the condition on m heavily constraints their use. We will use it with a slight generalization of Cantor bases which we call *self-folding bases*, and which allow even more aggressive optimization than what is done in [16, 15, 21] and may be of independent interest.

Let $\mathbb{F} = \mathbb{F}_{2^k}$ be a finite field of characteristic two.We consider the additive FFT of a polynomial $f \in \mathbb{F}[x]$, that is we evaluate f over the $\mathbb{F}_2$-linear span generated by m elements $\beta_0, \dots, \beta_m \in \mathbb{F}$ linearly independent over $\mathbb{F}_2$. This span contains $N = 2^m$ elements, and is defined if and only if $N \leq 2^k$, or equivalently $m \leq k$.

Taylor Expansion. An important subroutine of the Gao-Mateer additive FFT is the Taylor expansion, a slight variant of the usual notion of Taylor series. It consists of writing any polynomial $f \in \mathbb{F}[x]$ of degree $< N$ as follows:

$$f(x) = \sum_{i=0}^{N/2-1} h_i(x) \cdot (x^2 - 1)^i, \tag{13}$$

where each h_i is a polynomial of $\mathbb{F}[x]$ of degree at most 1. Algorithm 5 (provided in Appendix D) is an algorithm presented in [27] for computing the Taylor expansion of a polynomial in the case where m is generic (and not a power of 2), in which it is shown to require $\frac{1}{2}N \log N - \frac{1}{2}N$ field additions and no multiplication.

Basis folding. We abusively call basis any subset $B = \{\beta_0, \dots, \beta_{m-1}\} \subset \mathbb{F}^m$ of m elements of $\mathbb{F}$ linearly independent over $\mathbb{F}_2$.[3] For a basis B of length m and an integer $0 \leq i < 2^m$ which can be written as $i = \sum_{j=0}^{m-1} a_j 2^j$ with $a_j \in \{0, 1\}$, we will note:

$$B[i] = \sum_{j=0}^{m-1} a_j \beta_j. \tag{14}$$

We will say that $B[i]$ is the i-th element of $\langle B \rangle$. An important subroutine in [27] consists of what we call *folding* a basis, a process we recall in Algorithm 6.

Additive FFT. Gao and Mateer [27] proposed an algorithm for computing the additive FFT of a polynomial f over the subspace $\langle B \rangle$ generated by a basis B. This algorithm is described in Algorithm 7 (Appendix D), which costs $2N \log N - 2N + 1$ field additions, as well as $\frac{1}{4}N(\log N)^2 + \frac{3}{4}N \log N - \frac{N}{2}$ scalar field multiplications. No formal description of the inverse algorithm is given, but it is observed that an inverse additive FFT can be obtained by performing the inverse of each operation in the reverse order. Since the scalar field multiplications and their inversions can be precomputed, the cost of the inverse additive FFT is the same as for the forward algorithm.

Self-Folding Bases. One could assume that the choice of B is not too important, because the operations involving B can be precomputed anyway. However, we show in this

[3]Here our notation differ slightly from [27], where B denotes the linear space spanned by the basis. We find it more natural for our purposes to denote by B the basis.

subsection that carefully choosing B can go a long way in making the additive FFT faster, simpler to implement and less costly in memory.

In fact, in the case where m is a power of two, by taking $\beta_{m-1} = 1$ for their Cantor basis, Bernstein *et al.* [16] showed that one could saved up some of the multiplication operations (namely the computation that involved the inverse of β_{m-1}) for the top-level recursion cases. We take this idea further and propose a specific kind of bases such that $\beta_{m-1} = 1$ at every layer of the recursion. In the following, those kind of bases are called self-folding bases.

Definition 9 (Self-folding bases)**.** Let $\mathbb{F}$ be a finite field of characteristic two and let $m \geq 1$. A self-folding basis $B = \{\beta_0, \ldots, \beta_{m-1}\} \subset \mathbb{F}^m$ is a basis which verifies the two following conditions:

1. $\beta_{m-1} = 1$;
2. for $i \in \{0, \ldots, m-2\}$, it holds that $\beta_i^2 - \beta_i = \beta_{i+1}$.

We have the following properties about self-folding bases:

Proposition 1. *Let $\mathbb{F}$ be a finite field of characteristic two and let $m \geq 1$. The following properties hold:*

1. *Let $B = \{\beta_0, \ldots, \beta_{m-1}\}$ be a self-folding basis and $G, D \leftarrow \mathsf{Fold}(B)$. We have $G = \{\beta_0, \beta_1, \ldots, \beta_{m-2}\}$ and $D = \{\beta_1, \beta_2, \ldots, \beta_{m-1}\}$.*
2. *Let $\mathbb{F}'$ be a subfield of $\mathbb{F}$ of cardinality $2^{k'}$. For any $m \leq k'$, there exists a self-folding basis B of m elements such that $\langle B \rangle$ is included in $\mathbb{F}'$.*

Proof. The first item is immediate from the definition. The second item is proven in the special case $\mathbb{F}' = \mathbb{F}$ and $m = k'$ in the appendix of [27], and the proof is constructive. From there, the general case is immediate to obtain by embedding the solutions for $\mathbb{F}'$ in the larger field $\mathbb{F}$ and keeping only the m last elements of B. $\square$

From Proposition 1, we can see that B "folds onto itself": folding B into G, D yields subsets of B, and this self-folding property transfers to D.

The notion of self-folding basis is close to that of Cantor basis [17, 27]; the only difference is that the elements are taken in reverse order, and that no condition is imposed on the basis elements belonging to a subfield $\mathbb{F}_{2^m}$ (such a subfield does not always exist, thus self-folding bases are slightly more general objects than Cantor bases). The most important difference, however, is how they are used. In [27, 16, 15], Cantor bases are used to speed up the *specialized* additive FFT algorithm of [27], which only works on a restricted set of parameters (namely, when m is a power of two). In comparison, our improvements apply to speed up the *generic* algorithm of [27], which works for any value of m.

Half-FFT. Our improved (half-)FFT works with a self-folding basis and its iterated foldings. It is described in Appendix D. Its main advantage is that the step 3 of Algorithm 7 becomes unnecessary; in total, this saves us $N \log N$ scalar multiplications. Moreover, it divides by two the number of precomputed tables. As another algorithmic optimization, we make full use of the fact that for polynomial multiplications, half the inputs of the FFT's call are zero coefficients which leads to speed up to the computations by a factor two compared to a regular additive FFT. These optimizations applies in a similar way to the inverse additive FFT (with the difference here is that we cannot exploit anymore the fact that half of the polynomial coefficients are zero).

Complexity. In this section, we provide the complexity of our new algorithms (the detailed analysis is provided in Appendix D). Table 1 gives the operation counts for our variant of the Gao-Mateer additive FFT. For N being a power of 2 ($N = 2^m$), the multiplication of two polynomials of degree lower than N in a finite field $\mathbb{F}_{2^k}$ requires $(1/2)N\log^2(N) + 2N\log(N) - 2N$ additions in $\mathbb{F}_{2^k}$, $(3/2)N\log(N) - 2N$ multiplications by constants in $\mathbb{F}_{2^k}$ and $2N$ (bilinear) multiplications in $\mathbb{F}_{2^k}$. Our multiplication gadget (for an encoding order n) described in Subsection 5.1 has thus a total complexity of $(1/2)n\log^2(n) + 4n\log(n) + n$ additions in $\mathbb{F}_{2^k}$, $(7/2)n\log(n)$ multiplications by constants in $\mathbb{F}_{2^k}$, and $2n$ (bilinear) multiplications in $\mathbb{F}_{2^k}$. It further requires $2n\log(n) + 2n$ random elements from $\mathbb{F}_{2^k}$. Table 2 summarizes the operation counts for the different gadgets for GJR$^+$.

Table 1: Complexity of our variant of Gao-Mateer additive FFT.

	Gao-Mateer FFT [27]	HalfFFT	InverseFFT
Add	$\frac{1}{4} \cdot N \cdot (\log_2(N))^2 + \frac{3}{4} \cdot N \cdot \log_2(N) - \frac{1}{2} \cdot N$	$\frac{1}{8} \cdot N \cdot (\log_2(N))^2 + \frac{5}{8} \cdot N \cdot \log_2(N) - N$	$\frac{1}{4} \cdot N \cdot (\log_2(N))^2 + \frac{3}{4} \cdot N \cdot \log_2(N)$
Mult	$2 \cdot N \cdot \log_2(N) - 2N - 1$	$\frac{1}{2} \cdot N \cdot \log_2(N) - \frac{3}{4} \cdot N$	$\frac{1}{2}N \cdot \log_2(N) - \frac{1}{2} \cdot N$

Table 2: Operation counts for GJR$^+$.

	Mult	Add	Random
Mult. gadget ($\mathbb{F}_p$)	$5n\log(n) + 9n - 2$	$8n\log(n) + 11n$	$2n\log(n) + 2n$
Mult. gadget ($\mathbb{F}_{2^m}$)	$7n\log(n)/2 + 2n$	$n\log^2(n)/2 + 4n\log(2n)$	$2n\log(n) + 2n$
Addition gadget	0	n	0
Refresh gadget	$n\log(n)/2$	$n\log(n)$	$n\log(n)/2$

6.2 Implementation of ISW$^+$

Table 3 summarizes the operation counts of the different gadgets for ISW$^+$.

Table 3: Operation counts for ISW$^+$.

	Mult	Add	Random
Mult. gadget (ISW) [31]	n^2	$2n(n-1)$	$n(n-1)/2$
Addition gadget	0	n	0
Refresh gadget (BCPZ) [10]	0	$2n\log(n) - n$	$n\log(n) - n/2$

6.3 Masking of AES

We consider the AES cipher as an arithmetic circuit over $\mathbb{F}_{256}$, composed of additions, multiplications, squares, multiplication by constants, and a special gate computing the affine part of the AES s-box. Besides the multiplication, all these operations give rise to a *linear gadget* in the masked setting, *i.e.* a gadget which applies the operation share-wisely and refreshes the output sharing.

Algorithm 2 Masked AES

```
Require: $n$-sharings $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}$ of plaintext bytes $x_1, \ldots, x_{16} \in \mathbb{F}_{256}$, $n$-sharings $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{16}$ of the round key bytes $k_1^j, \ldots, k_{16}^j \in \mathbb{F}_{256}$, for $0 \leq j \leq r$
Ensure: $n$-sharing of $\mathsf{AES}(x, k)$
 1: $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow (G_n^{\oplus}(\boldsymbol{x}_1, \boldsymbol{k}_1^0), \ldots, G_n^{\oplus}(\boldsymbol{x}_{16}, \boldsymbol{k}_{16}^0)))$
 2: for $j = 1, \ldots, r-1$ do
 3:     $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow (G_n^{\mathsf{Aff}}(\mathsf{MaskedExp}(\boldsymbol{x}_1)), \ldots, G_n^{\mathsf{Aff}}(\mathsf{MaskedExp}(\boldsymbol{x}_{16})))$
 4:     $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow \mathsf{ShiftRows}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16})$
 5:     $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow \mathsf{MaskedMixColumns}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16})$
 6:     $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow (G_n^{\oplus}(\boldsymbol{x}_1, \boldsymbol{k}_1^j), \ldots, G_n^{\oplus}(\boldsymbol{x}_{16}, \boldsymbol{k}_{16}^j)))$
 7: $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow (G_n^{\mathsf{Aff}}(\mathsf{MaskedExp}(\boldsymbol{x}_1)), \ldots, G_n^{\mathsf{Aff}}(\mathsf{MaskedExp}(\boldsymbol{x}_{16})))$
 8: $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow \mathsf{ShiftRows}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16})$
 9: $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16}) \leftarrow (G_n^{\oplus}(\boldsymbol{x}_1, \boldsymbol{k}_1^r), \ldots, G_n^{\oplus}(\boldsymbol{x}_{16}, \boldsymbol{k}_{16}^r)))$
10: return $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{16})$
```

The masked description of AES is described in Algorithm 2. The linear gadget for the s-box affine transformation is denoted G_n^{Aff}. The $\mathsf{ShiftRows}$ transformation simply permutes the byte indexes and is virtually free (in particular it does not involve any computation on the shares). The $\mathsf{MaskedMixColumns}$ transformation simply applies the MixColumns transformation by using gadgets $G_n^{\oplus}$ and G_n^{xtimes} in place of xors and xtimes operations (*i.e.* multiplications by the constant $\mathsf{02}$ on the field $\mathbb{F}_{256}$). We consider the implementation described in [26] which involves 15 additions and 3 xtimes per column:

$$\begin{aligned}
&acc \leftarrow x_1 \oplus x_2 \oplus x_3 \oplus x_4 \\
&y_1 \leftarrow \mathsf{xtimes}(x_1 \oplus x_2) \oplus acc \oplus x_1 \\
&y_2 \leftarrow \mathsf{xtimes}(x_2 \oplus x_3) \oplus acc \oplus x_2 \\
&y_3 \leftarrow \mathsf{xtimes}(x_3 \oplus x_4) \oplus acc \oplus x_3 \\
&y_4 \leftarrow y_1 \oplus y_2 \oplus y_3 \oplus acc
\end{aligned}$$

A masked implementation of this process thus involves 18 linear gadgets as well as 15 refresh gadgets (for the variables used multiple times). The $\mathsf{MaskedExp}$ procedure is further depicted in Algorithm 3, which is based on the Rivain-Prouff scheme [40]. As explained above, a sharing in input of several gadgets is refreshed before each new usage.

Algorithm 3 MaskedExp

```
Require: $n$-sharing $\boldsymbol{x}$ of $x \in \mathbb{F}_{256}$
Ensure: $n$-sharing of $x^{254}$
1: $\boldsymbol{z} \leftarrow G_n^{(\cdot)^2}(\boldsymbol{x})$ ;    $\boldsymbol{x} \leftarrow G_n^{\mathsf{R}}(\boldsymbol{x})$
2: $\boldsymbol{y} \leftarrow G_n^{\otimes}(\boldsymbol{x}, \boldsymbol{z})$ ;   $\boldsymbol{z} \leftarrow G_n^{\mathsf{R}}(\boldsymbol{z})$
3: $\boldsymbol{w} \leftarrow G_n^{(\cdot)^4}(\boldsymbol{y})$ ;    $\boldsymbol{y} \leftarrow G_n^{\mathsf{R}}(\boldsymbol{y})$
4: $\boldsymbol{y} \leftarrow G_n^{\otimes}(\boldsymbol{y}, \boldsymbol{w})$ ; $\boldsymbol{w} \leftarrow G_n^{\mathsf{R}}(\boldsymbol{w})$
5: $\boldsymbol{y} \leftarrow G_n^{(\cdot)^{16}}(\boldsymbol{y})$
6: $\boldsymbol{y} \leftarrow G_n^{\otimes}(\boldsymbol{y}, \boldsymbol{w})$
7: $\boldsymbol{y} \leftarrow G_n^{\otimes}(\boldsymbol{y}, \boldsymbol{z})$
8: return $\boldsymbol{y}$
```

The gadget count of the masked AES is given in Table 4. According to Algorithm 3, the $\mathsf{MaskedExp}$ involves 4 multiplication gadgets, 3 linear gadgets and 4 refresh gadgets. According to the above description, we get a total of 72 linear gadgets and 60 refresh gadgets for the full MaskedMixColumns. One full round is composed of 16 calls to MaskedExp, one call to MaskedMixColumns, plus 32 linear gadgets (16 gadgets $G_n^{\oplus}$ and 16 gadgets G_n^{Aff}). A full AES computation is composed of 9 full rounds, 1 partial round (without the MixColumns) plus one key addition (16 gadgets $G_n^{\oplus}$).

Table 4: Gadget counts for masked AES.

	Mult.	Linear	Refresh
MaskedExp	4	3	4
MaskedMixColumns	0	72	60
One round	64	136	124
Full masked AES	640	1304	1180

6.4 Masking of MiMC

Let x be some plaintext and k be some secret key, both belonging to some large field $\mathbb{K}$. The MiMC cipher is defined as:

$$\mathsf{MiMC}(x, k) = F_{k,c_r} \circ \cdots \circ F_{k,c_1}(x) + k \ ,$$

with $F_{k,c_i}(x) = (x + k + c_i)^3$, with $r = \lceil \log(|\mathbb{K}|)/\log(3) \rceil$.

For our application, we consider the prime field variant of MiMC for which $\mathbb{K}$ can be chosen as any prime field $\mathbb{F}_p$ such that $\gcd(3, p-1) = 1$ (so that x^3 is invertible on $\mathbb{F}_p$). Since we wish to apply the GJR$^+$ scheme based on the NTT (as in the original GJR scheme), the chosen field must further satisfy $p - 1 = \alpha \cdot (2n_{max})$ for some odd integer α and some integer n_{max} which is a power of two. In practice n_{max} is the maximum masking order which can be achieved by the GJR$^+$ scheme. We hence choose a prime $p = \alpha \cdot 2^{\ell+1} + 1$ with $\gcd(\alpha, 3) = 1$ and $\ell = \log_2 n_{max}$. Specifically, for a given target field size $\lambda = \lceil \log_2 p \rceil$, we search for greatest integer ℓ and smallest integer α such that: *(i)* $3 \nmid \alpha$, *(ii)* $\log_2 \alpha + \ell + 1 < \lambda$, and *(iii)* $p = \alpha \cdot 2^{\ell+1} + 1$ is prime. For our application, we thus instantiate MiMC with such 128-bit and 256-bit prime fields:

- for $\lambda = 128$, we get $p = 407 \cdot 2^{119} + 1$, giving $n_{max} = 118$,
- for $\lambda = 256$, we get $p = 467 \cdot 2^{247} + 1$, giving $n_{max} = 246$.

Algorithm 4 gives a masked description of MiMC based on any standard circuit compiler. For the sake of clarity, we omit to apply the refresh gadget G_n^{R} to the output of an arithmetic gadget $G_n^{\oplus}$ or $G_n^{\otimes}$ and consider that the refresh is part of these gadget when necessary. For $\lambda = 128$ (resp. $\lambda = 256$), the number of rounds is $r = 81$ (resp. $r = 162$).

Algorithm 4 Masked MiMC

```
Require: $n$-sharing $\boldsymbol{x}$ of plaintext $x \in \mathbb{F}_p$, $n$-sharing $\boldsymbol{k}$ of secret key $k \in \mathbb{F}_p$
Ensure: $n$-sharing of $\mathsf{MiMC}(x, k)$
1: for $i = 1, \ldots, r$ do
2:     $\boldsymbol{x} \leftarrow G_n^{\oplus}(\boldsymbol{x}, \boldsymbol{k}, c_i)$
3:     $\boldsymbol{k} \leftarrow G_n^{\mathsf{R}}(\boldsymbol{k})$
4:     $\boldsymbol{y} \leftarrow G_n^{\otimes}(\boldsymbol{x}, \boldsymbol{x})$
5:     $\boldsymbol{x} \leftarrow G_n^{\otimes}(\boldsymbol{x}, \boldsymbol{y})$
6: $\boldsymbol{x} \leftarrow G_n^{\oplus}(\boldsymbol{x}, \boldsymbol{k})$
7: return $\boldsymbol{x}$
```

Table 5: Gadget counts for masked MiMC.

	Mult.	Linear	Refresh
One round	2	2	1
Full MiMC ($\lambda = 128$)	162	163	81
Full MiMC ($\lambda = 256$)	324	325	162

6.5 Performances and Comparison

Table 6 and Table 7 summarize the operation counts for full MiMC (with $\lambda = 128$) and full AES with the two masking schemes ISW$^+$ and with GJR$^+$ implemented over the same finite field. They show that our approach results in a 62% decrease in the randomness complexity and a 51% decrease of the number of multiplication for MiMC masked at order 128 and in a 46% (*resp.* 59%) decrease in the randomness complexity and a 20% (*resp.* 52%) decrease of the number of multiplication for AES masked at order 64 (*resp.* 128).

Table 6: Performances comparison for masked MiMC ($\lambda = 128$).

n		Mul	Add.	Random
8	Full MiMC with ISW$^+$	10416.0	45408.0	17544.0
	Full MiMC with GJR$^+$	40512.0	66128.0	20100.0
	Efficiency ratio (GJR$^+$/ISW$^+$)	3.89	1.46	1.15
16	Full MiMC with ISW$^+$	41600.0	153056.0	55856.0
	Full MiMC with GJR$^+$	100796.0	165968.0	51872.0
	Efficiency ratio (GJR$^+$/ISW$^+$)	2.43	1.09	0.93
32	Full MiMC with ISW$^+$	166208.0	513536.0	173984.0
	Full MiMC with GJR$^+$	240812.0	399360.0	127088.0
	Efficiency ratio (GJR$^+$/ISW$^+$)	1.45	0.78	0.74
64	Full MiMC with ISW$^+$	664320.0	1773696.0	555456.0
	Full MiMC with GJR$^+$	559740.0	933568.0	300864.0
	Efficiency ratio (GJR$^+$/ISW$^+$)	0.85	0.53	0.55
128	Full MiMC with ISW$^+$	2656000.0	6367744.0	1857664.0
	Full MiMC with GJR$^+$	1275388.0	2136832.0	695104.0
	Efficiency ratio (GJR$^+$/ISW$^+$)	0.49	0.34	0.38

For AES, the masking scheme ISW$^+$ can always be implemented over the $\mathbb{F}_{256}$ finite field. However, to achieve provable region-probing security without relying on Hypothesis 1, Corollary 1 imposes that the masking scheme GJR$^+$ is implemented over a larger finite field such as $\mathbb{F}_{2^{128}}$ (which has less efficient arithmetic). To compare the different complexities of the schemes GJR$^+$ and ISW$^+$, we can implement $\mathbb{F}_{2^{128}}$ as a degree 16 extension of $\mathbb{F}_{2^8} = \mathbb{F}_{256}$, so that: (1) an addition over $\mathbb{F}_{2^{128}}$ takes 16 additions over $\mathbb{F}_{256}$, (2) a multiplication over $\mathbb{F}_{2^{128}}$ takes 81 multiplications (and a large number of additions) over $\mathbb{F}_{256}$ using Karatsuba's algorithm (since a multiplication of two polynomials of degree at most $16 = 2^4$ over $\mathbb{F}_{256}$ requires $81 = 3^4$ multiplications, see [42, Section 8.1]) and (3) a random element of $\mathbb{F}_{2^{128}}$ requires 16 random elements of $\mathbb{F}_{256}$. The computational efficiency of the masking scheme GJR$^+$ for AES compared to ISW$^+$ is then only better for masking order $n \geq 8192$ and its randomness complexity is better for masking order $n \geq 2048$.

Table 7: Performances comparison for masked AES.

n		Mul	Add.	Random
8	Full AES with ISW$^+$	64896	297088	123520
	Full AES with GJR$^+$	157056	257408	110080
	Efficiency ratio (GJR$^+$/ISW$^+$)	2.43	0.87	0.9
16	Full AES with ISW$^+$	211712	926976	372480
	Full AES with GJR$^+$	396032	683776	286720
	Efficiency ratio (GJR$^+$/ISW$^+$)	1.88	0.74	0.77
32	Full AES with ISW$^+$	751104	2847232	1077760
	Full AES with GJR$^+$	955904	1725952	706560
	Efficiency ratio (GJR$^+$/ISW$^+$)	1.28	0.61	0.66
64	Full AES with ISW$^+$	2812928	8991744	3148800
	Full AES with GJR$^+$	2239488	4209664	1679360
	Efficiency ratio (GJR$^+$/ISW$^+$)	0.8	0.47	0.54
128	Full AES with ISW$^+$	10868736	29820928	9594880
	Full AES with GJR$^+$	5134336	10016768	3891200
	Efficiency ratio (GJR$^+$/ISW$^+$)	0.48	0.34	0.41

Acknowledgments

This work was partly supported by the French FUI-AAP25 VeriSiCC project and by the Innovate UK Research Grant 104423 (PQ Cybersecurity). The authors would like to thank Jean-Sébastien Coron as well as the anonymous reviewers for meaningful comments and suggestions.

References

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

[2] M. Ajtai. Secure computation with information leaking to an adversary. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM STOC*, pages 715–724. ACM Press, June 2011.

[3] M.-L. Akkar and C. Giraud. An implementation of DES and AES, secure against some attacks. In Çetin Kaya. Koç, D. Naccache, and C. Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 309–318. Springer, Heidelberg, May 2001.

[4] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, Dec. 2016.

[5] P. Ananth, Y. Ishai, and A. Sahai. Private circuits: A modular approach. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 427–455. Springer, Heidelberg, Aug. 2018.

[6] M. Andrychowicz, S. Dziembowski, and S. Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 586–615. Springer, Heidelberg, May 2016.

[7] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, F.-X. Standaert, and P.-Y. Strub. Improved parallel mask refreshing algorithms: Generic solutions with

parametrized non-interference & automated optimizations. Cryptology ePrint Archive, Report 2018/505, 2018. https://eprint.iacr.org/2018/505.

[8] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini. Strong non-interference and type-directed higher-order masking. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, Oct. 2016.

[9] A. Battistello, J.-S. Coron, E. Prouff, and R. Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In B. Gierlichs and A. Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Heidelberg, Aug. 2016.

[10] A. Battistello, J.-S. Coron, E. Prouff, and R. Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. Cryptology ePrint Archive, Report 2016/540, 2016. http://eprint.iacr.org/2016/540.

[11] S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud. Randomness complexity of private circuits for multiplication. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 616–648. Springer, Heidelberg, May 2016.

[12] S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud. Private multiplication over finite fields. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 397–426. Springer, Heidelberg, Aug. 2017.

[13] S. Belaïd, J.-S. Coron, E. Prouff, M. Rivain, and A. R. Taleb. Random probing security: Verification, composition, expansion and new constructions. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 339–368. Springer, Heidelberg, Aug. 2020.

[14] S. Belaïd, D. Goudarzi, and M. Rivain. Tight private circuits: Achieving probing security with the least refreshing. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 343–372. Springer, Heidelberg, Dec. 2018.

[15] D. J. Bernstein and T. Chou. Faster binary-field multiplication and faster binary-field MACs. In A. Joux and A. M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 92–111. Springer, Heidelberg, Aug. 2014.

[16] D. J. Bernstein, T. Chou, and P. Schwabe. McBits: Fast constant-time code-based cryptography. In G. Bertoni and J.-S. Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 250–272. Springer, Heidelberg, Aug. 2013.

[17] D. G. Cantor. On arithmetical algorithms over finite fields. *J. Comb. Theory, Ser. A*, 50(2):285–300, 1989.

[18] C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-order masking schemes for S-boxes. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, Heidelberg, Mar. 2012.

[19] G. Cassiers and F. Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[20] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Heidelberg, Aug. 1999.

[21] T. Chou. McBits revisited. In W. Fischer and N. Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 213–231. Springer, Heidelberg, Sept. 2017.

[22] J.-S. Coron. Higher order masking of look-up tables. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 441–458. Springer, Heidelberg, May 2014.

[23] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. Higher-order side channel security and mask refreshing. In S. Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 410–424. Springer, Heidelberg, Mar. 2014.

[24] J.-S. Coron, A. Roy, and S. Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In L. Batina and M. Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 170–187. Springer, Heidelberg, Sept. 2014.

[25] A. Duc, S. Dziembowski, and S. Faust. Unifying leakage models: From probing attacks to noisy leakage. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Heidelberg, May 2014.

[26] G. Fumaroli, A. Martinelli, E. Prouff, and M. Rivain. Affine masking against higher-order side channel analysis. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 262–280. Springer, Heidelberg, Aug. 2011.

[27] S. Gao and T. Mateer. Additive fast Fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, Dec 2010.

[28] L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya. Koç and C. Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, Heidelberg, Aug. 1999.

[29] D. Goudarzi, A. Joux, and M. Rivain. How to securely compute with noisy leakage in quasilinear complexity. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 547–574. Springer, Heidelberg, Dec. 2018.

[30] D. Goudarzi and M. Rivain. How fast can higher-order masking be in software? In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 567–597. Springer, Heidelberg, Apr. / May 2017.

[31] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, Aug. 2003.

[32] A. Journault and F.-X. Standaert. Very high order masking: Efficient implementation and security evaluation. In W. Fischer and N. Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 623–643. Springer, Heidelberg, Sept. 2017.

[33] H. Kim, S. Hong, and J. Lim. A fast and provably secure higher-order masking of AES S-box. In B. Preneel and T. Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 95–107. Springer, Heidelberg, Sept. / Oct. 2011.

[34] T. S. Messerges. Securing the AES finalists against power analysis attacks. In B. Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 150–164. Springer, Heidelberg, Apr. 2001.

[35] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A side-channel analysis resistant description of the AES S-box. In H. Gilbert and H. Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 413–423. Springer, Heidelberg, Feb. 2005.

[36] J. M. Pollard. The fast Fourier transform in a finite field. *Mathematics of Computation*, 25:365–374, 1971.

[37] T. Prest, D. Goudarzi, A. Martinelli, and A. Passelègue. Unifying leakage models on a Rényi day. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 683–712. Springer, Heidelberg, Aug. 2019.

[38] E. Prouff and M. Rivain. Masking against side-channel attacks: A formal security proof. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

[39] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. Consolidating masking schemes. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 764–783. Springer, Heidelberg, Aug. 2015.

[40] M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, Heidelberg, Aug. 2010.

[41] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.

[42] J. von zur Gathen and J. Gerhard. *Modern computer algebra (2. ed.)*. Cambridge University Press, 2003.

[43] J. Wang, P. K. Vadnala, J. Großschädl, and Q. Xu. Higher-order masking in practice: A vector implementation of masked AES for ARM NEON. In K. Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 181–198. Springer, Heidelberg, Apr. 2015.

[44] Y. Wang and X. Zhu. A fast algorithm for the Fourier transform over finite fields and its VLSI implementation. *IEEE Journal on Selected Areas in Communications*, 6(3):572–577, April 1988.

A General Definitions

We give hereafter the formal security properties of *uniformity* and *input-output separation* for general gadgets for binary operations which generalize the definitions given for the refresh gadgets in Subsection 3.1.

Definition 10 (Uniformity)**.** Let $\boldsymbol{v} \in \mathbb{K}^n$ and g be any binary operation $g : (x, y) \in \mathbb{K}^2 \mapsto z \in \mathbb{K}$. A $\boldsymbol{v}$-gadget G of g is *uniform* if for every $\boldsymbol{x} \in \mathbb{K}^n$ and $\boldsymbol{y} \in \mathbb{K}^n$, the output $G(\boldsymbol{x}, \boldsymbol{y})$ is a uniform $\boldsymbol{v}$-linear sharing of $g(\langle \boldsymbol{v}, \boldsymbol{x} \rangle, \langle \boldsymbol{v}, \boldsymbol{y} \rangle)$.

Let g be any binary operation $g : (x, y) \in \mathbb{K}^2 \mapsto z \in \mathbb{K}$ and G be a $\boldsymbol{v}$-gadget for g. In the following, we shall say that a triple of vector $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in (\mathbb{K}^n)^3$ is *admissible* for G if there exists a random tape $\boldsymbol{\rho}$ such that $\boldsymbol{z} = G^{\boldsymbol{\rho}}(\boldsymbol{x}, \boldsymbol{z})$. For an admissible triple $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ and a set $\mathcal{W} \subseteq [|G|]$, the wire distribution of G in $\mathcal{W}$ *induced* by $(\boldsymbol{x}, \boldsymbol{y})$, denoted $G_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, is the random vector $G^{\boldsymbol{\rho}}_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y})$, *i.e.* the tuple of wire values for the wire indexes in $\mathcal{W}$, obtained for a uniform drawing of $\boldsymbol{\rho}$ among the set $\{\boldsymbol{\rho} \in \mathbb{K}^q \,;\, G^{\boldsymbol{\rho}}_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{z}\}$.

Definition 11 (IOS)**.** Let $\boldsymbol{v} \in \mathbb{K}^n$ and g be any binary operation $g : (x, y) \in \mathbb{K}^2 \mapsto z \in \mathbb{K}$. A $\boldsymbol{v}$-gadget G of g is said *t-input-output separative (t-IOS)*, if it is uniform and if for every admissible triple $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ and every set of wires $\mathcal{W} \subseteq [s]$ with $|\mathcal{W}| \leq t$, there exists a (two-stage) simulator $\mathcal{S}_{G,\mathcal{W}} = \big(\mathcal{S}^{(1)}_{G,\mathcal{W}}, \mathcal{S}^{(2)}_{G,\mathcal{W}}\big)$ such that

1. $\mathcal{S}^{(1)}_{G,\mathcal{W}}(\perp) = (\mathcal{I}_1, \mathcal{I}_2, \mathcal{J})$ where $\mathcal{I}_1, \mathcal{I}_2, \mathcal{J} \subseteq [n]$, with $|\mathcal{I}_1| + |\mathcal{I}_2| \leq |\mathcal{W}|$ and $|\mathcal{J}| \leq |\mathcal{W}|$;

2. $\mathcal{S}^{(2)}_{G,\mathcal{W}}(\boldsymbol{x}_{|\mathcal{I}_1}, \boldsymbol{y}_{|\mathcal{I}_2}, \boldsymbol{z}_{|\mathcal{J}}) \overset{\text{id}}{=} G_{\mathcal{W}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$.

A $\boldsymbol{v}$-gadget is simply said to be *IOS* if it is *n-IOS*.

B Random Probing and Noisy Leakage Security

Ajtai introduced in [2] the so-called *random probing model* in which an adversary cannot choose a fixed number of arbitrary wires but instead the leaking wires are chosen independently, each with some given probability p. This model is therefore similar to the *binary erasure channel* used in coding theory and information theory.

Definition 12 (Random Probing Model)**.** Let $\varepsilon, p \in [0,1]$. A randomized arithmetic circuit $\widehat{C}$ is (p,ε)*-random probing secure* w.r.t. an encoding algorithm Encode if there exists a simulator $\mathcal{S}_{\widehat{C}}$ such that, sampling a set of wires $\mathcal{W} \subseteq \big[|\widehat{C}|\big]$, where each wire from $\widehat{C}$ belongs to $\mathcal{W}$ independently with probability p, and for every plain input $\boldsymbol{x}$, we have:

$$\mathcal{S}_{\widehat{C},\mathcal{W}}(\perp) \overset{\text{id}}{=} \widehat{C}_{\mathcal{W}}(\mathsf{Encode}(\boldsymbol{x})) \ . \tag{15}$$

with probability at least $1-\varepsilon$ over the random sampling of $\mathcal{W}$. A circuit compiler $(\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$ is (p,ε)*-random probing secure* if for every circuit C the compiled circuit $\widehat{C} = \mathsf{Compile}(C)$ is (p,ε)-random probing secure w.r.t. Encode (where p and ε might be a function of the encoding order and the circuit size).

Using the classical Chernoff's inequality, it is easy to prove that security in the region probing model implies security in the random probing model with appropriate parameters.

Proposition 2. *Let $r \in [0,1]$ and $\widehat{C}$ be a randomized circuit r-region probing secure w.r.t. an encoding algorithm* Encode *with a circuit partition $\widehat{C} \equiv (C_1, C_2, \ldots, C_m)$ where $|C_i| \geq t$ for each $i \in \{1, \ldots, m\}$. The circuit $\widehat{C}$ is (p,ε)-random probing secure w.r.t.* Encode *with*

$$p = \frac{r}{2} \text{ and } \varepsilon \leq m \cdot \exp(-p \cdot t/3).$$

Proof. More generally, let $\delta \geq 1$ and let us suppose that $p \leq r/(1+\delta)$. Let $\mathcal{W} \subseteq \big[|\widehat{C}|\big]$ be a wire set where each wire from $\widehat{C}$ belongs to $\mathcal{W}$ independently with probability p. For each subcircuit C_i of the circuit partition $\widehat{C} \equiv (C_1, C_2, \ldots, C_m)$, we denote $\mathcal{W}_i = \mathcal{W} \cap C_i$ for $i \in \{1, \ldots, m\}$. By Chernoff's bound, we have $\Pr[|\mathcal{W}_i| \geq (1+\delta)p|C_i|] \leq \exp(-\delta^2 p|C_i|/(\delta+2))]$ for $i \in \{1, \ldots, m\}$ and therefore

$$\Pr\big(|\mathcal{W}_i| \geq \lceil r|C_i|\rceil\big) \ \leq \ \exp(-\delta p \cdot t/3)$$

for $i \in \{1, \ldots, m\}$. Using the union bound over the m sets $\mathcal{W}_i$ for $i \in \{1, \ldots, m\}$ and setting $\delta = 1$, we get the claimed bounds. □

Noisy Leakage Model. The noisy leakage model was first formalized by Prouff and Rivain [38]. In this model, the adversary may learn information about every single wire; however, instead of learning exactly the value x of a wire (as in the probing model), the adversary learns a randomized function $f(x)$ of x. The generality of the noisy leakage model allows it to encompass several real-life instances of leakages, making it a very realistic leakage model. However, due to its somewhat analytical nature, security proofs are notoriously hard to do in it.

Thankfully, is has been shown that the noisy leakage and random probing models are equivalent: one implication was proven [25] and improved in [37], the other one was proven in [37]. As a workaround to the complexity of the noisy leakage model, one can therefore establish security proofs in the random probing model and subsequently transfer them in the noisy leakage model using the equivalence between the two. Proposition 2 provides an additional tool to this proof strategy and, combined with results of [25, 37], imply that a compiler secure in the region probing model is secure in the noisy leakage model.

Security of the GJR$^+$ scheme. We have the following corollary of Theorem 3.

Corollary 2. *If the FFT circuit is linear and under the t_n^{R}-IOS property of the refresh gadget, the GJR$^+$ compiler is (p_n, ε_n)-random probing secure with*

$$p_n = \frac{1}{2} \max_{t \le t_n^{\mathsf{R}}} \min\Big(\frac{(n-1) - 6t}{2 \cdot |\mathsf{FFT}_n|} , \frac{t}{|G_n^{\mathsf{R}}|} \Big) \tag{16}$$

and

$$\varepsilon_n = (2N^{\oplus} + 4N^{\otimes}) \cdot \exp(-n \cdot p_n) + 3N^{\otimes} \cdot \frac{n}{|\mathbb{F}|} , \tag{17}$$

where $N^{\oplus}$ (resp. $N^{\otimes}$) is the number of addition gates (resp. multiplication gates) in the original circuit.

Proof. Let C be an arithmetic circuit composed of $N^{\oplus}$ addition (or linear) gates and $N^{\otimes}$ multiplication gates and let $\widehat{C}$ be the corresponding compiled circuit output from the GJR$^+$ compiler. We consider a split of $\widehat{C}$ into different regions following the region probing security reduction of the GJR$^+$ scheme. Specifically, each addition (or sharewise) gadget and each refresh gadget consists in a single region while each multiplication gadget gives rise to three different regions: the first block (*i.e.* the circuit considered in Lemma 1), the internal refresh, and the second block (*i.e.* the circuit considered in Lemma 2). We hence get a total of $N_{reg} = 2N^{\oplus} + 4N^{\otimes}$ regions in $\widehat{C}$ (counting the refresh gadgets). In the random probing model, each wire leaks independently of the other wires with a given probability, denoted p_n here. We show hereafter that with overwhelming probability over the distribution of the indexes of the leaking wires, the full random probing leakage can be perfectly simulated. More specifically, we exhibit two failure events $\mathcal{F}_1$ and $\mathcal{F}_2$ that may prevent such a perfect simulation. Whenever none of these failure events occur, a perfect simulation is achieved in the same way as in the region probing security reduction given above.

The first failure event $\mathcal{F}_1$ occurs when the number of leaking wires in at least one region R exceeds the threshold $2p_n|R|$ where $|R|$ denotes the number of wires in R. Applying the Chernoff bound, this occurs in a given region with probability lower than

$$\exp\Big(- \frac{p_n \cdot |R|}{3} \Big) \le \exp(-n \cdot p_n) , \tag{18}$$

where the inequality holds since the minimal value of $|R|$ is obtained for the addition gadget with $|R| = 3n$. We deduce that the first failure event occurs with probability

$$\Pr(\mathcal{F}_1) \le N_{reg} \cdot \exp(-n \cdot p_n) . \tag{19}$$

Provided that the first failure event does not occur, the random probing simulator needs to simulate less than $2p_n|R|$ wires per region that is $r_n|R|$ wires per region where $r_n = 2p_n$ is as defined in Theorem 3 for $t_n^{\mathsf{FFT}} = (n-1)$. This translates to simulating at most $t_n^{\mathsf{FFT}} = (n-1)$ probes in each FFT circuit through the above security reduction. Our second failure event $\mathcal{F}_2$ occurs whenever the $n-1$ leaking wires within a FFT circuit cannot be perfectly simulated. Using Lemma 2 from [29] (see Appendix C) and thanks to the linearity of the FFT circuit, we have that for any choice of $n-1$ leaking wires from the FFT circuit, the probability that the leaking wires cannot be simulated is lower than $n/|\mathbb{F}|$. Besides the linearity of the FFT circuit, the only requirement for this upper bound to apply is that the choice of the leaking wires is made independently of ω, which occurs in the random probing model since the placement of the probes is randomly draw (with leakage probability p for each wire) independently of the random generation of ω. We deduce that the second failure event $\mathcal{F}_2$ occurs with probability

$$\Pr(\mathcal{F}_2) \leq 3N^{\otimes} \cdot \frac{n}{|\mathbb{F}|} \ . \tag{20}$$

Whenever no failure event occur, the leaking wires within each FFT circuit can be perfectly simulated which –following the above reduction– implies that the overall leaking wires can be perfectly simulated. It results that the GJR$^+$ scheme is (p_n, ε_n)-random probing secure with

$$\varepsilon_n \leq \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) \leq \Pr(\mathcal{F}_1) + \Pr(\mathcal{F}_2) \ , \tag{21}$$

which together with Equation 19 and Equation 20 concludes the proof. □

In the above random probing security proof, the tolerated number of probes in an FFT circuit is $t_n^{\mathsf{FFT}} = (n-1)$ which implies $\gamma \approx 1$ in Equation 8. On the other hand, our refresh gadget is such that $\beta = 3$. Assuming a FFT algorithm satisfying $|\mathsf{FFT}_n| = \alpha \cdot n \log n$, we finally get

$$p_n = \frac{r_n}{2} \approx \Big(\frac{1}{4\alpha + 36}\Big) \cdot \frac{1}{\log n} \ . \tag{22}$$

For instance, the NTT algorithm used in the original GJR scheme satisfies $\alpha \approx 6$, which gives $p_n \approx \frac{1}{60 \log n}$.

C Lemmas from [29]

We recall hereafter the key lemmas from [29] for the security of the GJR scheme. Let FFT_n be a linear FFT circuit on field $\mathbb{K}$ taking an ω-encoding as input. Every value v taken by a wire of FFT_ncan be expressed as

$$v = \sum_{i=0}^{n-1} \alpha_i a_i \tag{23}$$

where the α_i's are constant coefficients over $\mathbb{K}$. The lemmas use the following notation

$$[v] = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})^{\mathsf{T}} \tag{24}$$

for the column vector of coefficients of such a wire value. Similarly, we shall denote $[a] = (1, \omega, \omega^2, \ldots, \omega^{n-1})^{\mathsf{T}}$ for an ω-encoding $(a_1, \ldots, a_n)$ of a variable a since we have $a = \sum_{i=0}^{n-1} \omega^i a_i$ by definition. Moreover, $[v_0, v_1, \ldots, v_\ell]$ shall denote the matrix with column vectors $[v_0]$, $[v_1]$, …, $[v_\ell]$.

Lemma 3 (Lemma 1 of [29])**.** *Let $v_1, v_2, \ldots, v_\ell$ be the values taken by $\ell < n$ wires of* FFT_n *on input a uniform ω-encoding of a variable a. The distribution of the tuple $(v_1, v_2, \ldots, v_\ell)$ is statistically independent of a iff*

$$[a] \notin \text{span}([v_1, \ldots, v_\ell]) \ , \tag{25}$$

where span$(\cdot)$ *refers to the linear span of the input matrix.*

Lemma 4 (Lemma 2 of [29]). *Let ω be a uniform random element in $\mathbb{K}^*$. And let $v_1, v_2, \ldots, v_\ell$ be a set of $\ell < n$ intermediate variables of FFT_n on input an ω-encoding of a variable a. We have:*

$$\Pr\big([a] \in \mathrm{span}([v_1, \ldots, v_\ell])\big) \le \frac{\ell}{|\mathbb{K}| - 1} < \frac{n}{|\mathbb{K}|} , \tag{26}$$

where the above probability is taken over a uniform random choice of ω.

From these two lemmas, the values taken by any set of $\ell < n$ wires of FFT_n can be perfectly simulated without knowledge of a. The simulation simply works by taking a random a, picking a random ω-encoding of a, and evaluating the wires $v_1, \ldots, v_\ell$ accordingly leads to a perfect simulation. According to Lemma 2 of [29] such a simulation fails with probability lower than $n/|\mathbb{K}|$.

D Complements on Gao-Mateer additive FFT

D.1 Algorithms used in the Gao-Mateer additive FFT

This section presents the detailed algorithms used in the Gao-Mateer additive FFT.

Algorithm 5 TaylorExpansion(f, N)

```
Require: $f \in \mathbb{F}[x]$ of degree $< N$
Ensure: The Taylor expansion $\{h_0, \ldots, h_{N/2-1}\}$ of $f$ w.r.t. $(x^2 - x)$
 1: if $N \le 2$ then
 2:     return $\{f\}$
 3: $k \leftarrow N/4$
 4: $g_0 \leftarrow \sum_{i=0}^{2k} f_i \cdot x^i$          ▷ $g_0$ is the low-order coefficients of $f$
 5: $g_1 \leftarrow \sum_{i=0}^{2k} f_{i+2k} \cdot x^i$     ▷ $g_1$ is the high-order coefficients of $f$
 6: for $i = 0, \ldots, k-1$ do
 7:     $g_1(x) \leftarrow g_1(x) \oplus g_{1,i+k} \cdot x^i$
 8:     $g_0(x) \leftarrow g_0(x) \oplus g_{1,i} \cdot x^{i+k}$
 9: $V_0 \leftarrow$ TaylorExpansion$(g_0, N/2)$          ▷ Recursive call
10: $V_1 \leftarrow$ TaylorExpansion$(g_1, N/2)$          ▷ Recursive call
11: return $V_0 \| V_1$
```

Algorithm 6 Fold(B)

```
Require: A basis $B = \{\beta_0, \ldots, \beta_{m-1}\} \subset \mathbb{F}^m$
Ensure: Two new basis $G, D \subset \mathbb{F}^{m-1}$
 1: for $i = m-2, \ldots, 0$ do
 2:     $\gamma_i \leftarrow \beta_i / \beta_{m-1}$
 3:     $\delta_i \leftarrow \gamma_i^2 - \gamma_i$
 4: $G \leftarrow \{\gamma_0, \ldots, \gamma_{m-2}\}$
 5: $D \leftarrow \{\delta_0, \ldots, \delta_{m-2}\}$
 6: return $G, D$
```

Algorithm 7 $\mathsf{GaoMateerFFT}(f, m, B)$

```
Require: A polynomial $f \in \mathbb{F}[x]$ of degree $< 2^m$, a basis $B = \{\beta_0, \ldots, \beta_{m-1}\} \subset \mathbb{F}^m$
Ensure: The additive FFT of $f$ over the span $\langle B \rangle$
 1: if $m = 1$ then
 2:     return $\{f(0), f(\beta_0)\}$.
 3: Compute $g(x) = f(\beta_{m-1}x)$.
 4: Compute the Taylor expansion of $g$, i.e. $g(x) = \sum_i (g_{i,0} + g_{i,1}x) \cdot (x^2 - x)^i$.
 5: Let $g_0 \leftarrow \sum_i g_{i,0}x^i$ and $g_1 \leftarrow \sum_i g_{i,1}x^i$.
 6: Let $G, D \leftarrow \mathsf{Fold}(B)$, and $k = 2^{m-1}$.
 7: $\{u_0, u_1, \ldots, u_{\frac{N}{2}-1}\} \leftarrow \mathsf{GaoMateerFFT}(g_0, m-1, D)$
 8: $\{v_0, v_1, \ldots, v_{\frac{N}{2}-1}\} \leftarrow \mathsf{GaoMateerFFT}(g_1, m-1, D)$
 9: for $i = 0, \ldots, \frac{N}{2} - 1$ do
10:     $w_i \leftarrow u_i \oplus G[i] \cdot v_i$
11:     $w_{\frac{N}{2}+i} \leftarrow w_i \oplus v_i$
12: return $\{w_0, w_1, \ldots, w_{n-1}\}$
```

D.2 Half-FFT

Our improved (half-)FFT works with a self-folding basis and its iterated foldings. We clarify the notation: let $B = \{\beta_0, \ldots, \beta_{m-1}\}$ be a self-folding basis, and $B_k = \{\beta_{m-k}, \ldots, \beta_{m-1}\}$ for $k \in \{1, \ldots, m\}$. We also denote $G_k = \{\beta_{m-k-1}, \ldots, \beta_{m-2}\}$, so that $|B_k| = |G_k| = k$, and $(G_{k-1}, B_{k-1}) = \mathsf{Fold}(B_k)$.

For our purposes, it is adequate to precompute $G_j[i]$ for $1 \leq j < m$ and $0 \leq i < 2^j$. In this section, we describe how various algorithmic tricks can help to improve the additive FFT of [27] in our setting. First, we observe that from Proposition 1, it holds that for each B_k, the $(k-1)$-th element of B_k is 1, hence the step 3 of Algorithm 7 becomes unnecessary; in total, this saves us $N \log N$ scalar multiplications. Second, we note that since each G_k is a subset of G_{m-1}, storing precomputed multiplication tables for the step 10 does not require to store $\frac{N}{2} + \frac{N}{4} + \cdots + 1 = N - 1$ precomputed tables anymore, but rather $\frac{N}{2} - 1$ (not having to store the multiplication by zero saves an additional element). Added to the removal of step 3, this more than divides by two the number of precomputed tables. As a final algorithmic optimization, we make full use of the fact that for polynomial multiplications, half the inputs of the FFT's call are zero coefficients which leads to speed up to the computations by a factor two compared to a regular additive FFT. This optimized FFT is described in Algorithm 8.

Algorithm 8 HalfFFT(f, B)

```
Require: $f \in \mathbb{F}[x]$ of degree $< N = 2^{m-1}$, a self-folding basis $B = \{\beta_0, \ldots, \beta_{m-1}\}$ and
    its iterated foldings $G_{m-1}, \ldots, G_1$.
Ensure: The evaluation of $f$ over the $\mathbb{F}_2$-linear space generated by $B_m$
 1: $W \leftarrow 0^N$
 2: if $m = 2$ then                                          ▷ Bottom of the recursion
 3:     $W_0 \leftarrow f_0$                    ▷ Since $\forall j, G_j[0] = 0$, it simplifies the computation
 4:     $W_1 \leftarrow f_0 \oplus G_1[1] \cdot f_1$
 5:     $W_2 \leftarrow f_0 \oplus f_1$
 6:     $W_3 \leftarrow W_1 \oplus f_1$
 7:     return $W$
 8: $T \leftarrow$ TaylorExpansion$(f, N/2)$         ▷ $T$ is a list of $N/4$ polynomials $T_i = T_{i,0} \oplus T_{i,1}x$
 9: $g_0 \leftarrow \sum_{i=0}^{N/4-1} T_{i,0} \cdot x^i$
10: $g_1 \leftarrow \sum_{i=0}^{N/4-1} T_{i,1} \cdot x^i$
11: $U \leftarrow$ HalfFFT$(g_0, \{\beta_1, \ldots, \beta_{m-1}\})$                          ▷ Recursive call
12: $V \leftarrow$ HalfFFT$(g_1, \{\beta_1, \ldots, \beta_{m-1}\})$                          ▷ Recursive call
13: for $i = 0, \ldots, N/2$ do
14:     $W_i \leftarrow U_i \oplus G_{m-1}[i] \cdot V_i$
15:     $W_{i+N/2} \leftarrow W_i \oplus V_i$
16: return $W$
```

The optimizations we just described apply in a similar way to the inverse additive FFT. The only difference here is that we cannot exploit anymore the fact that half of the polynomial coefficients are zero: indeed, since we use the additive FFT to multiply two polynomials of degree at most $n - 1 = N/2 - 1$, we expect the degree of the result to be at most $N - 2$. Hence, simply applying the inverse of each operations of Algorithm 8 in reverse order would yield an incorrect result. Our optimized inverse additive FFT is described in Algorithm 9.

Algorithm 9 InverseFFT(W, B)

```
Require: A self-folding basis $B = \{\beta_0, \ldots, \beta_{m-1}\}$ and its foldings $G_{m-1}, \ldots, G_1$, the
    evaluation $W$ of $f$ over the $\mathbb{F}_2$-linear space generated by $B$
Ensure: A polynomial $f \in \mathbb{F}[x]$ of degree $< N = 2^m$
 1: if $m = 2$ then                                          ▷ Bottom of the recursion
 2:     $u \leftarrow W_1 \oplus W_2$
 3:     $v \leftarrow W_1 \oplus W_3$
 4:     $f_0 \leftarrow W_0 \oplus (G_1[0] \cdot u)$
 5:     $f_1 \leftarrow (G_1[1] \cdot (v)) \oplus f_0 \oplus W_0 \oplus u$
 6:     $f_2 \leftarrow f_1 \oplus v$
 7:     $f_3 \leftarrow f_1 \oplus f_2 \oplus W_0 \oplus W_2$
 8:     return $f_0 \oplus f_1 \cdot x \oplus f_2 \cdot x^2 \oplus f_3 \cdot x^3$
 9: for $i = 0, \ldots, N/2$ do
10:     $V_i \leftarrow W_i \oplus W_{i+N/2}$
11:     $U_i \leftarrow W_i \oplus G_{m-1}[i] \cdot V_i$
12: $U \leftarrow (U_i)_i, V \leftarrow (V_i)_i$
13: $g_0 \leftarrow$ InverseFFT$(U, \{\beta_1, \ldots, \beta_{m-1}\})$                        ▷ Recursive call
14: $g_1 \leftarrow$ InverseFFT$(V, \{\beta_1, \ldots, \beta_{m-1}\})$                        ▷ Recursive call
15: $T \leftarrow (g_{0,i} + g_{1,i} \cdot x)_{0 \leq i \leq N/2}$
16: $f \leftarrow$ inverseTaylorExpansion$(T, N)$
17: return $f$
```

D.3 Complexity Analysis

In this section, we provide the complexity of our new algorithms. We do not change the algorithm for the Taylor expansion, which cost remains the same: $N(\log N - 1)/2$ field additions and no multiplication.

Let $A(N)$ and $M(N)$ denote the number of field additions and multiplications entailed by Algorithm 8:

- We note that $M(4) = 1$ and that $M(N) = 2 \cdot M(N/2) + N/2$. From these two facts, one can show by induction that $M(N) = \frac{1}{2} \cdot N \cdot \log_2(N) - \frac{3}{4} \cdot N$.
- Similarly, we have $A(4) = 3$ and $A(N) = \frac{1}{4} N \log \frac{N}{2} - \frac{1}{4} N + 2A(N/2) + N$. It follows by induction that $A(N) = \frac{1}{8} \cdot N \cdot (\log_2(N))^2 + \frac{5}{8} \cdot N \cdot \log_2(N) - N$.

Using the same techniques, one can show that Algorithm 9 performs $\frac{1}{4} \cdot N \cdot (\log_2(N))^2 + \frac{3}{4} \cdot \log_2(N)$ additions and $\frac{1}{2} \cdot \log_2(N) - \frac{1}{2} \cdot N$ multiplications.

We note that thanks to our use of self-folding bases, we divide by about 4 the number of multiplications compared to [27]. Similarly, exploiting the degree of the input polynomials divides the number of additions in Algorithm 7 by two. For all practical purposes, our algorithms even perform better than the specialized additive FFT described in [27, Section IV]. This FFT requires slightly more multiplications ($\frac{1}{2} N \log_2(N)$) than ours, and the number of additions ($N \log_2(N) + \frac{1}{2} N \log_2(N) \log_2 \log_2 N$), while asymptotically better than ours, remains larger than the number of additions in Algorithm 9 for any $N \leq 256$. Since the specialized FFT works only for m a power of two, it will only start to outperform Algorithm 9 for $N \geq 2^{16}$, corresponding to a masking order $n = 32768$. It also remains to be studied whether this algorithm can exploit the degree of the input polynomials to reduce the number of additions, as done by Algorithm 7. We note that just like in [27], all the multiplications in our algorithms are field multiplications by a constant, which can be stored in precomputed tables.