Combined Fault and Leakage Resilience: Composability, Constructions and Compiler*

Sebastian Berndt[1], Thomas Eisenbarth[2], Sebastian Faust[3], Marc Gourjon[4,5], Maximilian Orlt[3], and Okan Seker[2,5]

[1] Institute for Theoretical Computer Science, University of Lübeck, Germany
s.berndt@uni-luebeck.de
[2] Institute for IT Security, University of Lübeck, Germany
thomas.eisenbarth@uni-luebeck.de
[3] TU Darmstadt, Germany, firstname.lastname@tu-darmstadt.de
[4] Hamburg University of Technology, Germany, firstname.lastname@tuhh.de
[5] NXP Semiconductors, Germany okan.seker@nxp.com

Abstract. Real-world cryptographic implementations nowadays are not only attacked via classical cryptanalysis but also via implementation attacks, including passive attacks (observing side-channel information about the inner computation) and active attacks (inserting faults into the computation). While countermeasures exist for each type of attack, countermeasures against combined attacks have only been considered recently. Masking is a standard technique for protecting against passive side-channel attacks, but protecting against active attacks with additive masking is challenging. Previous approaches include running multiple copies of a masked computation, requiring a large amount of randomness or being vulnerable to horizontal attacks. An alternative approach is polynomial masking, which is inherently fault-resistant.
This work presents a compiler based on polynomial masking that achieves linear computational complexity for affine functions and cubic complexity for non-linear functions. The resulting compiler is secure against attackers using region probes and adaptive faults. In addition, the notion of fault-invariance is introduced to improve security against combined attacks without the need to consider all possible fault combinations. Our approach has the best-known asymptotic efficiency among all known approaches.

1 Introduction

In classical cryptography, the security of cryptographic primitives is often analyzed in the black-box model. In this model, the adversary attacks the cryptographic algorithm via access to inputs and outputs but has no knowledge and no control over the inner workings of the algorithms. In particular, sensitive information

*Full version

such as the secret key is hidden from and out of the control of the adversary. Unfortunately, when running cryptographic algorithms on real-world devices countless attacks demonstrate that the black-box model is far too optimistic. Examples include *passive attacks*, where the adversary exploits physical phenomena such as the power consumption or running time of a device to extract sensitive information; or *active attacks*, where the adversary modifies temporary values via a laser or via heating up the device to introduce faulty computation.

Masking schemes. Masking schemes are a classical countermeasure to protect against passive side-channel attacks. Masking conceals sensitive information by secret sharing each value v from some finite field $\mathbb{F}$ into shares $v_0, \ldots, v_{n-1}$ such that $d+1$ shares are required to reconstruct the secret, while $\leq d$ shares reveal nothing about the sensitive value v. The most common sharing scheme is *additive* masking. Here, we choose $v_0, \ldots, v_{n-2}$ randomly from $\mathbb{F}$ and define v_{n-1} such that $v = v_0 \oplus v_2 \ldots \oplus v_{n-1}$, where $\oplus$ is the addition of the underlying field $\mathbb{F}$. The main challenge in designing masking schemes is to securely compute on the shared values. To this end, we design masked subcircuits, called *gadgets*, that securely compute on sharings and devise methods for securely composing such gadgets without violating overall security.

The security of a masking scheme is typically analyzed in the so-called probing model originally introduced by Ishai, Sahai and Wagner [ISW03]. In this model, the adversary can learn up to d values that are produced during the computation. The security proof is typically done by analyzing the d-probing security of the gadgets and then extending it towards security of an entire masked circuit via *composition*. To argue secure composition in the probing model, an important property is *(strong) non-interference (SNI)*. Intuitively, this property guarantees that all information gained by the attacker by probing d internal values of a gadget can also be obtained by probing at most d shares of the masked input. Furthermore, probes on the output sharing can be simulated from scratch in the case of the stronger notion.

Beyond passive security. As mentioned above, passive side-channel attacks are not the only threat to cryptographic implementations. In practice, an adversary may also be able to induce faults into the computation thereby breaking the cryptographic implementation. Even worse, a physical adversary may launch combined attacks, where the adversary both passively observes side-channel leakage and introduces faults to break the cryptographic implementation. While a masking scheme can be used to protect against the passive adversary, it is easy to see that it fails to offer security against faults. For instance, if an adversary succeeds in adding an offset $c \in \mathcal{F}$ to only one of the shares, the result of the computation is faulty, which may have catastrophic consequences for security [BS97]. Hence, we need to extend probing security to also include fault attacks, where in addition to placing d probes, the adversary is allowed to induce ϵ faults. In this work, we consider arbitrary *adaptive* faults that might even depend on the information obtained via previous probes.

With adaptive faults, we cover attacks where the adversary uses information from previous leakage to insert faults. While such attackers seem unrealistically strong, they are possible in the context of side-channel attacks: Firstly, the adversary has access to the device and could stop or slow down regions of the devices similar as in cold boot attacks. Secondly, the adversary could use the leaked information not immediately but in the next cycle of a circuit that uses multiple cycles, such as AES and Present. Finally, we note that adaptive adversaries are a stronger adversarial model. Thus, from a theoretical point of view it is interesting to explore what security can be achieved in this model.

In order to protect against combined attacks, two main approaches have been considered in the literature – duplicated masking and polynomial masking. The most common one is duplicated masking to replicate the masked computation [DN20b, FRSG22].

Duplicated Approach. In this setting, the masked circuit $\hat{C}$ is executed $\epsilon + 1$ times in parallel. After each gate, the masked outputs of the computation are checked for equality to detect faults. This requires that the $\epsilon + 1$ copies use the same randomness internally. Otherwise, the output sharings would not be equal. Moreover, re-using the randomness has an additional advantage. As generating randomness is costly, by re-using randomness in all $\epsilon + 1$ copies, the overall randomness used can be reduced by a factor of $O(\epsilon)$.

The duplication approach described above has two important shortcomings. First, affine operations that traditionally can be masked at very low cost (typically at an $O(n)$ complexity overhead for $n = \epsilon + d$) get significantly more expensive as each such masked affine operation is now computed $\epsilon + 1$ times. This is especially problematic, as many modern primitives, e.g., [BDPA13, GLR+20, ARS+15, AGR+16, AAB+20, HKL+22, GKR+21] aim to reduce the number of non-linear operations by increasing the number of affine operations significantly. Even worse, in terms of security, the duplication approach is very vulnerable to so-called horizontal attacks [BCPZ16].

Horizontal Attacks. Horizontal attacks are attacks in which an adversary exploits the fact that multiple computations share the same randomness or secret key material. In the context of side-channel attacks, horizontal attacks can be particularly devastating as the attacker can amplify the leaked information and thus recover sensitive information more easily [CFG+10, ORSW12, VGS14, BS21]. To protect against horizontal attacks, it is essential to ensure that different instances of computation use independent and fresh randomness or secret key material.

Clearly, the duplication method is particularly vulnerable to horizontal attacks, as all copies share the same randomness to ensure fault detection. This drawback was already observed and explicitly stated in [FRSG22]. To illustrate this issue more clearly, the appendix contains calculations describing the influence of the duplication on attacks in the random probing model. The random probing model is the standard method to analyze security of the masking countermeasure against horizontal attacks. In this model, the adversary can choose an *unbounded number of wires* and receives the value on each chosen wire with probability

p. Alternatively, such attacks can be modeled in the region probing model, also introduced in [ISW03]. Here, the threshold model is extended so that the threshold of probes applies to each gadget (or regions) in the circuit. In other words, the total number of probes increases with the number of gadgets. It has been shown by Duc et.al. [DDF14] that security in this model also implies security in the random probing model. Recently, this property has been used to construct secure compilers e.g. [ADF16, GPRV21]. In this work, we also allow up to t probes in each gadget to model horizontal attacks. One possible way to improve security against horizontal attacks is to use fresh randomness in each copy, but (a) it is not straightforward to detect faults in such randomized computation, and (b) the randomness complexity increases to $O(|C|n^3)$.

An alternative approach to executing the masked computation multiple times is to use an different sharing scheme. Recall that additive secret sharing is highly vulnerable to fault attacks as already a single fault is undetectable. Hence, using an additive sharing intuitively requires a large number of independent copies of the same computation to avoid these faults. To tackle this problem, an obvious idea is to resort to error detection codes, where one of the most promising candidates are Reed Solomon codes (often also called Shamir's secret sharing), which in addition to error detection also offer linearity. The latter is particularly useful for carrying out computation with sharings. In the literature, masking schemes based on Shamir's secret sharing are often called *polynomial masking*.

Polynomial Masking. Here, we need $|\mathbb{F}| \geq n+1$ and choose pairwise different support points $\alpha_0, \ldots, \alpha_{d-1} \neq 0$. To share a value v, we construct a polynomial $f \in \mathbb{F}[x]$ of degree d such that $f(0) = v$. The i-th share v_i is now defined as $f(\alpha_i)$. Polynomial masking [CPR12, GM11, RP12] is a well-known countermeasure against side channel attacks, which offers advantages over additive secret sharing based schemes due to its higher algebraic complexity. Moreover, they allow for a simple protection against faults: We can add *redundant* points $\alpha_d, \alpha_{d+1}, \ldots \alpha_{n-1}$ and corresponding shares $v_{d+1}, v_{d+2}, \ldots, v_{n-1}$, but will still use a polynomial of degree d. Due to the error-correcting properties of these polynomial codes, *valid* codewords, i.e., those sharings describing a polynomial of degree d, will differ in at least $n-d$ positions. If an attacker modifies less than $n-d$ shares, the underlying polynomial (which can be interpolated from the shares v_i) will have degree at least $d+1$ due to the fundamental theorem of algebra. Hence, modification of only $n-d$ shares results in an *invalid sharing*.

The idea of using polynomial sharing was already used in the context of multiparty computations in the now classical work of Ben-Or, Goldwasser, and Wigderson [BGW88]. Using a more complicated scheme, called *verifiable secret sharing*, they show how to achieve perfect security if the number of corrupted parties is strictly less than $n/3$. Inspired by this, Seker, Fernandez-Rubio, Eisenbarth, and Steinwandt [SFRES18] adapted the BGW scheme to protect against combined attackers. Their main idea is to simplify the BGW multiplication to avoid using verifiable secret sharing in such a way that faulted inputs will lead to a faulted output with high probability. This allowed them to show that $n = 2d+\epsilon+1$ shares are sufficient to protect against d probes and ϵ *additive* faults,

i.e., faults where the attacker can add an arbitrary value to a wire (independently from the actual value on that wire). Both the randomness requirement and the computational complexity of their multiplication gadgets are asymptotical identical to those using the duplication approach, i.e., $O(n^2)$ and $O(n^3)$ respectively. But, due to their linear number of shares, they can compute affine operations in *linear* time. In this work, we show that $n = d + \epsilon + 1$ are both sufficient and necessary to protect against combined attacks. In particular, in contrast to earlier works [SFRES18, DN20b, FRSG22, RFSG22], we show security in a stronger adversarial model where the adversary can induce *adaptive* faults into the computation and security holds in the *region probing* model.

Our approach has the same asymptotical complexities for multiplication as both the duplicated approach and the approach of [SFRES18] and a linear complexity for affine operations. Furthermore, in contrast to the duplicated approach, our solution is provably resistant against horizontal attacks.

1.1 Contribution.

Our contributions are threefold. First, we present combined security notions suitable for *polynomial masking*. Second, we propose the notion of *fault-invariance*, that allows us to transform gadgets secure against probing attacks into ones that are secure against combined attacks. Third, we propose two new compilers which use the optimal (*linear*) number of $n = e + d + 1$ shares and show security against horizontal attacks in the region probing model.

Combined Security Notions for Polynomial Sharing. In previous works [SFRES18, DN20b,FRSG22,RFSG22], an (d, ϵ)-attacker was able to choose d wires for probes and ϵ wires for faults (and corresponding fault operations from a class of possible faults). Then, the circuit was faulted according to these faults and the values of the d chosen wires were given to the attacker aiming to extract some sensitive information from these values. We strengthen the attackers significantly with regard to both probes and faults by allowing *region probes* and *adaptive faults*. Informally, region probes allow to perform d probes per *gadget*, in contrast to d probes in total. Furthermore, our attacker can choose the fault applied to a wire based on the already observed probes adaptively. A formal description about the attacker model is presented in Section 3.

A careful analysis of the differences between additive masking and polynomial masking reveals that the previously used security definitions do not transfer easily. In additive masking, we want to give an upper bound on the number of faulty outputs, while in polynomial masking we want to give a lower bound on the *degree* of the polynomial described by the sharing. We present new definitions adapted to this difference that allow to argue the composability of two secure gadgets. Here, composability means if gadgets satisfy certain security properties, these properties also hold for more complex computation that is composed of such gadgets.

Simplification of combined security analyzes. The previous approach to prove the security against combined attackers was to verify probing security of these gadget for *all* possible fault combinations [RFSG22]. This often leads to very complicated proofs with many case distinctions and many optimizations developed cannot be reused. We introduce the notion of *fault-invariance* of a gadget that allows us to *lift* probing-secure gadgets to also be secure against combined attacks *without* the need to consider all possible fault combinations. A fault-invariant gadget that is (S)NI stays (S)NI even in the presence of faults and thus allows us to reuse existing probing-secure gadgets.

A new countermeasure for combined attacks. Finally, we present two new compilers secure against combined attackers using adaptive faults in the region probing model. These are the first such compilers as the existing countermeasures using additive masking are very vulnerable to such attacks. Compared to [SFRES18], we significantly reduce the number of needed shares from $2e+d+1$ down to $n = e + d + 1$ (which we also show as *optimal*). Along the way, we also show how to fix their approach by presenting an SNI-secure refresh. Compared to [DN19], we significantly reduce the number of needed random values down to $O(n^2)$ and the computational complexity down to $O(n^3)$. Finally, we also show that our compilers are secure against *horizontal attacks*, a feature explicitly not shared by [DN20b, FRSG22]. All of the approaches using duplicated sharing [DN20b, FRSG22] need a *quadratic* number of shares, hence their complexity will always be suboptimal for affine circuits or circuits with a very large number of affine gates, a feature of many modern blockciphers, e.g., [AGR+16, AAB+20, HKL+22, GKR+21].

For a comparison of our work to other works protecting against combined attacks, we refer to Table 1. Analysing the cryptographic primitives Keccak [BDPA13], LowMC [ARS+15] or HadesMiMc [GLR+20] yields complexity estimations shown in Table 2. These estimations show that our approach outperforms the duplication approach due to the large number of linear operations.

Table 1: A comparison of the complexity of the addition, multiplication and refresh gadgets with regard to $n = e + d$.

	# Shares	Multiplication		Addition	Refresh		Horizontal Att.
		Rand	Compl	Compl	Rand	Compl	Security
[DN20b, FRSG22]	$O(n^2)$	$O(n^2)$	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^3)$	✖
[SFRES18]	$O(n)$	$O(n^2)$	$O(n^3)$	$O(n)$	insecure		✖
[DN19][6]	$O(n)$	$O(n^3)$	$O(n^5)$	$O(n)$	$O(n^3)$	$O(n^5)$	✖
This Work	$O(n)$	$O(n^2)$	$O(n^3)$	$O(n)$	$O(n^2)$	$O(n^3)$	$O(n^{-2})$

[6] This result is only present in the version `20190603:070457` of the eprint paper.

Table 2: A rough estimation of the number of operations of our approach compared to the duplication approach of [DN20b, FRSG22] for $n = 8$ for Keccack [BDPA13], LowMC ($R = 55, m = 20$) [ARS+15], and HadesMiMc ($R_F = 10$) [GLR+20]. The numbers given here depend on estimations given in the corresponding works.

	#Add	#Mult	#Ops [DN20b, FRSG22]	#Ops This Work
Keccak	422 400	38 400	46 694 400	23 040 000
LowMC	28 894 643	3 300	1 850 946 752	232 846 744
HadesMiMc	1 820	150	193 280	91 360

1.2 Related Work.

The study of private circuits was initiated by the work of Ishai, Sahai, and Wagner who presented a generic compiler to protect against probing attacks [ISW03]. In a follow-up work, Ishai, Prabhakaran, Sahai, and Wagner also considered fault attacks and presented a corresponding compiler [IPSW06]. Note that a combination of two protection mechanisms against probing attacks and fault attacks might actually *lower* the security of the protection mechanisms [REB+08, LFZD14]. Similar to the work of Ishai, Prabhakaran, Sahai, and Wagner, the use of error-detection codes together with threshold implementations was studied by Schneider, Moradi, and Güneysu [SMG16] and by De Cnudde and Nikova [CN16]. Recently, the use of explicit multi-party computation protocols as protection mechanisms was studied by Reparaz, De Meyer, Bilgin, Arribas, Nikova, Nikov, and Smart [RDB+18] and by Dhooghe and Nikova [DN20a]. Closest to this paper is the work of Seker, Fernandez-Rubio, Eisenbarth, and Steinwandt that introduced the model of statistical security against fault attacks [SFRES18][7]. They also showed that the classical multi-party protocol of Ben-Or, Goldwasser, and Wigderson [BGW88] can be adapted to this scenario and reduced the number of shares need from $n \geq 3d + 1$ down to $2d + e + 1$.

Previous security notion. The most common security notions against probing-only attacks are *non-interference (NI)* and its stronger counterpart *SNI*. The stronger security notion provides very useful composition results. Namely, it guarantees that the composition of d-SNI gadgets is d-SNI again. Now, using d-SNI gadgets prevents a probing-only attacker (where $e = 0$) from obtaining any information, but faults might still be used to obtain information. Security notions against probing-only attackers have been studied intensively and it was shown that the non-interference notions indeed prevent realistic attacks (see also Duc, Dziembowski, and Faust [DDF14]). An alternative approach, called *probe-isolating non-interference (PINI)* was introduced by Cassiers and Standaert [CS20] and also allows composability.

For fault attacks, the situation is not as easy, as different strategies might lead to different properties that are non-comparable. The simplest behavior,

[7] In this work we show that the construction has a bug, see Sec. B.5.

fault detection, aims to detect possible faults. Now, one can regularly check for the existence of these faults and abort the computation to prevent information leakage. In a more complex setting, *fault correction*, the computation would try to correct possible faults. While fault correction is a very useful property, it usually comes at prohibitive cost. We thus only focus on the detection of faults. Adding fault checks after every gate would detect the presence of faults as early as possible, but would increase the cost of the computation severely. Our first goal is thus to minimize the number of fault checks. The existence of multiple successive gates where no fault detection is used opens up the danger of *ineffective faults*, i.e., faults that only change some parts of the computation, but do not change the output. More informally, these faults cancel out at some point in the computation. As shown, e.g., by Clavier [Cla07] or Dobraunig, Eichlseder, Korak, Mangard, Mendel, and Primas [DEK+18] these (statistical) ineffective faults can be used by attackers in a devastating way. To protect against such faults, we design *fault-robust* gadgets: If these gadgets are given faulted inputs, their outputs will also contain faults.

As described earlier, Dhooghe and Nikova [DN20b] introduced the notion of (S)NINA, a combination of (strong) non-interference and non-accumulation, to protect against combined attackers using d probes and ϵ faults. They showed that a duplicate additive sharing is sufficient to obtain security by presenting a multiplication gadget and a refresh gadget that provided security against combined attacks. Richter-Brockmann, Feldtkeller, Sasdrich, and Güneysu [RFSG22] extended the (S)NINA notion to provide accurate definitions for the hardware context and constructed a tool, VERICA, to analyze gadgets with regard to (S)NINA. Finally, Feldtkeller, Richter-Brockmann, Sasdrich, and Güneysu [FRSG22] adapted the related notion of probe-isolating non-interference (PINI) presented by Cassiers and Standaert [CS20] to fault attacks and combined attacks. Similar to the work of Dhooghe and Nikova, they also used duplicate additive sharing and designed corresponding multiplication and refresh gadgets for these sharings that are secure against combined attacks. In contrast to our work, they only allowed static non-adaptive faults and a total number of d probes (i.e., their attacker only worked in the classical threshold probing model and not in the region probing model). Furthermore, as each copy of the computation uses the same randomness, their approach is very vulnerable to horizontal attacks, as shown in Sec. B.3. We summarize the efficiency with regard to $n = e + d$ of the constructions of the previous works in Table 1.

2 Background

In this section, we fix the notation used throughout this paper and give the needed background on polynomials and circuits.

Notation. We denote the set of the numbers between 0 and $n-1$ by $[n]$, i.e., $[n] = \{0, \ldots, n-1\}$. We write $r \leftarrow_{\$} S$ to denote that r is a random, uniformly distributed element from the finite set S. To simplify notation, if we are given a

vector $(v_0, \ldots, v_{n-1})$, we write $(v_i)_{i \in I}$ to denote a vector only consisting of the elements v_i with $i \in I$ for $I \subseteq [n]$ and thus also $(v_i)_{i \in [n]}$ instead of $(v_0, \ldots, v_{n-1})$. If D and D' are probability distribution over domain X, we write $D \equiv D'$, if $D(x) = D'(x)$ for all $x \in X$, i.e., if the distributions agree at each point. Random variables $X_0, X_1, \ldots X_{n-1}$ over a set $\mathbb{F}$ are independent if it holds for any $a_0, a_1, \ldots a_{n-1} \in \mathbb{F}$ that $\Pr[X_0 = a_0, X_1 = a_1, \ldots X_{n-1} = a_{n-1}] = \prod_{i \in [n]} \Pr[X_i = a_i]$. We write that $X_0, X_1, \ldots X_{n-1}$ are k-wise independent if for any subset $I \subset [n]$ with $|I| \leq k$, the random variables $(X_i)_{i \in I}$ are independent. For a matrix A, its rank is denoted by $\mathrm{rank}(A)$ and its kernel by $\ker(A)$. The dimension of a linear subspace H is denoted by $\dim(H)$. The weight of a vector $(a_i)_{i \in [n]}$ is the number of non-zero elements $\mathrm{weight}((a_i)_{i \in [n]}) = |\{a_i : a_i \neq 0\}|$. Further, we use polynomials in $\mathbb{F}[x]$ that are functions $f : \mathbb{F} \to \mathbb{F}$ with $f(x) = \sum_{i=0}^{k} f_i x^i$ for a natural number k and for all $f_i \in \mathbb{F}$. The degree $\deg(f)$ of f is the highest index of the non-zero f_i's. In detail, $\deg(f) = \max_i\{i\colon \text{ with } f_i \neq 0\}$.

We say that a probability distribution D is *perfectly simulatable* from a set S, if there exists a simulator Sim such that the output of $\mathsf{Sim}(S)$ has the same distribution as D. In detail, it holds for any possible x in the domain of D that $\Pr[\mathsf{Sim}(S) = x] = \Pr[D = x]$. In the following we denote this with $\mathsf{Sim}(S) \equiv D$.

Polynomial sharing. Throughout this work, we will fix a finite field $(\mathbb{F}, \oplus, \odot)$ with addition $\oplus$ and multiplication $\odot$ such that $|\mathbb{F}| \geq n + 1$, where n will be clear from the application. For the sake of simplicity, we will often also write $\cdot$ instead of $\odot$ and $+$ instead of $\oplus$. Throughout this paper, we fix n pairwise different support points $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{F}$ with $\alpha_i \neq 0$. We will often represent a polynomial $f \in \mathbb{F}[x]$ via the shares $(F_i)_{i \in [n]}$ with $F_i = f(\alpha_i)$ and say that $(F_i)_{i \in [n]}$ is a degree d sharing. To see that $(F_i)_{i \in [n]}$ is indeed a valid representation of f, consider the *Vandermonde matrix* $V_{n,d} := V_{n,d}[\alpha_0, \ldots, \alpha_{n-1}]$, where the i-th row has the form $(1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^d)$. It is now easy to see that $V_{n,d} \cdot (f_0, \ldots, f_d)^T = (f(\alpha_0), f(\alpha_2), \ldots, f(\alpha_{n-1}))^T = (F_0, \ldots, F_{n-1})^T$, i.e., the Vandermonde matrix can be used to evaluate the polynomial on the public support points α_i. Furthermore, it is well known that $\det(V_{n,d}) = \prod_{0 \leq i < j \leq n-1} (\alpha_i - \alpha_j)$. As the α_i are pairwise different and belong to a field $\mathbb{F}$ (which is free of nonzero zero divisors), this determinant is non-zero. Hence, $V_{n,d}$ is regular and the inverse matrix $V_{n,d}^{-1}$ thus exists. As $V_{n,d} \cdot (f_0, \ldots, f_d)^T = (F_0, \ldots, F_{n-1})^T$, we have $(f_0, \ldots, f_d)^T = V_{n,d}^{-1} \cdot (F_0, \ldots, F_{n-1})^T$. Hence, the inverse Vandermonde matrix can interpolate the coefficients f_i from the shares F_i via a linear operator.

To share a sensitive value $s \in \mathbb{F}$ into n shares, we will construct a polynomial f with $f(x) = \sum_{i=0}^{n-1} f_i x^i$ of degree $n-1$ where the coefficients $f_1, f_2, \ldots, f_{n-1}$ are chosen randomly and f_0 is equal to the sensitive value s. Then, the value $F_i = f(\alpha_i)$ is the i^{th} share. We denote this randomized procedure that outputs $(F_i)_{i \in [n]}$ from s by $(s_i)_{i \in [n]} \leftarrow \mathsf{Enc}(s)$ with $s_i = f(\alpha_i)$. To recover the sensitive value $f_0 = s$, we only need the first row of $V_{n,d}^{-1}$. We denote the i-th element of the first row of $V_{n,d}^{-1}$ by $\lambda_i^{(0)}$. To reconstruct the shared value, we use the well-known *interpolation lemma*.

Lemma 1 (Interpolation Lemma). *Let $f \in \mathbb{F}[x]$ be a polynomial of degree $d \leq n$, let $\alpha_0, \ldots, \alpha_{n-1}$ be pairwise different support points in $\mathbb{F} \setminus \{0\}$, and let $\lambda_i^{(0)}$ be the entries of the first row of the inverse Vandermonde matrix $V_{n,d}[\alpha_0, \ldots, \alpha_{n-1}]$. Then $(\lambda_1^{(0)}, \ldots, \lambda_n^{(0)}) \cdot (f(\alpha_0), \ldots, f(\alpha_{n-1})) = f(0)$.*

To simplify notation, we also write $v \leftarrow \mathsf{Dec}((v_i)_{i\in[n]})$ with $v_i = f(\alpha_i)$ for this. Since f is of degree d, the sensitive value v can be reconstructed from $(v_i)_{i\in I}$ with any subset $I \subset [n]$ with $|I| > d$ and $(v_i)_{i\in I}$ is independent of v if $|I| \leq d$.

An important fact that we will make use of throughout this paper is the fact that a sharing $(F_i)_{i\in[n]}$ of a non-zero polynomial with many zero entries corresponds to a polynomial of high degree, as captured by the following well-known fact.

Lemma 2 (Fundamental Lemma). *Let $f \in \mathbb{F}[x]$ be a polynomial of non-zero degree. If f has k distinct roots,* $\deg(f) \geq k$.

Circuit model. We represent the computation via a circuit on the field $(\mathbb{F}, \oplus, \odot)$, i.e., we consider directed acyclic graphs G where each node is labeled as (i) input gate, (ii) output gate, (iii) random gate, (iv) addition gate, (v) multiplication gate, or (vi) constant (transformation) gate. To compute the circuit on given inputs $x_1, x_2, \ldots$ we first initialize the input gates with the according inputs. Then, at each time step, we evaluate all gates that only have parents that are already evaluated. Random gates are evaluated by sampling an element uniformly at random from $\mathbb{F}$. Constant transformation gates have two constants a and b and evaluate $a \cdot x + b$ on input x. For $a = 0$ it is the usual constant gate initialized with b. We denote the resulting output distribution $y_1, y_2, \ldots$ of circuit C with inputs $x_1, x_2, \ldots$ by $(y_1, y_2, \ldots) \leftarrow \mathsf{C}(x_1, x_2, \ldots)$. In order to simplify notation, we also write $\mathsf{C}^R(x_1, x_2, \ldots)$ for the output of C if the samples from the random gates are taken according to the *random values* R. A *gadget* is simply a subgraph of a circuit. We stress that our definition allows for an arbitrary out-degree of a gate. Hence, there is no need for copy gates or similar. Instead of outputting the result of the computation, a circuit can also *abort* the computation by returning the abort signal $\perp$.

Compiler. A *compiler* $\mathfrak{C}$ is a function transforming a circuit C into another circuit C'. We are interested in compilers that output fault- and leakage-resilient circuits C'. This can be done with polynomial sharing $(\mathsf{Enc}(\cdot), \mathsf{Dec}(\cdot))$ described above such that the circuit C' only computes on encoded values. Therefore, each gate is transformed into a sub-circuit, a so-called *gadget*, that takes as input the encoded inputs of the gate and outputs encodings such that the decoded output represents the outputs of the gate. For security reasons, additional randomness can be injected by so-called refresh gadgets that take as input an encoding and outputs a randomized encoding in such a way that the decoded value of the input and output is the same. For any circuit $\mathsf{C} : \mathbb{F}^n \to \mathbb{F}^m$ with n inputs and m outputs, the resulting compiler $\mathfrak{C}$ generates $\mathsf{C}' \leftarrow \mathfrak{C}(\mathsf{C})$ such that for any input $x^0, x^1, \ldots x^{n-1}$ and $(y_i^0)_{i\in[n]}, (y_i^1)_{i\in[n]}, \ldots (y_i^{m-1})_{i\in[n]} \leftarrow \mathsf{C}'(\mathsf{Enc}(x^0), \mathsf{Enc}(x^0), \ldots \mathsf{Enc}(x^{n-1}))$ it holds

that $\mathsf{Dec}((y_i^0)_{i\in[n]}), \mathsf{Dec}((y_i^1)_{i\in[n]}), \ldots \mathsf{Dec}((y_i^{m-1})_{i\in[n]}) = \mathsf{C}(x^0, x^1, \ldots x^{n-1})$. In this case we also write that C and C' are arithmetic circuits over $\mathbb{F}$ and $\mathbb{F}^n$, respectively to highlight the fact that C' is working on the shared representation. Further, we say that C' is a *masked circuit* and has the same functionality as C. Section 4 gives a more detailed construction of the compiler and defines all the required gadgets.

Security Notions. When the adversary has access to a device to perform side-channel and fault attacks, we assume that the adversary can run the device and probe up to d intermediate values. The first and simplest security definition, *d-probing security* requires that the observation of up to d intermediate values in a masked circuit does not reveal anything about the unmasked variables.

Definition 1. *A masked circuit* C *with* k *inputs* $(x_i^j)_{i\in[n]} \leftarrow \mathsf{Enc}(x^j)$ *and* $j \in [k]$ *is* d-probing secure *if for any set* $\mathcal{L}$ *of* d *probes in*

$$\mathsf{C}((x_i^0)_{i\in[n]}, (x_i^1)_{i\in[n]}, \ldots, (x_i^{k-1})_{i\in[n]}))$$

there is a simulator Sim *only having access to* $\mathsf{C}(\cdot)$ *without the* k *secrets* x^j *such that* $\mathcal{L} \equiv \mathsf{Sim}(\mathsf{C}(\cdot))$ *for any secret* $x^0, x^1, \ldots x^{k-1}$.

Note that Sim has only access to the circuit C but not to the secrets $x^0, x^1, \ldots x^{k-1}$. In other words, the probes $\mathcal{L}$ are independent of the unmasked values. A stronger security definition is the d-region-probing model that also takes the circuit size into account. In detail it allows d-probes in each gadget of the masked circuit.

Definition 2. *A masked circuit* C *with* k *inputs* $(x_i^j)_{i\in[n]} \leftarrow \mathsf{Enc}(x^j)$ *and* $j \in [k]$ *is* d-region-probing secure *if for any set* $\mathcal{L}$ *of* d *probes in each gadget of*

$$\mathsf{C}((x_i^0)_{i\in[n]}, (x_i^1)_{i\in[n]}, \ldots, (x_i^{k-1})_{i\in[n]}))$$

there is a simulator Sim *only having access to* $\mathsf{C}(\cdot)$ *without the* k *secrets* x^j *such that* $\mathcal{L} \equiv \mathsf{Sim}(\mathsf{C}(\cdot))$ *for any secret* $x^0, x^1, \ldots x^{k-1}$.

It turned out that probing security of two circuits does not always imply probing security of its composition. Since composition results are very useful and allow the construction of compilers that work on a gate-by-gate basis, stronger definitions were subsequently developed. In the following, we give some stronger security definitions, well suited to masked circuits (or gadgets). To simplify presentation, we consider only gadgets having a single output sharing $(y_i)_{i\in[n]}$. We refer to Cassiers and Standaert for discussion of gadgets with multiple outputs [CS20]. An important notion to achieve composability in the presence of probing attacks is the notion of *d-Non-Interference* (d-NI) and *d-Strong-Non-Interference* (d-SNI) [BBD+16]. Both definitions guarantee that the leakage of up to d probes is independent of the shared secret.

Definition 3 (d-NI [BBD+15,BBD+16]). *A gadget* G *with one output sharing is* d-non-interfering (d-NI) *if and only if every set of* $d' \leq d$ *internal probes can be (perfectly) simulated with at most* d' *shares of each input sharing.*

The stronger d-SNI notion requires to distinguish between intermediate and output probes and guarantees that only the number of intermediate probes affects the number of inputs required by the simulator, easing the compilation of circuits.

Definition 4 (d-SNI [BBD$^+$16]). *A gadget with one output sharing is* d-strong-non-interfering (d-SNI) *if and only if for every set I of d_1 internal probes and every set O of d_2 output probes such that $d_1 + d_2 \leq d$, the set of probes $I \cup O$ can be (perfectly) simulated with d_1 shares of each input sharing.*

Note PINI is another useful security notion for compositions in the threshold model. We omit this definition in the main body since it is vulnerable to horizontal attacks, and thus does not provide good properties for proofs in the region probing model. A detailed analysis is given in Section B.2.

3 Combined Security Model

As mentioned in the introduction, many countermeasures defending against fault attacks *or* probing attacks have been studied in the literature, but the task to protect against both attacks at the same time has only received more attention in the last few years. In this section we analyze the combined security of both fault resilience and probe resilience. To understand the influence of different kind of faults, we will model these faults as set of functions. An adversary with access to the class of faults $\mathcal{F}$ is able to change the value x to $\zeta(x)$ for $\zeta \in \mathcal{F}$ during the computation. For the sake of simplicity, we always assume that the identity function id is part of every class $\mathcal{F}$. A fault attack now applies several of these faults to different wires of $\mathsf{C}^R \colon \mathbb{F}^k \to \mathbb{F}^l$. More formally, if the wires of the circuit C^R are numbered by $1, \ldots, W$, a *fault attack* T is a tuple of functions $T = (\zeta_1, \ldots, \zeta_W)$ with $\zeta_i \in \mathcal{F}$ for all $i = 1, \ldots, W$ that describes how the value of each wire i is faulted. This means that such a wire i gets a value z_i from its output gate, but the following gate gets as input the faulted value $\zeta_i(z_i, \boldsymbol{u}_i)$, where ζ_i is the i^{th} function in $T = (\zeta_1, \ldots, \zeta_W)$ and $\boldsymbol{u}_i$ are the values already revealed by probes. We write $\mathsf{A}(\mathcal{F})$ to refer to the set of all possible fault attacks using the fault-functions $\mathcal{F}$. To simplify notation, we will often only use the ordering of the wires implicitly. If we tamper the circuit C^R with a fault attack $T \in \mathsf{A}(\mathcal{F})$ we write $T[\mathsf{C}^R]$. Due to physical constraints, a typical attacker cannot fault arbitrary many wires and is thus restricted (for example, [SFRES18] considers at most 3 faults and [RDB$^+$18] at most 8 faults). For a fault attack $T \in \mathsf{A}(\mathcal{F})$, we write $|T|$ for the number of non-identity faults used, i.e., $|T| = |\{i \in \{1, \ldots, W\} \mid \zeta_i \neq \mathrm{id}\}|$ with $T = (\zeta_1, \ldots, \zeta_W)$. In the following we often consider different types of fault sets. In the most general case, we use *wire independent faults* $\mathcal{F}^{ind} := \{\text{all functions } \zeta \colon \mathbb{F} \times \mathbb{F}^* \to \mathbb{F}\}$ to show that the attacker can fault arbitrarily. We stress here that our model implies that the faults performed on different wires are somewhat *independent*, as they each only consider a single wire. An often studied restriction are *additive faults* $\mathcal{F}^+ := \{\zeta : \zeta(x, \boldsymbol{u}) = x + a \text{ for all } a \in \mathbb{F}\}$ that fault the wires value by adding an

arbitrary value. We give a detailed discussion about the fault sets in the appendix (Sec. B.1).

Security Experiment We are now ready to give a formal description of the underlying security experiment where an attacker is able to perform combined attacks. Therefore, we adjusted the security game of Dhooge and Nikova [DN20b] to allow adaptive faults and region probes. Let $\mathsf{C}\colon \mathbb{F}^k \to \mathbb{F}^l$ be a circuit with wires $W = \{w_i\}_i$ that is split into *regions* $R_1, \ldots, R_r$ with wires $W_1, \ldots, W_r$. An (d, e)-*attacker* A with respect to a fault class $\mathcal{F}$ takes part in the following experiment:

- The experiment chooses $b \leftarrow\!\!\$\ \{0, 1\}$ uniformly at random
- A is given input C and outputs the following:
  - a fault-attack $T \in \mathsf{A}(\mathcal{F})$ with $|T| \leq e$
  - for each region R_j, a subset $W'_j \subseteq W_j$ of wires with $|W'_j| \leq d$
  - two possible circuit inputs $x_0, x_1 \in \mathbb{F}^k$
- The experiment runs $\widetilde{y_b} \leftarrow T[\mathsf{C}](x_b)$ and the wire values corresponding to $W' = \bigcup_{j=1}^{r} W'_j$ are given to A. The attacker outputs a bit b'.

We say that C is ϵ-secure if $\Pr[b = b'] = 1/2$ and $\Pr[\widetilde{y_b} \in \{\perp, y_b\}] \geq 1 - \epsilon$ for any (d, e)-*attacker* A, where $y_b \leftarrow \mathsf{C}(x_b)$ is the output of a non-faulted run of C on x_b. In other words, the circuit is ϵ-secure if it is information-theoretic secure against leakage and detects erroneous values with probability at least $1 - \epsilon$.

In this work, we assume leakage-free encoding and decoding gadgets as defined in Definition 1. As a consequence, it is sufficient to prove that the probes can be simulated with less than d values of each masked input, if the circuit is masked with a degree d masking. The existence of such gadgets is commonly used and goes back to Ishai, Sahai, and Wagner [ISW03].

3.1 Privacy

First, we give a property that guarantees the (S)NI property of a gadget even in the presence of fault attacks. We emphasize that this property only gives probing security in the presence of faults, but ignores fault security notions such as error preservation and detection. For the general fault resilience we refer to Section 3.2. Next, we extend the probing security by requiring that a gadget is d-(S)NI even if the adversary inserts faults into the computation.

Definition 5 (fault resilient SNI). *A gadget* G *is* d fault resilient (strong-) non-interfering (d-fr(S)NI) *with respect to* $\mathcal{F}$ *if* $T[\mathsf{G}]$ *is* d*-(S)NI for any fault attack* $T \in \mathsf{A}(\mathcal{F})$.

Note that an (S)NI gadget is not always (S)NI in the presence of faults. In Section B.6, we give a detailed discussion and some examples. fr(S)NI is a relative strong property and in some cases it might be sufficient (or needed) to slightly weaken this notion. To do so, we will consider the situation introduced before where (a) the number of faults are bounded and, furthermore, (b) we will treat

these faults additionally as probes. This is justified, e.g., in the context of constant fault functions (also called *stuck faults*) that might set a random value to a fixed value known by the adversary, as this can easily be seen to be strictly stronger than a probe on this random value. If a circuit is fault resilient under these restrictions, we say that the circuit is wfr(S)NI.

Definition 6 (weak fault resilient NI). *A gadget* G *is* d weak fault resilient non-interfering (d-wfrNI) *with respect to* $\mathcal{F}$ *if every set of* d' *probes in* $T[\mathsf{G}]$ *can be (perfectly) simulated with* $d' + |T|$ *shares of each input sharing for any fault attack* $T \in \mathsf{A}(\mathcal{F})$ *with* $|T| + d' \leq d$.

Definition 7 (weak fault resilient SNI). *A gadget* G *is* d weak fault resilient strong-non-interfering (d-wfrSNI) *with respect to* $\mathcal{F}$ *if every set of* d_1 *internal probes and* d_2 *output probes in* $T[\mathsf{G}]$ *can be (perfectly) simulated with* $d_1 + \epsilon_1$ *shares of each input sharing for any fault attack* $T \in \mathsf{A}(\mathcal{F})$ *with* ϵ_1 *internal faults and* ϵ_2 *output faults such that* $d_1 + d_2 + \epsilon_1 + \epsilon_2 \leq d$.

Note that this weaker notion does not imply that the faulted gadget $T[\mathsf{G}]$ is $(d - \epsilon) - (S)NI$ with $\epsilon = |T|$, as our d-wfr(S)NI definition gives ϵ more input values to the simulator for the simulation. This new security notions allows us to use the same composition results as the standard (S)NI gadgets even in the presence of faults. For example, the composition of two d-frSNI gadgets is easily seen to be d-frSNI again.

Theorem 1. *The composition of two d-frSNI (or d-wfrSNI) gadgets with respect to* $\mathcal{F}$ *is d-frSNI (or d-wfrSNI) with respect to* $\mathcal{F}$ *if* $\mathcal{F} \subseteq \mathcal{F}^{ind}$.

We write adaptive if the security still holds under the assumption that the adversary can choose the function with the knowledge of the probes (before). Theorem 1 implies that the composition of an arbitrary number of SNI gadgets is SNI again. This easily follows by the fact that composed SNI gadgets can be seen as SNI gadgets again, and we can compose step-by-step arbitrary many gadgets together. Theorem 1 is only an example composition for SNI. Next, we give a general proof that also implies this theorem[8] and shows that all d-(S)NI composition rules apply as well to the d-frSNI and d-wfrSNI.

Theorem 2. *The composition rules for (S)NI also apply to d-fr(S)NI and d-wfr(S)NI.*

Proof. We start with the proof for the stronger security notion. The faults $\mathcal{F} \subseteq \mathcal{F}^{ind}$ only allow independent faults on each wire, and this allows us to split the adversary in multiple adversaries that tamper each gadget independently. So let C be an arbitrary composed circuit only using d-fr(S)NI gadgets G_0, G_1, ..., G_m with respect to $\mathcal{F}$ and let T be any fault attack $T \in \mathsf{A}(\mathcal{F})$. Since we keep the proof as general as possible we just assume that C has some security properties (e.g. SNI) that follow from the (S)NI properties of G_0, G_1. Now we show that the property also holds for any T. Since we can split the circuit attack

[8] Alternatively, we give a straight forward proof of Theorem 1 in the appendix (Sec. B.7)

T to gadget-wise attacks $T_0, T_1, \ldots, T_m$ it holds that $T[\mathsf{C}]$ can be also described as the composition of the (independently) faulted gadgets $T_0[\mathsf{G}_0]$, $T_1[\mathsf{G}_1]$, $\ldots$, $T_m[\mathsf{G}_m]$. The fr(S)NI properties still guarantees that each faulted gadget $T_j[\mathsf{G}_j]$ has the same (S)NI property as its unfaulted version G_j. Hence, the faulted gadgets $T_0[\mathsf{G}_0]$, $T_1[\mathsf{G}_1]$, $\ldots$, $T_m[\mathsf{G}_m]$ have the same composition properties as the unfaulted (S)NI gadgets G_0, G_1, $\ldots$, G_m and the faulted circuit $T[\mathsf{C}]$ has the same (S)NI properties as the original one C. Note that this holds for any fault attack $T \in \mathsf{A}(\mathcal{F})$, and it follows that the same composition rules apply for fr(S)NI as for (S)NI. The proof for the weaker notion is similar, only the fault attack is limited and the faults are counted as probes. □

Remember that d-(S)NI gadgets are also d-probing secure as defined in Definition 1. Similarly to Theorem 2 it easily follows that any d-frSNI circuit C with respect to $\mathcal{F}$ is also d-probing secure for any fault attack $T \in \mathsf{A}(\mathcal{F})$. More formally, $T[\mathsf{C}]$ is d-probing secure for any $T \in \mathsf{A}(\mathcal{F})$. The probing security of d-wfrSNI circuits is slightly weaker. Since the number of allowed probes in d-wfrSNI circuits is reduced by the number of faults, a d-wfrSNI circuit C with respect to $\mathcal{F}$ is only $(d-e)$-probing secure for any fault attack $T \in \mathsf{A}(\mathcal{F})$ with $|T| = e \leq d$.

Privacy analyzes. Analysing the (S)NI property in itself is often tedious and to study the (w)f-(S)NI notion means we also need to consider all possible fault attacks. To avoid the combinatorial explosion, we present an additional property such that the classical (S)NI property directly implies the faulty one. This property is called *fault-invariance*, as the amount of information that a probe gives is independent of the faults. As the internal values z_i of a circuit only depend on the internal randomness R and the inputs $x_0, x_i, \ldots, x_{l-1}$ we can also write them as functions $z_i = f_i^R(x_0, x_1, \ldots x_{k-1})$.

Definition 8 (fault-invariance). *A circuit* C *is* fault-invariant *with respect to a fault set* $\mathcal{F}$ *if for any* $T \in \mathsf{A}(\mathcal{F})$*, any intermediate value* f *in* C^R *and the according value* $\tilde{f}$ *in* $T[\mathsf{C}^R]$ *there are* $\zeta, \zeta_0, \zeta_1, \ldots, \zeta_{k-1} \in \mathcal{F}$ *such that it holds*

$$\tilde{f}^R(x_0, x_1, \ldots x_{m-1}) = \zeta(f^R(\zeta_0(x_0), \zeta_1(x_1), \ldots, \zeta_{m-1}(x_{m-1})))$$

for any input $(x_0, x_1, \ldots, x_{m-1})$ *and randomness* R*.*

In other words, Definition 8 says that all faults in $\mathcal{F}$ applied to a gadget can be pushed to the input or the output of the gadget.

Gadgets that are (S)NI and also have this property are directly (S)NI in the presence of faults.

Corollary 1. *If a gadget is d-(S)NI and fault-invariant with respect to* $\mathcal{F} \subseteq \mathcal{F}^{ind}$*, the gadget is d-fr(S)NI with respect to* $\mathcal{F}$*.*

Proof. We will prove that we can take the classical leakage simulator of the non-faulted gadget and transform it according to the faults due to the fault invariance. Fix a gadget G and some probed values $(p_0, p_1, \ldots, p_{d-1})^T$. Due to

the d-(S)NI property there is a simulator $S(a_0, a_1, \dots)$ that perfectly simulates the leakage with some input values $(a_0, a_1, \dots)$. This means it holds that

$$S(a_0, a_1, \dots) = (S_0(a_0, a_1, \dots), S_1(a_0, a_1, \dots), \dots S_{d-1}(a_0, a_1, \dots))^T$$

has the same distribution as $(p_0, p_1, \dots, p_{d-1})^T$, where $S_i(a_0, a_1, \dots)$ is the projection of the output of S to the wire indexed by probe p_i. Further, let any $T \in \mathsf{A}(\mathcal{F})$ be a fault-attack and $(p'_0, p'_1, \dots, p'_{d-1})^T$ be the according probes on the same wires in $T[\mathsf{G}]$. Due to the fault-invariance we know that there exist functions $\zeta_{i,j}, \zeta'_i \in \mathcal{F}$ such that the values

$$S'(a_0, a_1, \dots) = \begin{pmatrix} \zeta'_0(S_0(\zeta_{0,0}(a_0), \zeta_{0,1}(a_1), \dots)) \\ \zeta'_1(S_1(\zeta_{1,0}(a_0), \zeta_{1,1}(a_1), \dots)), \\ \vdots \\ \zeta'_{p-1}(S_{p-1}(\zeta_{p-1,0}(a_0), \zeta_{p-1,1}(a_1), \dots)) \end{pmatrix}$$

have the same distribution as $(p'_0, p'_1, \dots, p'_{d-1})^T$. Hence, the simulator S' can simulate the faulted gadget and with the same inputs as the simulator S of the unfaulted (S)NI gadget. This proves that $T[\mathsf{G}]$ is also (S)NI for any $T \in \mathsf{A}(\mathcal{F})$. □

In the following, we show that the stronger definition with regard to the additive faultset $\mathcal{F}^+$ directly implies the weaker definition for the more general independent $\mathcal{F}^{ind}$, if we consider fault-invariant gadgets.

Corollary 2. *If a gadget is d-fr(S)NI and fault-invariant with respect to $\mathcal{F}^+$, it is adaptively d-wfr(S)NI with respect to $\mathcal{F}^{ind}$.*

Proof. Let C be an d-fr(S)NI and fault-invariant circuit with respect to $\mathcal{F}^+$. We give a reduction and prove that any simulator Sim for d-wfr(S)NI with respect to $\mathcal{F}^{ind}$ can be simulated by a simulator $\widetilde{\mathsf{Sim}}$ for d-fr(S)NI with respect to $\mathcal{F}^+$ if the according gadget is fault-invariant with respect to $\mathcal{F}^+$.

So let Sim simulate a fault attack $T \in \mathsf{A}(\mathcal{F}^{ind})$ with $|T| = s$ and $d - s$ probes. Now we show that Sim can be simulated with a Simulator $\widetilde{\mathsf{Sim}}$ with fault attack $\widetilde{T} \in \mathsf{A}(\mathcal{F}^+)$ with $|\widetilde{T}| = s$ and d probes. Here, we use the fact that $\widetilde{\mathsf{Sim}}$ can simulate d values, and we can also simulate the faulted values to transform all faults into additive faults. In detail, let v_1, v_2,... v_s the values faulted by the fault functions $\zeta_1, \zeta_2, \dots, \zeta_s \in \mathcal{F}^{ind}$ due to T and $v_{s+1}, v_{s+2}, \dots, v_d$ the $d - s$ values simulated by Sim in $T[\mathsf{C}]$. Now $\widetilde{\mathsf{Sim}}$ can simulate the according values $\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_d$ in the unfaulted C. Next, it computes for all fault functions ζ_j with $a_j = \zeta_j(v_j) - v_j$ and constructs the new additive fault function $\tilde{\zeta}_j(x) = x + a_j$. It follows that for all faults it holds $\zeta_j(v_j) = \tilde{\zeta}_j(v_j)$. Due to the invariance $\widetilde{\mathsf{Sim}}$ can move all additive faults to the inputs and outputs. In other words $\widetilde{\mathsf{Sim}}$ can compute how the additive faults affect the simulated probes $\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_d$, and can compute the according probes $v'_1, v'_2, \dots, v'_d$ in the circuit faulted with the fault functions $\tilde{\zeta}_j(x)$. Since the faults are the same as $\zeta_j(x)$, the values $v'_{s+1}, v'_{s+2}, \dots, v'_d$ and $v_{s+1}, v_{s+2}, \dots, v_d$ have the same distribution. This implies the claim of the corollary because it

shows that if we have $\widetilde{\mathsf{Sim}}$, we also have Sim. In other words if the gadget is d-fr(S)NI and fault-invariant with respect to $\mathcal{F}^+$, it is d-wfr(S)NI with respect to $\mathcal{F}^{ind}$.

Note that the simulator can choose the fault function with the knowledge of the probes. This also means that if the adversary chooses the function adaptively, it does not affect the simulator. Hence the adversary can be adaptive. □

3.2 Error Preservation and Detection

As described before, a general way to understand the countermeasures against passive and active attacks, can be viewed as *encodings*. For a concrete (randomized) encoding scheme $\mathsf{Enc}\colon \mathbb{F} \to \mathbb{F}^n$, a value $y \in \mathbb{F}^n$ is a *valid* encoding if there is an $x \in \mathbb{F}$ and some randomness such that $\mathsf{Enc}(x; r) = y$. Similar to the fr(S)NI property we are interested in a property that guarantees that errors can be detected even when we compose multiple gadgets. In our case, we want to guarantee that errors are detected by the fact that the resulting encodings are invalid. More concretely, if y and y' are valid encodings, we want to increase their Hamming distance, denoted by $\mathrm{d}(y, y')$. To argue about the behavior of a gadget in the presence of faults that can introduce computation errors, we need to guarantee that errors already present in the computation (a) stay present in the computation (to detect them) and (b) that these errors cannot accumulate over time to lead to a valid encoding of an *incorrect* value. To model this, we assume that the inputs $(x_i)_{i\in[n]}$ of our gadgets might already be faulted by a *fault vector* $(v_i)_{i\in[n]}$, i.e., the inputs will always be $(x_i)_{i\in[n]} + (v_i)_{i\in[n]}$, where $(x_i)_{i\in[n]}$ is a valid encoding. In a first approach, one might require that the input of an invalid encoding, i.e., one where $(v_i)_{i\in[n]} \neq 0$, always leads to an invalid output encoding. But this is a very strict requirement that is nearly impossible to fulfill if we consider an addition gadget: The attacker might add $(v_i)_{i\in[n]}$ to one of the inputs and then later add $-(v_i)_{i\in[n]}$ to the output. Clearly, this gives a valid encoding although the input was invalid. We thus also allow that our gadget on input $(x_i)_{i\in[n]} + (v_i)_{i\in[n]}$ with $(v_i)_{i\in[n]} \neq 0$ can produce a valid encoding of the *correct value* but this effect has to be independent of input $(x_i)_{i\in[n]}$.

Definition 9 (e-f-robust). *A gadget G with one output sharing and two input sharings is* e-fault-robust *with respect to $\mathcal{F}$, if for any valid encoding $(x_i^{(0)})_{i\in[n]}$ and $(x_i^{(1)})_{i\in[n]}$, the output $(y_i)_{i\in[n]} \leftarrow \mathsf{G}((x_i^{(0)})_{i\in[n]}, (x_i^{(1)})_{i\in[n]})$ is also valid. Further, it holds for any fault vectors $(v_i^{(0)})_{i\in[n]}$, $(v_i^{(1)})_{i\in[n]}$, and $T \in \mathsf{A}(\mathcal{F})$ with $(y_i)_{i\in[n]} + (w_i^{(1)})_{i\in[n]} + (w_i^{(2)})_{i\in[n]} \leftarrow T[\mathsf{G}]((x^{(0)} + v_i^{(0)})_{i\in[n]}, (x^{(1)} + v_i^{(1)})_{i\in[n]})$, that there are numbers t_1 and t_2 with $t_1 + t_2 \leq |T|$ such that*

(*i*) $\mathrm{weight}(w') \leq t_1$ *with* $(w'_i)_{i\in[n]} = (v_i^{(0)})_{i\in[n]} + (v_i^{(1)})_{i\in[n]} - (w_i^{(1)})_{i\in[n]}$;

(*ii*) *and $(w_i^{(2)})_{i\in[n]}$ is the zero vector or produced by the following random experiment: A polynomial $p_w \in \mathbb{F}[x]$ is chosen such that the coefficients x^{d+1}, x^{d+2}, …, x^{n-t_2} are drawn uniformly at random from $\mathbb{F}$. Then, $w_i^{(2)} = p_w(\alpha_i)$.*

Note that any valid encoding $(y_i)_{i\in[n]}$ and any fault vector $(w_i)_{i\in[n]}$ with $\text{weight}(w) \leq e$ cannot result in a valid encoding with $(y_i)_{i\in[n]} + (w_i)_{i\in[n]}$ in our setting, as we use $d+1+e$ shares of a polynomial of degree d. Hence, $\text{weight}((y_i)_{i\in[n]} - (y'_i)_{i\in[n]}) \geq e+1$ for all valid sharings y and y'. Next we give a useful composition result for e-f-robustness.

Theorem 3. *The composition of two e-fault-robust gadgets with respect to $\mathcal{F}$ is e-fault-robust with respect to $\mathcal{F}$ if $\mathcal{F} \subseteq \mathcal{F}^{ind}$.*

Proof. Let G and G' be two gadgets such that G is given inputs $(x_i^{(0)})_{i\in[n]}$ and $(x_i^{(1)})_{i\in[n]}$ and produces output $(y_i^{(0)})_{i\in[n]}$. Furthermore, let G' have the inputs $(y_i^{(0)})_{i\in[n]}$ (i.e., the output of G) and $(y_i^{(1)})_{i\in[n]}$ and output $(z_i)_{i\in[n]}$. Let $\tilde{\mathsf{G}}$ be the complete construction and $v_{x^{(0)}}$, $v_{x^{(1)}}$, $v_{y^{(0)}}$, $v_{y^{(1)}}$, v_z be the fault vectors.

Fix any $\tilde{T} \in \mathsf{A}(\mathcal{F})$. Due to the independence of these faults, we can split them into two parts T and T', where T corresponds to the faults introduced in G and T' corresponds to the faults introduced in G'.

As G is e-fault-robust, its output fault vector $v_{y^{(0)}}$ is of the form $v_{y^{(0)}} = v_{x^{(0)}} + v_{x^{(1)}} + w^{(1)} + w^{(2)}$ where $\text{weight}(w^{(1)}) \leq t_1$ and $w^{(2)}$ is the zero vector or drawn randomly with highest coefficient x^{n-t_2} for some numbers t_1 and t_2 with $t_1 + t_2 \leq |T|$. Furthermore, as G' is e-fault-robust, its output fault vector v_z is of the form $v_z = v_{y^{(0)}} + v_{y^{(1)}} + w'^{(1)} + w'^{(2)}$ where $\text{weight}(w'^{(1)}) \leq t'_1$ and $w'^{(2)}$ is drawn randomly with highest coefficient $x^{n-t'_2}$ for some numbers t'_1 and t'_2 with $t'_1 + t'_2 \leq |T'|$. Hence, $v_z = v_{x^{(0)}} + v_{x^{(1)}} + w^{(1)} + w^{(2)} + v_{y^{(1)}} + w'^{(1)} + w'^{(2)}$, where $\text{weight}(w^{(1)} + w'^{(1)}) \leq t_1 + t'_1$ and $w^{(2)} + w'^{(2)}$ corresponds to a random polynomial with highest coefficient $x^{n-\min\{t_2,t'_2\}}$. □

The definition of e-fault-robustness can be simplified for non-adaptive attackers: We can then guarantee that either $w^{(1)}$ or $w^{(2)}$ are zero. The proof of composability is similar, but uses the fact that the sum of a random polynomial and a deterministic, *independent* polynomial is again a random polynomial.

The notion of e-fault-robustness now directly allows us to give a bound on the probability that a valid encoding of an invalid value is generated by a circuit.

Theorem 4. *If a circuit is e-fault-robust, the probability that $s \leq e$ faults can produce a valid encoding of an invalid value is at most q^{s-e-1} in the case of non-adaptive attackers and $q^{s-e} \cdot (d+e+1)^{t_1}$ for all $t_1 \leq s$ in the case of adaptive attackers.*

Proof. Let $w^{(1)}$ and $w^{(2)}$ be as in the definition of e-fault-robustness and let $p^{(1)}$ and $p^{(2)}$ be the corresponding polynomials. Lemma 2 implies that the fault polynomial $p^{(1)}$ has degree at least $n - t_1$. Hence, if $p^{(2)}$ is identical to zero, the attacker can not produce a valid encoding of an invalid value. Furthermore, if $p^{(1)}$ is identical to zero, the sharing is valid if and only if all coefficients of x^{d+1}, x^{d+2}, …, x^{n-s} of $p^{(1)}$ are zero, which happens with probability q^{s-e-1}.

If $p^{(2)}$ is not identical to zero, the polynomial $p = p^{(1)} + p^{(2)}$ corresponding to $w = w^{(1)} + w^{(2)}$ needs to have degree at most d. As $p^{(2)}$ has degree at most $n - t_2$,

this is only possible if $n - t_1 \leq n - t_2$, i.e., if $t_1 \geq t_2$ holds. We will now show that the number of different polynomials $p^{(1)}$ and $p^{(2)}$ such that p has degree at most d is very small. Clearly, the number of different polynomials possible for $p^{(2)}$ is q^{e-t_2} due to the fact that $p^{(2)}$ is randomly generated. Furthermore, the number of different polynomials possible for $p^{(1)}$ is $\binom{n}{t_1} \cdot q^{t_1} \leq n^{t_1} \cdot q^{t_1}$. Hence, the probability that there is a vector $w^{(1)}$ matching to the random vector $w^{(2)}$ is at most

$$\frac{q^{t_1}}{q^{e-t_2}} \cdot n^{t_1} = q^{t_1+t_2-e} \cdot n^{t_1}.$$

Hence, the probability that $s \leq e$ faults can produce a valid encoding of an invalid value is at most $q^{s-e} \cdot (d+e+1)^{t_1}$. □

3.3 Combined Security

Equipped with our new notions of d-fr(S)NI and e-fault-robustness, it is easy to see that the combination of these notions directly implies security against (d, e)-attackers.

Theorem 5. *If the circuit* C *is d-frSNI (or d-wfr(S)NI) and e-fault-robust, it is* q^{s-e-1}*-secure against any non-adaptive* (d, s)*-attacker (or* $(d-s, s)$*-attacker) with* $s \leq e$*. Furthermore, it is* $q^{s-e} \cdot (d+e+1)^{t_1}$*-secure against any adaptive* (d, s)*-attacker (or* $(d-s, s)$*-attacker) with* $t_1 \leq s \leq e$*.*

Proof. The perfect security with regard to the leakage (i.e., that the attacker is not able to determine the challenge bit b) directly follows from the fact that the circuit is d-SNI even in the presence of at most e faults. Furthermore, Theorem 4 directly implies that the probability that a valid encoding of an invalid value is ever output is at most q^{s-e-1} in the non-adaptive case and $q^{s-e} \cdot (d+e+1)^{t_1}$ in the adaptive case. □

4 Compiler

This section gives a compiler transforming an unprotected circuit into a fault and probe resilient circuit using a polynomial sharing with an optimal number of shares. As described in the previous section, the sensitive data v is masked with a $d+1$-out-of-n polynomial sharing. Therefore, a polynomial f with degree d such that $f(0) = v$ is generated, and the shares are given by $f(\alpha_i)$ for pairwise different non-zero inputs $\alpha_0, \ldots, \alpha_{n-1}$. The main goal of the compiler is now to take an ordinary circuit and transform it into a circuit operating on shares such that d probes and e faults do not reveal any sensitive data, i.e., into a circuit that is ϵ-secure against (d, e)-attackers. As usual, this is done by replacing gates of the original circuit with gadgets. For security reasons, refresh gadgets are also inserted that guarantee that intermediate sharings become independent. Refresh gadgets take as input an encoded secret and output a re-randomized encoding

of the same secret. This procedure reduces the dependencies that might occur when one value is used as input for multiple gadgets.

In the following, let $n = d + e + 1$. Unary gates (such as addition or multiplication with a constant) taking only a single input can be handled easily, as these operations are linear and can thus be computed locally on the shares. We thus only need to focus on binary gates, i.e., addition and multiplication gates that take two inputs. Let us consider two $d+1$-out-of-n sharings $(F_i)_{i\in[n]}$ and $(G_i)_{i\in[n]}$ of secrets f_0, respectively g_0. Similar to the unary gates, Algorithm 1 computes a *share-wise* addition such that its output $(Q_i)_{i\in[n]}$ represents the addition $f_0 + g_0$. This algorithm is a gadget computing an addition since it outputs a $d+1$-out-of-n sharing again, as the addition of two degree d polynomials also has degree d.

Algorithm 1 (n, d)-**SWAdd** for $n = d + \epsilon + 1$

Input: Shares of f_0 as $(F_i)_{i\in[n]}$ and g_0 as $(G_i)_{i\in[n]}$ with degree d
Output: Shares of $f_0 + g_0$ as $(Q_i)_{i\in[n]}$ with degree d.

1: **for** $i = 0$ **to** $n - 1$ **do**
2: $\quad Q_i \leftarrow F_i + G_i$
3: **return** $(Q_i)_{i\in[n]}$

Due to our attack model, all faults added to the input of any such share-wise (linear) gadget can also be added to the output without any change in the computation. Furthermore, as $n \geq d$, these gadgets are secure against d probes, as d values of a $d + 1$-out-of-n sharing do not reveal any information. We can thus state the following fact.

Theorem 6. *Share-wise affine gadgets are d-frNI with respect to $\mathcal{F}^+$ (or d-wfrNI with respect to $\mathcal{F}^{ind}$) and e-fault-robust with respect to $\mathcal{F}^{ind}$.*

Proof. The e-fault-robustness immediately follows from Definition 9(i). Next, we prove the frNI property. It is sufficient to show that a share-wise linear gadget is NI and fault invariant with respect to $\mathcal{F}^+$ because Corollary 1 shows that this implies d-frNI with respect to $\mathcal{F}^+$. Further, Corollary 2 shows that this also implies d-wfrNI with respect to $\mathcal{F}^{ind}$. Hence, it remains to prove that a share-wise linear gadget is (i) NI and (ii) fault invariant with respect to $\mathcal{F}^+$.

(i) It is well known that any share-wise gadget is NI, as all output (and intermediate) values only depend on input shares with the same index. For example, the share-wise addition with $c_i \leftarrow a_i + b_i$ only depends on the i^{th} share a_i and b_i for any $i \in [n]$. It follows that any d probes can be simulated by at most d shares of each input sharing because we never need more than one share of each input sharing for each probe. This implies the NI property.
(ii) The shift invariance w.r.t $\mathcal{F}^+$ follows from the linearity. Since an additive fault ζ only adds a constant value x on a wire, we can move the addition

x to the output or input due to the linearity. For example it holds for the share-wise addition $c_i \leftarrow a_i + b_i$ that $(a_i + b_i) + x = a_i + (b_i + x) = (a_i + x) + b_i$. In the first term, the fault is moved to the output $\zeta(c_i) = c_i + x$ and in the second two terms it is moved to the inputs $\zeta(b_i)$, and $\zeta(a_i)$, respectively. This is the property required for fault invariance. We give a more general proof in the appendix (Theorem 17, Sec. A.2).

With (i) and (ii) we can conclude the proof. □

As usual, the remaining gate, the binary multiplication gate is the most complicated gate, as it does not behave linearly. Nevertheless, Algorithm 2 computes a share-wise multiplication such that its output $(Q_i)_{i\in[n]}$ represents the multiplication $f_0 \cdot g_0$. Unfortunately, the multiplication of two polynomials of degree d results in a polynomial of degree $2d$. Therefore, Algorithm 2 outputs a $2d$ degree polynomial. For $n < 2d$, this means that the shares can not represent this polynomial properly and we thus lose the information about the shared value. Furthermore, even if $n > 2d$, the next multiplication gadget in the circuit could possibly lead to a polynomial of degree $4d$ and so on. Hence, Algorithm 2 can not be used as a multiplication gadget alone and further work is required. The classical approach due to Ben-Or, Goldwasser, and Wigderson [BGW88] performs a *degree reduction* to reduce the polynomial to degree d after the share-wise multiplication to prevent the exponential blowup of the degrees. Nevertheless, the polynomial of degree $2d$ needs to be stored and their approach thus requires at least $2d$ shares.

Algorithm 2 (n, d)-**SWMult** for $n = d + \epsilon + 1$

Input: Shares of f_0 as $(F_i)_{i\in[n]}$ and g_0 as $(G_i)_{i\in[n]}$ with degree d
Output: Shares of $f_0 g_0$ as $(Q'_i)_{i\in[n]}$ with degree $2d$.

1: **for** $i = 0$ **to** $n - 1$ **do**
2: $\quad Q'_i \leftarrow F_i \cdot G_i$
3: **return** $(Q'_i)_{i\in[n]}$

To construct a multiplication gadget, the state-of-the-art gadget due to Seker, Fernandez-Rubio, Eisenbarth, and Steinwandt [SFRES18] also follows the general approach of Ben-Or, Goldwasser, and Wigderson: They first apply the share-wise multiplication (Alg. 2) to compute sharing of degree $2d$ and then reduce the degree back to d afterwards such that subsequent operations can be performed. However, the degree reduction is relatively expensive and complex. We will discuss this strategy in more depth in the following section, Section 4.1. But due to the need to store a polynomial of degree $2d$, their approach can not need less than $2d$ shares. Furthermore, they also need an additional e shares to handle faults. Our solution circumvents the need for this large number of shares. Section 4.2 presents a new compiler that needs at most $d + e + 1$ shares.

4.1 Fixed State-of-the-Art

Here, we give a more thorough explanation of the binary multiplication gadget construction due to Seker, Fernandez-Rubio, Eisenbarth, and Steinwandt [SFRES18]. We note here that this gadget can only handle *additive* faults.

Figure 1b illustrates the high-level idea of the gadget. It first performs the share-wise multiplication which outputs sharing $(Q'_i)_{i\in[n]}$ of degree $2d$ that encodes the product of the secrets of $(F_i)_{i\in[n]}$ and $(G_i)_{i\in[n]}$ and then reduces the degree of $(Q'_i)_{i\in[n]}$ such that $(Q_i)_{i\in[n]}$ carries the same secret as $(Q'_i)_{i\in[n]}$ but has degree d. This construction was proven to be d-SNI secure. As mentioned above, this multiplication leads to the intermediate $2d$ degree sharing $(Q'_i)_{i\in[n]}$ that requires $n = 2d + e + 1$ shares. To handle faults, the following idea is used: Any attacker that can add e additive faults to the shares adds a polynomial of degree at least $n - e = d + 1$ to the sharing. As a valid sharing has degree at most d, this means that the higher-order monomial $d + 1$ is set to a non-zero value iff a fault was added. These higher-order monomials are kept unchanged by share-wise additions and by careful design, are also kept unchanged with probability at least $1 - (1/q)$ in the multiplication gadget (or, more generally $1 - q^{-e+s-1}$ for $s \leq e$ faults). Unfortunately, their refresh gadget is not (S)NI, as shown in Sec. B.5, and an alternative refresh is required. We remark that such a refresh gadget can be seen as multiplication with a fresh sharing of the value 1. It is thus sufficient to focus on addition and multiplication gadgets here. We refer to [BBD+16] for more detailed explanations. The result by Seker, Fernandez-Rubio, Eisenbarth, and Steinwandt [SFRES18] can thus be summed up via the following informal theorem:

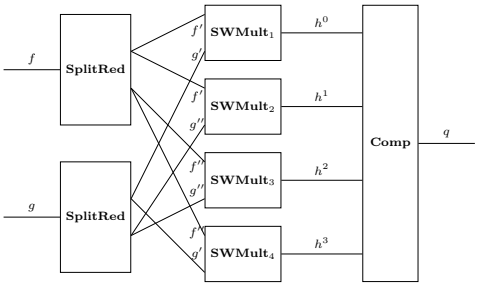
Theorem 7 (Fixed SotA). *For any $d, e \in \mathbb{N}$ there is a compiler that is given an arithmetic circuit $\mathcal{C}$ over $\mathbb{F}$ and outputs an arithmetic circuit $\mathcal{C}'$ on $\mathbb{F}^n$ where $n = 2d + e + 1$ with*

- *$\mathcal{C}'$ has the same functionality as $\mathcal{C}$*
- *$\mathcal{C}'$ is d-probing-secure,*
- *C' is e-fault-robust with respect to $\mathcal{F}^+$.*

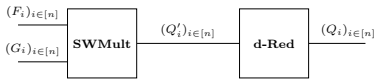
The proof follows from [SFRES18] with the multiplication use as SNI refresh, and the compiler presented in [BBD+16]. The idea is that the multiplication and refresh gadgets are error preserving and d-SNI and the addition gadget is error preserving and d-NI. Applying the compiler of [BBD+16] results in a d-probing secure circuit with error preserving gadgets and, consequently, the compiler outputs an error preserving circuit that is information-theoretic secure against d probes.

4.2 laOla Compiler

We improve our compiler in two steps. We first improve the (fr)SNI refresh, and we give a new multiplication gadget to reduce the number of shares from $2d+e+1$ to $d + e + 1$.



(a) Structure of the n, d-Multiplication defined in Algorithm 6



(b) Structure of the n, d-Multiplication used in [SFRES18].

Fig. 1: Our n, d-Multiplication and the multiplication in [SFRES18]

Refresh. We construct a new SNI refresh gadget only using d^2 random values. Therefore, we transform this problem to a gadget that generates a zero encoding. Assuming a "secure" zero encoding $(e_i)_{i\in[n]} \leftarrow \mathsf{Enc}(0)$ it is easy to see that the refreshed output $(y_i)_{i\in[n]}$ of a sharing $(x_i)_{i\in[n]}$ with $y_i = x_i + e_i$ is SNI. Any probe y_i is uniform random, and only the (additional) internal probe e_i requires the input x_i for a perfect simulation. However, we still ignored the internal probes in $(e_i)_{i\in[n]} \leftarrow \mathsf{Enc}(0)$ to generate the zero encoding. **sZEnc** depicted in Algorithm 4 is such a gadget that is SNI secure even in the presence of internal probes and faults (Theorem 11).

Further, we can show that the resulting refresh gadget even gives region probing security (Theorem 12). Using our new refresh gadget we can improve the compiler of [SFRES18]. The improved compiler (Imp. SotA, listed in Table 3) requires less randomness and guarantees higher security due to region probing security.

Multiplication. In order to avoid the dependency on $2d$, a trivial (insecure) approach is to *switch* the order of operations: I.e., we first reduce the degree of the input shares $(G_i)_{i\in[n]}$, $(F_i)_{i\in[n]}$ to sharings $(G'_i)_{i\in[n]}$, $(F'_i)_{i\in[n]}$ of degree $d/2$. The output of the share-wise multiplication $\mathbf{SWMult}((G'_i)_{i\in[n]}, (F'_i)_{i\in[n]})$ then has the right degree d again. However, it is easy to see that $(G'_i)_{i\in[n]}$, $(F'_i)_{i\in[n]}$ would reveal their secrets with $d/2 + 1 \leq d$ probes. Hence, this approach does not guarantee privacy against d probes.

While a naive implementation of this idea is insecure, our new construction depicted in Figure 1a still guarantees security. Instead of performing a simple degree reduction, our gadget **SplitRed** *simultaneously* constructs an *additive* sharing of the polynomials. Hence, an application of **SplitRed** on an input sharing described by the polynomial f produces *two* polynomials f' and f'' both of degree d such that the polynomial $f' + f''$ has only degree $d/2$. In other words, we produce two polynomials f' and f'' where the coefficients of the monomials $x^0, x^1, \ldots x^{d/2}$ are additive sharings of the corresponding coefficients of f and the coefficients of the monomials $x^{d/2+1}, \ldots, x^d$ are additive sharings of the all-

zero vector. We compute g' and g'' similarly such that each polynomial f', f'', g', and g'' considered individually is still a polynomial of degree d. Now we can apply the share-wise multiplication four times to compute $f' \cdot g'$, $f' \cdot g''$, $f'' \cdot g'$, and $f'' \cdot g''$ and sum all four outputs with our gadget **Comp**. It follows that the output $(H_i)_{i\in[n]}$ describes a polynomial h of degree d again because $f' \cdot g' + f' \cdot g'' + f'' \cdot g' + f'' \cdot g'' = (f' + f'') \cdot (g' + g'')$ is again a polynomial of degree d. The formal description of the algorithm is given in Section 6 and Section A provides a detailed security analysis against probes and faults.

Compiler The multiplication, and refresh, together with the share-wise addition, lead to a compiler using only $n = d + e + 1$ shares against additive faults.

Theorem 8 (laOla (additive)). *For any $d, e \in \mathbb{N}$ there is a circuit compiler that is given an arithmetic circuit $\mathcal{C}$ over $\mathbb{F}$ and outputs an arithmetic circuit $\mathcal{C}'$ over $\mathbb{F}^n$ where $n = d + e + 1$ with*

- *$\mathcal{C}'$ has the same functionality as $\mathcal{C}$,*
- *$T[\mathcal{C}']$ is probing secure for any $T \in \mathsf{A}(\mathcal{F}^+)$ and*
  - *(i) up to d probes in $T[\mathcal{C}']$ (threshold probing security), or*
  - *(ii) up to $d/2$ probes in every gadget of $T[\mathcal{C}']$ (region probing security).*
- *C' is e-fault-robust with respect to $\mathcal{F}^+$.*

Furthermore, we also show that our approach can handle more general faults, although this comes at the cost of needing more shares.

Theorem 9 (laOla-Compiler (general)). *For any $d, e \in \mathbb{N}$ there is a circuit compiler that is given an arithmetic circuit $\mathcal{C}$ over $\mathbb{F}$ and outputs an arithmetic circuit $\mathcal{C}'$ over $\mathbb{F}^n$ where $n = d + e + 1$ with*

- *$\mathcal{C}'$ has the same functionality as $\mathcal{C}$,*
- *$T[\mathcal{C}']$ is probing secure for any $T \in \mathsf{A}(\mathcal{F}^+)$ with $|T| \leq e$ and*
  - *(i) up to $d - e$ probes in $T[\mathcal{C}']$ (threshold probing security), or*
  - *(ii) with up to $d/2$ probes in every gadget of $T[\mathcal{C}']$ when the faults are counted as probes (region probing security).*
- *$T[\mathcal{C}']$ is $d - e$ probing secure for any $T \in \mathsf{A}(\mathcal{F}^{ind})$ with $|T| \leq e$,*
- *C' is e-fault-robust with respect to $\mathcal{F}^{ind}$.*

In both cases, Theorem 5 implies q^{s-e-1}-security against (d, s)-attackers.

Next, we will prove that our compiler is optimal for affine circuits. Since the addition gadget is share-wise, and cannot be further optimized for any compiler[9], we only show that the number of shares $n = d + e + 1$ is optimal if one wants to protect against d probes and e faults. If one assumes that the values are polynomially masked, this is relatively easy to see. To protect against d probes, the underlying polynomial needs to have degree at least d. If we use strictly less

[9] Assume an addition gadget where both input sharings and the output sharing have n shares. In total, we have $2n$ input shares and n output shares. It follows that at least one input share does not effect the output if we have only $n - 1$ (xor-)operations with two input wires. Hence, the gadget needs at least n operations.

Table 3: A concrete comparison with the compiler [SFRES18] fixed in this work, the constuction in [DN19], and our new compiler.

Compiler	# Shares	Randomness Cost		Comb. Sec. in the Reg. Prob. Model	Opt. for affine Circuits
		Mult. Gadget	SNI Refresh		
[DN19]	$d+e+1$	$\Theta(n^3)$	$\Theta(n^3)$	✖	✔
Fixed SotA [SFRES18]	$2d+e+1$	$2d^2+d(e+1)$	$2d^2+d(e+1)$	✖	✖
Imp. SotA[This Work]	$2d+e+1$	$2d^2+d(e+1)$	d^2	(✔)	✖
laOla [This Work]	$d+e+1$	$3d^2+2d(e+1)$	d^2	✔	✔

than $d+e+1$ shares, an attacker can fault e of these shares, which corresponds to adding a polynomial of degree strictly less than $d+e+1-e = d+1$ to the sharing. Hence, the attacker can modify the valid sharing, described by a polynomial of degree d, by adding a polynomial of degree d and thus obtain a *valid* sharing of a different values. Hence, there is no way to detect these modifications and fault-resilience is thus not possible.

One might now wonder whether this is an artifact of the polynomial sharing or whether it is inherently impossible to use a lower number of shares. We will show that the latter is true.

Theorem 10. *The number of shares n of any sharing procedure that protects against d probes and e faults is at least $n \geq d+e+1$.*

This follows from [Lin98] (Theorem 5.2.1), and we give a detailed proof in the appendix Sec. B.4.

5 Refresh Gadget

To construct a refresh gadget with input sharing $(x_i)_{i\in[n]}$ it is sufficient to generate a zero encoding $(e_i)_{i\in[n]}$ and output its sum $(y_i)_{i\in[n]} = (x_i)_{i\in[n]} + (e_i)_{i\in[n]}$. In this section we give two different gadgets to generate zero encodings. The first one is sufficient to inject randomness in our multiplication gadget. The latter gadget uses the weaker one and results in a d-SNI refresh gadget. Further we show that the gadget is even stronger, and we can use it to transform our d-probing secure circuit into a $d/2$-region-probing secure circuit.

***ZEnc** Gadget.* The gadget **ZEnc** depicted in Algorithm 3 generates a random zero encoding. We use the polynomial representation to describe the high level idea of the gadget. Since g is a random polynomial with $g(0) = 0$, it holds $g(x) = \sum_{i=1}^{d} r_j x^j$ with $r_1, r_2, \ldots r_d \in \mathbb{F}$. Our gadget generates such polynomials by choosing each r_i uniform at random and outputs g. More precisely, we use the polynomial masking where each polynomial is described by n points $g(\alpha_0), g(\alpha_1), \ldots g(\alpha_{n-1})$. Therefore, the algorithm does not compute $g(x)$ but

each $g(\alpha_i) = \sum_{i=1}^{d} r_j \alpha_i$, separately. Hence, the final output of Algorithm 3 represent an encoding of zero with $(g_i)_{i\in[n]} := (g(\alpha_i))_{i\in[n]}$.

Algorithm 3 $\mathbf{ZEnc}_n^d$

Output: A randomized (n, d)-Encoding of zero $(g_i)_{i\in[n]}$.

1: **for** $j = 1$ **to** d **do**
2: $\quad r_j \leftarrow\$ \mathbb{F}$
3: $\quad$ **for** $i = 0$ **to** $n - 1$ **do**
4: $\quad\quad g_{i+1} \leftarrow g_i \oplus r_j \alpha_i^j$
5: **return** $(g_i)_{i\in[n]}$

In the appendix (Sec. B.5), we show that this encoding does not suffice for an SNI refresh. However, next we show how to use **ZEnc** to construct an SNI secure refresh.

sZEnc *Gadget.* The gadget **sZEnc** depicted in Algorithm 4 generates a zero encoding because the sum of zero encodings is a zero encoding again.

Algorithm 4 $\mathbf{sZEnc}_n^d$

Output: A randomized (n, d)-Encoding of zero $(g_i)_{i\in[n]}$.

1: **for** $j = 0$ **to** d **do**
2: $\quad (g_i)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^d$
3: $\quad (y_i)_{i\in[n]} \leftarrow (y_i)_{i\in[n]} + (g_i)_{i\in[n]}$
4: **return** $(y_i)_{i\in[n]}$

Lemma 3 (Probing security). *For any set P with $d' \leq d$ probes it holds for $(e_i)_{i\in[n]} \leftarrow \boldsymbol{sZEnc}_n^d$: There is a sub set $A \subset [n]$ with $|A| = n - d'$ such that*

(i) $(e_i)_{i\in A}$ are still $(d-d')$-wise independent, independent from P and $(e_i)_{i\in[n]\setminus A}$,
(ii) P can be perfectly simulated with $(e_i)_{i\in[n]\setminus A}$

The proof is given in the appendix (Lemma 4, Sec. A.1). It is easy to see that a gadget with input sharing $(x_i)_{i\in[n]}$ and output sharing $(x_i + e_i)_{i\in[n]}$ with $(e_i)_{i\in[n]} \leftarrow \mathbf{sZEnc}_n^d$ is an SNI refresh:

Theorem 11 (Refresh). *The gadget G'_{G} (Alg. 5) with identity G is a d-frSNI w.r.t. $\mathcal{F}^+$ (or d-wfrSNI w.r.t. $\mathcal{F}^{ind}$) and e-fault-robust w.r.t. $\mathcal{F}^{ind}$ refresh gadget*

Proof. In the appendix (Sec. A) we give a detailed proof for SNI (Theorem 14, Sec. A.1) and fault-invariance w.r.t. $\mathcal{F}^+$ (Theorem 17, Sec. A.2). With Theorem 1 we get frSNI, and with Theorem 2 follows wfrSNI. Fault-robustness follows with Theorem 20, Sec. A.3. □

Algorithm 5 G'_{G} with $n \geq d + e + 1$

Input: The same input sharings as G. E.g., $(x_i)_{i\in[n]}$ and $(x'_i)_{i\in[n]}$, or only $(x_i)_{i\in[n]}$.
Output: A randomized output of $\mathsf{G}((x_i)_{i\in[n]}, (x'_i)_{i\in[n]})$ (or $\mathsf{G}((x_i)_{i\in[n]})$).

1: $(e_i)_{i\in[n]} \leftarrow \mathbf{sZEnc}_n^d$
2: $(y'_i)_{i\in[n]} \leftarrow \mathsf{G}((x_i)_{i\in[n]}, (x'_i)_{i\in[n]})$ (or $(y'_i)_{i\in[n]} \leftarrow \mathsf{G}((x_i)_{i\in[n]})$)
3: $(y_i)_{i\in[n]} \leftarrow (y_i)_{i\in[n]} + (e_i)_{i\in[n]}$
4: **return** $(y_i)_{i\in[n]}$

However, this refresh gadget is even more secure. Next, we show how to construct a region-probing secure compiler with the gadget depicted in Algorithm 5.

Theorem 12. *A d probing secure composition with d-NI and d-SNI secure gadgets G_i is $d/2$ region probing secure if each gadget is transformed into G'_{G} (Alg. 5), and outputs refreshed sharings.*

It immediately follows from Lemma 3, and the detailed proof is given in the appendix (Theorem 15, Sec. A.1). Assuming a (w)frSNI and error preserving multiplication, this theorem directly implies both Theorem 8 and Theorem 9. Using Theorem 5 then implies q^{s-e-1}-security against (d, s)-attackers.

Next, we give our frSNI and error preserving multiplication gadget only using $n = d + e + 1$ shares.

6 Multiplication Gadget

In this section we introduce our new improved gadget which securely performs a masked multiplication on a polynomial sharing with just $n = d + e + 1$ shares, whereas the state-of-the-art requires $2d + e + 1$ shares.

6.1 Concept and Overview

In the following we formally introduce the new multiplication gadget depicted in Figure 1a, its formal description is given in Algorithm 6.

For a better intuition of the gadget **Mult** with inputs $(F_i)_{i\in[n]}$ and $(G_i)_{i\in[n]}$, we use the polynomial representation f and g such that $f(\alpha_i) = F_i$ and $g(\alpha_i) = G_i$. As depicted in Figure 1a, the gadget **SplitRed** first splits the inputs g and f with secrets $f(0) = f_0$ and $g(0) = g_0$ into polynomials f', f'', g', and g'' such that each polynomial is uniform random with degree d but $f' + f'$ and $g' + g''$ have degree $\frac{d}{2}$ each and, furthermore, $f'(0) + f'(0) = f_0$ and $g'(0) + g'(0) = g_0$. This allows to avoid the intermediate polynomials that require $2d$ shares. Furthermore, we can use $\mathbf{SWMult}_i$ to compute the four polynomials $h^0(x) = f'(x)g'(x)$, $h^1(x) = f'(x)g''(x)$, $h^2(x) = f''(x)g'(x)$, and $h^3(x) = f''(x)g''(x)$. The last gadget **Comp** refreshes the polynomials and sums them up into

$$f'(x)g'(x) + f'(x)g''(x) + f''(x)g'(x) + f''(x)g''(x).$$

The sum results in a polynomial $q(x)$ with $(Q_i)_{i\in[n]} = q(\alpha_i)$ which encodes the correct value $q(0) = f_0 \cdot g_0$, as

$$\begin{aligned} q(0) &= f'(0)\cdot g'(0) + f'(0)\cdot g''(0) + f''(0)\cdot g'(0) + f''(0)\cdot g''(0) \\ &= (f'(0) + f''(0))\cdot(g'(0) + g''(0)) = g(0)\cdot f(0). \end{aligned}$$

The next sections introduce the sub-gadgets **SplitRed** (Sec. 6.2), **SWMult**, and **Comp** (Sec.6.3) needed for our multiplication gadget.

Algorithm 6 $(n, d) - \mathbf{Mult}$

Input: Shares of f_0 as $(F_i)_{0\le i<n}$ and shares of g_0 as $(G_i)_{0\le i<n}$.
Output: Shares of $f_0 g_0$ as $(\mathbf{Q}_i)_{0\le i<n}$.

1: $\left((F'_i)_{i\in[n]}, (F''_i)_{i\in[n]}\right) \leftarrow \mathbf{SplitRed}((F_i)_{i\in[n]})$
2: $\left((G'_i)_{i\in[n]}, (G''_i)_{i\in[n]}\right) \leftarrow \mathbf{SplitRed}((G_i)_{i\in[n]})$
3: $(H_i^0)_{i\in[n]} \leftarrow \mathbf{SWMult}((F'_i)_{i\in[n]}, (G'_i)_{i\in[n]})$
4: $(H_i^1)_{i\in[n]} \leftarrow \mathbf{SWMult}((F'_i)_{i\in[n]}, (G''_i)_{i\in[n]})$
5: $(H_i^2)_{i\in[n]} \leftarrow \mathbf{SWMult}((F''_i)_{i\in[n]}, (G'_i)_{i\in[n]})$
6: $(H_i^3)_{i\in[n]} \leftarrow \mathbf{SWMult}((F''_i)_{i\in[n]}, (G''_i)_{i\in[n]})$
7: $(Q_i)_{i\in[n]} \leftarrow \mathbf{Comp}((H_i^0)_{i\in[n]}, (H_i^1)_{i\in[n]}, (H_i^2)_{i\in[n]}, (H_i^3)_{i\in[n]})$
8: **return** $(Q_i)_{i\in[n]}$

6.2 SplitRedGadget

The general idea of **SplitRed**-LAOLA is best understood in the polynomial representation, on which we focus here. We are given a sharing $(F_i)_{i\in[n]}$ of a polynomial $f = \sum_i f_i x^i$, where f_0 encodes the sensitive information. We now want to split f into two polynomials f' and f''. To understand the general idea behind the algorithm, we will first focus on the case that no faults are present and later show how to adapt to faults. In this scenario, we aim for the following two properties.

(*) The sum of the sensitive information of f' and f'' is an *additive* sharing of the sensitive information of f, i.e., $(f' + f'')_0 = f_0$ (which is the *split* part of the gadget).
(**) The degrees of both f' and f'' are equal to $d = \deg(f)$, but the degree of the sum $f' + f''$ is only equal to $\frac{d}{2}$ (which is the *reduce* part of the gadget).

To obtain these properties, we proceed roughly as follows:

(i) We generate a random polynomial $g = \sum_i g_i x^i$ of degree d with $g_0 = 0$, i.e., all coefficients g_i are drawn uniformly at random for $i > 0$.

(ii) We generate another polynomial $g' = \sum_i g'_i x^i$ of degree d with $g'_0 = 0$. For $i \in [1, \frac{d}{2}]$, the coefficients g'_i are drawn uniformly at random. For $i > \frac{d}{2}$, we set $g'_i = -g_i$. This means that $\deg(g) = \deg(g') = d$, but $\deg(g + g') = \frac{d}{2}$.

(iii) Now, the second property (**) is fulfilled, but we still need to share the sensitive information of f into g and g'. Now, remember that share j holds the value $f(\alpha_j)$. We now set $g_0 = \sum_{j<n/2} \lambda_j^{(0)} f(\alpha_j)$ and $g'_0 = \sum_{j\geq n/2} \lambda_j^{(0)} f(\alpha_j)$. The interpolation lemma then implies the correctness, as $g_0 + g'_0 = \sum_j \lambda_j^{(0)} f(\alpha_j) = f(0) = f_0$.

While the correctness of this idea directly follows from the interpolation lemma, we need to be careful to secure the algorithm against both probes and faults. To obtain probing security, we simply need to generate more random polynomials and include the values $\lambda_j^{(0)} f(\alpha_j)$ more carefully over time. More concretely, for $j = 1, \ldots, n/2$, we first generate random polynomials $\hat{g}^{(j)}$ of degree d (with absolute term 0), and for $j = 1, \ldots, n$, we first generate random polynomials $\tilde{g}^{(j)}$ of degree $\frac{d}{2}$ (with absolute term 0). For $j < n/2$, we compute $g^{(j)} = \tilde{g}^{(j)} + \hat{g}^{(j)}$ and for $j \geq n/2$, we compute $g^{(j)} = \tilde{g}^{(j)} - \hat{g}^{(j-n/2)}$. Then, for $j = 1, \ldots, n$, we set $g^{(j)} = g^{(j)} + \lambda_j^{(0)} f(\alpha_j)$ and finally, obtain $f' = \sum_{j<n/2} g^{(j)}$ and $f'' = \sum_{j\geq n/2} g^{(j)}$. A careful inspection of the construction shows that the sensitive information is always sufficiently hidden against up to d probes.

To handle faults, we need to make sure that the error coefficients of f are also preserved. To do so, we do not only incorporate the terms $\lambda_j^{(0)} f(\alpha_j)$, but the term $\left(\lambda_j^{(0)} + \sum_{k>d} \lambda_j^{(k)} \alpha_i^k\right) f(\alpha_j)$, which we will denote by $\hat{\lambda}_j^{(i)} \cdot f(\alpha_j)$ in the following. Note that the interpolation lemma implies that $\sum_j \sum_i \hat{\lambda}_j^{(i)} f(\alpha_j) = f_0 + \sum_{k>d} f_k x^k$.

We show in the appendix that **SplitRed** is d-NI (Sec. A.1, Lem. 5) and transfers faults from its inputs to the output (Sec. A.3, Thm. 21).

6.3 Share-wise Multiplication and Compression Gadgets

The share-wise multiplication **SWMult** (Alg. 2) works similar to the addition. Remember that **SplitRed** shares two polynomials $f(x)$ and $g(x)$ into $f'(x)$, $f''(x)$, $g'(x)$, and $g''(x)$ such that for $\tilde{f}(x) = f'(x) + f''(x)$ and $\tilde{g}(x) = g'(x) + g''(x)$ it holds $\tilde{f}(x)$ and $\tilde{g}(x)$ have degree $\frac{d}{2}$ and $f(0) = \tilde{f}(0)$, $g(0) = \tilde{g}(0)$. The share-wise multiplication now might lead to degrees larger than d when they compute $h^0(x) = f'(x) \cdot g'(x)$, $h^1(x) = f'(x) \cdot g''(x)$, $h^2(x) = f''(x) \cdot g'(x)$, $h^3(x) = f''(x) \cdot g''(x)$, but the final gadget **Comp** sums up all h^i and this results into a polynomial $\sum_{i=0}^{3} h^i$ with degree d. This follows from the fact that we can alternatively write $\sum_{i=0}^{3} h^i(x) = (f'(x) + f''(x)) \cdot (g'(x) + g''(x)) = \tilde{f}(x) \cdot \tilde{g}(x)$. Since $\tilde{f}(x)$ and $\tilde{g}(x)$ have degree $\frac{d}{2}$, the product $\tilde{f}(x) \cdot \tilde{g}(x)$ has degree d. Hence the sum of the h^i results in a degree d polynomial with secret $f_0 \cdot g_0$. Note that we also add an encoding of zero in **Comp** to re-randomize the values, but this does not change the correctness of the gadget. We show in the appendix that all values of **Comp** can be simulated from a few inputs (Corollary 3) and both **SWMult** and **Comp** transfer faults from their inputs to their outputs (Theorem 22 and Theorem 23).

Algorithm 7 (n, d)-**SplitRed** for $n = d + \epsilon + 1$.

Input: Shares of f_0 as $(F_i)_{i\in[n]}$.
Output: Shares of f'_0 as $(F'{}_i)_{i\in[n]}$ and shares of f''_0 as $(F''{}_i)_{i\in[n]}$, such that $f_0 = f'_0 + f''_0$.

for $j \in [\frac{n}{2}]$ **do**
  $(\hat{g}_i^j)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^d$

for $j \in [\frac{n}{2}]$ **do**
  $(\tilde{g}_i^j)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^{\frac{d}{2}}$
  $(g_i^j)_{i\in[n]} \leftarrow (\tilde{g}_i^j)_{i\in[n]} + (\hat{g}_i^j)_{i\in[n]}$
for $j \in [\frac{n}{2}]$ **do**
  for $i \in [n]$ **do**
    $\mathcal{F}'^j_i \leftarrow \hat{\lambda}^i_j \cdot F_j$
for $j \in [\frac{n}{2}]$ **do**
  $(\mathcal{F}_i^j)_{i\in[n]} \leftarrow (\mathcal{F}'^j_i)_{i\in[n]} + (g_i^j)_{i\in[n]}$
for $j \in [\frac{n}{2}]$ **do**
  $(F'{}_i)_{i\in[n]} \leftarrow (F'{}_i)_{i\in[n]} + (\mathcal{F}_i^j)_{i\in[n]}$

for $j \in [\frac{n}{2}]$ **do**
  $(\tilde{g}_i^{j+\frac{n}{2}})_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^{\frac{d}{2}}$
  $(g_i^{j+\frac{n}{2}})_{i\in[n]} \leftarrow (\tilde{g}_i^j)_{i\in[n]} - (\hat{g}_i^j)_{i\in[n]}$
for $j \in [\frac{n}{2}]$ **do**
  for $i \in [n]$ **do**
    $\mathcal{F}'^{j+\frac{n}{2}}_i \leftarrow \hat{\lambda}^i_{j+\frac{n}{2}} \cdot F_{j+\frac{n}{2}}$
for $j \in [\frac{n}{2}]$ **do**
  $(\mathcal{F}_i^{j+\frac{n}{2}})_{i\in[n]} \leftarrow (\mathcal{F}'^{j+\frac{n}{2}}_i)_{i\in[n]} + (g_i^{j+\frac{n}{2}})_{i\in[n]}$
for $j \in [\frac{n}{2}]$ **do**
  $(F''{}_i)_{i\in[n]} \leftarrow (F''{}_i)_{i\in[n]} + (\mathcal{F}_i^{j+\frac{n}{2}})_{i\in[n]}$

return $(F'{}_i)_{i\in[n]}, (F''{}_i)_{i\in[n]}$

Algorithm 8 Comp for $n = d + \epsilon + 1$

Input: 4 Sharings $(H_i^j)_{i\in[n]}$ of h^j
Output: Sharing $(Q_i)_{i\in[n]}$ with $h^0 + h^1 + h^2 + h^3$

1: $(Q_i)_{i\in[n]} \leftarrow \mathbf{sZEnc}_n^d$
2: $(Q_i)_{i\in[n]} \leftarrow [[[(Q_i)_{i\in[n]} + (H_i^0)_{i\in[n]}] + (H_i^1)_{i\in[n]}] + (H_i^2)_{i\in[n]}] + (H_i^3)_{i\in[n]}$
3: **return** $(Q_i)_{i\in[n]}$

6.4 Security Analysis of the Multiplication Gadget

In this section we show that the multiplication **Mult** is frSNI and e-fault-robust. Corollary 1 shows that SNI and fault-invariance implies frSNI. The detailed security proofs can be found in the appendix.

Theorem 13. *The multiplication gadget **Mult** depicted in Algorithm 6 is d-fr(S)NI with respect to $\mathcal{F}^+$ or (d-wfr(S)NI with respect to $\mathcal{F}^{ind}$) and e-fault-robust with respect to $\mathcal{F}^{ind}$.*

Proof (sketch). In Appendix A.3 (Theorem 24) we prove e-fault-robustness with respect to $\mathcal{F}^{ind}$. It remains to prove the frSNI property. Due to Corollary 1, it is sufficient to show that the multiplication gadget is SNI and fault invariant with respect to $\mathcal{F}^+$ because this implies d-frSNI with respect to $\mathcal{F}^+$. Further, Corollary 2 shows that this also implies d-wfrSNI with respect to $\mathcal{F}^{ind}$. Hence, it remains to prove that the gadget **Mult** is (i) SNI and (ii) fault invariant with respect to $\mathcal{F}^+$.

(i) We give the detailed proof in Appendix A.1 (Theorem 16): We first analyze the different subroutines of **Mult** separately. Combining these results shows that the complete gadget is t-SNI.
(ii) We give the detailed proof in Appendix A.2 (Theorem 17): The proof is similar to the fault-invariance proof of linear gadgets. The only difference is that the gadget has (only) one non-linear layer – the share-wise multiplication. However, the remaining computation of the multiplication gadget is linear again and the definition of fault-invariance only requires that we can move the faults to the inputs *or* outputs. The idea is that all faults before the non-linear layer can be moved to the inputs, and the faults after the non-liner layer can be moved to the outputs as in step (ii) of the proof of Theorem 6.

Acknowledgment

This work was partly supported by the German Research Foundation (DFG) via the DFG CRC 1119 CROSSING (project S7), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and by the European Commission(ERCEA),

ERC Grant Agreement 101044770 CRYPTOLAYER. This work has been partially supported by BMBF through the VE-Jupiter project grants 16ME0231K and 14ME0234.

References

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.*, 2020(3):1–45, 2020.

[ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 586–615. Springer, Heidelberg, May 2016.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015.

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 457–485. Springer, Heidelberg, April 2015.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Heidelberg, August 2016.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Heidelberg, May 2013.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, Heidelberg, August 1997.

[BS21] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms. *IACR*

TCHES, 2021(3):202–234, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8973.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *ICICS 10*, volume 6476 of *LNCS*, pages 46–61. Springer, Heidelberg, December 2010.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 181–194. Springer, Heidelberg, September 2007.

[CN16] Thomas De Cnudde and Svetla Nikova. More efficient private circuits II through threshold implementations. In *FDTC 2016*, pages 114–124. IEEE Computer Society, 2016.

[CPR12] Jean-Sébastien Coron, Emmanuel Prouff, and Thomas Roche. On the use of shamir's secret sharing against side-channel analysis. In *CARDIS*, volume 7771 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2012.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Heidelberg, May 2014.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting ineffective fault inductions on symmetric cryptography. *IACR TCHES*, 2018(3):547–572, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7286.

[DN19] Siemen Dhooghe and Svetla Nikova. My gadget just cares for me - how NINA can prove security against combined attacks. Cryptology ePrint Archive, Report 2019/615, 2019. https://eprint.iacr.org/2019/615.

[DN20a] Siemen Dhooghe and Svetla Nikova. Let's tessellate: Tiling for security against advanced probe and fault adversaries. In Pierre-Yvan Liardet and Nele Mentens, editors, *CARDIS 2020*, volume 12609 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2020.

[DN20b] Siemen Dhooghe and Svetla Nikova. My gadget just cares for me - how NINA can prove security against combined attacks. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 35–55. Springer, Heidelberg, February 2020.

[FRSG22] Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. CINI MINIS: domain isolation for fault and combined security. In *CCS*, pages 1023–1036. ACM, 2022.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 674–704. Springer, Heidelberg, May 2020.

[GM11] Louis Goubin and Ange Martinelli. Protecting AES with Shamir's secret sharing scheme. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 79–94. Springer, Heidelberg, September / October 2011.

[GPRV21] Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):599–640, 2021.

[HKL+22] Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 581–610. Springer, Heidelberg, May / June 2022.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: Keeping secrets in tamperable circuits. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 308–327. Springer, Heidelberg, May / June 2006.

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

[LFZD14] Pei Luo, Yunsi Fei, Liwei Zhang, and A. Adam Ding. Side-channel power analysis of different protection schemes against fault attacks on AES. In *ReConFig 2014*, pages 1–6. IEEE, 2014.

[Lin98] J. H. Van Lint. *Introduction to Coding Theory*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 1998.

[ORSW12] Yossef Oren, Mathieu Renauld, François-Xavier Standaert, and Avishai Wool. Algebraic side-channel attacks beyond the hamming weight leakage model. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 140–154. Springer, Heidelberg, September 2012.

[RDB+18] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. CAPA: The spirit of beaver against physical attacks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 121–151. Springer, Heidelberg, August 2018.

[REB+08] Francesco Regazzoni, Thomas Eisenbarth, Luca Breveglieri, Paolo Ienne, and Israel Koren. Can knowledge regarding the presence of countermeasures against fault attacks simplify power attacks on cryptographic devices? In Cristiana Bolchini, Yong-Bin Kim, Dimitris Gizopoulos, and Mohammad Tehranipoor, editors, *DFT 2008*, pages 202–210. IEEE Computer Society, 2008.

[RFSG22] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. VERICA - verification of combined attacks automated formal verification of security against simultaneous information leakage and tampering. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):255–284, 2022.

[RP12] Thomas Roche and Emmanuel Prouff. Higher-order glitch free implementation of the AES using secure multi-party computation protocols - extended version. *Journal of Cryptographic Engineering*, 2(2):111–127, September 2012.

[SFRES18] Okan Seker, Abraham Fernandez-Rubio, Thomas Eisenbarth, and Rainer Steinwandt. Extending glitch-free multiparty protocols to resist fault injec-

tion attacks. *IACR TCHES*, 2018(3):394–430, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/7281`.

[SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI – towards combined hardware countermeasures against side-channel and fault-injection attacks. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 302–332. Springer, Heidelberg, August 2016.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 282–296. Springer, Heidelberg, December 2014.

Appendix

A Detailed Security Analysis

In this section we analyze the security of the refresh gadget defined in Section 5, and multiplication gadget **Mult** defined in Section 6. In Section A.1, we first analyze the probing-resilience of our constructions, and then, in Section A.2, we show fault-invariance. Using Corollary 1 and Corollary 2, this implies that the gadgets are d-(w)fr(S)NI, i.e., it does not reveal any sensitive information even in the presence of faults. Then, in Section A.3, we analyze the error preservation of our refresh and multiplication gadgets and prove that it is e-fault-robust. Using Theorem 4 then implies that the probability that faults introduced into the circuit lead to a valid sharing of an invalid value is bounded.

A.1 Probe-Resilience

In this section we prove the probe resilience of Refresh and **Mult**. In detail, we analyze the underlying gadgets **sZEnc**, **SplitRed**, **Comp**, and **SWMult**. This results in the d-SNI property of **Mult** (Theorem 16), and Refresh (Theorem 14). In Theorem 17, we furthermore show that the constructions are fault-invariant, which implies that the gadgets are also d-fr(S)NI.

Algorithm 9 $\mathbf{ZEnc2}_n^d$

Output: $d+1$-out-of-n sharing $(y_i)_{i\in[n]}$ of zero.

1: **for** $j \in [d]$ **do**
2: $\quad(g_i^j)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^{j+1}$
3: $\quad(y_i)_{i\in[n]} \leftarrow (y_i)_{i\in[n]} + (g_i^j)_{i\in[n]}$ ▷ Noted as $(y^{(j)}{}_i)_{i\in[n]}$
4: **return** $(y_i)_{i\in[n]}$

Probe-Resilience of sZEnc – Proof of Lemma 3. Next we prove Lemma 3. Therefore, we do not give a concrete simulator but we prove that such a simulator exists. In detail, let $(e_i)_{i\in[n]} \leftarrow$ **sZEnc**, and P the probes in **sZEnc** with $|P| = d'$. It is clear that there is a simulator that perfectly simulates P with $(e_i)_{i\in[n]}$. Due to the redundancy there is such an simulator only requiring $(e_i)_{i\in C}$ with $C \subset [n]$ and $|C| = d$ as well. To show that there is a simulator that can simulate P only using d' output values we do the opposite – We show which values the simulator does not need. Let $(e_i)_{i\in A}$ be the outputs that the simulator does not need with $|A| = n - d'$. Hence, we want to argue that there is a simulator that simulates P

with $(e_i)_{i\in[n]\setminus A}$. In other words we have to show that all $(e_i)_{i\in A}$ are not effected by the choices of of P **and** $(e_i)_{i\in[n]\setminus A}$. Formally, this proves the following claim.

Algorithm 10 $\mathbf{sZEnc}_n^d$

Output: $d+1$-out-of-n sharing $(y_i)_{i\in[n]}$ of zero.

1: **for** $j \in [d]$ **do**
2: $\quad(g_i^j)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^d$
3: $\quad(y_i)_{i\in[n]} \leftarrow (y_i)_{i\in[n]} + (g_i^j)_{i\in[n]}$ ▷ Noted as $(y^{(j)}{}_i)_{i\in[n]}$
4: **return** $(y_i)_{i\in[n]}$

Lemma 4 (Lemma 3). *For any set P with $d' \leq d$ probes it holds for $(e_i)_{i\in[n]} \leftarrow \boldsymbol{sZEnc}_n^d$: There is a sub set $A \subset [n]$ with $|A| = n - d'$ such that*

(i) $(e_i)_{i\in A}$ *are still $(d-d')$-wise independent, and*
(ii) $(e_i)_{i\in A}$ *are independent of P* ***and*** $(e_i)_{i\in[n]\setminus A}$

Note that this means that an adversary only learning P can be upper-bounded with an adversary that knows P **and** $(e_i)_{i\in[n]\setminus A}$. However, such an "upper-bound adversary" does not learn more about the output of $(e_i)_{i\in[n]} \leftarrow \mathbf{sZEnc}_n^d$ than an adversary only probing the outputs $(e_i)_{i\in[n]\setminus A}$. In other words it exist a simulator that can simulate P only using values in $(e_i)_{i\in[n]\setminus A}$, since $(e_i)_{i\in A}$ are independent of P **and** $(e_i)_{i\in[n]\setminus A}$. Hence, an adversary that only probes the outputs can learn the same about the outputs as a simulator that can also probe internal values. This is exactly what we want to have when we consider an gadget as leakage-free. Next we prove the leakage-free property:

Proof. We show that for any set P of d' probes there is an index set $A \subset [n]$ with $|A| = n - d'$ such that the outputs $(e_i)_{i\in A}$ are still $d-d'$-wise independent and independent of the probes and the other outputs $(e_i)_{i\in[n]\setminus A}$. In detail, we show that for any $A' \subset A$ with $|A'| = d - d'$ the values $(e_i)_{i\in A'}$ are independent and independent of P and $(e_i)_{i\in[n]\setminus A}$.

Group K: All probes in the set $K = \bigcup_{l\in[n]} K_l$ with K_l the internal values in $(g_i^l)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^d$ and $(g_i^l)_{i\in[n]}$. We write *probes of Group K* when we refer to all probes that form a subset of K.
Group J: All probes in the set $J = \bigcup_{i\in[n]} J_i$ with

$$J_i = \{y_i^{(0)}, y_i^{(1)}, \dots y_i^{(\frac{n}{2}-1)}\},$$

where $(y^{(j)}{}_i)_{i\in[n]}$ is j^{th} loop iteration with $(y^(j)_i)_{i\in[n]} = \sum_{k=0}^{j}(g_i^j)_{i\in[n]}$. We write *probes of Group J* when we refer to all probes that form a subset of J.

Next, we give a simulator to simulate all Probes P. We can group P into $\tilde{K} \subset K$ and $\tilde{J} \subset J$ such that $P = \tilde{K} \cup \tilde{J}$. For sake of simplicity, we use an upper-bound

for $\tilde{J}$ and simulate all values in J_i if at least one value is probed in J_i. In the following C is the index set i of all sets J_i with at least one probe of $\tilde{J}$. Hence the simulator simulates all values in $\bigcup_{i\in C} J_i$. In the following we show that an adversary only probing $(y_i)_{i\in C}$ with does not learn more than an adversary probing the intermediate values $\bigcup_{i\in C} J_i$ and $\tilde{K}$. Next, we proof that $A = [n] \setminus C$ is exactly the set described above. For this reason we do an upper-bound and assume for any probe in K that all values leak of the according $\mathbf{ZEnc}_n^d$ gadget. With this assumption we can easily simulate all the according $\mathbf{ZEnc}_n^d$ gadget. However, we have at least $d - |\tilde{K}|$ unleaked outputs $(g_i^l)_{i\in[n]}$ that are still d-wise independent and independent of $\tilde{K}$ (and the according $\mathbf{ZEnc}_n^d$ gadget) Next, we can simulate $\tilde{J}$. If the according $(g_i^l)_{i\in[n]}$ are already simulated due to the probes in $\tilde{K}$ we simply take them. Otherwise we can choose the according $(g_i^l)_{i\in[n]}$ uniform random because the probe threshold is d and $(g_i^l)_{i\in[n]}$ are d-wise independent. Therefore, the remaining $(g_i^l)_{i\in[n]}$ that are not fixed by $\tilde{K}$ sum up to an output $(e_i)_{i\in A}$ $d - |C|$-wise independent and independent of P **and** $(e_i)_{i\in[n]\setminus A}$.

Probe resilience of Refresh – Proof of Theorem 11 & 12. With Lemma 4 we can give the security results of our refresh gadget.

Theorem 14 (Theorem 11). *The gadget* G'_{G} *(Alg. 5) with identity* G *is a d-SNI refresh gadget*

Proof. Let I the index set of output probes in $(y_i)_{i\in[n]}$ and P the internal probes in $(e_i)_{i\in[n]} \leftarrow$ **sZEnc** (we also count the probes in $(e_i)_{i\in[n]}$ as internal probes). With Lemma 4 we get that there is an index set A with $|A| = n - |P|$ such that $(e_i)_{i\in A}$ are $(d - |P|)$-wise independent and independent of P **and** $(e_i)_{i\in[n]\setminus A}$.

(*i*) It follows we can choose $(e_i)_{i\in[n]\setminus A}$ according to P, and simulate P and $(e_i)_{i\in[n]\setminus A}$. Due to Lemma 4 this does not change the distribution of $(e_i)_{i\in A}$.
(*ii*) The simulator gets all input shares $(x_i)_{i\in[n]\setminus A}$
(*iii*) We can choose a uniform random value to simulate any output probe in $(y_i)_{i\in A}$ due to the randomness of $(e_i)_{i\in A}$. Further, for any probe in $(y_i)_{i\in[n]\setminus A}$ the simulator has both $(e_i)_{i\in[n]\setminus A}$ (Step (*i*)), and $(x_i)_{i\in[n]\setminus A}$ (Step (*ii*)), Hence it can also simulate those values with $y_i = x_i + e_i$.

It remains to argue that the number of required input values is the number of internal probes:

$$|[n] \setminus A| = n - |A| = n - (n - |P|) = |P| \ .$$

Further we know that $(e_i)_{i\in A}$ are $d-|P|$ wise independent. Hence, we can consider them as uniform random for the simulation if the probe threshold is d.

Theorem 15 (Theorem 12). *A d probing secure composition with d-NI and d-SNI secure gadgets is* $d/2$ *region probing secure if each gadget outputs sharings refreshed by* ***sZEnc***

Proof. The Simulator first splits the circuit C into the sub gadgets $\tilde{\mathsf{G}}_l$. The gadget $\tilde{\mathsf{G}}_l$ can be further split into the (S)NI sub-gadgets G_l and refresh sub-gadgets $\mathbf{sZEnc}_l$.

We first consider the probes P_l in $\mathbf{sZEnc}_l$. Due to Lemma 3 we know that we can simulate P_l with output sets $(e_i^l)_{i\in A_l}$ of $\mathbf{sZEnc}_l$ with $A_l \leq P_l$.

First the simulator simulates all probes in the gadgets $(y_i^l)_{i\in[n]} \leftarrow G_l$ together with the outputs $(y_i)_{i\in A_l}$.

Note that all gadgets have such a simulator due to the (S)NI probes. Due to the threshold of $d/2$ probes in each gadget the simulators need at most $d/2$ input values to simulate the probes. If the simulator simulates a gadget that uses input wires of the circuit we take those values. (Hence we need at most $d/2$ input shares of each input sharing of C).

For the other gadgets that use inputs that are not outputs of an other gadget, we can do something that differs from proofs in the threshold model. In this case we only choose uniform random values to feed the simulator.

Next we argue why we can feed the simulators of the intermediate G_l with uniform random values. Therefore, we have to analyze the refresh gadgets **sZEnc**: Note that the whole gadget $\tilde{\mathsf{G}}_l$ first adds an zero encoding to the output of its underlying (S)NI gadget G_l.

Let us assume we have simulated $d/2$ output probes of the previous sub gadget G_l and need $d/2$ probes to simulate the probes of the next sub gadget G_{l+1}. We can chose all outputs of $\mathbf{sZEnc}_l$ such that the final $d/2$ outputs of the whole gadgets $\tilde{\mathsf{G}}_l$ are consistent to the $d/2$ inputs in $\tilde{\mathsf{G}}_{l+1}$. Note that the outputs of $\mathbf{sZEnc}_l$ are still uniform random since we have feed the intermediate simulators with uniform random values. It remains to simulate all P_l with the simulator of $\mathbf{sZEnc}_l$, and the chosen T_l. Hence the simulator only needs $d/2$ inputs of each input sharing of C to simulate $d/2$ probes in each gadget $\tilde{\mathsf{G}}_l$. However, we did not use the property of $\mathbf{sZEnc}_l$ yet. Since an output of G_l is randomized by $\mathbf{sZEnc}_l$ it is uniform random and

Probe-Resilience of SplitRed We get the following security results for the gadget.

Lemma 5. ***SplitRed*** *is d-NI.*

Proof. In the proof we distinguish three groups of probes – Group K, Group I, and Group J.

Group K: All probes in the set $K = (\bigcup_{l\in[n]} K_l) \cup (\bigcup_{l_\in[n/2]} K'_l)$ with

K_l – the internal values in $(\tilde{g}_i^l)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^{\frac{d}{2}}$ and $(\tilde{g}_i^l)_{i\in[n]}$,
K'_l – the internal values in $(\hat{g}_i^l)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^{d}$ and $(\hat{g}_i^l)_{i\in[n]}$.

Hence, K is the set of all values of the loops in Line 2, 4, 14, incl. the probes in **ZEnc**. Further, we write *probes of Group K* when we refer to all probes that form a subset of $K = (\bigcup_{l\in[n]} K_l) \cup (\bigcup_{l_\in[n/2]} K'_l)$.

Group I: All probes in the set $I = \bigcup_{l\in[n]} I_l$ with

$$I_l = \{F_l, \mathcal{F}_0^l, \mathcal{F}_1^l, \dots \mathcal{F}_{n-1}^l, g_0^l, g_1^l, \dots g_{n-1}^l, \mathcal{F}_0^l + g_0^l, \mathcal{F}_1^l + g_1^l, \dots, \mathcal{F}_{n-1}^l + g_{n-1}^l\}$$

Further, let

$$I_l(i) := \{\mathcal{F}_i^l, g_i^l, \mathcal{F}_i^l + g_i^l, \}$$

be the set that depends on the input F_l and the random values g_i^l. Again, we write *probes of Group I* when we refer to all probes that form a subset of I.
Group J: All probes in the set $J = \bigcup_{i\in[n]}(J_i' \cup J_i'')$ with

$$J_i' = \{{y'}_i^{(0)}, {y'}_i^{(1)}, \dots {y'}_i^{(\frac{n}{2}-1)}\} \text{ and } J_i'' = \{{y''}_i^{(0)}, {y''}_i^{(1)}, \dots {y''}_i^{(\frac{n}{2}-1)}\}\,.$$

We write *probes of Group J* when we refer to all probes that form a subset of J.

Next, we discuss the three cases when a random value g_i^l is not uniform random due to probes in Group K.

D. Case 1: The first dependence case is a probe in K_l and K_l' with $l < n/2$ because this means that $\tilde{g}_i^l$ and $\hat{g}_i^l$ are not independent of the probes in K. Hence, its sum $g_i^l = \tilde{g}_i^l + \hat{g}_i^l$ depends on the probes in K as well.
D. Case 2: The second reason is a probe in K_l and $K_{l-n/2}'$ with $l \geq n/2$, because it follows with the same argument as in Case 1 that $g_i^l = \tilde{g}_i^l + \hat{g}_i^{l-n/2}$ depends on the probes in K.
D. Case 3: The last dependence case is a probe in K_l' and no probes in K_l and $K_{l+n/2}$ (otherwise it is case one or two). Such a probe leads to a d'-wise independence in $(g_i^l)_{i\in[n]}$ and $(g_i^{l+n/2})_{i\in[n]}$ with $d/2 \leq d' < d$, because we need to assume that $(\hat{g}_i^l)_{i\in[n]}$ is not d-wise independent due to the probes in K_l'. However, $(g_i^l)_{i\in[n]}$ is still randomized by the $d/2$-wise independent random values $(\tilde{g}_i^l)_{i\in[n]}$; and $(g_i^{l+n/2})_{i\in[n]}$ is still randomized by the $d/2$-wise independent random values $(\tilde{g}_i^{l+n/2})_{i\in[n]}$. **Note:** If the probes do not depend on more of then $d/2$ probes on $(g_i^l)_{i\in[n]}$, the random values are still uniform random and independent of $(g_i^{l+n/2})_{i\in[n]}$ (**D. Case 3(a)**). And vice verse: If the probes do not depend on more of then $d/2$ probes on $(g_i^{l+n/2})_{i\in[n]}$, the random values are still uniform random and independent of $(g_i^l)_{i\in[n]}$ (**D. Case 3(b)**). Hence, in this case we only need to consider one of the two encodings and can assume that the other provides uniform random values.

Next, we can give the input set S for the simulator. The high-level idea is that S consists of all F_i that depend on the probes and cannot be randomized by uniform random g_i^l.

Input set S for the simulator:

(K) For probes in Group K: If at least two simultaneous probes are in the sets K_l, $K_{l+\frac{n}{2}}, K_l'$ with $l \in [n/2]$ the simulator gets both F_l and $F_{l+\frac{n}{2}}$.

$$S \leftarrow S \cup \{F_l, F_{l+\frac{n}{2}}\}$$

Note that we add at most one input share per probe in Group K. This covers all cases where (two) probes lead to not uniform random $(g_i^l)_{i\in[n]}$ and $(g_i^{l+n/2})_{i\in[n]}$ (**D. Case 1 and 2**).

(I) For probes in Group I: If there is a probe in I_l, the simulator gets F_l.

$$S \leftarrow S \cup \{F_l\}$$

Note that we add at most one input share per probe in Group I. This covers all cases where probes in I_l might reveal F_l.

(J) For probes in Group J: Here, we distinguish two events that are disjoint due to the probe threshold (and disjoint from the upper event).

(i) The event that there are more than $d/2$ probes in the set $\bigcup_{i=0}^{n/2-1}(I_i \cup J_i')$: If there is a probe in K_l' (and not in K_l, $K_{l+\frac{n}{2}}$), the simulator gets F_l (**D. Case 3(b)**).

$$S \leftarrow S \cup \{F_l\}$$

(ii) The event that there are more than $d/2$ probes in the set $\bigcup_{i=0}^{n/2-1}(I_{i+\frac{n}{2}} \cup J_i'')$: If there is a probe in K_l' (and not in K_l, $K_{l+\frac{n}{2}}$), the simulator gets $F_{l+\frac{n}{2}}$ (**D. Case 3(a)**).

$$S \leftarrow S \cup \{F_{l+\frac{n}{2}}\}$$

This covers the case where $(g_i^l)_{i\in[n]}$ and $(g_i^{l+n/2})_{i\in[n]}$ are not d-wise independent (**D. Case 3**). Since event (i) and (ii) are disjoint due to the probe threshold we add at most one input share per probe in J.

Next, we can give the simulator using the input set S. For sake of simplicity, we give an upper-bound and assume that the simulator simulates all intermediate values ${y'}_i^{(j)}$ with $j = 0 \dots n/2$ if at least one intermediate ${y'}_i^{(j)}$ was probed. Similarly, the simulator simulates all intermediate values ${y''}_i^{(j)}$ with $j = 0 \dots n/2$ if at least one intermediate ${y''}_i^{(j)}$ was probed. In other words, the simulator simulates all values of J_i' (or J_i'') if there is at least one probe in J_i' (or J_i'').

Simulator with input set S:

(K) Simulation of Group K – Cases when $(g_i^l)_{i\in[n]}$ is not a d-wise independent zero encoding (D. Case 1-3): If there is at least one probe in K_l' or two simultaneous probes in K_l and $K_{l+\frac{n}{2}}$ with $l \in [n/2]$, the simulator also simulates the following g_i^l:

(i) If there is a probe in $I_l(i)$ with $l \in [n/2]$, and at least two simultaneous probes are in the sets K_l, $K_{l+\frac{n}{2}}$, K_l', the simulator simulates $g_i^l = \tilde{g}_i^l + \hat{g}_i^l$ by computing the distribution of $\tilde{g}_i^j$ and $\hat{g}_i^j$ (D. Case 1).

(ii) If there is a probe in $I_{l+\frac{n}{2}}(i)$ with $l \in [n/2]$, and at least two simultaneous probes are in the sets K_l, $K_{l+\frac{n}{2}}$, K_l', the simulator simulates $g_i^{l+\frac{n}{2}} = -\tilde{g}_i^l + \hat{g}_i^{l+\frac{n}{2}}$ by computing the distribution of $\tilde{g}_i^l$ and $\hat{g}_i^{l+\frac{n}{2}}$ (D. Case 2).

(iii) If there is a probe in K_l' (and not in K_l, $K_{l+\frac{n}{2}}$), and

(a) there are more than $d/2$ probes in the set $\bigcup_{i=0}^{n/2-1}(I_i \cup J'_i)$, the simulator simulates $g_i^l = \tilde{g}_i^l + \hat{g}_i^l$ by computing the distribution of $\tilde{g}_i^j$ and $\hat{g}_i^j$ (D. Case 3(a)).

(b) there are more than $d/2$ probes in the set $\bigcup_{i=0}^{n/2-1}(I_{i+\frac{n}{2}} \cup J''_i)$, the simulator simulates $g_i^l = -\tilde{g}_i^l + \hat{g}_i^{l+\frac{n}{2}}$ by computing the distribution of $\tilde{g}_i^j$ and $\hat{g}_i^j$ (D. Case 3(b)).

In detail, all required $\tilde{g}_i^j$ and $\hat{g}_i^j$ mentioned above, and all probes in $(\bigcup_{l\in[n]} K_l) \cup (\bigcup_{l_\in[n/2]} K'_l)$ can be simulated according to the algorithm.

(I) Simulation of Group I: For all probes in $\bigcup_{l\in[n]} I_l$ the simulator does the following: For any $i \in [n]$ and $l \in [n]$: If there is a probe in $I_l(i)$ the simulator has F_l due to the generation of S. Further, due to Step K, it has g_i^l if it depends on probes in K. Otherwise, it can choose in g_i^l uniform random. Hence, the simulator can simulate all elements in $I = \bigcup_{i\in[n]} \bigcup_{l\in[n]} I_l(i)$.

(J) Simulation of Group J: For all probes in $\bigcup_{i\in[n]}(J'_i \cup J''_i)$ the simulator does the following:

(J') For any $i \in [n]$: If there is a probe in J'_i, the simulator simulates all values of J'_i. Since all elements in J'_i are sums of $(\hat{\lambda}_l^i F_l) + g_i^l$ with $l \in [n/2]$, we only need to show how to simulate all $(\hat{\lambda}_l^i F_l) + g_i^l$.

(i) If g_i^l is not uniform random due to the probes of Group K and I, the simulator has generated g_i^l in Step K or I. Further, it holds $F_l \in S$ and the simulator can perfectly simulate $(\hat{\lambda}_l^i F_l) + g_i^l$.

(ii) If g_i^l is uniform random and independent of the probes in Group K and I, the simulator can perfectly simulate $(\hat{\lambda}_l^i F_l) + g_i^l$ by choosing a uniform random value because it is randomized by the unknown g_i^l.

(J") For any $i \in [n]$: If there is a probe in J''_i, the simulator simulates all values of J''_i. Since all elements in J''_i are sums of $(\hat{\lambda}_{l+\frac{n}{2}}^i F_{l+\frac{n}{2}}) + g_i^{l+\frac{n}{2}}$ with $l \in [n/2]$ we only need to show how to simulate all $(\hat{\lambda}_{l+\frac{n}{2}}^i F_{l+\frac{n}{2}}) + g_i^{l+\frac{n}{2}}$.

(i) If $g_i^{l+\frac{n}{2}}$ is not uniform random due to the probes of Group K and I, the simulator has generated $g_i^{l+\frac{n}{2}}$ in Step K or I. Further, it holds $F_{l+\frac{n}{2}} \in S$ and the simulator can perfectly simulate $(\hat{\lambda}_l^i F_l) + g_i^l$.

(ii) If g_i^l is uniform random and independent of the probes in Group K and I, the simulator can perfectly simulate $(\hat{\lambda}_{l+\frac{n}{2}}^i F_{l+\frac{n}{2}}) + g_i^{l+\frac{n}{2}}$ by choosing a uniform random values because it is randomized by the unknown $g_i^{l+\frac{n}{2}}$.

For the security proof it remains to show that the simulator can perfectly simulate any d probes P with at most d input values. We can split any set of probes P in **SplitRed** into the three Groups K, I, J, and it holds $|S| \leq |P| \leq d$ due to the definition of the input set S. Next, we prove that the simulator can perfectly simulate all values in P only using the set S.

Group K: The simulation of all probes in Group K are independent of the inputs and can be perfectly simulated.
Group I: Due to the definition of S it holds that $F_l \in S$ if there is a probe in I_l. Since the set consists of intermediate values that only depend on the input value F_l and random values $(g_i^l)_{i\in[n]}$, it can perfectly simulate all probes with S. The g_i^l is already simulated by the simulator (Step K) if it depends on probes in Group K. Otherwise, it can be chosen uniform random. (Note that each probe in I_l only depends on one g_i^l at most, and we will argue later why g_i^l can be chosen uniform random when it was not generated in Step K of the simulator)
Group J: For every $i \in [n]$ the simulator simulates all values in J_i' (or J_i'') if there is at least one probe in J_i' (or J_i'').

- The simulation of all values in J_i' is equivalent to simulate all $(\hat{\lambda}_l^i F_l) + g_i^l$ with $l \in [n/2]$. Due to the definition of the input set S the simulator has F_l if g_i^l was generated in Step K or Step I. Hence, it can perfectly simulate $(\hat{\lambda}_l^i F_l) + g_i^l$ in this case. If g_i^l was not generated in Step K (and it was not already chosen uniform random in Step I), the simulator can chose uniform random value for $(\hat{\lambda}_l^i F_l) + g_i^l$ because g_i^l is unknown and uniform random. (Note that all probes in J_i' only depend on the i^{th} random value of each $(g_i^l)_{i\in[n]}$ and we will argue later why all g_i^l are still uniform random when they are not generated in Step K of the simulator)
- The simulation of all values in J_i'' is equivalent to simulate all values $(\hat{\lambda}_{l+n/2}^i F_{l+n/2}) + g_i^{l+n/2}$ with $l \in [n/2]$. Due to the definition of the input set S the simulator has $F_{l+n/2}$ if $g_i^{l+n/2}$ was generated in Step K or Step I. Hence, it can perfectly simulate $(\hat{\lambda}_{l+n/2}^i F_{l+n/2}) + g_i^{l+n/2}$ in this case. If $g_i^{l+n/2}$ was not generated in Step K (and it was not already chosen uniform random in Step I), the simulator can chose an uniform random value for $(\hat{\lambda}_l^i F_l) + g_i^l$ because g_i^l is unknown and uniform random. (Note that all probes in J_i' only depend on the i^{th} random value of each $(g_i^l)_{i\in[n]}$. Next, we argue why all g_i^l are still uniform random when they are not generated in Step K of the simulator)

Now we argue why we can consider the g_i^l as uniform random if they are not generated in Step K of the simulator. As mentioned above each probe only depends on at most all i^{th} random values of $(g_i^l)_{i\in[n]}$ with $l \in [n/2]$, or it only depends on at most all i^{th} random values of $(g_i^{l+n/2})_{i\in[n]}$ with $l \in [n/2]$. Note that d values in $(g_i^l)_{i\in[n]}$ (or $(g_i^{l+n/2})_{i\in[n]}$) are still independent and uniform random if there is no probe in K_l' and K_l (or K_l' and $K_{l+n/2}$). Here the probe threshold d comes into account because for each probe there is an

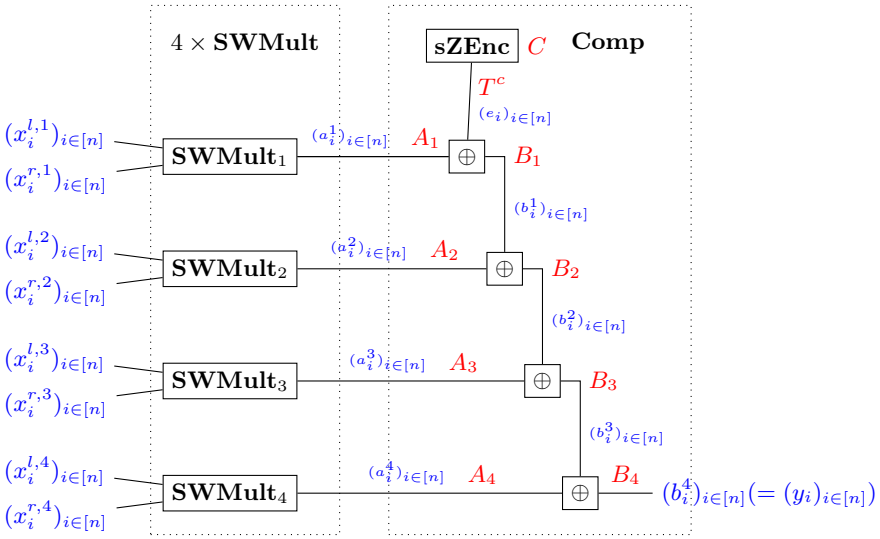


Fig. 2: Probe Propagation of **Comp** (Alg. 8)

$i \in [n]$ such that the simulator uses at most the random values

$$M_i' = \{g_i^0, g_i^1, \ldots, g_i^{\frac{n}{2}-1}\} \text{ or } M_i'' = \{g_i^{\frac{n}{2}}, g_i^{\frac{n}{2}+1}, \ldots, g_i^{n-1}\}$$

to randomize the probes. Hence, the simulator never uses more then d random values of each encoding, and the values in $(g_i^l)_{i\in[n]}$ and $(g_i^{l+n/2})_{i\in[n]}$ are d wise independent if there is no probe in K_l', K_l, and $K_{l+n/2}$. It remains to argue why we can also consider one of the encodings $(g_i^l)_{i\in[n]}$ and $(g_i^{l+n/2})_{i\in[n]}$ as random in **D. Case 3**. Again this follows with the d threshold. The simulator uses at most $d/2$ random values of at least one of the encodings. Hence those values are still independent and uniform random. And at most one of the encodings in only one of the encodings $(g_i^l)_{i\in[n]}$ and $(g_i^{l+n/2})_{i\in[n]}$ leads to dependent random values. This case is covered by Step K(iii) in the simulator, and the simulator has all dependent (and required) g_i^l.

This completes the proof, and **SplitRed** is d-NI, because $|S| \leq |P|$ for any set of Props P with $|P| \leq d$.

Probe-Resilience of Comp and SWMult We analyze the final computation of the gadget depicted in Figure 2. It consists of one **sZEnc**, four share-wise addition gadgets **SWAdd** ($\oplus$) and four **SWMult**. We labeled all edges with its intermediate values $(x_i^{l,j})_{i\in[n]}$, $(x_i^{r,j})_{i\in[n]}$, $(a_i^j)_{i\in[n]}$, $(b_i^j)_{i\in[n]}$, $(e_i)_{i\in[n]}$, and $(y_i)_{i\in[n]}$, as show in Figure 2. Let C be the set of internal probes in **sZEnc**. Lemma 3 guarantees the existence of an index set T with $|T| \geq n - |C|$ such that $(e_i)_{i\in T}$ is uniform random and $(n-|C|)$-wise independent from C where $(e_i)_{i\in[n]}$

are the outputs of **sZEnc**. Hence, we can ignore C and analyze how to simulate the remaining leakage with index sets A_j and B_j as depicted in Figure 2 with $(e_i)_{i\in T}$. For the sake of simplicity, we assume that we have to simulate all three values $x_i^{l,j}$, $x_i^{r,j}$, and a_i^j if at least one value is probed and use the index set A_j for such probes. The next corollary, Corollary 3, claims that all values $(b_i^1)_{i\in B_1}$, $(b_i^2)_{i\in B_2}$, $(b_i^3)_{i\in B_3}$, $(b_i^4)_{i\in B_4}$ and C can be simulated with

$$S = \bigcup_{i\neq j} (B_i \cap B_j) \cup \bigcup_{j=1}^{4} (B_j \cap T^c) \ ,$$

which also gives a index set for each $(x_i^{r,j})_{i\in[n]}$ and $(x_i^{l,j})_{i\in[n]}$ to simulate the entire circuit.

Corollary 3. *All values $(a_i^1)_{i\in A_1}$, $(a_i^2)_{i\in A_2}$, $(a_i^3)_{i\in A_3}$, $(a_i^4)_{i\in A_4}$, $(b_i^1)_{i\in B_1}$, $(b_i^2)_{i\in B_2}$, $(b_i^3)_{i\in B_3}$, $(b_i^4)_{i\in B_4}$, and C can be simulated from the inputs*

$$(x_i^{r,j})_{i\in A_j\cup S} \text{ and } (x_i^{l,j})_{i\in A_j\cup S} \text{ with } j = 1,2,3,4 \ .$$

*Further, it holds $|S| \leq \frac{1}{2}t'$ where t' is the number of probes in **Comp** ($t' = \sum_{j=1}^{4} |B_j| + |C|$).*

Proof. With Lemma 3, we can describe the probes C with an index set T such that $|T| \geq n-|C|$ and $(e_i)_{i\in T}$ are uniform random and $(n-|C|)$-wise independent from C where $(e_i)_{i\in[n]}$ are the outputs of **sZEnc**. If further probes depend on k random values e_i of the remaining values $(e_i)_{i\in T^c}$, we can assume that the adversary knows such e_i and it follows that the values $(e_i)_{i\in T}$ are still l-wise independent from those probes with $l = n - |C| - k$ (Lemma 3). Hence, we can ignore C and define the remaining leakage with index sets A_j and B_j as depicted in Figure 2 and analyze how to simulate them with $(e_i)_{i\in T}$. We split the circuit into two sub-circuits **Comp** and the $4 \times$ **SWMult** as depicted in Figure 2. For each $(a_i^j)_{i\in[n]}$ with $j = 1,2,3,4$, we first analyze the needed index set S'_j to simulate the leakage of **Comp**. Then we analyze the index set S_j of all $x_i^{l,j}$ and $x_i^{r,j}$ that the simulator requires for the simulation of $(a_i^j)_{i\in A_j}$ and $(a_i^j)_{i\in S'_j}$. Hence, we give index sets S_1, S_2, S_3, S_4 such that we can simulate the leakage of the entire circuit of Figure 2 with $(x_i^{r,j})_{i\in S_j}$ and $(x_i^{l,j})_{i\in S_j}$:

S_1 All probes in $(b_i^1)_{i\in B_1}$, $(b_i^2)_{i\in B_2}$, $(b_i^3)_{i\in B_3}$, $(b_i^4)_{i\in B_4}$ depend on the sharings $(a_i^1)_{i\in[n]}$ and $(e_i)_{i\in[n]}$ we can distinguish two cases. The first case is that such probe has an index i in T the other one is that i is in T^c. If $i \in T^c$, we assume that e_i is already known by the adversary. Hence, we need a_i to simulate the probe. This can be done due to Lemma 3 because the adversary never probes more the t values and the values $(e_i)_{i\in T}$ will always stay independent enough. If $i \in T$ we can randomize the probe with e_i and do not need a_i for the simulation. Hence, we need all $(a_i^1)_{i\in S_1}$ with $S'_1 = \left(\bigcup_{j=1}^{4} B_j\right) \cap T^c$ to simulate the leakage in **Comp** and therefore, we need all $x_i^{l,1}$ and $x_i^{r,1}$ with

index in S_1' or A_1 to simulate the entire circuit of Figure 2. In other words the simulator needs $(x_i^{r,1})_{i\in S_1}$ and $(x_i^{l,1})_{i\in S_1}$ with

$$S_1 = \left(\left(\bigcup_{j=1}^{4} B_j\right) \cap T^c\right) \cup A_1 \ .$$

S_2 All probes in $(b_i^2)_{i\in B_2}$, $(b_i^3)_{i\in B_3}$, $(b_i^4)_{i\in B_4}$ depend on the sharings $(a_i^2)_{i\in[n]}$ and $(e_i)_{i\in[n]}$. If we compare $(a_i^2)_{i\in[n]}$ with $(a_i^1)_{i\in[n]}$ it turns out that $(b_i^1)_{i\in B_1}$ does not depend on $(a_i^2)_{i\in[n]}$. But if we have to simulate a probe in $(b_i^2)_{i\in B_2}$, $(b_i^3)_{i\in B_3}$, $(b_i^4)_{i\in B_4}$ with index $i \in B_1$ we cannot randomize such values with a random value e_i because the value is determined by b_i^j. It follows that all a_i^2 with $i \in T^c \cup B_1$ are required to simulate the leakage if the simulator has to simulate at least on of the values b_i^2, b_i^3, b_i^3. The simulation of the leakage in **Comp** requires all $(a_i^2)_{i\in S_2'}$ with $S_2' = \left(\left(\bigcup_{j=2}^{4} B_j\right) \cap (B_1 \cup T^c)\right)$, and the simulator of the entire circuit of Algorithm 2 needs all $(x_i^{r,2})_{i\in S_2}$ and $(x_i^{l,2})_{i\in S_2}$ with

$$S_2 = \left(\left(\bigcup_{j=2}^{4} B_j\right) \cap (T^c \cup B_1)\right) \cup A_2 \ .$$

S_3 All probes in $(b_i^3)_{i\in B_3}$, $(b_i^4)_{i\in B_4}$ depend on the sharings $(a_i^2)_{i\in[n]}$ and $(e_i)_{i\in[n]}$. If we compare $(a_i^3)_{i\in[n]}$ with $(a_i^2)_{i\in[n]}$ similar to S_2 it turns out that also $(b_i^2)_{i\in B_2}$ does not depend on $(a_i^2)_{i\in[n]}$. But again we cannot randomize $(b_i^3)_{i\in B_3}$, $(b_i^4)_{i\in B_4}$ with index $i \in B_2$ and thus we obtain the set $S_3' = \left(\left(\bigcup_{j=3}^{4} B_j\right) \cap \left(\bigcup_{j=1}^{2} B_j \cup T^c\right)\right)$ and the simulator needs the information $(x_i^{r,3})_{i\in S_3}$ and $(x_i^{l,3})_{i\in S_3}$ with

$$S_3 = \left(\left(\bigcup_{j=3}^{4} B_j\right) \cap \left(\bigcup_{j=1}^{2} B_j \cup T^c\right)\right) \cup A_3 \ .$$

S_4 With the same techniques we get $S_4' = \left(\left(\bigcup_{j=4}^{4} B_j\right) \cap \left(\bigcup_{j=1}^{3} B_j \cup T^c\right)\right)$ and the simulator needs $(x_i^{r,4})_{i\in S_4}$ and $(x_i^{l,4})_{i\in S_4}$ with

$$S_4 = \left(\left(\bigcup_{j=4}^{4} B_j\right) \cap \left(\bigcup_{j=1}^{3} B_j \cup T^c\right)\right) \cup A_4 \ .$$

S To simplify the results we give an upper-bound $S_i' \subset S$ with

$$S = \bigcup_{i\neq j} (B_i \cap B_j) \cup \bigcup_{j=1}^{4} (B_j \cap T^c) \ .$$

S results in the first claim of the corollary because all $S_j \subset S \cup A_j$. For the second claim we can give an upper-bound of $|S|$ since $|T^c| \leq |C|$ and an index i is only in S if it is in at least two sets of B_i and T^c.

This results in the claim of the corollary. □

Probe-Resilience of Mult (Alg 6) With the claims of the previous sections we can finally prove the leakage-resilience of our gadget **Mult**.

Theorem 16. *The multiplication gadget **Mult** depicted in Algorithm 6 is t-SNI.*

Proof. As depicted in Figure 1a, the algorithm consists of three different sub-gadgets **SplitRed**, **SWMult**, and **Comp**. Let t' be the number of output probes, t'' the number of internal probes in **SWMult** and **Comp**, and t''' the number of internal probes in **SplitRed** with $t' + t'' + t''' \leq t$. In the following we give a simulator that simulates all $t' + t'' + t'''$ probes using only $t'' + t'''$ input shares of each input sharing. We split the simulator into two sub-simulators: the first one simulates the $t' + t''$ probes in **SWMult** and in **Comp**; the second one simulates the t''' probes in **SplitRed** and the values that the first simulator requires.

With Corollary 3, it follows that the $t' + t''$ probes in **SWMult** and in **Comp** can be simulated with at most t'' output probes of each **SplitRed**. Since **SplitRed** is a t-NI gadget (Lemma 5), there exists another simulator such that the t'' output probes and the t''' internal probes can be simulated with at most $t'' + t'''$ shares of each input sharing. The composition of both simulators results in a simulator that simulates all $t' + t'' + t'''$ probes using only $t'' + t'''$ input shares of each input sharing and this proves the claim of the theorem. □

A.2 Fault-invariance

In the previous section we have proven the (S)NI properties of our gadgets. As proven in Corollary 1, it is sufficient to show the (S)NI property and fault-invariance to prove the fr(S)NI property. Next, we show that all our gadgets are fault-invariant.

Theorem 17. *All gadgets of our compiler are fault-invariant with respect to $\mathcal{F}^+$.*

Proof. Since all faults $\zeta \in \mathcal{F}^+$ can be described with a $a = a_\zeta \in \mathbb{F}$ such that the faulted value is $\zeta(x) = x + a$ we can use the commutative property of '+'. In detail it holds for any $x, x' \in \mathbb{F}$ that

$$x + \zeta(x') = x + x' + a = x + a + x' = \zeta(x) + x'.$$

This means that gadgets only using addition gates are fault-invariant with respect to $\mathcal{F}^+$ because we can step by step move such faults to an input or output values and this is independent from the inputs. Hence, the share-wise addition gadget is fault-invariant.

Next we consider the multiplication of value x with a public constant c. Here, we have for $\zeta \in \mathcal{F}^+$ with $\zeta(x) = x + a$ that

$$c \cdot \zeta(x) = c \cdot (x + a) = (c \cdot x) + (c \cdot a) = \zeta'(c \cdot x)$$

where $\zeta' \in \mathcal{F}^+$ with $\zeta'(x) = x + (c \cdot a)$. As we consider multiplication with a public and fix value c, the function ζ' can be easily derived from ζ and c. We can ignore faults on c with $\zeta(c)$ here[10] because the circuit consists of constant transformation gates that compute $a \cdot x + b$ for input x and constants a, b. This means we can also move such faults step by step to the inputs and outputs. This proves the fault-invariance of all gadgets except our multiplication gadget, which uses multiplication gates with two sensitives inputs.

A close inspection of the multiplication gadget (Alg. 6) shows that the only operations that are not a constant multiplication or addition are the four share-wise multiplications in line 3 to 6. Note that all of these four share-wise multiplications can be run in parallel and do not depend on each other. Hence, there are no intermediate values computed in during the computation of these share-wise multiplications. It is thus sufficient to consider the values immediately before and immediately after this multiplication, which were produced by addition and constant multiplication gates. All intermediate faults before the share-wise multiplication can thus be moved step by step to an equivalent fault on the input value and all intermediate faults after the share-wise multiplication can be moved step by step to an equivalent fault on any intermediate value that depends on the original faulted value.

Hence, we have shown for every faulted gadget that any intermediate value $\tilde{f}^R(x_0, x_1, \dots x_{m-1})$ can be written as $\zeta(f^R(\zeta_0(x_0), \zeta_1(x_1), \dots \zeta_{m-1}(x_{m-1})))$ for any input $(x_0, x_1, \dots x_{m-1})$ and randomness R when $f^R(x_0, x_1, \dots x_{m-1})$ is the according intermediate value of the unfaulted gadget. □

A.3 Fault-Resilience

In the following, we study the fault-resilience of our gadgets. We first analyze all algorithms on themself and then combine the analysis of the parts to obtain Theorem 24 that sums up the fault-resilience of a single iteration of our multiplication gadget **Mult**. Our major workhorse will be the fundamental lemma that allows to relate the number of faults with the degree of the underlying polynomial.

Fault-Resilience of ZEnc In order to guarantee fault-resilience, we will show that every change to the output of Algorithm 3 introduced by s *internal* faults can also be achieved by s direct faults on the output. For $k = 0, \dots, n-1$, we denote the value $\sum_{j=1}^{k} r_j \cdot \alpha_i^j$ by $e_i[k]$. Now, an attacker can use the faults to fault a set $J \subseteq \{1, \dots, t\}$ of random values, i.e., r_j is faulted to $r_j + \Delta_j$ for $j \in J$. Furthermore, they can fault a set $I \subseteq \{1, \dots, t\} \times \{0, \dots, n-1\}$ of values $e_i[k]$,

[10] We could also add such faults in the wfr(S)NI setting as shown in the proof of Corollary 8

i.e., these values are now $e_i[k] + \Delta_{i,k}$. For two sets J and I, let $D_{J,I}$ be the output distribution of Algorithm 3 if the values are faulted according to these fault sets.

Lemma 6. *For all $J \subseteq \{1, \ldots, t\}$ and all $I \subseteq \{1, \ldots, t\} \times \{0, \ldots, n-1\}$, we have $D_{J,I} \equiv D_{\emptyset,I}$.*

Proof. For $(e_i)_{i\in[n]} \leftarrow\!\!\$\ D_{J,I}$, it is easy to see that

$$e_i = \sum_{j \notin J} \left[r_j \alpha_i^{(j)} \right] + \sum_{j \in J} \left[(r_j + \Delta_j) \alpha_i^{(j)} \right] + \sum_{(i,k)\in I} \Delta_{i,k}.$$

Let r'_j with $r'_j = r_j$ for $j \notin J$ and $r'_j = r_j + \Delta_j$ for $j \in J$ with

$$e_i = \sum_{j \in [n]} \left[r'_j \alpha_i^{(j)} \right] + \sum_{(i,k)\in I} \Delta_{i,k}.$$

Since r_i is i.i.d., it holds $r'_j \equiv r_j$, and hence

$$e_i \equiv \sum_{j \in [n]} \left[r_j \alpha_i^{(j)} \right] + \sum_{(i,k)\in I} \Delta_{i,k}.$$

This implies the statement. □

Lemma 7. *For all $I \subseteq \{1, \ldots, t\} \times \{0, \ldots, n-1\}$, there exists a sharing $(f_i)_{i\in[n]}$ with $|\operatorname{supp}(f_i)_{i\in[n]}| \leq |I|$ such that $D_{\emptyset,I} = D_{\emptyset,\emptyset} + (f_i)_{i\in[n]}$.*

Proof. A short calculation shows that for $(e_i)_{i\in[n]} \leftarrow\!\!\$\ D_{\emptyset,I}$ we have

$$e_i = \sum_{j} \left[r_j \alpha_i^{(j)} \right] + \sum_{(i,k)\in I} \Delta_{i,k}.$$

Now, define $f_i = \sum_{(i,k)\in I} \Delta_{i,k}$. Clearly, $(e_i)_{i\in[n]} - (f_i)_{i\in[n]}$ is distributed according to $D_{\emptyset,\emptyset}$ and, furthermore, $|\operatorname{supp}(e_i)_{i\in[n]}| \leq |I|$. □

A combination of these lemmata implies the following theorem about the fault-resilience of **ZEnc**, which allows us to concentrate only on outputs faults of **ZEnc**.

Theorem 18. *Every change to the output of **ZEnc** introduced by s* internal *faults can also be achieved by s direct faults on the output.*

Since the gadget **sZEnc** is only a share-wiese addition of **ZEnc** the claim of Theorem 18 also hold for **sZEnc**

Theorem 19. *Every change to the output of **sZEnc** introduced by s* internal *faults can also be achieved by s direct faults on the output.*

Fault-Resilience of Refresh With the fault resilience of **sZEnc** follows the fault resilience of our refresh gadget.

Theorem 20 (Refresh). *The gadget* G'_{G} *(Alg. 5) with identity* G *is an e-fault-robust w.r.t.* $\mathcal{F}^{ind}$ *refresh gadget*

Proof. Due to Theorem 19 each fault can be transformed to a output fault. Hence the e-fault-robustness follows from the security of the underlying encoding. Hence, the gadget is e fault robust if $n \geq d + e + 1$. □

Fault-Resilience of SplitRed To analyze the fault-resilience of **SplitRed**, we first notice that all operations before Line 8 are simple shared-wise addition operations or calls to **ZEnc**. Theorem 18 implies that we can ignore faults that are internal to **ZEnc**. Hence, we can assume that the first time that a fault is introduced in the run of **SplitRed** is in Line 8, i.e., in the computation of ${\mathcal{F}'}_i^{(j)}$ by faulting the share F_j to the value $F_j + \Delta_{i,j}$ for a set $J \subseteq [n]^2$. All operations afterwards are share-wise additions (Theorem 6), which are easily fault-resilient, i.e., all faults can be performed after the additions w.l.o.g. Hence, the remaining faults are described by two sets $I', I'' \subseteq \{0, \ldots, n-1\}$ where the value of F'_i is replaced by $F'_i + \Delta'_i$ for $i \in I'$ and the value of F''_i is replaced by $F''_i + \Delta''_i$ for $i \in I''$. For the sake of readability, we expand the definition of the values Δ_i also for non-faulted shares: Let $\bar{\Delta}'_i = 0$ for $i \notin I'$ and $\bar{\Delta}'_i = \Delta_i$ for $i \in I'$, and, similarly, be $\bar{\Delta}''_i = 0$ for $i \notin I''$ and $\bar{\Delta}''_i = \Delta_i$ for $i \in I''$. For simplicity, let $J_i = \{j \mid (i,j) \in J\}$. We first show the following useful lemma.

Lemma 8. *Let* $J \subseteq [n]$ *and let* $\Lambda_J = (\hat{\lambda}_j^{(i)})_{i,j}$ *be a matrix containing the* $\hat{\lambda}_j^{(i)}$ *for* $i \in [n]$ *and* $j \in J$*. Then* $\operatorname{rank}(\Lambda_J) = \min\{|J|, e+1\}$.

Proof. Consider the inverse Vandermonde matrix $V_{n,n}^{-1}$ that has full rank n. Now, note that each column $(\hat{\lambda}_j^{(i)})_i$ in Λ_J can be written as $A(\lambda_j^{(i)})_i$ for a column $(\lambda_j^{(i)})_i$ of $V_{n,n}^{-1}$. Here, A is obtained by taking the the original Vandermonde matrix $V_{n,n}$ ad replacing all entries in the first d columns with 0. Clearly, for $|J| \leq e$, we have $\operatorname{rank}(A) = |J|$ and for $|J| \geq e$, we have $\operatorname{rank}(A) = \operatorname{rank}(V_{n,n}) - d = n - d = e + 1$. As $\Lambda_J = A \cdot V_{n,n}^{-1}$, we have $\operatorname{rank}(\Lambda_J) = \min\{|J|, e+1\}$. □

Hence, the total result of the computation is

$$
\begin{aligned}
F'_i &= \sum_{j \in [n/2]} \left[{g'}_i^{(j)} \right] + \sum_{j \in [n/2], j \notin J_i} \left[{\mathcal{F}'}_i^{(j)} \right] + \sum_{j \in [n/2], j \in J_i} \left[{\mathcal{F}'}_i^{(j)} \right] + \bar{\Delta}_i \\
&= \sum_{j \in [n/2]} \left[{g'}_i^{(j)} \right] + \sum_{j \in [n/2], j \notin J_i} \left[\hat{\lambda}_j^{(i)} F_j \right] + \sum_{j \in [n/2], j \in J_i} \left[\hat{\lambda}_j^{(i)} (F_j + \Delta_j) \right] + \bar{\Delta}_i \\
&= \sum_{j \in [n/2]} \left[{g'}_i^{(j)} \right] + \sum_{j \in [n/2], j \notin J_i} \left[\hat{\lambda}_j^{(i)} F_j \right] + \sum_{j \in [n/2], j \in J_i} \left[\hat{\lambda}_j^{(i)} (F_j + \Delta_j) \right] + \bar{\Delta}_i \\
&= \sum_{j \in [n/2]} \left[{g'}_i^{(j)} \right] + \sum_{j \in [n/2]} \left[\hat{\lambda}_j^{(i)} F_j \right] + \sum_{j \in [n/2], j \in J_i} \left[\hat{\lambda}_j^{(i)} \Delta_j \right] + \bar{\Delta}_i.
\end{aligned}
$$

As a single fault value Δ_j might now influence all shares F'_i, we can not simply define a corresponding fault polynomial as in the proof of Theorem 18. Nevertheless, our choice of $\hat{\lambda}_j^{(i)}$ will enable us still argue the fault resilience of the construction. Applying the interpolation lemma to the faulted values shows that the induced fault polynomial Δ has the form

$$\Delta(x) = \sum_{i\in[n]} \sum_{j\in J_i} \Delta_j(x) + \sum_{i\in I} \bar{\Delta}_i(x),$$

where each $\Delta_j(x)$ has the form

$$\Delta_j(x) = \sum_{k\geq 0} \left[\sum_{i\in[n]} \lambda_i^{(k)} \hat{\lambda}_j^{(i)} \Delta_j \right] x^k$$

and each $\bar{\Delta}_i(x)$ has the form

$$\bar{\Delta}_i(x) = \sum_{k\geq 0} \left[\sum_{i\in[n]} \lambda_i^{(k)} \bar{\Delta}_i \right] x^k.$$

Now, the fundamental lemma says that $\deg(\sum_{i\in I} \bar{\Delta}_i(x)) \geq n - |I|$. Furthermore, Lemma 8 implies that $\deg(\Delta) \geq n - |J|$.

Claim. We have $\deg(\Delta) \geq n - |J|$ for $J \neq \emptyset$.

Proof. Suppose that $\deg(\Delta) < n - |J|$. Due to the interpolation theorem, we have for each $k > \deg(\Delta)$ that

$$\left[\sum_{i\in[n]} \sum_{j\in J_i} \lambda_i^{(k)} \hat{\lambda}_j^{(i)} \Delta_j \right] = 0.$$

As all of these equations are linear in the variables Δ_j, we can rewrite these $n - \deg(\Delta) > |J|$ constraints into a linear equation system with constraint matrix A. Here, $A = V_{n,n}^{-1} \cdot \Lambda_{J'}$, where $\Lambda_{J'} = (\hat{\lambda}_j^{(i)})_{i,j}$ is the matrix containing the $\hat{\lambda}_j^{(i)}$ for $i \in [n]$ and $j \in J'$ where $J' = \{j \mid \exists i : j \in J_i\}$. Now, Lemma 8 shows that $\operatorname{rank}(\Lambda_{J'}) = \min\{|J'|, e+1\}$, and as $\operatorname{rank}(V_{n,n}^{-1}) = n$, we have $\operatorname{rank}(A) = |J'|$, as $|J| \leq e$. The rank-nullity theorem then implies that $\dim(\ker(A)) = 0$, hence $\ker(A) = \{0\}$. Hence, $\Delta_j = 0$ for all $j \in J$, which is a contradiction to the assumption that $J \neq \emptyset$. □

As a consequence of our analysis, using s_{int} faults against a polynomial containing s_{input} input faults will always result in a polynomial of degree $n - s_{\text{int}} - s_{\text{input}}$.

Theorem 21. *If the input to **SplitRed** has degree at least $n - s_{input}$, the outputs have degree at least $n - s_{int} - s_{input}$, if s_{int} faults are performed during the computation.*

Fault-Resilience of Comp and SWMult As both **Comp** and **SWMult** are share-wise algorithms, i.e., they operate on each share independently, faults from the input polynomials will directly transfer to faults of the output polynomial of the gadgets, as shown in Theorem 6.

Theorem 22. *If the input to* ***Comp*** *has degree at least $n - s_{input}$, the outputs have degree at least $n - s_{int} - s_{input}$, if s_{int} faults are performed during the computation.*

Theorem 23. *If the input to* ***SWMult*** *has degree at least $n - s_{input}$, the outputs have degree at least $n - s_{int} - s_{input}$, if s_{int} faults are performed during the computation.*

Fault-Resilience of Mult (Algorithm 6) In the following, we focus on the *polynomial representation* of the involved sharings. Consider **Mult** (Algorithm 6) for this, which has inputs $f(x) = \sum_{i=0}^{n} f_i x^i$ and $g(x) = \sum_{i=0}^{n} g_i x^i$. In the first line, **SplitRed** (Algorithm 7) outputs two polynomials f' and f'' such that

$$f'(x) = f'_0 + \sum_{k=1}^{t} r_k^{(f')} x^k + \sum_{k=1}^{d} \frac{n}{2} r_k^{(f)} x^k + \sum_{k=d+1}^{n} f'_k x^k$$
$$f''(x) = f''_0 + \sum_{k=1}^{t} r_k^{(f'')} x^k - \sum_{k=1}^{d} \frac{n}{2} r_k^{(f)} x^k + \sum_{k=d+1}^{n} f''_k x^k$$

with the following properties:

(*) Summing up f'_k and f''_k gives the original value f_k for $k \in \{0, d+1, d+2, \ldots, n\}$

(**) The values $r_k^{(f')}$, $r_k^{(f'')}$, and $r_k^{(f)}$ are i.i.d. Note that both f' and f'' contain the same terms $\sum_{k=1}^{d} \frac{n}{2} r_k^{(f)} x^k$.

Similarly, the second line uses **SplitRed** (Algorithm 7) to output two polynomials g' and g'' such that

$$g'(x) = g'_0 + \sum_{k=1}^{t} r_k^{(g')} x^k + \sum_{k=1}^{d} \frac{n}{2} r_k^{(g)} x^k + \sum_{k=d+1}^{n} g'_k x^k$$
$$g''(x) = g''_0 + \sum_{k=1}^{t} r_k^{(g'')} x^k - \sum_{k=1}^{d} \frac{n}{2} r_k^{(g)} x^k + \sum_{k=d+1}^{n} g''_k x^k.$$

If we allow faults during the computations of **SplitRed** (Algorithm 7), this possibly introduces different polynomials at different positions. Instead of computing $f' \cdot g'$, Line 3 in **Mult** (Algorithm 6) would thus compute $(f' + \Delta^{(f',g')}) \cdot (g' + \Delta^{(g',f')})$ for fault-induced polynomials $\Delta^{(f',g')}$ and $\Delta^{(g',f')}$. Similarly, Line 4 computes $(f' + \Delta^{(f',g'')}) \cdot (g'' + \Delta^{(g'',f')})$, Line 5 computes $(f'' + \Delta^{(f',g'')}) \cdot (g'' + \Delta^{(g'',f')})$, and Line 6 computes $(f'' + \Delta^{(f'',g'')}) \cdot (g'' + \Delta^{(g'',f'')})$. Note that all operations in **SplitRed** (Algorithm 7) and in **ZEnc** (Algorithm 3) are random

sampling, addition of shares, and multiplication of shares by public constants (α_i^j in Algorithm 3 or $\hat{\lambda}_j^{(i)}$ in Algorithm 7).

While the relation between these fault-induced polynomials might be complicated, we know that the degree of all fault-induced polynomials $\Delta^{(\cdot,\cdot)}$ is at least $d+1$ (or equal to 0, if no fault was induced) by assumption. Denote the k-th coefficient of $\Delta^{(\cdot,\cdot)}$ by $\Delta_k^{(\cdot,\cdot)}$.

We now analyse the polynomial

$$\begin{aligned} h = \\ &(f' + \Delta^{(f',g')}) \cdot (g' + \Delta^{(g',f')}) + \\ &(f' + \Delta^{(f',g'')}) \cdot (g'' + \Delta^{(g'',f')}) + \\ &(f'' + \Delta^{(f',g'')}) \cdot (g'' + \Delta^{(g'',f')}) + \\ &(f'' + \Delta^{(f'',g'')}) \cdot (g'' + \Delta^{(g'',f'')}), \end{aligned}$$

corresponding to the output polynomial of Algorithm 6.

Lemma 9. *If no faults are introduced, $h(0) = f(0) \cdot g(0)$ and $\deg(h) \leq d$.*

Proof. Properties (*) and (**) imply that a non-faulted computation of Algorithm 6 on non-faulted inputs f and g (i.e., inputs with degree at most d) outputs a non-faulted polynomial (that represents the correct value). □

Similar to the discussion above, we will now show that h is constructed from a polynomial of uniformly random distributed higher-order coefficients.

Theorem 24. *The gadget **Mult** is e-fault-robust.*

Proof. Consider the coefficient $d+1$ of h, which equals the sum of the coefficients $d+1$ of the four polynomials, that themself are products of polynomials.

Let us first take a look at the coefficient $d+1$ of the first polynomial $(f' + \Delta^{(f',g')}) \cdot (g' + \Delta^{(g',f')})$. The coefficient $d+1$ of this polynomial is given by the sum of products

$$\begin{aligned} &(f'_0 + \Delta_0^{(f',g')}) \cdot (g'_{d+1} + \Delta_{d+1}^{(g',f')}) + \\ &(r_1^{(f')} + \frac{n}{2} r_1^{(f)} + \Delta_1^{(f',g')}) \cdot (\frac{n}{2} r_d^{(g)} + \Delta_d^{(g',f')}) + \\ &\quad \cdots \\ &(r_t^{(f')} + \frac{n}{2} r_t^{(f)} + \Delta_t^{(f',g')}) \cdot (\frac{n}{2} r_{t+1}^{(g)} + \Delta_{t+1}^{(g',f')}) + \\ &(\frac{n}{2} r_{t+1}^{(f)} + \Delta_{t+1}^{(f',g')}) \cdot (r_t^{(g')} + \frac{n}{2} r_t^{(g)} + \Delta_t^{(g',f')}) + \\ &\quad \cdots \\ &(\frac{n}{2} r_d^{(f)} + \Delta_d^{(f',g')}) \cdot (r_1^{(g')} + \frac{n}{2} r_1^{(g)} + \Delta_1^{(g',f')}) + \\ &(f'_{d+1} + \Delta_{d+1}^{(f',g')}) \cdot (g'_0 + \Delta_0^{(g',f')}). \end{aligned}$$

The coefficients for the other three polynomials can be derived similarly.

Now, consider the terms of the coefficient $d+1$ of h that involve the random coefficient $r_1^{(f')}$, which are

$$(r_1^{(f')} + \frac{n}{2} r_1^{(f)} + \Delta_1^{(f',g')}) \cdot (\frac{n}{2} r_d^{(g)} + \Delta_d^{(g',f')}) +$$
$$(r_1^{(f')} + \frac{n}{2} r_1^{(f)} + \Delta_1^{(f',g'')}) \cdot (-\frac{n}{2} r_d^{(g)} + \Delta_d^{(g'',f')}).$$

Rearranging these terms shows that they are a sum of terms involving the term $r_1^{(f')} \cdot (\Delta_d^{(g',f')} + \Delta_d^{(g'',f')})$. For the sake of simplicity, we denote the error polynomial involving $\varphi \in \{f', f'', g', g''\}$ as $\Delta[\varphi]$, e.g., $\Delta[f'] = \Delta^{(g',f')} + \Delta^{(g'',f')}$.

Now, if the coefficient $\Delta_d[f']$ of $r_1^{(f')}$ is non-zero, the coefficient $d+1$ of h is uniformly distributed, as $r_1^{(f')}$ is uniformly distributed and does not appear anywhere else in the coefficient $d+1$ of h. Following the same line of reasoning, if a coefficient $\Delta_{d+i}[f']$ is non-zero, it is paired into the coefficient $d+i+j$ of h with the random value $r_j^{(f')}$ for all $j \geq 1$. Similar arguments hold for the coefficients of $r_i^{(f'')}$, $r_i^{(g')}$, and $r_i^{(g'')}$ and their corresponding error coefficients $\Delta_{d+1}[f'']$, $\Delta_{d+1}[g']$, and $\Delta_{d+1}[g'']$.

Now, consider any $\varphi \in \{f', f'', g', g''\}$ and the corresponding error polynomial $\Delta[\varphi]$. By assumption, at most $s \leq e$ faults were used in the run of Algorithm 6 to produce $\Delta[\varphi]$, The fundamental lemma then implies that $\deg(\Delta[\varphi]) \geq d+e+1-s$. As $s \leq e$, there is at least one index $i \in \{0, \ldots, e\}$ such that the coefficient $d+i$ of $\Delta[\varphi]$ is non-zero. Ergo, the coefficients $d+i+j$ of h are uniformly distributed for all $j \geq 1$. □

B Additional Examples and Motivations

B.1 Fault-sets $\mathcal{F}$

In this work we consider *wire independent faults* $\mathcal{F}^{ind} := \{\text{all functions } \zeta \colon \mathbb{F} \to \mathbb{F}\}$ and *additive faults* $\mathcal{F}^{+} := \{\zeta : \zeta(x) = x + a \text{ for all } a \in \mathbb{F}\}$ that fault the wires value by adding an arbitrary value. It is easy to see that

$$\mathcal{F}^{+} \subset \mathcal{F}^{ind} .$$

Assuming the $\mathbb{F}_{2^n}$ this means that we can arbitrary flip the bits of an element $a \in \mathbb{F}_{2^n}$ with additive faults. However, there are also other attacks such as set, attacks those attacks are covered by $\mathcal{F}^{ind}$. Wire independent faults can do even more when we consider larger fields. For example it could set the first half bits of a to zero iff a has some special properties (e.g. the hamming weight is 2). Further, assuming fields other fields (e.g. prime fields). It is clear that additive faults do not describe bit flips. But the wire independent faults can also do bit-wise faults such as flips and set-faults.

B.2 Threshold Probing model

In this section we discuss the limitations of the composition results in the threshold probing model.

Probe-Isolating Non-Interference (PINI). This definition give the best known composition results in the threshold model. The high-level idea is that the definition describes the index relation of the probed output wires and the input wires used for the simulation of the probes.

Definition 10 (PINI [CS20]). *A gadget with m input shares $(x_i^{(j)})_{i\in[n]}$ with $j \in [m]$ and l output shares $(y_i^{(j)})_{i\in[n]}$ with $j \in [l]$ is PINI if for any $t_1 \in N$, any set of t_1 intermediate variables and any subset O of output indices, there exists a subset $I \subset [n]$ of input indices with $|I| \leq t_1$ such that the t_1 intermediate variables and all output shares $(y_i^{(0)})_{i\in O}, (y_i^{(1)})_{i\in O}, \dots (y_i^{(l-1)})_{i\in O}$ can be perfectly simulated from the input shares $(x_i^{(0)})_{i\in O\cup I}, (x_i^{(1)})_{i\in O\cup I}, \dots (x_i^{(m-1)})_{i\in O\cup I}$.*

Note that share-wise operation fulfill the PINI definition. However it is quite clear that the composition of such gadgets do not give region-probing security. A trivial example is the composition of two share-wise copy gadgets for $d + 1$ shared secrets. If the adversary is allowed to probe d wires in total, the probes are still independent of the secret. However, if the adversary probes d wires ind each gadget (region), it can probe all shares and reconstruct the secret.

Probe-Isolating Non-Interference (PINI). The SNI Property defined in the main body does not provide as good composition results as PINI when we consider the threshold probing model. However, it already gives gives an countermeasure to the attack describes in the PINI paragraph. In detail it showes that the number of

the simulator's required input values only depend on the internal probes, and not on the output probes. Unfortunately it is still not efficient to allow compositions in the region probing model. As an example we consider a d-SNI multiplication, where both outputs are refreshed by a d-SNI refresh gadgets. Assume we probe d' values in the multiplication gadget, and we probe d'' values in each refresh gadget such that the simulator of both gadgets requires d'' input values. Hence, (due to the composition argument of SNI), the simulator of the multiplication gadget has to simulate the d' probes in the multiplication, the d'' input values of the first refresh and the d'' input values of the first refresh. In total the simulator of the multiplication gadget has to simulate $d' + 2d''$ probes. In the threshold model this works since we assume give a threshold for the whole circuit $d' + 2d'' \leq d$, and the simulator of the multiplication gadget can simulate all $d' + 2d''$ values due to the SNI property. This is not the case if the adversary can probe up to d values in each gadget (region). Here, the adversary can place the probes in such a way that the simulator of the multiplication gadget has to simulate $d' + 2d'' = 3d$ probes. It is obvious that the d-SNI property does not provide such a simulator. Note that even the assumption that the adversary only probes $d/2$ bits in each region (as we do in Theorem 9) leads to $d' + 2d'' > d$.

B.3 Leakage resilience of Error-detection codes

In the section we compare the random probing security of the polynomial sharing used in this paper with the duplicated sharing. Therefore we analyze the encoding itself without further computational leakage and assume a leakage probability p. Let e be the number of allowed faults and d the number of allowed probes. The duplicated sharing first shares a secret into $d+1$ random values x_i with $\sum_{i=0}^{d} x_i = x$, and then it copies it generates $e+1$ copies such that $x_i^j = x_i$ for all $i \in [d]$ and $j \in [e]$. It follows (assuming a leakage probability of p) that we learn x_i with a probability of $1-(1-p)^e$ because each x_i^j leaks with probability p. So the probability that we learn the secret x is the probability that we learn all x_i, i.e.,

$$(1-(1-p)^{e+1})^{d+1}.$$

The polynomial sharing used in this work has only $d+e+1$ shares and the adversary learns the secret if at least $d+1$ values are leaked. This probability is

$$\sum_{i=d+1}^{e+d+1} \binom{e+d+1}{i} p^i (1-p)^{e+d+1-i}.$$

As a reference we can use a sharing without any redundancy. Hence it has only $d+1$ shares and the secret leaks with probability p^{d+1}. In Figure 3 we illustrate the improvement of polynomial sharing compared to the duplicated one when we consider $e, d = O(n)$. Note that the figure uses a logarithmic scale, otherwise the graphs would divagate exponential.

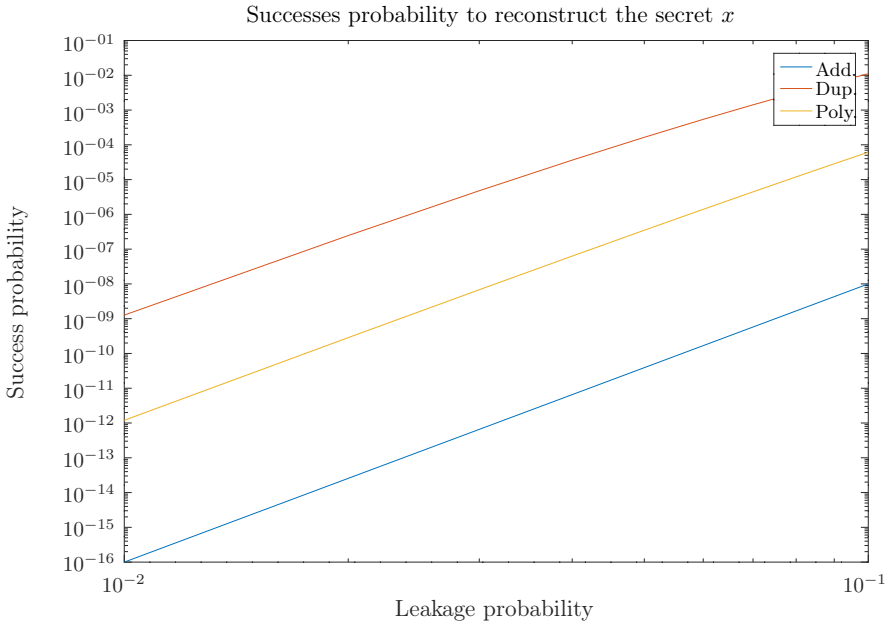


Fig. 3: The Figure gives the success probability to reconstruct the secret of a single encoding with $e = d = 4$

B.4 Proof of Theorem 10 – Optimality of $n = d + e + 1$

If one assumes that the values are polynomially masked, this is relatively easy to see. To protect against d probes, the underlying polynomial needs to have degree at least d. If we use strictly less than $d + e + 1$ shares, an attacker can fault e of these shares, which corresponds to adding a polynomial of degree strictly less than $d + e + 1 - e = d + 1$ to the sharing. Hence, the attacker can modify the valid sharing, described by a polynomial of degree d, by adding a polynomial of degree d and thus obtain a *valid* sharing of a different values. Hence, there is no possible way to detect these modifications and fault-resilience is thus not possible.

One might now wonder whether this is an artifact of the polynomial sharing or whether it is inherently impossible to use a lower number of shares. We will show that the latter is true by making use of known bounds from coding theory. An $(n, m, dist)$-*code* C over $\mathbb{F}$ is a subset $C \subseteq \mathbb{F}^n$ with $|C| = m$ such that for all $c, c' \in C$, we have $d_H(c, c') \geq \text{dist}$, where d_H denotes the *Hamming distance* of two strings, i.e., the number of positions where these strings differ. An abstract way of looking at a sharing of a value $v \in \mathbb{F}$ is to see this sharing as a code over $\mathbb{F}$. With this perspective, the polynomial sharing used in this work is easily seen to be a Reed-Solomon-Code. Suppose that we use a (n, m, dist)-code C over $\mathbb{F}$ to share our values of $\mathbb{F}$. Clearly, to prevent an attacker that is able to introduce e faults

from producing a valid codeword (sharing), we need dist $\geq e+1$. Furthermore, as we want to protect against d probes. As each probe reveals a value of $\mathbb{F}$, we need at least $m \geq q^{d+1}$ codewords: If we have less codewords, d probes will result in an uncertainty clearly less than q and thus provide the attacker with some information about the shared value. We can now apply the so called *Singleton bound*.

Theorem 25 (Theorem 5.2.1 in [Lin98]). *Let C be an $(n, m, dist)$-code over $\mathbb{F}$. Then $m \leq q^{n-dist+1}$.*

Based on the inequalities dist $\geq e+1$ and $m \geq q^{d+1}$ derived earlier, we immediately obtain the bound $q^{d+1} \leq m \leq q^{n-\text{dist}+1} \leq q^{n-e-1+1} = q^{n-e}$. Rearranging these terms shows that $n \geq d+e+1$ and implies the following theorem, which shows that the number of used shares in our approach is optimal.

B.5 Attack on the construction in [SFRES18]

In this section we attack the refresh gadget in [SFRES18]. The authors claim to have a d-SNI secure refresh gadget depicted in Algorithm 11. Next, we give an example to illustrate that this is actually not the case. Let us consider the field $\mathbb{F}_p$ with a (large enough) prime number p and $d = 3$ where $\alpha_0 = 1$ and $\alpha_1 = -1 (\bmod p)$ and $\alpha_2 \notin \{-1, 0, 1\}$ arbitrary. Let us consider the output probes y_0, y_1, and the intermediate probe r_2. With the SNI property would follow that there is a simulator that simulates all probes only using one input share. However we will show that the simulator needs x_0 **and** x_1. It holds

$$y_0 = r_1 \cdot (1)^1 + r_2 \cdot (1)^2 + r_3 \cdot (-1)^3 + x_0 \qquad \text{and}$$
$$y_1 = r_1 \cdot (-1)^1 + r_2 \cdot (-1)^2 + r_3 \cdot (-1)^3 + x_0 \,.$$

When we add both shares, we get $y_0 + y_1 = x_0 + x_1 - 2 \cdot r_2$, and we can compute $x_0 + x_1 = y_0 + y_1 + 2 \cdot r_2$. Hence the adversary can compute $x_0 + x_1$ if it probes y_0, y_1 and r_2. It is easy to see that the simulator of y_0, y_1 and r_2 needs the inputs x_0 **and** x_1.

Algorithm 11 *Refresh* given in [SFRES18]

Input: A $(x_i)_{i\in[n]}$
Output: A randomized (n,t)-Encoding $(y_i)_{i\in[n]}$ of $\mathsf{Dec}((y_i)_{i\in[n]})$.

1: $(y_i)_{i\in[n]} \leftarrow (x_i)_{i\in[n]}$
2: **for** $j = 1$ **to** d **do**
3: $\quad r_j \leftarrow_\$ \mathbb{F}$
4: $\quad$ **for** $i = 0$ **to** $n-1$ **do**
5: $\quad\quad y_{i+1} \leftarrow y_i \oplus r_j \alpha_i^j$
6: **return** $(y_i)_{i\in[n]}$

B.6 Non fault resilient SNI gadget

In this section we illustrate that not every SNI gadget stays SNI if an adversary injects Faults. For example, the refresh gadget given in Algorithm 12. Let us consider a probe on the output y_0 and an additive fault $\zeta(x) = x - 1$ on the input wire when the gadget computes $r_0 r_1$. Hence, we get

$$((r_0\zeta(r_1)) + (r_0(1 - r_1))) + x_0 = ((r_0(r_1 - 1)) + (r_0(1 - r_1))) + x_0 = x_0$$

and the gadget is not SNI anymore because the simulator needs x_0 if it has to simulate y_0 of the faulted refresh. Mote that this only describes an attack that brakes our stronger frSNI definition. However, this is only an example for the intuition. Next we give an example where we can even break the weaker notion. In detail we construct a secure gadget that is (S)NI but can be broken with a single fault: Therefore, we use the same technique as before. Let

$$f(r', r'', r) = r'r + r(r'')$$

be a sub gadget. It is easy to see that $f(r', 1-r', r) = r$ and $f(r', r', r) = 0$. So let $\mathsf{G}_{r_0,r_1,\ldots,r_m}$ be a (S)NI secure gadget with the internal randomness $r_0, r_1, \ldots, r_m$ to mask the computation. Let us consider a gadget that does the following.

$r', r_0, r_1, \ldots, r_m \leftarrow_\$ \mathbb{F}$
$r'' \leftarrow 1 - r'$
$r_i \leftarrow f(r', r'', r_i)$ for all $i = 0, 1, \ldots m$
run $\mathsf{G}_{r_0,r_1,\ldots,r_m}$ and also output its output

Clearly, it is easy to see that this gadget is (S)NI as well. Note that the probes on r' and r'' do not affect the r_i and we get the same probing security as $\mathsf{G}_{r_0,r_1,\ldots r_m}$. But a single fault $\zeta(r'')$ with $\zeta(x) = x - 1$ sets all r_i to zero. This means that whole gadget becomes deterministic because it always computes $\mathsf{G}_{0,0,\ldots 0}$. Hence the construction is not SNI only with only one fault. In other word this shows that we even break our wfr(S)NI definition. Note that the given example is artificial, but it illustrates two things:

- The same random value should not effect the security of to many shares,
- and the we have to be careful when we consider faults in large circuits since large circuits might produce unexpected randomness relations.

Algorithm 12 $wRefresh$

Input: A additive sharing (x_0, x_1)
Output: A randomized sharing (y_0, y_1) with $x_0 + x_1 = y_0 + y_1$.

1: $r_0, r_1, r_2 \leftarrow_\$ \mathbb{F}$
2: $y_0 \leftarrow ((r_0 r_1) + (r_0(1 - r_1))) + x_0$
3: $y_1 \leftarrow -r_1 + x_1$
4: **return** (y_0, y_1)

B.7 Proof of Theorem 1

Proof. We start with the proof for d-frSNI compositions. Let C be a composition of two d-f(S)NI gadgets G_0, G_1 with respect to $\mathcal{F}$ and let T be any fault attack $T \in \mathsf{A}(\mathcal{F})$. Since $\mathcal{F} \subseteq \mathcal{F}^{ind}$ only allows independent faults on each wire, we can split the complete attack T into gadget-wise attacks T_0, T_1. Hence, $T[\mathsf{C}]$ can be described as the composition of the (independently) faulted gadgets $T_0[\mathsf{G}_0]$,$T_1[\mathsf{G}_1]$. Due to the definition of fr(S)NI both faulted gadgets are still SNI and it follows that the composition of the faulted gadgets $T[\mathsf{C}]$ is also SNI for any $T \in \mathsf{A}(\mathcal{F})$ due to the composition result of SNI. The proof for d-wfrSNI compositions is similar to the proof of d-frSNI, only the fault attack is limited and the faults are counted as probes. □

B.8 Useful properties for $frSNI$

It is not hard to see that the stronger definition implies the weaker one.

Corollary 4. *Any d-fr(S)NI Gadget with respect to a fault set $\mathcal{F}$ is a d-wfr(S)NI gadget with respect to $\mathcal{F}$.*

Proof. If the gadget is secure even in the presence of arbitrary many faults it is also secure against a limited number of faults. The simulation even might need less input shares for the simulation. □