# My Gadget Just Cares For Me - How NINA Can Prove Security Against Combined Attacks

No Author Given

No Institute Given

**Abstract** Differential Power Analysis and Differential Fault Analysis threaten the security of even the most trustworthy cryptographic primitives. It is important we protect their implementation such that no sensitive information is leaked using side channels and it withstands injected faults or combined physical attacks.

In this work, we propose security notions tailored against advanced physical attacks consisting of both faults and probes on circuit wires. We then transform the security notions to composable security notions. This compositional property eases the verification time of implementations; helps to create secure components by allowing for search methods; and isolates primitives in larger protocols such as modes of operation. We dub our notion *NINA*, which forms the link between the established *Non-Interference (NI)* property and our composable active security property, *Non-Accumulation (NA)*.

To illustrate the NINA property, we use it to prove the security of two multiplication gadgets: an error checking duplication gadget and an error correcting duplication gadget. The NINA proofs for error detecting gadgets capture the effect of Statistical Ineffective Fault Analysis (SIFA), an attack vector which threatens most current masked implementations. Additionally, we study error correcting techniques. We show that error correcting gadgets can attain the *Independent NINA* property. A stronger property which captures a clear separation between the effect of faults and probes. Thus, we show that clever error correcting gadgets improve on error detecting ones by achieving significant higher levels of combined security along with guaranteed output delivery.

**Keywords:** Combined Security · DFA · DPA · Masking · SIFA

## 1 Introduction

Differential Fault Analysis (DFA), proposed by Biham and Shamir in 1997 [8], is an attack on a physical device which effectively reveals the secret key of a cipher using well-placed faults in the encryption procedure. Differential Power Analysis (DPA) is an attack which uses a cryptographic device's power consumption to launch a divide-and-conquer attack on the private key as first described by Kocher et al. in 1999 [25]. To facilitate key-extraction, several physical attacks can be used against the implementation, we differentiate passive, active, and

combined attacks. Passive attacks observe the behaviour of a device during its process, such as observing the process time or the device's power consumption. Active attacks tamper with the device's functioning, such as inducing computational errors by fault injections. Using passive and/or active attacks for either enhanced tampering or observation of the device's reaction to tampering are called combined attacks.

In order to defend against physical attacks without using expensive custom hardware such as shields and detectors it is the algorithm that needs to counteract passive, active, and combined attacks by securing it in a formal security model. Passive adversary models and their corresponding security notions have improved significantly over the last five years, largely due to the introduction of the probing adversary by Ishai et al. [24]. This adversary is capable of reading the exact values on a number of circuit wires. The minimal number of wires the adversary observes to learn a sensitive variable is defined as the order of probing security. Duc et al. showed that the noisy leakage model reduces to the probing model assuming the presence of sufficient noise and independent wire leakage, and more specifically that an implementation's signal to noise ratio is exponentially related to its probing security order [17]. While the probing model helps to verify implementations, the time complexity is exponential in the security order which is therefore not cost effective for larger implementations such as symmetric ciphers. To streamline this verification procedure, Barthe et al. proposed a composable security definition called Strong Non-Interference (SNI) [3]. This approach views circuits as the composition of several components and forms a sufficient security condition, such that when multiple components are linked together the total circuit is probing secure. Composable security definitions allow designers to verify and optimise separate circuit components which are small enough for a brute force verification technique. This technique has been adopted in several tools to quickly verify implementations based on modular designs [4, 7, 15]. The importance of a formal security notion, such as the probing model, includes the need of assurance in high-end secure devices. To guarantee such assurance, the Common Criteria was proposed as an international standard. These criteria specify the security and assurance users can have in their sensitive devices where the strongest criterion requires a target of evaluation to have a verified design which is only possible with formal security notions [19].

Apart from security models, the current literature provides countermeasures against passive attacks. One example is the methodology of Ishai, Sahai and Wagner (ISW) which guarantees protection of arbitrary circuits against passive attacks using the previous discussed probing model [24]. This countermeasure led to further study to increase its security and efficiency [5–7, 9, 14, 18, 21, 32]. Another methodology to secure implementations is described in Threshold Implementations by Nikova et al. [27]. By extensively using the masking scheme's and the cipher's properties, they minimise the countermeasure's latency and randomness costs and, as a result, the method has been used to defend various symmetric primitives [2, 13, 22, 26, 28].

2

Despite having formal security notions and countermeasures against passive attacks, there are only few works which consider active and combined attacks. The first is Private Circuits II [23] which provides a countermeasure where the active adversary is modelled as one who faults a bounded number of wires per clock cycle. By viewing faults as probes, the work naturally offers protection against a combined adversary. However, the implementation of the countermeasure and its efficiency is currently still a challenge [12]. Later on, the work of ParTI [31] proposes to encode intermediate variables with error correcting codes to detect errors. To protect against passive attacks, they apply threshold implementations on top of the encoding. The results are promising as they succeed in protecting the LED cipher on FPGA. However, they only provide argumentation for active security leaving out combined security and a formal adversary model. As efficiency is a major concern for practical applications, the work of Impeccable Circuits [1] only focuses on active attacks to find very efficient countermeasures. They consider an adversary who faults up to a given number of gates and consider compositional security, i.e., they look at the propagation of faults in their components. Previous works looked at adversaries faulting and reading separate wires, the work of CAPA [29] considers stronger adversaries. They use multiparty computation to provide provable security against combined attacks by proposing a new adversary model, the tile probe-and-fault model. This model considers an adversary who is capable of reading and faulting whole areas in the implementation thus ensuring hardware protection against combined attacks. However, due to their security model the countermeasures are heavy.

The adversaries considered in Private Circuits II and Impeccable Circuits are a good start towards formalising active and combined security but they do not yet allow for composable combined security definitions which are needed by designers. In this work, we combine the wire faulting adversary with the usual probing adversary to consider an attacker who can read and fault a given number of wires in a circuit. Similar to the proposition of Non-Interference by Barthe et al. [3], we build further on our adversary model by considering a modularised circuit and proposing sufficient security conditions (SNA and SNINA) such that modular compositions remain secure.

## 1.1 Contributions

The focus of our work is to propose compositional security notions which capture active and combined attacks. Our motivation to propose these security notions is threefold.

- **Efficient Verification.** The probing and wire faulting security model's verification time is exponential in their security order. As a result, verifying larger circuits such as a symmetric cipher quickly becomes infeasible. To leverage this complexity constraint, composable security definitions allow for the verification of smaller blocks even for larger security orders.
- **Optimised Building Blocks.** Composable security notions allow designers to create smaller circuits enabling search techniques to maximise efficiency.

Since a proof of composable security considers arbitrary security orders, these designs are flexible to provide the preferred security/efficiency trade-off. Moreover, they allow to secure arbitrary protocols.

– **Compartmentalisation.** Symmetric ciphers are not often used in a stand-alone setting. Instead they are used in a mode of operation as an encryption scheme; to generate a stream cipher for authenticated encryption; or as a compression for hash functions. A cipher secured with a composable notion is guaranteed secure even in a more complex structure contrary to non-compositional notions.

To show our security notions in action, we propose two SNINA secure multiplication gadgets.

– **Error Detection:** We propose a multiplication gadget using duplicated Boolean shares with an error detecting mechanism. We show that the gadget is vulnerable to a Statistical Ineffective Fault Attack (see [16]) but the probability for the attack to succeed can be made arbitrarily small by increasing the number of shares. Thus, we prove that the gadget still attains SNINA security. Last, we provide an abort mechanism to show the gadget does not rely on an ideal abort command.
– **Error Correction:** For the second construction, we adapt the error detecting gadget and add error correction methods. The result is a gadget which is impervious to ineffective faults and, moreover, we show the gadget achieves the stronger notion of SININA. This SININA notion proves that the level of combined security is higher than the error detection variant, i.e., the adversary does not gain any advantage by using faults in addition to probes. This shows that, although error correction techniques are more expensive, they give a significant increase in protection against combined attacks as well as guaranteed output delivery.

## 2 The Circuit Model and Secret Sharing

We introduce gadgets, private circuits, and the notion of simulatability. Similar to [24], we represent computations in arithmetic circuit form, a directed acyclic graph whose nodes are operations over a finite field $\mathbb{F}$ and whose edges are wires. Additionally, we consider probabilistic arithmetic circuits, meaning circuits with nodes having no input and uniform random elements over $\mathbb{F}$ as output; this randomness is independent and identically distributed, and the correctness of the circuit is not dependent on it. In order to resist fault attacks, we consider nodes with no output and which can abort the computation. This abort signal works as a broadcast making all wires in the circuit read $\perp$ when the signal is sent out.[1] The adversary also receives this abort signal as it can view from the state of the output whether the circuit aborted or not.

---

[1] On hardware this functionality is replaced a specialised mechanism such as a cascading gadget from [23].

In order to defend algorithms against side-channel attacks a sound and widely deployed approach is the masking countermeasure which was introduced at the same time by Chari et al. [10] and by Goubin and Patarin [20]. The technique splits each key-dependent variable $x$ in the algorithm into shares $x_i$ such that $x = \sum_i x_i$ over a finite field $\mathbb{F}$. In case this field is binary, this masking method is referred to as Boolean masking. If no $d$ shares give information on the secret we say that the masking scheme has a passive threshold $d$.

To defend an algorithm against fault attacks the core idea is to utilise redundancy to enable detection of the injected faults. This redundancy is found in encoding intermediate variables using error detecting codes. A popular encoding method is to duplicate intermediate variables, such that, by checking whether all duplicates are equal, an algorithm can detect injected faults. If all sets of $k$ faulty shares in a share vector are detectable, we say that the encoding scheme has an active threshold $k$.

Using masking and encoding of variables as the core idea to protect secrets against passive and active attacks, we introduce terminology to protect algorithms. A probabilistic circuit with shared inputs/outputs and, if needed, the capability to abort the computation is dubbed a gadget.

**Definition 1 (Gadget).** *A gadget $G$ is a probabilistic circuit with input in $\mathbb{F}^{nm}$ (m inputs where each input is divided into n shares), uniform randomness $r \in \mathbb{F}^{\alpha}$, and a shared output in $\mathbb{F}^{nm'}$ or abort $\perp$.*

Concerning symmetric primitives, the secrets are each potential intermediate variable of the primitive. In other words, to protect the primitive against passive or active attacks, it works solely over shared variables. The secrets are thus the secret shared variables.

Additionally, we define private circuits as probabilistic circuits consisting of a gadget, where its inputs are first shared and the shared outputs are reconstructed.

**Definition 2 (Private Circuit [24]).** *A private circuit implementing the function $f : \mathbb{F}^m \to \mathbb{F}^{m'}$ is defined by a triple $(\mathcal{I}, \mathcal{C}, \mathcal{O})$, where*

- *$\mathcal{I} : \mathbb{F}^m \to \mathbb{F}^{nm}$ is a probabilistic circuit with uniform randomness, called input encoder;*
- *$\mathcal{C} : \mathbb{F}^{nm} \to \mathbb{F}^{nm'}$ is a gadget with uniform randomness and the ability to abort;*
- *$\mathcal{O} : \mathbb{F}^{nm'} \to \mathbb{F}^{m'}$ is a circuit with the ability to abort, called output decoder.*

Since we will be working with composable security definitions, we typically consider that private circuits are composed of several gadgets, i.e., the output of one gadget forms the input of another.

We aim to protect against passive, active or combined adversaries as those who interact with a circuit by placing probes, faults, or both respectively. The circuit responds to this adversary by setting or toggling the values on the faulted wires and returning the values on the probed wires. The state of the abort signal (true or false) is returned as well.
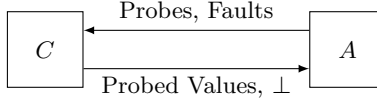
**Figure 1.** Interaction between a circuit $C$ and an adversary $A$.

In order to make simulation based proofs for the secrecy of shared variables in gadgets, we define simulatability similar to the definitions proposed in [5,9]. However, we additionally consider that up to $k$ wires in that gadget have been faulted and that the gadget can abort. Here the adversary (distinguisher) is either interacting with the actual gadget or with a simulator. This simulator is given only a part of the input and does not know the secrets of the gadget. The distinguisher's goal is to determine whether it is interacting with the simulator or with the actual gadget. A failure to do so implies that the adversary can know at most the shares given to the simulator and as a result only some inputs of the gadget.

**Definition 3 (Simulatability).** *Let $P = \{p_1, ..., p_d\}$ be a set of $d$ probes of a gadget $C$ with $m$ inputs where each input is divided into $n$ shares. Let the set of $q$ shares of each input given to the simulator be denoted by $I = \{(i_1, j_1), ..., (i_m, j_q)\} \subset \{1, ..., m\} \times \{1, ..., n\}$. Let $F = \{(f_1, e_1), ..., (f_k, e_k)\}$ be a set of $k$ injected faults $e_i$ (either set or add) on the wire $f_i$ in $C$. Denote $C_{P,F}$ as the circuit $C$ with probed wires as per $P$ and injected faults as per $F$. Finally, let $\perp \in \{0, 1\}$ denote the state of the abort signal in the circuit.*

*We define the simulator $S$ and distinguisher $D$ as the following probabilistic functions.*

$$S : \mathbb{F}^q \times \mathbb{F}^k \to \mathbb{F}^d \times \{0, 1\}$$

$$D : \mathbb{F}^d \times \{0, 1\} \times \mathbb{F}^k \times \mathbb{F}^{nm} \to \{0, 1\}$$

*We say that the set of probes $P$ and the state of the abort signal $\perp$ of the faulted circuit $C_F$ can be simulated with the set of values on the input wires $I$ if there exists a simulator $S$, such that for any distinguisher $D$ and any inputs $a_{*,*}$, we have that*

$$\left| \Pr[D(C_{P,F}(a_{*,*}), F, a_{*,*}) = 1] - \Pr[D(S(I, F), F, a_{*,*}) = 1] \right|$$

*is negligible in the passive threshold of the sharing scheme, where the probability is taken over the random coins in $C, S$ and $D$.*

We note that for composable security, as we will see later on, we require that the probability for the distinguisher to view the difference between the circuit and the simulator is negligible and we should take care composing gadgets when it is not.

# 3 Security Definitions

In this section we specify orders of passive, active, and combined security and expand them to composable security notions which is the focus of our work.

## 3.1 Orders of Security

**Passive Security.** To model passive security we consider the known probing adversary who can read the exact values of up to a threshold number of wires in a gadget. The order of passive security is the well-known order of probing security.

**Definition 4 (Order of passive security [24, 30]).** *A private circuit is $d^{th}$-order passive secure ($d^{th}$-order probing secure) if every d-tuple of the gadget's intermediate variables is independent of any sensitive variable.*

**Active Security.** We ensure protection against an adversary who is capable of faulting a given number of wires in the circuit. We note that similar adversaries have been proposed in Private Circuits II [23] and Impeccable Circuits [1]. The order of active security is determined by the number of wires in the circuit the adversary needs to fault in order to create an incorrect output. Such incorrect outputs are important as they can activate DFA attacks, thus to secure implementations we require that the private circuit either gives back a correct output or the process is aborted.

**Definition 5 (Order of active security).** *A private circuit is $k^{th}$-order active secure if any set of k faults on the gadget's intermediate variables results in either abort $\bot$ or a correct output.*

Note that active security guarantees output correctness and does not consider fault attacks which target the privacy of a scheme such as ineffective faults.

**Combined Security.** We protect against a combined adversary who both faults and probes wires and consider a private circuit secure if it retains both its privacy and correctness against the combined adversary. This gives us the following combined security definition.

**Definition 6 (Order of combined security).** *A private circuit is $(d, k)$-order combined secure if for any set of k faults and d probes on the gadget's intermediate variables, the following holds.*

*(a) Privacy: The probed d-tuple with the state of the abort signal is independent of any sensitive variable.*
*(b) Correctness: The circuit either aborts $\bot$ or gives a correct output.*

The combined security model with $d = 0$ still differs from the active security model as the combined security model considers that an adversary can use the knowledge on the state of the abort signal to derive the private circuit's internal variables. The difference between the two models thus lies in the combined security model looking at both the privacy and correctness of a circuit while active security only considers its correctness.

## 3.2   Composable Notions of Security

We note that the previously discussed security conditions are not composable, i.e., the composition of multiple secure gadgets can be insecure. Thus, the previous security conditions should be applied to the entire implementation, instead we look at composable security notions.

**Passive Security.** The security notion for composable passive security has been studied by Barthe et al. [3] who defined the notion of Non-Interference (NI) using simulation based security (see Definition 3).

**Definition 7** ($d$ **Non-Interferent** ($d$-**NI**) [3])**.** *A gadget $G$ is $d$-NI if any set of at most $d' \leq d$ probes can be simulated with at most $d'$ shares of each input.*

Intuitively, the above model grants composable security since a probed value in a gadget can be simulated with an input share, which on its turn is the output share of a previous gadget. In case the latter gadget is also non-interferent, this output value can again be simulated with an input share. This chains until we reach the encoding function in a private circuit (Definition 2). Since the adversary can only probe $d$ values we only need to use a secret sharing scheme of passive threshold at least $d$ to protect against our probing adversary. While the notion of non-interference is a good start and captures a composable security notion over the serial composition of gadgets, the notion is not sufficient to provide protection when gadgets are composed in parallel (e.g., when two gadgets share the same input). To this end Barthe et al. introduced the notion of Strong Non-Interference (SNI).

**Definition 8** ($d$-**Strong Non-Interferent** ($d$-**SNI**) [3])**.** *A gadget $G$ is $d$-SNI if any set of $d_1$ probes on its intermediate variables and every set of $d_2$ probes on its output shares such that $d_1 + d_2 \leq d$, the totality of the probes can be simulated by only $d_1$ shares of each input.*

We note that intermediate variables can also be the input or output variables of the gadget.

When the above notion of non-interference is combined with a sharing scheme with a high enough passive threshold, the composable notion provides for probing security.

**Active Security.** Recall that we defined the order of active security as the maximal number of faulty wires such that the circuit still returns a correct output. We now make this into a composable notion, thus we look at the effect of a fault in a gadget which is part of a larger whole. Ideally an injected fault in the gadget is not propagated, i.e., the fault does not affect the output of that gadget. However, the adversary can always fault its output directly, meaning that we can never guarantee that all outputs of a faulted gadget are correct. Instead, we are interested in gadgets which do not accumulate faults. In other words, we need a fault on a single input or intermediate wire to affect only a single output of the gadget. We relax this requirement by allowing countermeasures to abort the computation (e.g., by using error detecting methods). We thus find composable active security notions which are similar in nature to the definitions of NI and SNI discussed earlier. Our first notion is Non-Accumulation (NA).

**Definition 9 ($k$-Non-Accumulative ($k$-NA)).** *A gadget $G$ is $k$-NA if for any set of $k' \leq k$ errors, the gadget either aborts or gives an output with at most $k'$ errors.*
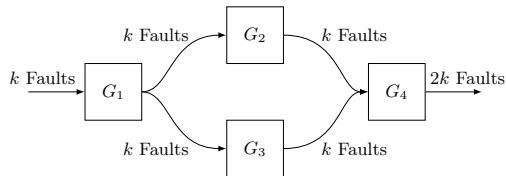


**Figure 2.** An example of the propagation of faults over several $k$-NA gadgets for which a stronger composability notion is needed.

For a gadget which is $k$-NA, $k$ faults on its intermediate variables result in the gadget giving an output with at most $k$ faults. When composing gadgets, a stronger notion of non-accumulation is needed to guarantee the security of the composition. For example, consider the case given in Figure 2 where each gadget $G_i$ is $k$-NA. If an adversary injects $k$ faults in the input of $G_1$, the gadget will give an output with at most $k$ faulty shares. These faults propagate to the inputs of $G_2$ and $G_3$ which, because both gadgets are $k$-NA, results in a worst case scenario where $G_4$ gets an input with a total of $2k$ faulty shares. The end result is a sharing with $2k$ faulty shares even though only $k$ faults were injected. To avoid such an accumulation of faults, one needs gadgets which are capable of erasing the errors from their input. The following definition of Strong Non-Accumulation (SNA) is sufficient to arbitrarily compose gadgets and be assured of their active security.

**Definition 10 ($k$-Strong Non-Accumulative ($k$-SNA)).** *A gadget $G$ is $k$-SNA if for any set of $k_1$ errors on each input and every set of $k_2$ errors on the intermediate variables, with $k_1 + k_2 \leq k$, the gadget either aborts or gives an output with at most $k_2$ errors.*

When the non-accumulation notions are combined with a sharing scheme with a high enough active threshold, the composable notions provide active security.

**Combined Security.** We now look at composable security notions considering circuits which are both probed and faulted. First, we need to guarantee the correctness of the output of each gadget. To capture the effect of faults in compositions of gadgets, we use an argument similar to the one on active security. Thus,we need that an injected fault in a gadget propagates to at most one output share. However, the adversary can now place probes and thus learn part of the computation made in the gadgets. As a result, the combined security notion needs to capture the probability of an adversary breaking the correctness of a gadget given several faults and probes. In this work we only propose countermeasures where the correctness can not be broken, to give an example of a countermeasure for which this probability is non-trivial we refer the reader to the CAPA countermeasure [29]. Apart from guaranteeing the correctness of a gadget, we also guarantee its sensitive variable privacy for which we use simulation based arguments similar to non-interference. As mentioned by Clavier et al. [11], fault injections can act as a probing tool (think of an adversary faulting away the randomness in a gadget). Thus,we treat faults as probes giving extra shares to the simulator per injected fault (though we see later on that this is not always needed). Additionally, to give the designer the freedom to make countermeasures more efficient we consider security with abort. To capture the effect of the abort signal potentially revealing secrets in the gadget, we require the simulator to reproduce this signal given the injected errors and some input shares. As a result, we design a composable security notion of order $(d, k)$ such that the gadget is $(d', k')$-order combined security for all sets of $d' + k' \leq d$ probes and $k' \leq k$ faults. We dub our notion NINA derived from the concatenation of the names Non-Interference (NI) and Non-Accumulation (NA).

**Definition 11 ($(d, k)$-NINA).** *A gadget $G$ is $(d, k)$-NINA if for any set of $k' \leq k$ errors and any set of $d'$ probes, such that $d' + k' \leq d$, the following holds.*

(a) *Privacy: The probes and the abort signal can be simulated with $d' + k'$ shares of each input and the injected errors.*
(b) *Correctness: The gadget either aborts or gives an output with at most $k'$ errors.*

The NINA notion, combined with a sharing scheme having a sufficient passive and active threshold, implies the notion of combined security (see Definition 6). This follows from the simulation based security stating that the adversary can learn up to a threshold number of the gadget's inputs which, if lower than the passive threshold of the sharing scheme, gives no information on the gadget's secrets. Similarly, since the adversary can only fault up to a threshold number of outputs, a decoding gadget can detect or correct those errors given that the sharing scheme has enough redundancy in it. A formal proof of this implication is found in Appendix A.

**Theorem 1.** *A $(d, k)$-NINA gadget $G$ with input encoding $\mathcal{I}$ and output decoding $\mathcal{O}$ using a secret sharing scheme with passive threshold at least $d$ and active threshold at least $k$ is $(d', k')$-order combined secure for any $d' + k' \leq d$ and $k' \leq k$.*

As a result, if we prove a gadget is NINA, we know it is combined secure. However, just as with non-interference, the NINA notion is not sufficient for composability. To this end we introduce "Strong NINA" (SNINA).

**Definition 12 ($(d, k)$-SNINA).** *A gadget $G$ is $(d, k)$-SNINA if for any set of $k_1$ errors on each input and $k_2$ intermediate errors, any set of $d_1$ intermediate probes, any set of $d_2$ probes on the output, such that $d_1 + d_2 + k_1 + k_2 \leq d$ and $k_1 + k_2 \leq k$, the following holds.*

(a) *Privacy: The probes and the abort signal can be simulated with $d_1 + k_1 + k_2$ shares of each input and the injected errors.*
(b) *Correctness: The gadget either aborts or gives an output with at most $k_2$ errors.*

The notion of SNINA is sufficient for composability. In other words the composition of two SNINA gadgets is again SNINA (for a proof see Appendix B).

**Theorem 2.** *The composition of two $(d, k)$-SNINA gadgets is $(d, k)$-SNINA.*

The above theorem together with Theorem 1 implies that the notion of SNINA is a sufficient condition to achieve composable combined security. The relations between the SNINA notion and other security models is shown in Figure 3.

*Remark 1.* We note that there are variants in between NINA and SNINA, these hybrid definitions (SNI-NA and NI-SNA) are given in Appendix F. While these definitions are not sufficient to guarantee arbitrary composition of gadgets, they allow for a secure countermeasure using minimal error check and refresh gadgets similar to what was done in [7].
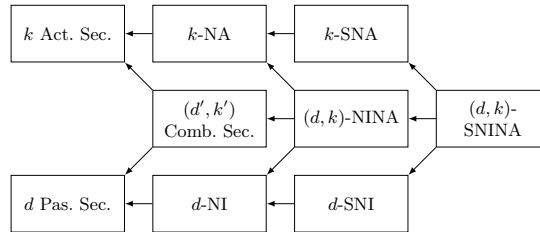


**Figure 3.** A short overview of security models and relations between them.

Nevertheless, we find that there is a stronger property than NINA which gives improved protection. In case we use error correcting techniques instead of

error detecting ones, specialised gadgets can attain a stronger security condition where faults are no longer modelled as probes. Thus,we propose a security notion where we claim an adversary can not learn anything by faulting a gadget which manifests itself in the security definition as the simulator not getting an extra input share for an injected fault. The result of this change is captured in the following definition which we dub "Independent NINA" or ININA.

**Definition 13 ($(d, k)$-ININA).** *A gadget $G$ is $(d, k)$-ININA if for any set of $k' \leq k$ errors and any set of $d'$ probes, such that $d' \leq d$, the following holds.*

(a) *Privacy: The probes can be simulated with $d'$ shares of each input and the injected errors.*
(b) *Correctness: The gadget gives an output with at most $k'$ errors.*

The above definition can again be made into a property which is sufficient for arbitrary compositions. This gives us the notion of "Strong Independent NINA" or SININA for short.

**Definition 14 ($(d, k)$-SININA).** *A gadget $G$ is $(d, k)$-SININA if for any set of $k_1$ errors on each input and $k_2$ intermediate errors, any set of $d_1$ intermediate probes, any set of $d_2$ probes on the output, such that $d_1 + d_2 \leq d$ and $k_1 + k_2 \leq k$, the following holds.*

(a) *Privacy: The probes can be simulated with $d_1$ shares of each input and the injected errors.*
(b) *Correctness: The gadget gives an output with at most $k_2$ errors.*

It is evident that the ININA notions are stronger than the NINA notions, thus the above notions also provide combined security. However, the notion provides directly $(d, k)$-combined security instead of $(d', k')$-order combined secure for any $d' + k' \leq d$ and $k' \leq k$. The proof of the following theorem is given in Appendix C.

**Theorem 3.** *A $(d, k)$-ININA gadget $G$ with input encoding $\mathcal{I}$ and output decoding $\mathcal{O}$ using a secret sharing scheme with passive threshold at least $d$ and active threshold at least $k$ is $(d, k)$-order combined secure.*

As a result, using the same masking scheme, a SININA secure gadget provides significant improved combined protection over an SNINA secure gadget.

Similar to SNINA, SININA is sufficient for composability. In other words the composition of two SININA gadgets is again SININA (for a proof see Appendix D).

**Theorem 4.** *The composition of two $(d, k)$-SININA gadgets is $(d, k)$-SININA.*

## 4 Combined Secure Duplicated Boolean Masking

In this section we introduce a combined secure methodology for an arbitrary security order. We work over bits $\mathbb{F}_2$, share values using Boolean secret sharing and encode using duplication. We first quickly introduce the secret sharing scheme and then move on to show our methodology. In Appendices H and I we prove the security of the gadgets.

## 4.1 Duplicated Boolean Masking

For the proposed countermeasures, we make use of a duplicated Boolean masking approach which shares a secret $x$ as a vector

$$\left(x_{1,1}, ..., x_{1,k+1}, x_{2,1}, ..., x_{d+1,k+1}\right),$$

such that $\sum_{i=1}^{d+1} x_{i,\ell} = x$ for all $\ell \in [k+1]$ and $x_{i,1} = ... = x_{i,k+1}$ for all $i \in [d+1]$. This method has a passive threshold $d$ meaning that no $d$ shares give information on the secret $x$ and an active threshold $k$ meaning that any faults on at most $k$ shares could be detected in the share vector.

## 4.2 Duplicated Boolean Methodology

We recall that our secret sharing scheme has a passive threshold $d$, meaning that an adversary needs to view at least $d + 1$ shares to recover the secret, and an active threshold $k$, thus an adversary needs to inject at least $k + 1$ errors for the fault to be undetectable. We note that our methodology is similar to the one from Private Circuits II [23]. The pseudo-code to secret share a value is given in Algorithm 1.

---

**Algorithm 1:** Duplicated Boolean sharing a secret $a$

> **Input:** Secret $a$ and uniform random values $r_i$
> **Output:** Duplicated Boolean shares of $a$
>
>     **for** $\ell \leftarrow 1$ **to** $k + 1$ **do**
>         **for** $i \leftarrow 1$ **to** $d$ **do**
>             $a_{i,\ell} \leftarrow r_i$;
>         **end**
>         $a_{d+1,\ell} \leftarrow a + \sum_{i=1}^{d} a_{i,\ell}$;
>     **end**

---

The addition between independent shared variables is quite simple and needs only component-wise addition between the shares. Thus, the addition between the sharing of $a$ and $b$, giving a sharing of $c = a + b$, is made by $c_{i,\ell} = a_{i,\ell} + b_{i,\ell}$. To secure operations between shares and constants we ensure that the constant is not a single point of failure, as such it also needs to be duplicated, namely each constant is replicated $(k + 1)$ times to form a $(k + 1)$ tuple which is the encoded value of the constant. The addition of a shared value $a$ with a constant $c$ is done by adding the duplicated constant to the duplicated first Boolean share of the variable.

$$\forall \ell \in [k + 1]: \; a_{1,\ell} \leftarrow a_{1,\ell} + c_\ell$$

A multiplication with a constant is done by multiplying the duplicated constant to each share.

$$\forall i \in [d + 1], \forall \ell \in [k + 1]: \; a_{i,\ell} \leftarrow a_{i,\ell} \cdot c_\ell$$

Since the above operations are all local, they are evidently $(d, k)$-NINA.

While linear operations are easily implemented, the multiplication between shared and encoded variables is more difficult. We give pseudo-code of our multiplication gadget in Algorithm 2. The gadget starts by multiplying two independent share vectors of $a$ and $b$ to create all cross products of the form $a_i b_j$. These cross products are then remasked by adding unique randomness $r_{i,j}$ (which is important for the SNI property). Since we add the same randomness over all duplicated cross products ($u_{i,j,\ell}$ for $\ell \in [k+1]$) all these cross products should equal each other if no fault was injected. As a result, we can error check them (which is important for the SNA property).[2] To detect errors in the cross products it is sufficient to compare a share to all its duplicated versions, in symbols:

$$\forall i, j \in [d+1], \ \forall \ell \in [k+1]: \ u_{i,j,1} = u_{i,j,\ell}.$$

Since we are working over bits, this translates to aborting the computation in case one of the $u_{i,j,1} + u_{i,j,\ell}$ is equal to 1. This abort operation is considered as a command causing all variables in the implementation to read $\perp$ as explained in Section 2 (in Section 4.3, we describe a cascading gadget in case an abort operation is not available). In case no error is detected, the gadget sums up all the cross products for different indices $j$ and returns a duplicated Boolean sharing of $ab$. The proof that this multiplication procedure is SNINA is given in Section H.2. From this proof we see that there is a statistical ineffective fault attack (see [16]) which breaks the privacy of the algorithm. This attack works as follows, the adversary adds a non-zero fault to one of the $a_{i,\ell}$ shares (similarly $b_{i,\ell}$ shares). In case the operation does not abort, the adversary learns that all $b_{i,\ell} = 0$ (similarly all $a_{i,\ell} = 0$), which means the adversary learns an input secret and breaks the privacy of the gadget. The probability for this attack to succeed is equal to $1/|\mathbb{F}_2|^{d+1}$. Due to the attack aborting the computation when it fails, this attack does not threaten the composability of the gadget.[3] To increase the protection against the ineffective fault, the probability for the attack to succeed needs to be made sufficiently small which is done by increasing the number of shares or by increasing the field size $|\mathbb{F}|$. In Section 5 we look at an error correcting variant of the multiplication gadget which is not vulnerable to an ineffective fault.

In Algorithm 3 we provide a method to refresh the randomness of a shared variable and check whether there are errors present on its shares. In Appendix I we prove that Algorithm 3 is SNINA. We note that this gadget can be used to transform a NINA secure operation into its SNINA variant by serially composing the NINA gadget with Algorithm 3. This follows from Theorem 5 which states

---

[2] Note that if an adversary injects a fault directly in one of the random values $r_{i,j}$, it would not be detected. Nevertheless, the gadget still outputs a valid duplicated Boolean sharing so it does not affect the correctness of the gadget. This fault should be carefully investigated for its effects on the gadget's privacy.

[3] To clarify, the passive threshold of the sharing does not need to increase to assure composability due to the only attack, causing simulator failure, aborting the computation on success.

**Algorithm 2:** Multiplying duplicated Boolean shared values

**Input:** Independent shares of $a$ and $b$, and uniform random $r_{i,j}$

**Output:** Shares of $ab$ or $\perp$

for $\ell \leftarrow 1$ to $k+1$ do
    for $i \leftarrow 1$ to $d+1$ do
        $u_{i,i,\ell} \leftarrow a_{i,\ell} b_{i,\ell}$;
        for $j \leftarrow i+1$ to $d+1$ do
            $u_{i,j,\ell} \leftarrow a_{i,\ell} b_{j,\ell} + r_{i,j}$;
            $u_{j,i,\ell} \leftarrow a_{j,\ell} b_{i,\ell} + r_{i,j}$;
        end
    end
end
for $\ell \leftarrow 2$ to $k+1$ do
    for $i \leftarrow 1$ to $d+1$ do
        for $j \leftarrow 1$ to $d+1$ do
            $t_{i,j,\ell} \leftarrow u_{i,j,1} + u_{i,j,\ell}$;
            if $t_{i,j,\ell} = 1$ then return $\perp$;
        end
    end
end
for $\ell \leftarrow 1$ to $k+1$ do
    for $i \leftarrow 1$ to $d+1$ do
        $c_{i,\ell} \leftarrow \sum_{j=1}^{d+1} u_{i,j,\ell}$;
    end
end

that the serial composition between a NINA gadget and an SNINA gadget is again SNINA. We leave the proof of this theorem for Appendix E.

**Theorem 5.** *The serial composition of a single input, output $(d, k)$-NINA gadget with a $(d, k)$-SNINA gadget is again $(d, k)$-SNINA.*

Thus,sometimes one can substitute SNINA gadgets with NINA ones without sacrificing security. This reduces costs as NINA secure gadgets are generally more efficient than their SNINA variants.

---

**Algorithm 3:** Refreshing and checking a duplicated Boolean sharing

**Input:** Duplicated Boolean shares of $a$ and uniform random values $r_{i,j}$
**Output:** Refreshed and checked shares of $a$ or $\perp$

> **for** $\ell \leftarrow 2$ **to** $k + 1$ **do**
>     **for** $i \leftarrow 1$ **to** $d + 1$ **do**
>         $t_{i,\ell} \leftarrow a_{i,1} + a_{i,\ell}$;
>         **if** $t_{i,\ell} = 1$ **then return** $\perp$;
>     **end**
> **end**
> **for** $\ell \leftarrow 1$ **to** $k + 1$ **do**
>     **for** $i \leftarrow 1$ **to** $d + 1$ **do**
>         **for** $j \leftarrow i + 1$ **to** $d + 1$ **do**
>             $a_{i,\ell} \leftarrow a_{i,\ell} + r_{i,j}$;
>             $a_{j,\ell} \leftarrow a_{j,\ell} + r_{i,j}$;
>         **end**
>     **end**
> **end**

---

Together, all gadgets described in this section form a methodology to secure arbitrary circuits as each algorithm over a finite field can be described in terms of additions and multiplications.

### 4.3 A Cascading Gadget

In case an abort mechanism is not available, we provide a circuit which erases all data when a fault is detected. This method is similar to the cascading gadget described in [23] and thus we lend its name. We first make variables for the abort flag, we consider $\perp_\ell$ for $\ell \in [k]$. A priori, all $\perp_\ell$ are equal to zero, however, when a fault is injected we require that each $\perp_\ell$ is set to one. In case the abort flag equals all one, no $k - 1$ faults can change each $\perp_\ell$ back to zero. The above described functionality is implemented by duplicating the error checks in Algorithms 2 and 3. For example, the error checking component (the first lines) of Algorithm 3 would be changed to the following.

```
for m ← 1 to k do
    for ℓ ← 2 to k + 1 do
        for i ← 1 to d + 1 do
            ⊥_m ← (a_{i,1} + a_{i,ℓ}) ∨ ⊥_m;
        end
    end
end
```

From the above algorithm it is clear that in case one of the $a_{i,1}$ does not equal $a_{i,\ell}$, all $\perp_m$ are set to one and no $k-1$ faults can set them all back to zero.

With the above abort flag as a global variable and its functionality as described above, we can easily describe a gadget which erases its input in case a $\perp_m$ is equal to one. We give the pseudo-code of this gadget in Algorithm 4.

---

**Algorithm 4:** Cascading a duplicated Boolean sharing

**Input:** Shares of $a$ and the abort state $\perp_m$ for $m \in [k]$
**Output:** The shares of $a$ or all 0

```
for ℓ ← 1 to k + 1 do
    for i ← 1 to d + 1 do
        a_{i,ℓ} ← a_{i,ℓ} ∏_{m=1}^{k}(1 + ⊥_m);
    end
end
```

---

In case Algorithm 4 is serially composed with each Algorithm 2 or Algorithm 3, our duplicated Boolean masking methodology is secure against combined attacks without the need of an ideal abort command.

## 5  A Correcting Multiplication

In the previous section we gave a combined secure methodology based on detecting errors using duplicated Boolean shares. From its proof of security in Theorem 9 we can see that Algorithm 2 is vulnerable against a statistical ineffective fault. To avoid this vulnerability one can use an error correction method instead of an error detection one. As there is no longer an abort signal, a fault does not change the state of the output and as a result ineffective faults are now actually ineffective. Note that this comes at the increased cost of using extra shares and operations to enable error correction.

Instead of just replacing the error detection mechanisms with error correction ones, we go one step further and create an error correcting variant of Algorithm 2

which attains Strong Independent NINA security (Definition 14). Whereas Algorithm 2 was secure against $d$ probes and $k$ faults where the combined number of probes and faults do not exceed $d$, our new algorithm does not require this restriction thus it is secure against up to $d$ probes and $k$ faults at the same time. In other words, a $k$-active adversary faulting the new multiplication gadget does not harm the privacy of the gadget.

---

**Algorithm 5:** Multiplying shares with error correction

---

**Input:** Independent shares of $a$ and $b$, and uniform random $r_{i,j,\ell}$
**Output:** Shares of $ab$

    **for** $\ell \leftarrow 1$ **to** $2k+1$ **do**
        **for** $i \leftarrow 1$ **to** $d+1$ **do**
            $u_{i,i,\ell} \leftarrow a_{i,\ell}b_{i,\ell}$;
            **for** $j \leftarrow i+1$ **to** $d+1$ **do**
                $u_{i,j,\ell} \leftarrow a_{i,\ell}b_{j,\ell}$;
                $u_{j,i,\ell} \leftarrow a_{j,\ell}b_{i,\ell}$ ;
                **for** $m \leftarrow 1$ **to** $k+1$ **do**
                    $u_{i,j,\ell} \leftarrow u_{i,j,\ell} + r_{i,j,m}$;
                    $u_{j,i,\ell} \leftarrow u_{j,i,\ell} + r_{i,j,m}$;
                **end**
            **end**
        **end**
    **end**
    **for** $\ell \leftarrow 1$ **to** $2k+1$ **do**
        **for** $i \leftarrow 1$ **to** $d+1$ **do**
            **for** $j \leftarrow 1$ **to** $d+1$ **do**
                $v_{i,j,\ell} \leftarrow \mathrm{Maj}(u_{i,j,1}, ..., u_{i,j,2k+1})$;
            **end**
        **end**
    **end**
    **for** $\ell \leftarrow 1$ **to** $2k+1$ **do**
        **for** $i \leftarrow 1$ **to** $d+1$ **do**
            $c_{i,\ell} \leftarrow \sum_{j=1}^{d+1} v_{i,j,\ell}$;
        **end**
    **end**

---

We introduce the error correcting multiplication gadget. We again work over bits $\mathbb{F}_2$, share values using $d$ Boolean secret shares, but now encode using $2k+1$ duplicated shares (instead of $k+1$ shares). As such, the secret sharing scheme has a passive threshold $d$, meaning that an adversary needs to view at least $d+1$ shares to recover the secret, and an active threshold $2k$, thus an adversary needs to inject at least $k+1$ errors for the fault to be uncorrectable (note the difference with the undetectability of faults). We give the pseudo-code of the multiplication gadget in Algorithm 5. The error correcting gadget works similar

to the error detecting one. It starts by multiplying two independent share vectors of $a$ and $b$ to create all cross products. These cross products are then remasked by adding $k + 1$ random elements $r_{i,j,\ell}$ to each of them. As a result, since each cross product is masked by $k + 1$ random values, no set of $k$ faults can remove all random values on a cross product. Since we add the same randomness over all duplicated cross products ($u_{i,j,\ell}$ for $\ell \in [2k + 1]$) all these cross products still equal each other if no fault was injected. As a result, we can error correct them. An error correction on duplicated shares is done by majority voting the shares. If at least $k + 1$ out of $2k + 1$ cross products were equal to zero, the result of this majority vote is zero otherwise it is equal to one. For brevity, we denote this operation "Maj", where we assume for simplicity that a probing adversary can view all arguments given to the Maj function with one probe. We stress that this error correction procedure is independently applied to each cross product, such that a single fault can only affect one cross product. Our multiplication gadget again ends by summing up all the cross products for different indices $j$ and returns a duplicated Boolean sharing of $ab$. The proof that this multiplication procedure is SININA is given in Theorem 11.

## 6 Conclusion

We provided security notions considering circuits with probed and/or faulted wires. We then extended them to active and combined composable notions similar to the extension from the probing model to Non-Interference (NI). The first notion of Non-Accumulation (NA) addresses composable active security which states that a gadget is secure if injected faults affect only one output each. The second is the notion of composable combined security (NINA). A gadget is considered NINA if an injected fault only affects one output and a fault or probe can be simulated using only one input. We discussed both error detection and error correcting gadgets and showed that the error detection mechanism is prone to ineffective faults whereas error correction comes at an increased cost but gives significantly improved protection (Independent NINA).

The notions for composable security offer the ability to efficiently verify building blocks of larger implementations and allow for the search of efficient functions which achieve security in the corresponding model. Moreover, these composable notions enable us to use secured primitives in a larger whole such as modes of operation.

## References

1. Aghaie, A., Moradi, A., Rasoolzadeh, S., Schellenberg, F., Schneider, T.: Impeccable circuits. Cryptology ePrint Archive, Report 2018/203 (2018)
2. Arribas, V., Bilgin, B., Petrides, G., Nikova, S., Rijmen, V.: Rhythmic keccak: SCA security and low latency in HW. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 269–290 (2018). https://doi.org/10.13154/tches.v2018.i1.269-290, https://doi.org/10.13154/tches.v2018.i1.269-290

3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 116–129. ACM (2016). https://doi.org/10.1145/2976749.2978427, https://doi.org/10.1145/2976749.2978427

4. Barthe, G., Belaïd, S., Fouque, P., Grégoire, B.: maskverif: a formal tool for analyzing software and hardware masked implementations. IACR Cryptology ePrint Archive **2018**, 562 (2018), https://eprint.iacr.org/2018/562

5. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Randomness complexity of private circuits for multiplication. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 616–648. Springer (2016). https://doi.org/10.1007/978-3-662-49896-5_22, https://doi.org/10.1007/978-3-662-49896-5_22

6. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 397–426. Springer (2017). https://doi.org/10.1007/978-3-319-63697-9_14, https://doi.org/10.1007/978-3-319-63697-9_14

7. Belaïd, S., Goudarzi, D., Rivain, M.: Tight private circuits: Achieving probing security with the least refreshing. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 343–372. Springer (2018). https://doi.org/10.1007/978-3-030-03329-3_12, https://doi.org/10.1007/978-3-030-03329-3_12

8. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Jr., B.S.K. (ed.) Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1294, pp. 513–525. Springer (1997). https://doi.org/10.1007/BFb0052259, https://doi.org/10.1007/BFb0052259

9. Cassiers, G., Standaert, F.: Improved bitslice masking: from optimized non-interference to probe isolation. IACR Cryptology ePrint Archive **2018**, 438 (2018), https://eprint.iacr.org/2018/438

10. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999). https://doi.org/10.1007/3-540-48405-1_26, https://doi.org/10.1007/3-540-48405-1_26

11. Clavier, C.: Secret external encodings do not prevent transient fault analysis. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings. Lecture Notes in Computer Science,

vol. 4727, pp. 181–194. Springer (2007). https://doi.org/10.1007/978-3-540-74735-2_13, https://doi.org/10.1007/978-3-540-74735-2_13

12. Cnudde, T.D., Nikova, S.: More efficient private circuits II through threshold implementations. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016. pp. 114–124. IEEE Computer Society (2016). https://doi.org/10.1109/FDTC.2016.15, https://doi.org/10.1109/FDTC.2016.15

13. Cnudde, T.D., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with d+1 shares in hardware. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9813, pp. 194–212. Springer (2016). https://doi.org/10.1007/978-3-662-53140-2_10, https://doi.org/10.1007/978-3-662-53140-2_10

14. Coron, J.: High-order conversion from boolean to arithmetic masking. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 93–114. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_5, https://doi.org/10.1007/978-3-319-66787-4_5

15. Coron, J.: Formal verification of side-channel countermeasures via elementary circuit transformations. In: Preneel, B., Vercauteren, F. (eds.) Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10892, pp. 65–82. Springer (2018). https://doi.org/10.1007/978-3-319-93387-0_4, https://doi.org/10.1007/978-3-319-93387-0_4

16. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 547–572 (2018)

17. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 423–440. Springer (2014). https://doi.org/10.1007/978-3-642-55220-5_24, https://doi.org/10.1007/978-3-642-55220-5_24

18. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable masking schemes in the presence of physical defaults & the robust probing model. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 89–120 (2018). https://doi.org/10.13154/tches.v2018.i3.89-120, https://doi.org/10.13154/tches.v2018.i3.89-120

19. Gollmann, D.: Computer Security (3. ed.). Wiley (2011), http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118801326.html

20. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1717, pp. 158–172. Springer (1999). https://doi.org/10.1007/3-540-48059-5_15, https://doi.org/10.1007/3-540-48059-5_15

21. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In: Bil-

gin, B., Nikova, S., Rijmen, V. (eds.) Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016. p. 3. ACM (2016). https://doi.org/10.1145/2996366.2996426, https://doi.org/10.1145/2996366.2996426

22. Groß, H., Schaffenrath, D., Mangard, S.: Higher-order side-channel protected implementations of KECCAK. In: Kubátová, H., Novotný, M., Skavhaug, A. (eds.) Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017. pp. 205–212. IEEE Computer Society (2017). https://doi.org/10.1109/DSD.2017.21, https://doi.org/10.1109/DSD.2017.21

23. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.A.: Private circuits II: keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings. Lecture Notes in Computer Science, vol. 4004, pp. 308–327. Springer (2006). https://doi.org/10.1007/11761679_19, https://doi.org/10.1007/11761679_19

24. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_27, https://doi.org/10.1007/978-3-540-45146-4_27

25. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25

26. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: A very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 69–88. Springer (2011). https://doi.org/10.1007/978-3-642-20465-4_6, https://doi.org/10.1007/978-3-642-20465-4_6

27. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4307, pp. 529–545. Springer (2006). https://doi.org/10.1007/11935308_38, https://doi.org/10.1007/11935308_38

28. Poschmann, A., Moradi, A., Khoo, K., Lim, C., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2, 300 GE. J. Cryptology **24**(2), 322–345 (2011). https://doi.org/10.1007/s00145-010-9086-6, https://doi.org/10.1007/s00145-010-9086-6

29. Reparaz, O., De Meyer, L., Bilgin, B., Arribas, V., Nikova, S., Nikov, V., Smart, N.P.: CAPA: the spirit of beaver against physical attacks. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol.

10991, pp. 121–151. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_5, https://doi.org/10.1007/978-3-319-96884-1_5

30. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer (2010). https://doi.org/10.1007/978-3-642-15031-9_28, https://doi.org/10.1007/978-3-642-15031-9_28

31. Schneider, T., Moradi, A., Güneysu, T.: Parti - towards combined hardware countermeasures against side-channel and fault-injection attacks. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 302–332. Springer (2016). https://doi.org/10.1007/978-3-662-53008-5_11, https://doi.org/10.1007/978-3-662-53008-5_11

32. Ueno, R., Homma, N., Sugawara, Y., Nogami, Y., Aoki, T.: Highly efficient $gf(2^8)$ inversion circuit based on redundant GF arithmetic and its application to AES design. In: Güneysu, T., Handschuh, H. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9293, pp. 63–80. Springer (2015). https://doi.org/10.1007/978-3-662-48324-4_4, https://doi.org/10.1007/978-3-662-48324-4_4

## A  Proof of Theorem 1

In this appendix we give the proof that NINA (Definition 11) implies combined security (Definition 6).

*Proof.* We recall that per Definition 2 an input encoding function $\mathcal{I}$ is a probabilistic circuit which shares the input using uniform randomness and is called before the gadget $G$. The output decoding function $\mathcal{O}$ gives back the decoding of its input shares or an abort signal $\perp$ and is called on the output of $G$. We take an arbitrary $d', k'$ set of probes and faults such that $d' + k' \leq d$ and $k' \leq k$ and prove the correctness and privacy of $G$ with $\mathcal{I}$ and $\mathcal{O}$ following Definition 11.

First, we prove that the adversary does not violate the correctness of $G$ with $\mathcal{I}$ and $\mathcal{O}$ given the knowledge of any $d'$ intermediate values and any set of $k'$ faults in the intermediate values. From Definition 11 of NINA we know that $G$ gives an output with less than $k'$ faulty shares or an abort signal. Since $k' \leq k$, we know from our secret sharing scheme that the output decoder $\mathcal{O}$ either corrects its input and gives a correct output or detects an error and aborts the computation.

Second, we prove the privacy of $G$ with $\mathcal{I}$ and $\mathcal{O}$. Since we defined simulatability (Definition 3) with computational security, we consider $d$ large enough for the probability of simulation failure for $G$ to be negligible. From Definition 11 of NINA we know that the adversary learns at most $d' + k'$ input shares of $G$ from the injected probes and faults. Since $d' + k' \leq d$, the passive security threshold of the secret sharing scheme is not reached and hence no information

of $G$'s secret inputs are revealed due to the properties of $\mathcal{I}$. Additionally, the adversary is given the state of the abort signal of the circuit. From the definition of NINA we know that given those $d' + k'$ input shares, we can simulate the abort signal, i.e., we can decide whether or not the abort signal was flagged. As we simulated the abort signal and the faults on the output shares, we can also simulate whether $\mathcal{O}$ gives back a correct output or an abort. As a result, the privacy of $G$ is proven. $\qquad\square$

# B   Proof of Theorem 2

In this appendix, we show that the composition of SNINA secure gadgets is again SNINA where we assume for simplicity that each gadget has one shared output.

*Proof.* Consider two gadgets $G_1$ and $G_2$ and their composition $G_3$. We prove that $G_3$ is $(d, k)$-SNINA given that both $G_1$ and $G_2$ are $(d, k)$-SNINA. We assume that $G_1$ and $G_2$ share at least one wire otherwise the proof is evident. Without loss of generality we assume the output of $G_2$ is not an input to $G_1$ (thus $G_1$ is evaluated before or at the same time as $G_2$). Since we defined simulatability (Definition 3) with computational security, we consider the passive threshold of the sharing scheme to be large enough for the probability of simulation failure for $G_1$ and $G_2$ to be negligible.

Take an arbitrary set of $k_1$ errors on each input of $G_3$, arbitrary sets of $k_2$ intermediate errors and $d_1$ probes on $G_3$, and an arbitrary set of $d_2$ probes on the output of $G_3$, such that $d_1 + d_2 + k_1 + k_2 \leq d$ and $k_1 + k_2 \leq k$. Because $G_3$ is the composition of $G_1$ and $G_2$, we divide the $d_1$ probes as $d_3$ probes on $G_1$ and $d_4$ probes on $G_2$ such that $d_3 + d_4 = d_1$. Similarly, we have $k_3$ faults on $G_1$ and $k_4$ faults on $G_2$ such that $k_3 + k_4 = k_2$.

We first prove the correctness of $G_3$, since $G_1$ is instantiated with $k_1$ errors on each input and $k_3$ intermediate errors, such that $k_1 + k_3 \leq k$, and since $G_1$ is $(d, k)$-SNINA it either aborts or its output has at most $k_3 \leq k_2$ errors on it. Now $G_2$ is instantiated with either $k_1$ or $k_3$ faults on each input and has $k_4$ intermediate faults, since $G_2$ is $(d, k)$-SNINA and $k_1 + k_4 \leq k$, $k_3 + k_4 \leq k$, it follows that $G_2$ either aborts or its output contains at most $k_4 \leq k_2$ errors. Since the combined outputs of $G_1$ and $G_2$ have less than $k_2$ faults on them, we have proven the correctness of $G_3$.

Second, we prove the privacy of $G_3$. First if $G_1$ aborts, then we can simulate this abort signal with $d_3 + k_1 + k_3$ of $G_1$'s input shares and the injected errors. Recall that $G_2$ is instantiated with either $k_1$ or $k_3$ faults on each input and there are $k_4$ faults on its intermediate variables. Similarly, there are $d_2$ probes on $G_2$'s output and $d_4$ intermediate probes. Since $G_2$ is $(d, k)$-SNINA, the adversary learns at most $d_4 + k_4$ input shares of $G_2$. Additionally, the adversary learns $\max(k1, k3)$ shares due to the faults on $G_2$'s inputs. Since both $G_1$ and $G_2$ are $(d, k)$-SNINA and less than $k$ faults are injected, we know that each fault causes the adversary to learn at most one share of each input per fault. From this we

24

learn that the $k_3$ injected faults in $G_1$ cause the adversary to learn $k_3$ input shares of $G_1$ and since at most $k_3$ faults propagate to its output, the adversary learns at most $k_3$ input shares of $G_2$. We also see that the $k_1$ input faults on $G_3$ give the adversary knowledge of at most $k_1$ shares of $G_1$ and of $G_2$, however these do not propagate from $G_1$ to $G_2$. Finally, since the output of $G_1$ is either the output of $G_3$ or the input to $G_2$ and since $d_3 + d_4 + d_2 + k_1 + k_3 + k_4 \leq d$, we can simulate the probed values and the abort signals of $G_1$ and $G_2$ with at most $d_3 + d_4 + k_1 + k_3 + k_4 \leq d_1 + k_1 + k_2$ shares of each input. This proves the privacy of $G_3$ and thus that $G_3$ is $(d, k)$-SNINA. □

## C Proof of Theorem 3

In this appendix we give the proof that ININA (Definition 13) implies combined security (Definition 6).

*Proof.* We recall that per Definition 2 an input encoding function $\mathcal{I}$ is a probabilistic circuit which shares the input using uniform randomness and is called before the gadget $G$. The output decoding function $\mathcal{O}$ gives back the decoding of its input shares and is called on the output of $G$. We prove the correctness and privacy of $G$ with $\mathcal{I}$ and $\mathcal{O}$ following Definition 13.

First, we prove that the adversary does not violate the correctness of $G$ with $\mathcal{I}$ and $\mathcal{O}$ given the knowledge of any $d$ intermediate values and any set of $k$ faults in the intermediate values. From Definition 13 of ININA we know that $G$ gives an output with less than $k$ faulty shares. We know from our secret sharing scheme that the output decoder $\mathcal{O}$ gives a correct output.

Second, we prove the privacy of $G$ with $\mathcal{I}$ and $\mathcal{O}$. Since we defined simulatability (Definition 3) with computational security, we consider $d$ large enough for the probability of simulation failure for $G$ to be negligible. From Definition 13 of ININA we know that the adversary learns at most $d$ input shares of $G$ from the injected probes and faults. Thus, the passive security threshold of the secret sharing scheme is not reached and hence no information of $G$'s secret inputs are revealed due to the properties of $\mathcal{I}$. As a result, the privacy of $G$ is proven. □

## D Proof of Theorem 4

In this appendix, we show that the composition of SININA secure gadgets is again SININA where we assume for simplicity that each gadget has one shared output.

*Proof.* Consider two gadgets $G_1$ and $G_2$ and their composition $G_3$. We prove that $G_3$ is $(d, k)$-SININA given that both $G_1$ and $G_2$ are $(d, k)$-SININA. We assume that $G_1$ and $G_2$ share at least one wire otherwise the proof is evident. Without loss of generality we assume the output of $G_2$ is not an input to $G_1$ (thus $G_1$ is evaluated before or at the same time as $G_2$). Since we defined simulatability (Definition 3) with computational security, we consider the passive threshold of

the sharing scheme to be large enough for the probability of simulation failure for $G_1$ and $G_2$ to be negligible.

Take an arbitrary set of $k_1$ errors on each input of $G_3$, arbitrary sets of $k_2$ intermediate errors and $d_1$ probes on $G_3$, and an arbitrary set of $d_2$ probes on the output of $G_3$, such that $d_1 + d_2 \leq d$ and $k_1 + k_2 \leq k$. Because $G_3$ is the composition of $G_1$ and $G_2$, we divide the $d_1$ probes as $d_3$ probes on $G_1$ and $d_4$ probes on $G_2$ such that $d_3 + d_4 = d_1$. Similarly, we have $k_3$ faults on $G_1$ and $k_4$ faults on $G_2$ such that $k_3 + k_4 = k_2$.

We first prove the correctness of $G_3$, since $G_1$ is instantiated with $k_1$ errors on each input and $k_3$ intermediate errors, such that $k_1 + k_3 \leq k$, and since $G_1$ is $(d, k)$-SININA its output has at most $k_3 \leq k_2$ errors on it. Now $G_2$ is instantiated with either $k_1$ or $k_3$ faults on each input and has $k_4$ intermediate faults, since $G_2$ is $(d, k)$-SININA and $k_1 + k_4 \leq k$, $k_3 + k_4 \leq k$, it follows that $G_2$'s output contains at most $k_4 \leq k_2$ errors. Since the combined outputs of $G_1$ and $G_2$ have less than $k_2$ faults on them, we have proven the correctness of $G_3$.

Second, we prove the privacy of $G_3$. There are $d_2$ probes on $G_2$'s output and $d_4$ intermediate probes. Since $G_2$ is $(d, k)$-SININA, the adversary learns at most $d_4$ input shares of $G_2$. Since the output of $G_1$ is either the output of $G_3$ or the input to $G_2$ and since $d_3 + d_4 + d_2 \leq d$, we can simulate the probed values of $G_1$ and $G_2$ with at most $d_3 + d_4 \leq d_1$ shares of each input. This proves the privacy of $G_3$ and thus that $G_3$ is $(d, k)$-SININA. □

# E   Proof of Theorem 5

We show that the serial composition of a NINA and an SNINA gadget is again SNINA where we assume for simplicity that each gadget has one shared output.

*Proof.* For brevity, we only show the proof where $G_1$ is NINA and $G_2$ is SNINA. Assume we have two gadgets $G_1$ and $G_2$ where we call their serial composition $G_3$. We prove that $G_3$ is $(d, k)$-SNINA given that $G_1$ is $(d, k)$-NINA and $G_2$ is $(d, k)$-SNINA. Since we defined simulatability (Definition 3) with computational security, we consider $d$ large enough for the probability of simulation failure for $G_1$ and $G_2$ to be negligible.

Take an arbitrary set of $k_1$ errors on the input of $G_3$, arbitrary sets of $k_2$ intermediate errors and $d_1$ probes on $G_3$, and an arbitrary set of $d_2$ probes on the output of $G_3$, such that $d_1 + d_2 + k_1 + k_2 \leq d$ and $k_1 + k_2 \leq k$. Because $G_3$ is the combination of $G_1$ and $G_2$ we divide the $d_1$ probes as $d_3$ probes on $G_1$ and $d_4$ probes on $G_2$ such that $d_3 + d_4 = d_1$. The same with the faults where we have $k_3$ faults on $G_1$ and $k_4$ faults on $G_2$ such that $k_3 + k_4 = k_2$. We need to prove that $G_3$ is correct and private.

We first prove the correctness. We know from $G_1$ being $(d, k)$-SNINA that $G_1$ gives an output containing at most $k_3$ errors or it aborts. This holds since $G_1$ is faulted by $k_3$ intermediate errors where its inputs are instantiated with $k_1$ faults such that $k_1 + k_3 \leq k$. Now $G_2$ is instantiated with $k_1 + k_3$ faults on the input and $k_4$ intermediate faults. Since $G_2$ is $(d, k)$-SNINA and $k_1 + k_3 + k_4 \leq k$,

26

we know that $G_2$ gives back an output with at most $k_4 \leq k_2$ errors or it aborts. This proves the correctness of $G_3$.

Second, we prove the privacy of $G_3$. First, if $G_1$ aborts, we know that we can simulate this abort signal with $d_3 + k_1 + k_3$ input shares of $G_1$ and the injected errors. We look at the case where $G_1$ does not abort. From the previous paragraph we know that $G_2$ is instantiated with at most $k_1 + k_3$ faults on its input and with $k_4$ intermediate faults where there are $d_2$ probes on $G_2$'s output and $d_4$ intermediate probes. Thus the simulator can simulate the probes and the abort signal of $G_2$ using the knowledge from $d_4 + k_1 + k_3 + k_4$ of its input values and the injected errors. Since $G_1$ is $(d, k)$-NINA and $k_1 + k_3 \leq k$, each fault causes at most one output share to be faulty and each fault causes the adversary to learn at most one input share. Thus, we have that the $k_1 + k_3$ faults on $G_1$ including the faulty inputs to $G_2$ can be simulated by $k_1 + k_3$ inputs of $G_1$. Thus we can simulate the probes of $G_1$ and $G_2$ with $d_3 + d_4 + k_1 + k_3 + k_4 = d_1 + k_1 + k_2$ shares of the input of $G_3$. This proves the privacy of $G_3$ and thus that $G_3$ is $(d, k)$-SNINA. $\qquad\square$

## F    Variants of the NINA Definition

We define two notions which are in between the NINA and SNINA properties, meaning that they imply NINA but are not sufficient for arbitrary compositions.

**Definition 15 $((d, k)$ SNI-NA).** *A gadget $G$ is $(d, k)$ SNI-NA if for any set of $d_1$ probes and $k' \leq k$ errors on the intermediate variables and any set of $d_2$ probes on the output, such that $d_1 + d_2 + k' \leq d$, the following holds.*

(a) *Privacy: The probes and the abort signal can be simulated with $d_1 + k'$ shares of each input and the injected errors.*
(b) *Correctness: The protocol either aborts or gives an output with at most $k'$ errors.*

**Definition 16 $((d, k)$ NI-SNA).** *A gadget $G$ is $(d, k)$ NI-SNA if for any set of $k_1$ errors on each input and any set of $d'$ probes and $k_2$ errors on the intermediate variables, such that $d' + k_1 + k_2 \leq d$ and $k_1 + k_2 \leq k$, the following holds.*

(a) *Privacy: The probes and the abort signal can be simulated with $d' + k_1 + k_2$ shares of each input and the injected errors.*
(b) *Correctness: The protocol either aborts or gives an output with at most $k_2$ errors.*

While these notions do not allow general composition (e.g., the arbitrary composition between SNI-NA gadget might not be SNI-NA), a careful designer might use these relaxed notions to attain better performances.

## G    Correctness and Passive Security of Algorithm 2

In this appendix, we prove the correctness and the passive security of the duplicated Boolean multiplication gadget (Algorithm 2), where the proofs are made using the definitions given in Section 3.2.

### G.1 Correctness of Algorithm 2

We verify that multiplying two duplicated Boolean masked values following Algorithm 2 results in a duplicated Boolean sharing of the product of the two input secrets.

**Theorem 6.** *Given correct input sharings of a and b, Algorithm 2 does not abort and the output shares $c_i$ are duplicated Boolean shares of ab.*

*Proof.* We first show that the algorithm does not abort. Since for all $i, j \in [d+1]$ and $\ell \in [k+1]$, $a_{i,1} = a_{i,\ell}$ and $b_{j,1} = b_{j,\ell}$, we know that $u_{i,j,1} = u_{i,j,\ell}$. Since we work over a binary field we know that $u_{i,j,1} + u_{i,j,\ell} = 0$ and thus the algorithm does not abort.

Secondly, we show that $c_{i,1} = c_{i,\ell}$ for all $i \in [d+1]$ and $\ell \in [k+1]$. From the previous paragraph we know that $u_{i,j,1} = u_{i,j,\ell}$ and that $a_{i,1}b_{j,1} = a_{i,\ell}b_{j,\ell}$ thus the sum of these is still equal.

Lastly, we show that $\sum_{i=1}^{d+1} c_{i,\ell} = ab$ for all $\ell \in [k+1]$. We know that $c_{i,1} = c_{i,\ell}$ so we show the proof for $\ell = 1$.

$$\sum_{i=1}^{d+1} c_{i,1} = \sum_{i=1}^{d+1} \left( a_{i,1}b_{i,1} + \sum_{j \neq i} u_{i,j,1} \right) =$$

$$\sum_{i=1}^{d+1} \left( a_{i,1}b_{i,1} + \sum_{j < i} (a_{i,1}b_{j,1} + r_{i,j}) + \sum_{j > i} (a_{i,1}b_{j,1} + r_{j,i}) \right) =$$

$$\sum_{i=1}^{d+1} \left( a_{i,1}b_{i,1} + \sum_{j < i} a_{i,1}b_{j,1} + \sum_{j > i} a_{i,1}b_{j,1} \right) =$$

$$\sum_{i=1}^{d+1} a_{i,1} \sum_{j=1}^{d+1} b_{j,1} = b \sum_{i=1}^{d+1} a_{i,1} = ab$$

$\square$

### G.2 Passive Security of Algorithm 2

We prove that Algorithm 2 is $d$-SNI as defined in Section 3. We recall that the proof concerns the creation of a simulator which, given a limited set of shares, is capable of replicating the distribution of the adversary's probed values of the circuit proper. Since Algorithm 2 is given unique and independent uniform random values, it suffices to show that the adversaries probed values are uniformly distributed or directly related to the circuit's inputs.

**Theorem 7.** *Algorithm 2 is d-SNI.*

*Proof.* Take an arbitrary set of $d_1$ probes on the intermediate variables of Algorithm 2 and $d_2$ probes on its output such that $d_1 + d_2 \leq d$. We construct a

simulator for the adversary's probed values which makes use of at most $d_1$ shares of the secrets $a, b$. We classify the internal variables in the following groups where we bundle all duplicated shares.

(1) $a_i = \{ a_{i,\ell} \mid \ell \in [k+1] \}$, $b_i = \{ b_{i,\ell} \mid \ell \in [k+1] \}$
(2) $r_{i,j}$
(3) $u_{i,j} = \{ u_{i,j,\ell} \mid \ell \in [k+1] \}$
(4) $c_i = \{ c_{i,\ell} \mid \ell \in [k+1] \}$

We define two sets of indices $I$ and $J$ such that $|I| \leq d_1$, $|J| \leq d_1$ and the probed values can be perfectly simulated given only the knowledge of $\{a_{i,1}\}_{i \in I}$ and $\{b_{j,1}\}_{j \in J}$. The sets are constructed as follows.

- For every probe in group $(1)$ add $i$ to $I$ and to $J$.
- For every probe in groups $(2), (3)$ add $i$ to $I$ and $j$ to $J$.
- For every probe in group $(4)$ add $i$ to $I$ and for every probed $u_{i,j}$ or $r_{i,j}$ add $j$ to $J$.

Since the adversary is allowed to make at most $d_1$ internal probes, we have $|I| \leq d_1$ and $|J| \leq d_1$. We now show how the simulator behaves for the internal observed values.

- For each observation as in group $(1)$, by definition of $I$ and $J$ the simulator has access to $a_{i,1}$ and $b_{i,1}$ which are equal to $a_{i,\ell}, b_{i,\ell}$ for $\ell \in [k+1]$. Thus we can perfectly simulate the values.
- For each observation as in group $(2)$, we simulate $r_{i,j}$ as a uniform random variable.
- For each observation as in group $(3)$, the simulator has access to $a_{i,1}$ and $b_{j,1}$, and $r_{i,j}$ can be simulated as a uniform random variable if it was not simulated before. Since $a_{i,1}, b_{j,1}$ is equal to $a_{i,\ell}, b_{j,\ell}$ for $\ell \in [k+1]$ and all $u_{i,j,\ell}$ for $\ell \in [k+1]$ are masked with the same $r_{i,j}$, we can perfectly simulate the $u_{i,j,\ell}$ values.
- For each observation as in group $(4)$, by definition $i \in I, J$. At first we assign a random value to every term $u_{i,j,\ell}$, with $j \in [d+1]$. Then, if one of the terms $u_{i,j,\ell}, u_{j,i,\ell}$ or $r_{i,j}$ was already been probed, since by definition $i \in J$ and $j \in J$ the simulator knows $a_{i,\ell}$ and $b_{j,\ell}$ and can simulate $u_{i,j,\ell}$. Otherwise $r_{i,j}$ is not viewed by any other probe and thus is perfectly simulated as a uniform random value.

We now simulate the outputs $c_{i,\ell}$ using no information from the inputs. We have to take into account the following cases.

- If the attacker has already observed some of the internal values, then the partial sums $u_{i,j,\ell}$ or $a_{i,\ell} b_{i,\ell}$ previously probed are already simulated. As for the remaining terms, we note that by definition of the scheme there always exists one random bit $r_{i,j}$ which does not appear in the computation of any other observed element. Therefore the simulator can assign a random and independent value to $c_{i,\ell}$.

– If the attacker has only observed output shares, then we point out that, following Algorithm 2, each of them is composed by $d$ random bits $r_{i,j}$ and at most one of them can enter in the computation of each other output variable $c_{j,\ell}$. Since the adversary may have previously probed at most $d-1$ of them, there exist one random bit $r_{i,j}$ which does not appear in the computation of any other observed value. Thus the simulator can assign a random and independent element to $c_{i,\ell}$, completing the proof.

$\square$

## H  Active and Combined Security of Algorithm 2

In this section, we prove the active and combined security of the error detecting duplicated Boolean multiplication gadget (Algorithm 2) using our composable security definitions given in Section 3.2. We note that for completeness and comparison purposes, we give a proof of the correctness and the SNI security of Algorithm 2 in Appendix G.

### H.1  Active Security of Algorithm 2

We prove the correctness of Algorithm 2 in the presence of active attacks using the $k$-SNA model.

The proof consists of showing that if any errors are present on the input of Algorithm 2, the algorithm aborts the computation and if an error is injected in one of the intermediate variables of the algorithm, the fault either affects only one output share or the algorithm aborts.

**Theorem 8.** *Algorithm 2 is $k$-SNA.*

*Proof.* Take an arbitrary $k_2$ faults on the intermediate variables and $k_1$ faults on each input such that $k_1 + k_2 \leq k$. We show that Algorithm 2 either aborts or gives back a result with at most $k_2$ errors. We classify the internal variables in the following groups.

(1) $a_{i,\ell}, b_{i,\ell}$
(2) $r_{i,j}$
(3) $u_{i,j,\ell}$
(4) $t_{i,j,\ell}$
(5) $c_{i,\ell}$

We go over all possible faults and determine whether the algorithm aborts or which outputs $c_{i,\ell}$ are affected. Recall that Algorithm 2 has a total of $(d+1)^2 k$ error checks by verifying that $u_{i,j,1} = u_{i,j,\ell}$ for all $i,j \in [d+1]$ and $\ell \in [k+1]$. We distinguish the following cases.

– The adversary used no faults. Then we know from the proof of correctness (Theorem 6) that Algorithm 2 gives back a correct result.

30

- The adversary injected a fault in group (1). For ease we consider that $a_i$ was faulted. We go over the following sub-cases.
  - No cross products $u_{i,j,\ell}$ are faulty. In this case all $b_{j,\ell}$ for $j \in [d+1]$ were equal to zero. As a result, the faults on the $a$ shares are removed, the algorithm does not abort and gives back a correct output.
  - At least one $u_{i,j,\ell}$ is faulty. In this case there are $k$ error checks performed on the faulty $u_{i,j,\ell}$. Since the adversary can only inject $k-1$ other faults, it can not bypass all checks and the algorithm aborts.
- The adversary injected a fault in group (2). This fault affects all $u_{i,j,\ell}$ and $u_{j,i,\ell}$ for $\ell \in [k+1]$. As a result, this fault goes undetected. However, it does not affect the correctness of the algorithm as the output is still a duplicated Boolean sharing of $ab$.
- The adversary injected a fault in group (3). This fault would be detected by the $k$ error checks on $u_{i,j,\ell}$ for $\ell \in [k+1]$. As the adversary can only fault $k-1$ other variables, it can not bypass these checks and the error will be detected.
- The adversary injected a fault in group (4). As this value is only used to check whether an error is present on the cross products, it does not influence the output of the algorithm.
- The adversary injected a fault in group (5). In this case $c_{i,\ell}$ is faulty.

As the only faults which affect the output shares are those in group (5) and since the adversary can only inject $k_2$ intermediate errors, Algorithm 2 gives an output with at most $k_2$ erroneous output shares or it aborts. Thus we have proven that Algorithm 2 is $k$-SNA. $\qquad\square$

## H.2 Combined Security of Algorithm 2

Now we prove the SNINA security of Algorithm 2 as defined in Section 3.2. Intuitively, the proof is a combination of the SNI and SNA security where the simulator now needs to simulate the abort signal. This simulator is given a share of each input per probe and fault injected in the gadget. Since our multiplication gadget is a duplicated version of the Ishai et al. multiplication gadget with $d+1$ shares, similar proofs on its passive (SNI) security already exist in the literature such as the one from [18].

We note that the following proof shows there is an attack which has a probability of violating the privacy of the gadget. This probability can be made negligible in $d$ thus we show that Algorithm 2 is secure as per Definition 3.

**Theorem 9.** *Algorithm 2 is $(d,k)$-SNINA.*

*Proof.* The proof is split in two parts. In the first part we prove that a probing and faulting adversary can not get an output with more than the injected number of errors of erroneous shares. In the second part we prove that this adversary also can not break the privacy of the scheme.

We take arbitrary inputs and injected randomness. Take arbitrary $k_1$ errors on each input and $k_2$ intermediate errors such that $k_1 + k_2 \leq k$. We take $d_1$

31

arbitrary probes on the intermediate variables and $d_2$ probes on the output such that $d_1 + d_2 + k_1 + k_2 \leq d$.

First, we prove that given the faults and probes, the algorithm does not give an output with more than $k_2$ erroneous outputs. From Theorem 8 we know that for any set of $k_1$ errors on each input and $k_2$ errors on the intermediate variables, the gadget does not give an output with more than $k_2$ erroneous shares. Thus, the information given by the probes can not help the adversary to create an incorrect output.

Second, we prove the privacy of the scheme. We classify the internal variables in the following groups.

(1) $a_{i,\ell}, b_{i,\ell}$
(2) $r_{i,j}$
(3) $u_{i,j,\ell}$
(4) $t_{i,j,\ell}$
(5) $c_{i,\ell}$

We define two sets of indices $I$ and $J$ such that $|I| \leq d_1 + k_1 + k_2$, $|J| \leq d_1 + k_1 + k_2$ and such that the probed values and the abort signal can be simulated given only the knowledge of $\{a_{i,1}\}_{i \in I}$ and $\{b_{i,1}\}_{i \in J}$. The sets are constructed as follows.

- For every probe or fault as in group (1) add the corresponding $i$ to either $I$ or $J$.
- For every probe or fault as in groups $(2), (3), (4)$ add $i$ to $I$ and $j$ to $J$.
- For every probe or fault as in group (5) add $i$ to $I$ and for every probed $u_{i,j,\ell}$, $r_{i,j}$ or $t_{i,j,\ell}$ add $j$ to $J$.

Since the adversary is allowed to make at most $d_1$ internal probes, $k_1$ faults on each input and $k_2$ intermediate faults, we have that $|I| \leq d_1 + k_1 + k_2$ and $|J| \leq d_1 + k_1 + k_2$. We now show how the simulator behaves for the internal observed values.

- For each observation as in group (1), since the probed input and the injected errors are given to the simulator, it can perfectly simulate this observation.
- For each observation as in group (2), we simulate $r_{i,j}$ as a uniform random variable.
- For each observation as in group (3), the simulator has access to $a_{i,1}$ and $b_{j,1}$ (a priori $a_{i,1} = a_{i,\ell}$ and $b_{j,1} = b_{j,\ell}$). In case $r_{i,j}$ was not directly probed it is simulated as a uniform random variable, otherwise it was already simulated. In case there were errors injected in the $a_{i,\ell}$, $b_{j,\ell}$, $r_{i,j}$ or $u_{i,j,\ell}$, the simulator is given these errors. As a result, the simulator can perfectly simulate the $u_{i,j,\ell}$.
- For each observation as in group (4). By definition $i \in I$ and $j \in J$. Since the simulator is given the injected faults, it knows the $a_{i,1}, b_{j,1}$ and $a_{i,\ell}, b_{j,\ell}$ used in the $u_{i,j,1}$ and $u_{i,j,\ell}$. Thus the simulation follows as in the previous step.

– For each observation as in group (5), by definition $i \in I, J$. In case the simulator knows the algorithm will abort, it simulates the value $c_{i,j,\ell}$ as $\perp$. Otherwise, we assign a random value to every term $u_{i,j,\ell}$ for $j \in [d+1]$. Then if one of the terms $u_{i,j,\ell}, u_{j,i,\ell}, t_{i,j,\ell}, t_{j,i,\ell}$ or $r_{i,j}$ was already probed or faulted, since by definition $i \in I$ and $j \in J$ the simulator knows $a_{i,\ell}$ and $b_{j,\ell}$ and can simulate $u_{i,j,\ell}$. Otherwise $r_{i,j}$ is not viewed or faulted by any other probe or fault and thus is perfectly simulated as a uniform random value.

We now simulate the outputs $c_{i,\ell}$ using no additional information from the inputs. Recall that faults are treated as probes. We have to take into account the following cases.

– In case the simulator knows the abort signal will be flagged, it simulates all $c_{i,\ell}$ as $\perp$.
– If the attacker has already observed some of the internal values, then the partial sums $u_{i,j,\ell}$ or $a_{i,\ell}b_{i,\ell}$ previously probed are already simulated. As for the remaining terms, we note that by definition of the scheme there always exists one random bit $r_{i,j}$ which does not appear in the computation of any other observed element. Therefore the simulator can assign a random and independent value to $c_{i,\ell}$ unless the $c_{i,\ell}$ was directly faulted then the simulator simulates it as the injected fault.
– If the attacker has only observed output shares, then we point out that, following Algorithm 2, each of them is composed by $d$ random bits $r_{i,j}$ and at most one of them can enter in the computation of each other output variable $c_{j,\ell}$. Since the adversary may have previously probed at most $d-1$ of them, there exist one random bit $r_{i,j}$ which does not appear in the computation of any other observed value. Thus the simulator can assign a random and independent element to $c_{i,\ell}$ or as an injected fault, completing the proof.

We go through the possible faults and show we can simulate the abort signal. We note that since we treat faults as probes and since the simulator is given the location and value of the errors, it knows if the injected faults were trivial or not, i.e., if the injected faults changed the original value. This also holds for when there are multiple faults on the same variable.

– For each fault as in group (1). In case there was an injected error which differed from the original value, the simulator aborts the algorithm. For brevity, we assume $a_{i,\ell}$ was faulted. From Theorem 8 we know that the real algorithm aborts unless all $b_{j,\ell} = 0$ for $j \in [d+1]$ in which case the simulator fails to simulate the real algorithm. However, since the input shares were randomised the probability this event occurs is $1/|\mathbb{F}_2|^{d+1}$. In case more than one input share was faulty, the probability of failing the simulation is smaller than $1/|\mathbb{F}_2|^{d+1}$.
– For each fault as in group (2), the simulator does not abort. From Theorem 8 we know that in case there is an error on the $r_{i,j}$, the algorithm never aborts thus this simulation is perfect.
– For each fault as in group (3), the simulator goes over all errors on $a_{i,\ell}$, $b_{j,\ell}$, $r_{i,j}$, $u_{i,j,\ell}$ and $t_{i,j,\ell}$ and checks if the total error is non-trivial in which

33

case the simulator aborts the algorithm. From Theorem 8 we know that in case there is an error on the $u_{i,j,\ell}$, the algorithm always aborts thus this simulation is perfect.

  – For each fault as in group (4). Similar to the previous step, the simulator checks the total error on both $u_{i,j,1}$ and $u_{i,j,\ell}$. Since the simulator is given both $a_{i,1}$ and $b_{j,1}$, it can perfectly simulate whether the algorithm aborts.
  – For each fault as in group (5), the simulator does not abort.

From Step 1 of the simulation of the abort signal we see that the simulation fails with a maximal probability of $1/|\mathbb{F}_2|^{d+1}$ for a fault in group (1). Since the simulator aborts in case such a fault occurs, the overall probability for the simulation to fail is $1/|\mathbb{F}_2|^{d+1}$ which can be made negligible in $d$.

Since $|I| \leq d_1 + k_1 + k_2$ and $|J| \leq d_1 + k_1 + k_2$ and since the abort signal and the probed variables are simulated using only the knowledge of $\{a_{i,1}\}_{i \in I}$ and $\{b_{i,1}\}_{i \in J}$, Algorithm 2 is $(d, k)$-SNINA. □

*Remark 2.* We have proven in Appendix B that the composition of SNINA gadgets is again an SNINA gadget but this does not hold if there is a non-negligible probability that the simulation fails, thus when $d$ is not taken sufficiently large. However, we note that Algorithm 2 is still composable as a failed attack causes the operation to abort and a successful attack breaks the privacy of the scheme. Since this is the only attack, the probability of a bad event stays the same when composed with itself or other SNINA gadgets.

## I    Combined Security of Algorithm 3

For completeness we show that the refresh and error check gadget (Algorithm 3) is $(d, k)$-SNINA. The proof follows closely to the one from Theorem 9. We again build a simulator given limited shares which is capable of simulating the probed values and the abort signal even when faults occur and we show that each fault can only one output or abort the algorithm.

**Theorem 10.** *Algorithm 3 is $(d, k)$-SNINA.*

*Proof.* The proof is split in two parts. In the first part we prove that the adversary who is given both probes and faults can not get an output with more than the injected number of errors of erroneous shares. In the second part we prove that this adversary also can not break the privacy of the scheme.

We take arbitrary inputs and injected randomness. Take arbitrary $k_1$ errors on each input and $k_2$ intermediate errors such that $k_1 + k_2 \leq k$. We take $d_1$ arbitrary probes on the intermediate variables and $d_2$ probes on the output such that $d_1 + d_2 + k_1 + k_2 \leq d$.

First, we prove that given the faults and probes, the algorithm does not give an output with more than $k_2$ erroneous outputs. From the error checking step $a_{i,1} + a_{i,\ell}$ for $i \in [d+1]$ and $\ell \in [k+1]$. We have that all $a_{i,m} = a_{i,\ell}$ for $i \in [d+1]$ and $m, \ell \in [k+1]$. Since the adversary can inject at most $k$ errors, any of the $k_1$ errors on the input would be detected and the algorithm would abort. We thus assume that $k_1 = 0$ and the algorithm does not abort. We distinguish two faults.

34

(a) A fault on $a_{i,\ell}$ after the error check.
(b) A fault on $r_{i,\ell}$.

A fault in group $(a)$ only affects one output share. A fault in group $(b)$ affects two output shares, but the output remains a duplicated Boolean sharing of $a$ thus the fault does not affect the correctness of the algorithm. Since the adversary can inject up to $k_2$ of these errors, only up to $k_2$ output shares can be faulty. As a result the correctness of the scheme is guaranteed following the definition of SNINA.

Second, we prove the privacy of the scheme. We classify the internal variables in the following groups.

(1) $a_{i,\ell}$
(2) $t_{i,\ell}$
(3) $r_{i,j}$
(4) $a_{i,\ell}$

We define a set of indices $I$ such that $|I| \leq d_1 + k_1 + k_2$ and such that the probed values and the abort signal is simulated given only the knowledge of $\{a_{i,1}\}_{i \in I}$. This set is constructed by adding $i$ to $I$ for every probe or fault as in groups (1)-(4). Since the adversary is allowed to make at most $d_1$ internal probes, $k_1$ faults on the input and $k_2$ intermediate faults, we have that $|I| \leq d_1 + k_1 + k_2$.

We now show that the simulator can simulate the probed values.

– For each observation as in group (1), since the probed input and the injected errors are given to the simulator, it can perfectly simulate this observation.
– For each observation as in group (2), the simulator has access to $a_{i,1}$ (a priori $a_{i,1} = a_{i,\ell}$). In case there were errors injected in $a_{i,1}$ or $a_{i,\ell}$, the simulator can perfectly simulate this as it has the injected errors.
– For each observation as in group (3), we simulate $r_{i,j}$ as a uniform random variable.
– For each observation as in group (4), by definition $i \in I$. In case the simulator knows the algorithm will abort, it simulates $a_{i,\ell}$ as $\perp$. We assign a random value to every term $r_{i,j}$ for $j \in [d+1]$. If one of the terms $r_{i,j}$ composing $a_{i,\ell}$ has been probed, it was already assigned a value otherwise we simulate it as a uniform random value. Since the simulator has access to the injected errors and since the creation of the $a_{i,\ell}$ is linear with respect to the $r_{i,j}$, the simulator can perfectly simulate an erroneous $a_{i,\ell}$.

We now simulate the outputs which we denote as $a'_{i,\ell}$. We have to take into account the following cases. Recall that faults are treated as probes.

– In case the simulator knows the abort signal will be flagged, it simulates all $a'_{i,\ell}$ as $\perp$.
– If the attacker has already observed some of the internal values, then $a_{i,\ell}$ is already given to the simulator. As for the remaining terms, we note that by definition of the scheme there always exists one random bit $r_{i,j}$ which does not appear in the computation of any other observed element. Therefore the simulator can assign a random and independent value to $a'_{i,\ell}$ unless the $a'_{i,\ell}$ was directly faulted then the simulator simulates it as the injected fault.

– If the attacker has only observed output shares we see that, following Algorithm 3, each of the $a'_{i,\ell}$ is composed by $d$ random bits $r_{i,j}$ and at most one of them can enter in the computation of each other output variable $a'_{j,\ell}$. Since the adversary may have previously probed at most $d-1$ of them, there exist one random bit $r_{i,j}$ which does not appear in the computation of any other observed value. Thus the simulator can assign a random and independent element to $a'_{i,\ell}$ or as an injected fault, completing the proof.

We then show that we can simulate the abort signal given the inputs $\{a_{i,1}\}_{i \in I}$. Since Algorithm 3 only aborts if an input is faulty and the simulator is given the faulted input, it is clear that we can simulate the abort signal.

As a result, since the probed variables are simulated using only the knowledge of $\{a_{i,1}\}_{i \in I}$ and $|I| \le d_1 + k_1 + k_2$, Algorithm 3 is $(d, k)$-SNINA. $\qquad\square$

## J   Combined Security of Algorithm 5

Since the proof of SNA for the error correcting duplicated Boolean multiplication gadget (Algorithm 5) would be similar to the one of the error detecting case (see Theorem 8), we only discuss the privacy of the algorithm following the definition of Strong Independent NINA (Definition 14).

**Theorem 11.** *Algorithm 5 is $(d, k)$-SININA.*

*Proof.* We take arbitrary inputs and injected randomness. Take arbitrary $k_1$ errors on each input and $k_2$ intermediate errors such that $k_1 + k_2 \le k$. We take $d_1$ arbitrary probes on the intermediate variables and $d_2$ probes on the output such that $d_1 + d_2 \le d$.

We only prove the privacy of the scheme. We classify the internal variables in the following groups.

(1) $a_{i,\ell}, b_{i,\ell}$
(2) $r_{i,j,\ell}$
(3) $u_{i,j,\ell,m}$
(4) $v_{i,j,\ell}$
(5) $c_{i,\ell}$

We define two sets of indices $I$ and $J$ such that $|I| \le d_1$, $|J| \le d_1$ and such that the probed values can be simulated given only the knowledge of $\{a_{i,1}\}_{i \in I}$ and $\{b_{i,1}\}_{i \in J}$. The sets are constructed as follows.

– For every probe as in groups $(1), (5)$ add $i$ to $I$ and $J$.
– For every probe as in groups $(3), (4)$ add $i$ to $I$ and $j$ to $J$.

Since the adversary is allowed to make at most $d_1$ internal probes, we have that $|I| \le d_1$ and $|J| \le d_1$. We now show how the simulator behaves for the internal observed values.

36

– For each observation as in group $(1)$, since the probed input and the injected errors are given to the simulator, it can perfectly simulate this observation.
– For each observation as in group $(2)$, we simulate $r_{i,j,\ell}$ as a uniform random variable.
– For each observation as in group $(3)$, the simulator has access to $a_{i,1}$ and $b_{j,1}$ (a priori $a_{i,1} = a_{i,\ell}$ and $b_{j,1} = b_{j,\ell}$). In case any of the $r_{i,j,m}$ were not directly probed they are simulated as uniform random variables, otherwise they were already simulated. In case there were errors injected in the $a_{i,\ell}$, $b_{j,\ell}$, $r_{i,j,m}$ or $u_{i,j,\ell}$, the simulator is given these errors. As a result, the simulator can perfectly simulate the $u_{i,j,\ell}$.
– For each observation as in group $(4)$. By definition $i \in I$ and $j \in J$. Since the simulator is given the injected faults, it knows the $a_{i,1}, b_{j,1}$ and $a_{i,\ell}, b_{j,\ell}$ used in the $u_{i,j,m}$. Thus the simulation follows as in the previous step.
– For each observation as in group $(5)$, by definition $i \in I, J$. Since each cross product $u_{i,j,\ell}$ is masked by $k+1$ random values, we know that the adversary could not have faulted all of them. From the error correcting properties of the $Maj$ function, we know that the $v_{i,j,\ell}$ can not contain errors unless directly faulted. As a result, we assign a random value to every term $v_{i,j,\ell}$ for $j \in [d+1]$ unless the term or $u_{i,j,\ell}$ was previously already probed and thus can be perfectly simulated.

We now simulate probed outputs $c_{i,\ell}$ using no information from the inputs. Recall that faults are treated as probes. We have to take into account the following cases.

– If the attacker has already observed some of the internal values, then the partial sums $v_{i,j,\ell}$ or $a_{i,\ell}b_{i,\ell}$ previously probed are already simulated. As for the remaining terms, we note that by definition of the scheme there always exists one random bit $r_{i,j,m}$ which does not appear in the computation of any other observed element. Therefore the simulator can assign a random and independent value to $c_{i,\ell}$ unless the $c_{i,\ell}$ was directly faulted then the simulator simulates it as the injected fault.
– If the attacker has only observed output shares, then we point out that, following Algorithm 5, each of them is composed by $d$ random bits $r_{i,j,m}$ and at most one of them can enter in the computation of each other output variable $c_{j,\ell}$. Since the adversary may have previously probed at most $d-1$ of them, there exist one random bit $r_{i,j,m}$ which does not appear in the computation of any other observed value. Thus the simulator can assign a random and independent element to $c_{i,\ell}$ or as an injected fault, completing the proof.

Since $|I| \leq d_1$ and $|J| \leq d_1$ and the probed variables are simulated using only the knowledge of $\{a_{i,1}\}_{i \in I}$ and $\{b_{i,1}\}_{i \in J}$, Algorithm 5 is $(d,k)$-SININA. $\square$