CINI MINIS: Domain Isolation for Fault and Combined Security

Jakob Feldtkeller
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
jakob.feldtkeller@rub.de

Jan Richter-Brockmann
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
jan.richter-brockmann@rub.de

Pascal Sasdrich
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
pascal.sasdrich@rub.de

Tim Güneysu
Ruhr University Bochum
Horst Görtz Institute for IT Security
Bochum, Germany
tim.gueneysu@rub.de

ABSTRACT

Observation and manipulation of physical characteristics are well-known and powerful threats to cryptographic devices. While countermeasures against passive side-channel and active fault-injection attacks are well understood individually, combined attacks, i.e., the combination of fault injection and side-channel analysis, is a mostly unexplored area. Naturally, the complexity of analysis and secure construction increases with the sophistication of the adversary, making the combined scenario especially challenging. To tackle complexity, the side-channel community has converged on the construction of small building blocks, which maintain security properties even when composed. In this regard, Probe-Isolating Non-Interference (PINI) is a widely used notion for secure composition in the presence of side-channel attacks due to its efficiency and elegance. In this work, we transfer the core ideas behind PINI to the context of fault and combined security and, from that, construct the first trivially composable gadgets in the presence of a combined adversary.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**.

KEYWORDS

Side-Channel Analysis; Fault-Injection Analysis; Combined Attacks; Gadgets; Probe-Isolating Non-Interference

ACM Reference Format:
Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. 2022. CINI MINIS: Domain Isolation for Fault and Combined Security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3560614

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11.
https://doi.org/10.1145/3548606.3560614

1 INTRODUCTION

In contrast to standard assumptions in theoretical models of cryptography, physical devices are not black boxes that perfectly hide internal computations. Instead, an adversary can observe and manipulate physical characteristics to reveal and determine intermediates, ultimately violating given security guarantees. To capture *passive* Side-Channel Analysis (SCA), which exploits dependencies between secrets and characteristics such as timing [30], instantaneous power consumption [31], or electromagnetic emanations [24], the theoretical models were extended with access to intermediates [11, 18, 26]. A well researched countermeasure against SCA is *masking* [11] that randomizes intermediate values via secret sharing and benefits from a broad theoretical and formal foundation. In contrast, for *active* Fault Injection Analysis (FIA), which manipulates physical characteristics for example through clock glitches [16], voltage glitches [45], electromagnetic pulses [14, 19], or focused laser beams [43], theoretical models were extended with the ability to manipulate internal computations [39]. Here, a well researched countermeasure is *redundancy*, either in time, space, or information [1, 33].

Combined Attacks. After reaching a deep understanding of theoretical models for both passive and active attacks separately, the research community is now ready to combine both models and consider an adversary able to launch SCA and FIA simultaneously, denoted by the term Combined Analysis (CA). Already existing research in this area has shown that the trivial combination of masking and redundancy is insufficient to protect against such a powerful adversary [36] and, hence, dedicated and formal methods are required that consider reciprocal effects, e.g., inspired by Multi-Party Computation (MPC) [35].

Composable Gadgets. The complexity of security analysis and verification increases with the power of the adversary and the size of the design under consideration and rapidly becomes prohibitive. Nevertheless, to construct formally proven systems, the research community in SCA focuses on the construction of secure building blocks, so called *gadgets*, that maintain their security properties when composed to larger circuits. Unfortunately, the broadly used security guarantees of the Ishai-Sahai-Wagner (ISW) probing security [26] are not sufficient for secure composition and additional notions needed to be introduced. In essence, those composability

notions restrict the propagation of leakage by a combination of isolation and re-randomization. For this, Barthe et al. introduced Non-Interference (NI) [3] and Strong Non-Interference (SNI) [4] which achieve composition by preventing the propagation of leakage between different gadgets. Later, Cassiers and Standaert proposed Probe-Isolating Non-Interference (PINI) [10] which introduces and isolates so called *share domains*. For most cases, PINI has proven to be more efficient, as it allows the trivial implementation and composition of linear functions.

Similarly, definitions of composability notions for FIA and CA, based on NI and SNI, were proposed by Dhooghe and Nikova [15] and later refined by Richter-Brockmann et al. [36]. Those notions come in different variants, a simpler version where the order of FIA security is dependent on the order of SCA protection and a stronger variant with independence between SCA and FIA. However, there are currently no composable hardware gadgets for CA as the first proposal by Dhooghe and Nikova is not transferable to hardware and their second proposal has been shown to be flawed by Richter-Brockmann et al. [36].

Contributions. In this work, we provide multiple composability notions, inspired by PINI, for both FIA and CA. As a warm-up, we introduce Fault-Isolating Non-Interference (FINI) (Section 3) for FIA that formalizes the well known security and composability guarantees from spacial replication. For CA, we define a *Shared Redundancy Domain (SRD)* and introduce two different composability notions based on the isolation of both share domains and SRDs. While the first notion, Combined-Isolating Non-Interference (CINI) (Section 4), introduces a dependency between the order of SCA and FIA security, the stronger Independent Combined-Isolating Non-Interference (CINI_{ind}) (Section 5) eventually achieves independence between SCA and FIA security.

For all three composability notions we provide practical gadgets and formally prove their security. Here, we focus on the combination of masking with spacial replication in the context of CA. Further, we integrated our notions into a state-of-the-art verification tool for CA [36] and circuit compiler [22] (Section 6). Finally, we provide a rigorous evaluation of all proposed gadgets both for security and implementation costs in hardware (Section 7).

Please note, the proofs to all theorems can be found in the extended version of the paper [23].

2 PRELIMINARIES

2.1 Notation

Important notations are given in Table 1. We use subscripts to denote the share index and superscripts to denote the replication index. Further, we use the notion of a *faulty value* as shorthand for a fault that was injected in the gate that produces the corresponding value.

2.2 Circuit Model

For the sake of simplicity, but without loss of generality, we restrict the set of combinational gates to $\mathcal{G}_{\mathsf{c}} = \{\mathsf{not}, \mathsf{and}, \mathsf{nand}, \mathsf{xor}, \mathsf{xnor}\}$ and the set of memory gates to clocked registers $\mathcal{G}_{\mathsf{m}} = \{\mathsf{reg}\}$. In addition, we define a randomness gate $\mathcal{G}_{\mathsf{rand}} = \{\mathsf{rand}\}$ that has no input and for each clock cycle outputs an independent and uniformly chosen random value.

Table 1. Notations used throughout this work.

	Notation	Description
SCA	d	Security order of a masking scheme (countermeasure).
	s	Number of shares used by a masking scheme.
	i, j	Index of share domain
FIA	k	Security order of redundancy scheme (countermeasure).
	n	Number of duplications used by a redundancy scheme.
	ℓ, ℓ'	Index of redundancy domain
	t	Fault type
Misc	C, G	Represents a digital logic circuit or gadget, respectively.
	F	Functions are written in sans serif font.
	$\mathcal{S}$	Sets are denoted as upper-case characters in calligraphic font.
	$\text{Reg}[a]$	Hardware register with input a.

Then, we model a digital logic circuit C as a *directed acyclic graph* $\mathcal{D} = \{\mathcal{V}, \mathcal{E}\}$, where vertices $v \in \mathcal{V}$ represent logical gates $g \in \mathcal{G}_{\mathsf{c}} \cup \mathcal{G}_{\mathsf{m}} \cup \mathcal{G}_{\mathsf{rand}}$ and edges $e \in \mathcal{E}$ represent wires connecting two gates and carrying an element of the finite field $\mathbb{F}_2$. Please note, as we do not consider leakage from transitions[1], it is sufficient to analyze unrolled implementations even when there are loops in a circuit.

2.3 Security via Simulation

Simulation is a proof technique for security arguments that is useful for statements about composability [8, 34]. For that, we define a *real* and an *ideal* game, where the ideal game is trivially secure (under some adversary model). The real game is secure iff there exists no adversary who can distinguish the real from the ideal game with a probability higher than $\frac{1}{2}$. We will define the ideal game as a probabilistic polynomial-time simulator that reproduces the view of the adversary without access to any secret.

2.4 Side-Channel Security

Adversary Model. An adversary $\mathcal{A}_p$ in the context of stateless probing security [26], is given access to a circuit C that can be invoked multiple times. Prior to each invocation, $\mathcal{A}_p$ can select up to d wires of C, so called *probes*. The view of the adversary $\mathcal{A}_p$ is defined by the glitch-extended probes [20], i.e., by the exact values of all registers a probed wire directly depends on[2]. Further, a probe propagates into another wire whenever this wire is required for simulation of the probe [10].

Probing Security. In this context, *probing security* [26] is defined as the view of the adversary $\mathcal{A}_p$ always being independent of any secret, i.e., all probes can be simulated without any knowledge apart from the structure of C. Please note, that probing security does not capture horizontal attacks [5].

Masking. A promising and well studied countermeasure against SCA is *Boolean masking* [11], where each value $x \in \mathbb{F}_2$ is replaced by a vector $\langle x_0, \dots x_{s-1} \rangle \in \mathbb{F}_2^s$ such that each x_i is uniform random from $\mathbb{F}_2$ for $i \leq s-2$ and $x_{s-1} = \bigoplus_{i=0}^{s-2} x_i \oplus x$. We call x_i a *share* of x with *share index* i. The construction ensures that all subsets of up

[1] Transitions are physical defaults that occur due to switching activities within a circuit.
[2] Glitches are physical defaults that occur when there are timing differences in the propagation path of signals. Hence, providing $\mathcal{A}_p$ with all stable inputs of a probed wire captures all possible leakage via glitches [20].

to $s-1$ shares are independent of x. We say a share x_i violates the *independence property* of Boolean sharing iff the subset of all other shares is not independent of x. Similarly, a circuit is transformed to a *shared circuit* by transferring each operation to a representation that operates over the share vector of its inputs and outputs a share vector of its output. Hence, the initial sharing and final unsharing operation are not part of the shared circuit and cannot be probed by $\mathcal{A}_p$ [2, 26].

Probe-Isolating Non-Interference. The notation of probing security is not composable on its own, i.e., combining two probing secure circuits does not necessarily result in a probing secure circuit itself. For this, additional composability notations are introduced that define how to construct atomic building blocks, so called *gadgets*, that can be securely combined to larger circuits. PINI [10] ensures composability by dividing the circuit into *share domains*:

Definition 2.1 (Share Domain). The *share domain i* of a shared circuit is defined by all wires with share index *i*.

Given this, PINI ensures that information leakage cannot propagate from one share domain into another, i.e., isolating share domains and, hence, allows trivial composition as long as the inputs and outputs are a valid Boolean sharing. The efficiency of PINI comes from the fact that the share-wise implementation of the addition (xor) is PINI.

Definition 2.2 (Probe-Isolating Non-Interference [10]). A gadget G is d-PINI iff for any set of d_1 internal probes and any set $\mathcal{S}_2$ of d_2 share domains, such that $d_1 + d_2 \leq d$, there exists a set $\mathcal{S}_1$ of at most d_1 share domains such that the outputs of the share domains in $\mathcal{S}_2$ and the probes can be simulated with the inputs of the share domains in $\mathcal{S}_1 \cup \mathcal{S}_2$.

Hardware Private Circuits. Cassiers et al. [9] propose two multiplication gadgets fulfilling PINI, namely HPC_1 and HPC_2 (Algorithm 2 and 4 without highlighted parts, respectively). Intuitively, both gadgets stop cross-domain leakage by refreshing the masked input b before performing a shared multiplication. However, HPC_2 requires less randomness as it avoids a refreshing in the multiplication step via the *masked shares multiplication trick* [10], computing $(a_i+1)\cdot r_{i,j} + a_i \cdot (b_j + r_{i,j})$ instead of $a_i \cdot (b_j + \tilde{r}_{i,j}) + r_{i,j}$, as is done in HPC_1 (cf. Algorithm 2). For that, the security of HPC_2 rests on the fact that if $a_i = 0$ then $r_{i,j}$ is only observable in $(a_i + 1) \cdot r_{i,j}$ (as $a_i \cdot (b_j + r_{i,j}) = 0$) and otherwise $r_{i,j}$ is only observable in $a_i \cdot (b_j + r_{i,j})$, which ensures a proper masking of b_j. Recently, Knichel and Moradi [28] proposed HPC_3 as a latency optimized alternative, similarly based on the masked shares multiplication trick.

2.5 Fault-Injection Security

Adversary Model. An adversary $\mathcal{A}_f$ in the context of fault security [36] is given access to a circuit C that can be invoked multiple times. Prior to each invocation, $\mathcal{A}_f$ selects up to k gates in C and a fault type from the set of allowed fault types $\mathcal{T}$ for each gate. Faults are modeled by transforming the selected gates to a different gate type which is specified by the fault type $t \in \mathcal{T}$ [39]. Typical fault types are *set*, *reset* (replacing the targeted gate with a constant one or zero, respectively), or *bit flips* (inversion of the gate). A fault propagates into another wire whenever the value of the wire is influenced by the fault. The view of the adversary $\mathcal{A}_f$ is defined by the abort signal (if existent) and the correctness of the outputs. Correctness is defined by equivalence to the *golden circuit*, which is a fault-free version of C.

Fault Security. Let G^D be a circuit realizing a fault detection or correction mechanism for up to k faults. Then *fault security* is defined as the output of the concatenation $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$ always being equal to the output of the golden circuit of C or aborting [36]. Please note, that fault security is not composable on its own.

Redundancy. Protection mechanisms against fault attacks usually depend on redundancy (e.g., in time, space, or information). The simplest form of redundancy is *replication*, where all data and operations are implemented multiple times in parallel. When using $k+1$ instances of a circuit C, up to k faults can be detected by comparing all instances and aborting when one instance is different. Similarly, $2k+1$ instances allow the correction of up to k faults via a majority function correcting to the value that occurs most often in the different instances. Again, the initial replication and final error detection/correction is not part of the replicated circuit and cannot be faulted by $\mathcal{A}_f$ [36]. Naturally, redundancy can be applied at the level of single bits. Please note, that all replications share the same random values wherever fresh randomness is required.

2.6 Combined Security

Adversary Model. An adversary $\mathcal{A}_c$ in the context of combined security [36] is the combination of the adversaries $\mathcal{A}_p$ and $\mathcal{A}_f$. Hence, $\mathcal{A}_c$ is given access to a circuit C that can be invoked multiple times and prior to each invocation $\mathcal{A}_c$ can select up to d wires of C that are probed and up to k gates of C that are faulted according to a fault type $t \in \mathcal{T}$. The view of the adversary $\mathcal{A}_c$ is defined by the glitch-extended probes, the abort signal (if existent), and the correctness of the outputs of the concatenation $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$, where G^D is a circuit realizing a fault detection or correction mechanism for up to k faults. Correctness is again defined by equivalence to the golden circuit of C. Here, however, the golden circuit of C incorporates the faults targeting randomness gates $g \in \mathcal{G}_{\mathsf{rand}}$, i.e., $g \in \mathcal{G}_{\mathsf{rand}}$ are replaced by some g' according to the fault model (see [36] for more details). Please note, that faulting some gates gives A_c knowledge about the changed distribution of the dependent values, however, not necessarily the concrete values. This knowledge of changed distributions can cause additional probes that need to be simulated when the independence property of an intermediate Boolean sharing is violated. In general, value distributions are sufficient for simulation and the changed distribution is given to $\mathcal{A}_c$ for faulty gadget inputs. A detailed discussion and justification of this model is given by Richter-Brockmann et al. [36].

Combined Security. Given some adversary $\mathcal{A}_c$ *combined security* is defined as the view of $\mathcal{A}_c$ being independent of any secret, i.e., the abort signal and the probes can be simulated only with the knowledge of the faults and the structure of C (*privacy*), and the concatenation $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$ always being equal to the output of the golden circuit of C or aborting (*correctness)* [36]. In accordance with probing and fault security, $\mathcal{A}_c$ is not allowed to probe or fault the initial sharing and replication nor the final unsharing and error

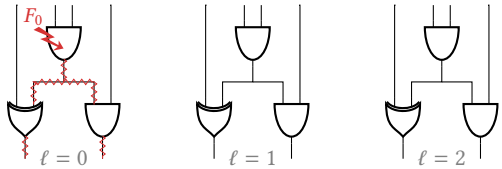

Figure 1. Isolation of fault propagation within redundancy domains.

detection/correction. Again, combined security is not sufficient for composition.

3 Fault-Isolating Non-Interference

When considering circuits, hardened against fault attacks by replication, we can observe that each injected fault can only propagate within the affected redundant part. Then, having more replications than allowed adversarial faults (maximum fault cardinality) can ensure the desired level of fault security. In addition, we observe that this property is inherently composable as long as the isolation of different redundant parts remains intact. Breaking this down to the core principles, namely the isolation of domains with regard to faults, replication reveals a strong similarity with PINI [10]. While PINI isolates probe propagation within share domains, replication isolates fault propagation within *redundancy domains*.

Definition 3.1 (Redundancy Domain). The *redundancy domain* ℓ of a redundant circuit is defined by all gates and wires with replication index ℓ.

In order to generalize the core principle of replication and allow a formal treatment, we now introduce the notion of Fault-Isolating Non-Interference. FINI is the dual to PINI in that it introduces redundancy domains and requires them to be isolated in terms of fault propagation (as illustrated in Figure 1). Then, assuming sufficient redundancy domains, detection or correction is always possible by comparing the values in different redundancy domains, regardless of the fault propagation within a single redundancy domain.

In this work, we focus on a realization of FINI via replication since it is an obvious match and transports the ideas and principles more easily. However, we intentionally construct FINI as a general notion that can be applied to other redundancy-based countermeasures with an appropriate definition of redundancy domains. When considering replication, the redundancy domain is defined by the replication index, i.e., all values with replication index ℓ belong to the redundancy domain ℓ.

As FINI, similar to PINI, introduces an isolation between different redundancy domains instead of an isolation between gadgets, faults at inputs are allowed to propagate to outputs of the same redundancy domain. Further, faults injected inside the gadget are restricted to only propagate to outputs belonging to a single redundancy domain. We now give a more formal definition in Definition 3.2.

Definition 3.2 (Fault-Isolating Non-Interference). A gadget G is k-FINI iff the following holds:

(i) For any set $\mathcal{F}_1$ of k_1 faulty redundancy domains and every set of k_2 faults injected in gates of G, with $k_1 + k_2 \leq k$, there exists a set of at most k_2 redundancy domains $\mathcal{F}_2$, such that the gadget either aborts or gives an output where all values, except those belonging to the redundancy domains $\mathcal{F}_1 \cup \mathcal{F}_2$, are equal to the values of the golden circuit.

(ii) There exists a decoding gadget G^{D}, such that given an input with at most k faulty redundancy domains and an abort signal, G^{D} either aborts or outputs a correct result.

Algorithm 1: FINI-secure correction gadget.

```
1 function Correct_FINI(a^0, ..., a^{n-1}):
      Require: n = 2k + 1
2     for ℓ = 0 to n − 1 do
3         b^ℓ ← maj(a^0, ..., a^{n-1})
4     return b^0, ..., b^{n-1}
```

3.1 FINI Security and Composition

FINI formalizes the intuitive security and compositional properties of replication codes. Here, the main argument for fault security comes from the fact that at most k redundancy domains can be manipulated while up to k manipulated redundancy domains can be detected or corrected by the corresponding decoding function.

Theorem 3.3. *A k-FINI gadget is k-fault secure.*

Now we argue that an arbitrary composition of FINI gadgets results in a (larger) FINI gadget. Again, this follows from the properties of replication codes, which introduce a natural isolation of different replications, i.e., a fault injected to one redundancy domain cannot propagate to another redundancy domain. This ensures that the upper bound of faulty redundancy domains remains unchanged after composition.

Theorem 3.4. *The composition of two k-FINI gadgets is k-FINI.*

Remark 1. *The connection of gadgets has to be consistent, i.e., redundancy domain ℓ of a gadget is connected only to redundancy domain ℓ of subsequent gadgets. However, we can permute the index of domains when necessary.*

3.2 FINI Gadgets

The construction of FINI gadgets for combinational gates follows a simple method: replication of the gate. By applying the compositional property of Theorem 3.4, the construction method can be generalized to all combinational or sequential circuits (which also includes the simple case of a single gate).

Theorem 3.5. *The $(k+1)$-times replication of any circuit is k-FINI.*

In Theorem 3.5 we instantiate the number of replications with $k + 1$, which is the minimum number required for fault detection via comparison. It is trivial to see that the same claim is true for implementations with more then $k + 1$ replications, as the comparison still detects faulty values. Similar, when using at least $2k + 1$ replications, we can use a majority function as decoding gadget additionally resulting in error correction.

The only gadgets that cannot be trivially constructed according to Theorem 3.5 are gadgets that combine different redundancy domains. The most prominent examples of this category are detection and correction modules. Here, the logic for detection/correction

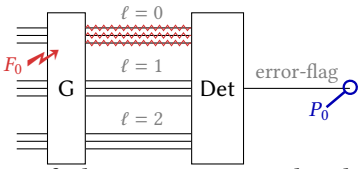

Figure 2. Insecure fault propagation in combined attack model. If all shares within one redundancy domain are faulty with a known biased distribution then the error flag indicates whether the faults represent a valid guess of the shares.

needs to be replicated for each redundancy domain, such that faults injected into this logic only affect one redundancy domain at the output [1, 42]. We show an example for this in Algorithm 1 considering a correction module[3].

4 Combined-Isolating Non-Interference

Combining both PINI and FINI is an obvious step to construct a composability notation for CA that is based on the isolation of domains. However, due to reciprocal effects and the dual nature of faults, which can both manipulate internal values and serve as a probe [12, 44], it is insufficient to isolate faults in redundancy domains (FINI) and probes in share domains (PINI) and a more complex notion is required. Those reciprocal effects can be easily seen when considering any HPC gadget where some randomness is faulted to a known value which nullifies the provided security guarantees. Similarly, with known faults, identifying correct *guesses* of shares becomes possible. Consider the example illustrated in Figure 2, where a gadget G with three shares and three redundancy domains is connected to a detection module. Further, assume that all shares in the first redundancy domain are faulty with a known and biased distribution (e.g., set/reset), representing an implicit guess on those shares. Then the error flag leaks information about all faulted shares, since it indicates the correctness of the guess, regardless of how the detection module is realized.

As a result the domain definition for fault propagation in the CA setting has to be more restrictive than given pure fault attacks. In particular, faults are restricted to propagate only in the combination of both share and redundancy domain. This ensures that a fault can leak at most one share domain even in circumstances where it can be used to learn values (similar to placing a probe).

Definition 4.1 (Shared Redundancy Domain). The *Shared Redundancy Domain (SRD)* (i, ℓ) of replicated and shared circuits is defined by all gates and wires with share index i and replication index ℓ.

Here, in contrast to faults, it is not necessary to further restrict the propagation of probes. On the contrary, as all replicated wires carry the same value, it is sufficient to probe one of those wires to learn all those values. Hence, each share domain (Definition 2.1) consists of multiple SRDs.

We now define CINI as a composability property for combined security, where faults are isolated within SRDs and probes within share domains. That is, for every set of probes and faults the number of input share domains $\mathcal{A}_c$ can learn is bounded by the sum of the cardinality and position of the probes and faults, and the number of faulty output SRDs is bounded by the cardinality and position of the faults, as visualized in Figure 3.

[3]The actual implementation of the function maj is irrelevant for the FINI property.

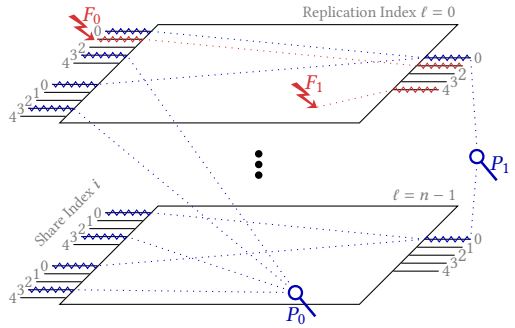

Figure 3. Propagation of probes and faults in the CINI context. While probes leak entire share domains (all inputs with same share index), faults are restricted to influence at most one output SRD. Further, probes at outputs restrict the leaked share domain of the inputs to be the same. Similar, input faults are restricted to affect the same SRD at the output.

Definition 4.2 (Combined-Isolating Non-Interference). A gadget G is (d, k)-CINI iff for any set $\mathcal{F}_1$ of k_1 faulty SRDs, every set of k_2 faults injected in gates of G, any set of d_1 probes placed on intermediate values, and any set $\mathcal{S}_2$ of d_2 share domains, such that $k_1 + k_2 \leq k$ and $d_1 + d_2 + k_1 + k_2 \leq d$, there exists a set $\mathcal{F}_2$ of at most k_2 SRDs and a set $\mathcal{S}_1$ of at most $d_1 + k_2$ share domains such that the following holds:

Correctness: The gadget either aborts or gives an output where all values, except those belonging to the SRDs $\mathcal{F}_1 \cup \mathcal{F}_2$, are equal to the golden circuit, and there exists a decoding gadget G^{D}, such that given an input with at most k faulty SRDs and an abort signal, G^{D} either aborts or outputs a correct result.

Privacy: The abort signal, the outputs of the share domains in $\mathcal{S}_2$, the outputs violating the independence property of Boolean sharing, and the probes can be simulated with the inputs of the share domains in $\mathcal{S}_1 \cup \mathcal{S}_2$ and knowledge of the faults both injected and on inputs in $\mathcal{F}_1$.

Please note, that CINI restricts the number of probes and faults together to be smaller than or equal to the order of probing security d. Hence, the order of fault security is always dependent on the order of probing security. In Section 5 we show how to achieve independence between fault and probing security.

4.1 CINI Security and Composition

Intuitively, the security of CINI comes from the fact, that there are always more share domains than the amount of probes and faults an adversary is allowed to place or inject. This is sufficient for security, as the isolation of probe and fault propagation within share domains and SRDs, respectively, restricts the possible leakage.

Theorem 4.3. *A (d, k)-CINI gadget is (d, k)-combined secure.*

Proof. Let G be a (d, k)-CINI gadget, with notation as in Definition 4.2. Further, let G^{D} be a gadget realizing a decoding function D,

such that, given an input with at most k faults and an abort signal, G^D either aborts or outputs a correct result.

Correctness: By definition of CINI, G either aborts or outputs a result where all values are correct, except for those belonging to the SRDs $\mathcal{F}_1 \cup \mathcal{F}_2$. Further, by definition it holds that $|\mathcal{F}_1 \cup \mathcal{F}_2| \leq k_1 + k_2 \leq k$. Hence, the concatenation $G^D(G(\cdot))$ either aborts or outputs a correct result.

Privacy: By definition of CINI the abort signal, the outputs of the share domains in $\mathcal{S}_2$, and the probes can be simulated with the inputs in the share domains in $\mathcal{S}_1 \cup \mathcal{S}_2$ and knowledge of the faults injected and on inputs in $\mathcal{F}_1$. Knowledge of the input faults does not reveal any additional information about the corresponding non-faulted value, as only distributions are leaked (a fault with biased distribution erases the original value and non-biased faults leak nothing on its own). It holds that $|\mathcal{S}_1 \cup \mathcal{S}_2| \leq d_1 + d_2 + k_2 \leq d < s$. Therefore, due to the independence of the input encoding of the shares, the inputs in the share domains $\mathcal{S}_1 \cup \mathcal{S}_2$ are independent of any sensitive signals. □

We continue by showing that the arbitrary combination of CINI gadgets again construct a CINI gadget, as long as there is no loop in the design. This is trivially true for all combinational circuits. Sequential logic with potential loops can be reduced to this case by unrolling the design for the analysis, which is valid when transitional leakage [20] is not considered (as for this work).

Theorem 4.4. *The loop-free composition of two (d,k)-CINI gadgets is (d,k)-CINI.*

Proof. Let G_1 and G_2 be arbitrary (d,k)-CINI gadgets and G_3 an arbitrary composition of G_1 and G_2, such that there is no loop within G_3. Without loss of generality, we say no output of G_2 is connected to an input of G_1 (as there is no loop).

Let $\mathcal{F}_1$ be a set of k_1 SRDs with faulty inputs to G_3 and $\mathcal{S}_2$ a set of d_2 share domains with probed outputs of G_3. Further, let there be k_2 faults injected in gates of G_3, of which k_2^1 target gates in G_1 and k_2^2 target gates in G_2. In addition, assume d_1 probes placed on wires in G_3, of which d_1^1 target wires in G_1 and d_1^2 target wires in G_2. As the gadgets G_1 and G_2 are disjoint it holds that $k_2^1 + k_2^2 = k_2$ and $d_1^1 + d_1^2 = d_2$. We chose d_1, d_2, k_1, and k_2 such that $k_1 + k_2 \leq k$ and $d_1 + d_2 + k_1 + k_2 \leq d$. We first prove correctness and then privacy of G_3.

Correctness: As no output of G_2 is connected to an input of G_1, the set $\mathcal{F}_1$ contains all SRDs with faulty inputs to G_1 and there are k_2^1 faults injected in gates of G_1. It holds that $k_1 + k_2^1 \leq k_1 + k_2 \leq k$. Hence, with CINI of G_1 it follows that G_1 either aborts or there exists a set $\mathcal{F}_2^1$ of k_2^1 SRDs, such that at most the outputs belonging to the SRDs $\mathcal{F}_1 \cup \mathcal{F}_2^1$ are faulty.

Outputs of G_1 can be connected to inputs of G_2, thus, the set of SRDs with possible faulty inputs to G_2 is the set $\mathcal{F}_1 \cup \mathcal{F}_2^1$. Further, there are k_2^2 faults injected in gates of G_2. It holds that $k_1 + k_2^1 + k_2^2 = k_1 + k_2 \leq k$. Hence, with CINI of G_2 it follows that G_2 either aborts or there exists a set $\mathcal{F}_2^2$ of k_2^2 SRDs, such that at most the outputs belonging to the SRDs $\mathcal{F}_1 \cup \mathcal{F}_2^1 \cup \mathcal{F}_2^2$ are faulty.

The outputs of G_3 can be both outputs of G_1 and G_2 and, therefore, at most the SRDs in $\mathcal{F}_1 \cup \mathcal{F}_2^1 \cup \mathcal{F}_2^2$ carry faulty outputs or G_3 aborts (if G_1 or G_2 abort). It holds that $|\mathcal{F}_1 \cup \mathcal{F}_2^1 \cup \mathcal{F}_2^2| \leq k_1 + k_2^1 + k_2^2 = k_1 + k_2 \leq k$. In addition, as G_1 and G_2 are (d,k)-CINI there exists a decoding gadget G^D, such that given an input with at most k faulty SRDs and an abort signal, G^D either aborts or outputs a correct result. From this follows correctness of G_3.

Privacy: As all outputs of G_2 are outputs of G_3 the set of probed output share domains of G_2 is $\mathcal{S}_2$. In addition, there are d_1^2 probes placed on wires in G_2, $k_1 + k_2^1$ SRDs with faulty inputs, and k_2^2 faults injected to G_2. It holds that $d_1^2 + d_2 + k_1 + k_2^1 + k_2^2 \leq d_1 + d_2 + k_1 + k_2 \leq d$. Hence, with CINI of G_2 there exists a set $\mathcal{S}_1^2$ of $d_1^2 + k_2^2$ share domains, such that the abort signal of G_2, the outputs of G_2 belonging to $\mathcal{S}_2$, and the probes placed on wires in G_2 can be simulated with the inputs of G_2 belonging to the share domains $\mathcal{S}_1^2 \cup \mathcal{S}_2$ and knowledge of the faults. Denote this simulator with S_2. Please note, the knowledge of the faults at inputs to G_2 can be derived by fault propagation.

Outputs of G_1 can be connected to inputs of G_2 and a simulator for G_1 needs to simulate all inputs of G_2 required for S_2. Hence, the set of probed output share domains is $\mathcal{S}_1^2 \cup \mathcal{S}_2$. In addition, there are d_1^1 probes placed on wires within G_1, k_1 SRDs with faulty inputs, and k_2^1 faults injected to G_1. It holds that $d_1^1 + d_1^2 + k_2^2 + d_2 + k_1 + k_2^1 = d_1 + d_2 + k_1 + k_2 \leq d$. Hence, with CINI of G_1 there exists a set $\mathcal{S}_1^1$ of $d_1^1 + k_2^1$ share domains, such that the abort signal of G_1, the outputs of G_1 belonging to the share domains $\mathcal{S}_1^2 \cup \mathcal{S}_2$, and the probes placed on wires in G_1 can be simulated with inputs of G_1 in the share domains $\mathcal{S}_1^1 \cup \mathcal{S}_1^2 \cup \mathcal{S}_2$ and knowledge of the faults. We call this simulator S_1.

Together, S_1 and S_2 build a simulator such that the abort signal of G_3, the outputs of G_3 belonging to the share domains in $\mathcal{S}_2$, and the probes placed on wires in G_3 can be simulated with the inputs of G_3 belonging to the share domains in $\mathcal{S}_1^1 \cup \mathcal{S}_1^2 \cup \mathcal{S}_2$ and knowledge of the faults. It holds that $|\mathcal{S}_1^1 \cup \mathcal{S}_1^2| \leq d_1^1 + k_2^1 + d_1^2 + k_2^2 = d_1 + k_2$. □

4.2 CINI Gadgets

One explicit goal of CINI is the trivial implementation of linear functions, e.g., addition, by simply sharing and replicating them. This is possible, as replication and Boolean sharing are linear themselves.

Theorem 4.5. *An implementation with $d+1$ shares and $(k+1)$-times replication of a linear function is (d,k)-CINI in the glitch-robust probing model.*

Proof. Let G be the trivial gadget implementation of a linear function, i.e., the linear function is replicated both in the dimension of fault and share domains. Then it holds that each SRD is functionally separated.

Correctness: Due to the separation of SRDs, a fault in a SRD (i,ℓ) can only propagate to an output in the SRD (i,ℓ). Hence, $\mathcal{F}_2$ is the union of all SRDs a fault is injected in and it always holds that $|\mathcal{F}_2| \leq k_2$. As input faults can also only propagate within the same SRD, all outputs are correct except for those belonging to the SRDs $\mathcal{F}_1 \cup \mathcal{F}_2$. In addition, as there are $k+1$ replications up to k faults can be detected by a equivalence check and, hence, a decoding gadget exists.

Privacy: Due to the separation of SRDs, each probe and output is only dependent on inputs of one SRD. As a SRD is a part of a share domain, the same is true for share domains. Hence, the outputs belonging to $\mathcal{S}_2$ can be simulated with the inputs belonging to $\mathcal{S}_2$.

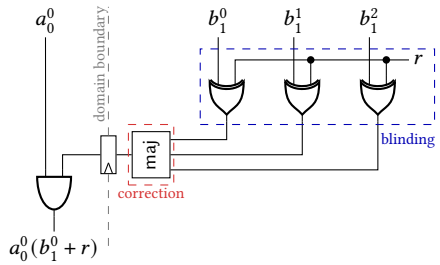

Figure 4. CINI requires all values crossing a domain boundary to be both correct and blinded.

Similarly, the probes can be simulated with the inputs belonging to the probed share domain. Hence, the simulator requires the inputs in $\mathcal{S}_1 \cup \mathcal{S}_2$ and knowledge of the faults, where $\mathcal{S}_1$ is the set of probed share domains (it trivially holds that $|\mathcal{S}_1| \leq d_1 \leq d_1 + k_2$). □

Implementing non-linear functions, e.g., multiplication, is more complex since values need to be combined across share-domain boundaries. Hence, similar to PINI, we have to ensure that the masks of such terms are refreshed beforehand. In addition, a faulty value is not allowed to cross SRD boundaries, which requires fault detection or correction for terms combining different share domains. Both principles are illustrated in Figure 4. As no fault is allowed to affect more than one SRD, each intermediate value is only allowed to be used in one SRD (except for generated randomness), requiring detection/correction within gadgets for values that are used across SRD boundaries.

Based on those design principles, we can extend the refresh-then-multiply technique, used in the HPC_1 gadget [9] (originally PINI), to a refresh-and-correct-then-multiply technique to construct a CINI gadget (given in Algorithm 2). The major difference to the original HPC_1 gadget is the replication of the logic and the majority function for correction in Line 12. Also, and in contrast to HPC_1, all intermediate values are only allowed to be used once (see Remark 3). In the following we prove CINI for the derived gadget.

Algorithm 2: $\mathsf{HPC}_1^{\mathsf{C}}$: CINI multiplication
(with difference to HPC_1 highlighted).

1 **function** $\mathsf{HPC}_1^{\mathsf{C}}(a_0^0, \ldots, a_d^n, b_0^0, \ldots, b_d^n)$:
  Require: $n = 2k + 1$
  Require: $a_i^\ell = a_i^{\ell'}$ and $b_i^\ell = b_i^{\ell'}$ for $0 \leq \ell, \ell' \leq n, 0 \leq i \leq d$
  Require: $\sum_{j=0}^{d} a_j^\ell = a$ and $\sum_{j=0}^{d} b_j^\ell = b$ for $0 \leq \ell \leq n$
  // Initialize randomness
2 **for** $i = 0$ *to* d **do**
3 **for** $j = i + 1$ *to* d **do**
4 $\tilde{r}_{i,j} \xleftarrow{\$} \mathbb{F}_2; \tilde{r}_{j,i} \leftarrow \tilde{r}_{i,j}$
5 $r_{i,j} \xleftarrow{\$} \mathbb{F}_2; r_{j,i} \leftarrow r_{i,j}$
  // Refreshing
6 **for** $\ell = 0$ *to* $n - 1$ **do**
7 **for** $j = 0$ *to* d **do**
8 $\tilde{v}_j^\ell \leftarrow b_j^\ell + \sum_{i=0, i \neq j}^{d} \tilde{r}_{i,j}$
  // Correction
9 **for** $\ell = 0$ *to* $n - 1$ **do**
10 **for** $i = 0$ *to* d **do**
11 **for** $j = 0$ *to* d **do**
12 $v_{i,j}^\ell \leftarrow \mathsf{maj}(\tilde{v}_i^0 \ldots \tilde{v}_i^{n-1})$
  // Multiplication
13 **for** $\ell = 0$ *to* $n - 1$ **do**
14 **for** $i = 0$ *to* d **do**
15 $w_i^\ell \leftarrow a_i^\ell \cdot \mathsf{Reg}[v_{i,i}^\ell]$
16 **for** $j = 0$ *to* d, $j \neq i$ **do**
17 $z_{i,j}^\ell \leftarrow a_i^\ell \cdot \mathsf{Reg}[v_{j,i}^\ell] + r_{i,j}$
18 $c_i^\ell \leftarrow \mathsf{Reg}[w_i^\ell] + \sum_{j=0; j \neq i}^{d} \mathsf{Reg}[z_{i,j}^\ell]$
  Ensures: $c_i^\ell = c_i^{\ell'}$ for $0 \leq \ell, \ell' \leq n, 0 \leq i \leq d$
  Ensures: $\sum_{i=0}^{d} c_i^\ell = a \cdot b$ for $0 \leq \ell \leq n$
19 **return** $c_0^0, \ldots, c_d^n$

Theorem 4.6. *The gadget* $\mathsf{HPC}_1^{\mathsf{C}}$ *as defined in Algorithm 2 with a register-free majority function is* (d, k)*-CINI in the glitch-robust probing model.*

Proof. Let $\mathcal{F}_1$ be a set of k_1 SRDs and $\mathcal{S}_2$ a set of d_2 share domains. Further, let there be k_2 faults injected to the gates of the gadget and d_1 probes placed on internal wires. Without loss of generality, we restrict the probes to only capture $v_{i,j}^\ell$, w_i^ℓ, $z_{i,j}^\ell$, and c_i^ℓ as other glitch-extended probes are less powerful. In particular, all probes within the majority function (Line 12) are less powerful than probes on $v_{i,j}^\ell$ as the majority function is implemented register free. We first show correctness and then privacy of the gadget.

Correctness: For correctness, faults in a random value $r_{i,j}$ can be ignored, as otherwise the correctness without any fault would depend on the concrete value of $r_{i,j}$ and the gadget would output a wrong result half of the time. By construction of the gadget each value with three indices i, j, and ℓ only influences the output c_i^ℓ and, hence, faults injected to those values can only influence the SRD (i, ℓ). This is true for $v_{i,j}^\ell$, $z_{i,j}^\ell$, and each intermediate value in $\mathsf{maj}(\tilde{v}_i^0 \ldots \tilde{v}_i^{n-1})$ (regardless of the implementation of the majority as it is true for $v_{i,j}^\ell$). Similar, the values a_i^ℓ, w_i^ℓ, and c_i^ℓ are only used for the computation of c_i^ℓ and can only influence the SRD (i, ℓ). The only values influencing more than one SRD are b_j^ℓ and $\tilde{v}_j^\ell$. However at most k of those values can be faulted and, hence, are corrected by the majority function. Therefore, the only way how a fault in b_j^ℓ or $\tilde{v}_j^\ell$ can have an impact on a SRD (i, ℓ), with $i \neq j$, is by manipulating the computation of $v_{i,j}^\ell$ in which case there is an additional fault for the SRD (i, ℓ).

In conclusion, each fault in a SRD at the input can cause a fault at the same SRD at the output. In addition, the set $\mathcal{F}_2$ is the union of all SRDs (i, ℓ) such that the computation of an intermediate value containing the indices i and ℓ was faulted. Then it holds that $|\mathcal{F}_2| \leq k_2$ as each faulty intermediate value can influence at most one SRD. As there are $2k+1$ repetitions of each output, the decoding gadget G^{D} can be constructed as a majority function.

Privacy: Algorithm 3, given a set of probes $\mathcal{P}$, a set of share domains $\mathcal{S}_2$, and a set of faults $\mathcal{F}$ returns a set of share domains $\mathcal{X}$ required for the simulation of the probes, the outputs belonging to $\mathcal{S}_2$, and the outputs that violate the independence property of

Algorithm 3: Share-Domain Chooser for the Simulator of $\mathsf{HPC}_1^{\mathsf{C}}$.

```
1  function DomainChooser^{HPC1^C}(P, S2, F2):
2      X ← ∅
3      for ℓ = 0 to n − 1 do
4          for i = 0 to d do
5              if w_i^ℓ or c_i^ℓ is probed or i ∈ S2 then
6                  X ← X ∪ {i}
7              for j = 0 to d, j ≠ i do
8                  if z_{i,j}^ℓ is probed then
9                      X ← X ∪ {i}
10                 if v_{i,j} is probed ∧ (i ∈ X or j ∈ X) then
11                     X ← X ∪ {i, j}
12                 else if v_{i,j}^ℓ is probed then
13                     X ← X ∪ {i}
14                 if r̃_{i,j} is faulted ∧ (i ∈ X or j ∈ X) then
15                     X ← X ∪ {i, j}
16             if ∀j : r_{i,j} is faulted then
17                 X ← X ∪ {i}
18     return X
```

Boolean sharing. We set $\mathcal{S}_1 \leftarrow \mathcal{X} \setminus \mathcal{S}_2$. Then all required intermediate values and outputs (with manipulation according to faults) can be computed exactly as in Algorithm 2 using the inputs belonging to the share domains in $\mathcal{X}$, except for some $\tilde{v}_j^\ell$.

First, we see that Algorithm 3 adds at most one share domain per probe, share domain in $\mathcal{S}_2$, and internal fault. Therefore, it always holds that $|\mathcal{S}_1| \leq d_1 + k_2$. We continue, by showing that the share domains in $\mathcal{X}$ and knowledge of the faults are sufficient to simulate the probes in $\mathcal{P}$, the outputs in $\mathcal{S}_2$, and the outputs that violate the independence property of Boolean sharing. (Please note, that Algorithm 2 has no abort signal).

We define a simulator that computes all required values exactly as defined in Algorithm 2 (required randomness is generated), except for $\tilde{v}_j^\ell$. For this value, we distinguish between the following cases: (i) if $j \in \mathcal{X}$ compute $\tilde{v}_j^\ell$ according to Algorithm 2. (ii) if $j \notin \mathcal{X}$ then draw a fresh random value r and set $\forall \ell : \tilde{v}_j^\ell \leftarrow r$. Afterwards, all values are manipulated according to the given faults.

Please note, if some $\tilde{v}_j^\ell$ is replaced by randomness then there is no fault injected in $\tilde{r}_{i,j}$, since then $j \notin \mathcal{X}$ and there is a probe dependent on $\tilde{v}_j^\ell$. Faults in $b_j^{\ell'}$ or $\tilde{v}_j^{\ell'}$, for $\forall \ell'$, have no impact on $v_{i,j}^\ell$, since there can be at most k faults and there are at least $k+1$ correct values. All other faults are either not observable by the probes or the faulted values are used exactly as in Algorithm 2 and the simulator can do the same manipulation as the fault in the gadget.

This simulator results in the same output distribution as the probes for the following reason: All values are computed exactly as in the gadget (Algorithm 2) except for $\tilde{v}_j^\ell$ in Case (ii). In this case, we argue that $\tilde{r}_{i,j}$ is only observable through one intermediate value and, hence, the simulation is correct. This is true for the following reason: Assume some $\tilde{v}_j^\ell$ is replaced by some randomness. This means $j \notin \mathcal{X}$ and some value dependent on $\tilde{v}_j^\ell$ is probed (i.e., $\tilde{v}_j^\ell$, $v_{i,j}^\ell$, w_i^ℓ, $z_{i,j}^\ell$, or c_i^ℓ). From $j \notin \mathcal{X}$ follows that there is no probe placed on any $\tilde{v}_j^{\ell'}$, $v_{j,i}^{\ell'}$, $w_j^{\ell'}$, $z_{j,i}^{\ell'}$, or $c_j^{\ell'}$, for $\forall i, \ell'$ and, hence, $\tilde{r}_{i,j}$ is only observable through $\tilde{v}_j^\ell$ and the simulation is correct.

Note that the output of Algorithm 2 is always a valid Boolean sharing due to the mask refreshing in Line 17 except when all random values $r_{i,j}, \forall j$ are faulted. In this case $c_i^\ell, \forall \ell$ violate the independence property, however, can be simulated as $i \in \mathcal{X}$. With the given simulator follows privacy of Algorithm 2. □

Remark 2. *We emphasize that faults targeted at randomness count as* internal *faults, i.e., contribute to k_2.*

Remark 3. *For security it is essential, that the majority function (Line 12) is computed for each SRD individually.*

Remark 4. *For security it is also essential that the mask refreshing in Line 8 is done one by one, i.e., $\tilde{v}_j^\ell \leftarrow (((b_j^\ell + r_{j,0}) + \ldots) + r_{j,d})$.*

Remark 5. *The assumption that the majority function is implemented register free is done for simplicity and readability of the proof. In fact, when there are register in maj then probes on $v_{i,j}^\ell$ and potential probes within maj are strictly less powerful as a probe on $v_{i,j}^\ell$ without registers in maj.*

As mentioned, the security of HPC_2 [9] relies on the fact that cross-domain leakage is prevented by some $r_{i,j}$ that is observable in only one of $(a_i+1)\cdot r_{i,j}$ or $a_i \cdot (b_j + r_{i,j})$ and, hence, ensures a proper masking of b_j (if b_j is not an input to the simulator). However, this only holds because both terms get the same a_i as input and one of them is using the negated form. Now, when replicating the gadget there exist some terms $(a_i^0 + 1) \cdot r_{i,j}$ and $a_i^1 \cdot (b_j^1 + r_{i,j})$ such that a_i^ℓ comes from different replications. Hence, by faulting a_i^0 or a_i^1 it is possible to force both $(a_i^0 + 1)$ and a_i^1 to be true and, hence, $r_{i,j}$ is observable in both corresponding terms. However, in a combined attack setting this flaw cannot be exploited for $d \leq 3$ and $k \leq 3$, as the corresponding attack requires one fault (at a_i^0) and two probes (at c_i^0 and c_i^1). Hence, for the remaining cases, we describe an HPC_2-based CINI gadget in Algorithm 4 and prove the security in Section 7.1 via a tool-based analysis. As HPC_3 [28] also depends on the masked shares multiplication trick it suffers from the same limitations. Interestingly, $\mathsf{HPC}_2^{\mathsf{C}}$ achieves a tight realization of CINI, meaning that a well placed fault indeed reduces the order of probing security by one (cf. Section 7.3). In contrast, $\mathsf{HPC}_1^{\mathsf{C}}$ actually achieves a higher security level than strictly required for some orders, e.g., the $(2,2)$-$\mathsf{HPC}_1^{\mathsf{C}}$ gadget is also $(2,1)$-$\mathrm{CINI}_{\mathrm{ind}}$ and $(1,2)$-$\mathrm{CINI}_{\mathrm{ind}}$ (see Section 5). This indicates some overhead in terms of randomness for $\mathsf{HPC}_1^{\mathsf{C}}$.

While potentially more efficient, gadgets based on detection have to avoid SCA leakage caused by fault propagation and construct a SCA-secure tree of detection flags. Both are non-trivial problems that we leave for future work.

5 Independent Combined-Isolating Non-Interference

The presented CINI definition requires that the number of probes and faults together is smaller than or equal to the order of probing

Algorithm 4: $\mathsf{HPC}_2^{\mathsf{C}}$: CINI multiplication for $d \leq 2, k \leq 2$ (with difference to HPC_2 highlighted).

1 **function** $\mathsf{HPC}_2^{\mathsf{C}}(a_0^0, \ldots, a_d^n, b_0^0, \ldots, b_d^n)$:
  Require: $d \leq 2, k \leq 2, n = 2k+1$
  Require: $a_i^\ell = a_i^{\ell'}$ and $b_i^\ell = b_i^{\ell'}$ for $0 \leq \ell, \ell' \leq n, 0 \leq i \leq d$
  Require: $\sum_{j=0}^d a_j^\ell = a$ and $\sum_{j=0}^d b_j^\ell = b$ for $0 \leq \ell \leq n$
  `// Initialize randomness`
2 **for** $i = 0$ *to* d **do**
3 **for** $j = i+1$ *to* d **do**
4 $r_{i,j} \xleftarrow{\$} \mathbb{F}_2; r_{j,i} \leftarrow r_{i,j}$
  `// Masking`
5 **for** $\ell = 0$ *to* $n-1$ **do**
6 **for** $i = 0$ *to* d **do**
7 **for** $j = 0$ *to* $d, j \neq i$ **do**
8 $\tilde{v}_{i,j}^\ell \leftarrow b_j^\ell + r_{i,j}$
9 **for** $\ell = 0$ *to* $n-1$ **do**
  `// Correction and partial products`
10 **for** $i = 0$ *to* d **do**
11 $w_i^\ell \leftarrow a_i^\ell \cdot \mathsf{Reg}[b_i^\ell]$
12 **for** $j = 0$ *to* $d, j \neq i$ **do**
13 $u_{i,j}^\ell \leftarrow (a_i^\ell + 1) \cdot \mathsf{Reg}[r_{i,j}]$
14 $v_{i,j}^\ell \leftarrow \mathsf{maj}(\tilde{v}_{i,j}^0 \ldots \tilde{v}_{i,j}^{n-1})$
15 $z_{i,j}^\ell \leftarrow a_i^\ell \cdot \mathsf{Reg}[v_{i,j}^\ell]$
  `// Reduction`
16 **for** $i = 0$ *to* d **do**
17 $c_i^\ell \leftarrow \mathsf{Reg}[w_i^\ell] + \sum_{j=0, j\neq i}^d (\mathsf{Reg}[u_{i,j}^\ell] + \mathsf{Reg}[z_{i,j}^\ell])$
18 **Ensures:** $c_i^\ell = c_i^{\ell'}$ for $0 \leq \ell, \ell' \leq n, 0 \leq i \leq d$
  Ensures: $\sum_{j=0}^d c_j^\ell = a \cdot b$ for $0 \leq \ell \leq n$
19 **return** $c_0^0, \ldots, c_d^n$

security d. This means that it is of course possible to build a gadget that can resist d' probes together with k' faults for arbitrary d', and k', however, it requires an implementation with at least $d'+k'+1$ shares and is, hence, effectively a $(d'+k', k')$ gadget. This results in a significant overhead in the number of shares and in consequence also in other metrics.

To avoid this overhead, we define CINI_{ind}, a composability notion for CA security where the order of probing and fault security can be selected independently. This independence requires only a small change compared to the CINI definition, namely that we now have $d_1 + d_2 \leq d$ instead of $d_1 + d_2 + k_1 + k_2 \leq d$, separating the number of injected faults and the order of probing security.

Definition 5.1 (Independent Combined-Isolating Non-Interference). A gadget G is (d,k)-CINI_{ind} iff for any set $\mathcal{F}_1$ of k_1 faulty SRDs, every set of k_2 faults injected in gates of G, any set of d_1 probes placed on intermediate values, and any set $\mathcal{S}_2$ of d_2 share domains, such that $k_1 + k_2 \leq k$ and $d_1 + d_2 \leq d$, there exists a set $\mathcal{F}_2$ of at most k_2 SRDs and a set $\mathcal{S}_1$ of at most d_1 share domains such that the following holds:

Correctness: The gadget either aborts or gives an output where all values, except those belonging to the SRDs $\mathcal{F}_1 \cup \mathcal{F}_2$, are equal to the golden circuit, and there exists a decoding gadget G^{D}, such that given an input with at most k faulty SRDs, G^{D} outputs a correct result.

Privacy: The outputs of the share domains in $\mathcal{S}_2$ and the probes can be simulated with the inputs of the share domains in $\mathcal{S}_1 \cup \mathcal{S}_2$ and knowledge of the faults both injected and on inputs in $\mathcal{F}_1$.

Please note, that in contrast to CINI the parameters for probing and fault security are now clearly separated. This also requires the use of correction countermeasures, as detection violates this separation [15]. To see this, assume an intermediate value that gets randomized by some randomness, i.e., $x_i^\ell + r$, where a reset fault is injected on x_i^ℓ. The detection then removes the r in some other fault domain ℓ' and an appropriate probe leaks x_i (cf. Figure 5 in [36]).

5.1 CINI_{ind} Security and Composition

The security and composition of CINI_{ind} relies on the same fundamental properties as for CINI, i.e., more domains than allowed attack points and isolation of probe and fault propagation within the respective domains.

Theorem 5.2. *A (d,k)-CINI_{ind} gadget is (d,k)-combined secure.*

Theorem 5.3. *The loop-free composition of two (d,k)-CINI_{ind} gadgets is (d,k)-CINI_{ind}.*

The proofs of both theorems follow closely the argumentation for their CINI counterparts (Theorem 4.3 and Theorem 4.4), where for *privacy* aspects related to injected faults are removed.

5.2 CINI_{ind} Gadgets

One can observe that the provided CINI gadget for linear functions already adheres to the more restrictive CINI_{ind} property. The reason is the natural isolation of each SRD when implementing the function for each share and replication individually.

Theorem 5.4. *An implementation with $d+1$ shares and $(2k+1)$-times replication of a linear function is (d,k)-CINI_{ind} in the glitch-robust probing model.*

Again, the proof follows closely the argumentation from Theorem 4.5.

As with PINI and CINI, implementing a non-linear gadget is more complex and requires careful separation of probe and fault propagation. Of course, the same design principles as for CINI apply also for CINI_{ind}, namely (i) the masking of values crossing share-domain boundaries need to be refreshed, and (ii) values crossing SRD boundaries need to be corrected. In addition CINI_{ind} requires (iii) re-masking is done with more than k random values. This ensures that an attacker cannot remove the randomness from a refreshed value.

Interestingly, increasing the amount of randomness is sufficient to make the CINI multiplication gadget CINI_{ind} in the case of $\mathsf{HPC}_1^{\mathsf{C}}$.

Theorem 5.5. *The gadget $\mathsf{HPC}_1^{\mathsf{I}}$ as defined in Algorithm 5 with a register-free majority function is (d,k)-CINI_{ind} in the glitch-robust probing model.*

Remark 3, Remark 4, and Remark 5 also hold for the CINI_{ind} gadget in Algorithm 5 with the majority function in Line 13 and the

Algorithm 5: $\mathsf{HPC}_1^{\mathsf{I}}$: $\mathrm{CINI}_{\mathrm{ind}}$ multiplication.

1 **function** $\mathsf{HPC}_1^{\mathsf{I}}(a_0^0, \ldots, a_d^n, b_0^0, \ldots, b_d^n)$:
  Require: $n = 2k + 1$
  Require: $a_i^\ell = a_i^{\ell'}$ and $b_i^\ell = b_i^{\ell'}$ for $0 \le \ell, \ell' \le n, 0 \le i \le d$
  Require: $\sum_{j=0}^{d} a_j^\ell = a$ and $\sum_{j=0}^{d} b_j^\ell = b$ for $0 \le \ell \le n$
  // Initialize randomness
2 **for** $i = 0$ *to* d **do**
3 **for** $j = i + 1$ *to* d **do**
4 **for** $m = 0$ *to* $k - 1$ **do**
5 $\tilde{r}_{i,j,m} \xleftarrow{\$} \mathbb{F}_2; \tilde{r}_{j,i,m} \leftarrow \tilde{r}_{i,j,m}$
6 $r_{i,j,m} \xleftarrow{\$} \mathbb{F}_2; r_{j,i,m} \leftarrow r_{i,j,m}$
  // Refreshing
7 **for** $\ell = 0$ *to* $n - 1$ **do**
8 **for** $j = 0$ *to* d **do**
9 $\tilde{v}_j^\ell \leftarrow b_j^\ell + \sum_{i=0, i\neq j}^{d} \sum_{m=0}^{k-1} \tilde{r}_{i,j,m}$
  // Correction
10 **for** $\ell = 0$ *to* $n - 1$ **do**
11 **for** $i = 0$ *to* d **do**
12 **for** $j = 0$ *to* d **do**
13 $v_{i,j}^\ell \leftarrow \mathrm{maj}(\tilde{v}_i^0 \ldots \tilde{v}_i^{n-1})$
  // Multiplication
14 **for** $\ell = 0$ *to* $n - 1$ **do**
15 **for** $i = 0$ *to* d **do**
16 $w_i^\ell \leftarrow a_i^\ell \cdot \mathrm{Reg}[v_{i,i}^\ell]$
17 **for** $j = 0$ *to* $d, j \neq i$ **do**
18 $z_{i,j}^\ell \leftarrow a_i^\ell \cdot \mathrm{Reg}[v_{j,i}^\ell] + \sum_{m=0}^{k-1} r_{i,j,m}$
19 $c_i^\ell \leftarrow \mathrm{Reg}[w_i^\ell] + \sum_{j=0; j\neq i}^{d} \mathrm{Reg}[z_{i,j}^\ell]$
  Ensures: $c_i^\ell = c_i^{\ell'}$ for $0 \le \ell, \ell' \le n, 0 \le i \le d$
  Ensures: $\sum_{i=0}^{d} c_i^\ell = a \cdot b$ for $0 \le \ell \le n$
20 **return** $c_0^0, \ldots, c_d^n$

masking in Line 9 and 19. Please note, the gadget $\mathsf{HPC}_2^{\mathsf{C}}$ can not be easily transferred to $\mathrm{CINI}_{\mathrm{ind}}$ since the contradiction mentioned in Section 4.2 then also occurs for smaller orders ($d \ge 2$ and $k \ge 1$).

6 TOOL SUPPORT

In the last years, many tools supporting the verification and construction of circuits were presented in the literature. We extend the recently published tool VERICA[4] [36] with our novel composability notions. Additionally, we added support for our gadgets to SAIREDA[5] [22] to construct secure cryptographic primitives from insecure implementations.

Integration into a Verification Tool. VERICA is a verification tool for combined countermeasures which has been coalesced from the SCA verification tool SILVER [29] and the FIA verification tool FIVER [38]. Hence, the tool analyzes a given gate-level netlist with respect to the protection against physical attacks. More precisely, VERICA can verify stand-alone protection against SCA and FIA but also combined protection considering reciprocal effects between fault-injections and side-channel analysis. These reciprocal effects are analyzed by applying a *active-then-passive* verification approach meaning that the tool injects a valid fault combination followed by a passive side-channel analysis. In this context, VERICA is limited to analyze combined composability only for the C-NI, C-SNI, and C-$\mathrm{SNI}_{\mathrm{ind}}$ security notions [36]. In order to extend VERICA to support our novel security notion, we first extend the fault verification strategies (i.e., error detection and error correction) with a feature to support verification of the FINI notion. Hence, the inputs and outputs require an additional annotation to identify the different redundancy domains. This information is used to track the redundancy domains of erroneous outputs and faulted inputs of a design under test. Based on Definition 3.2, we remove all input redundancy domains from the set of output redundancy domains and check if the cardinality of the resulting set exceeds the number of internal faults (if not, the design is FINI). So far, VERICA only supported two fault composability notions relying on ideas of non-accumulation (see [15, 36] for more details).

Similarly, we extend VERICA to support the verification of the CINI notion introduced in Section 4. In order to check the correctness, we implemented a similar strategy as for FINI but instead of relying on redundancy domains, we use SRDs as defined in Definition 4.1. Note that all faults on randomness are treated as internal faults, as discussed before. To verify the privacy, we modify the integrated PINI strategy of VERICA such that the number of allowed probes depends on the number of injected faults (cf. Definition 4.2).

For the integration of the $\mathrm{CINI}_{\mathrm{ind}}$ notion, we independently inject faults and subsequently check the the composability in the PINI model. The composability with respect to fault injections is accomplished by applying the same checks as for CINI (i.e., determining the number of faults in different SRDs).

Gadget Insertion. To automate the process of integrating the proposed gadgets into a hardware netlist, we added support for the gadgets to SAIREDA, which is a circuit compiler based on the Multi-Level Intermediate Representation (MLIR) framework [32]. As input serves an unprotected FIRRTL [27] design restricted to netlist operations (which can be derived from Chisel or Verilog) while the tool produces a protected Verilog description. Here, we support the composability notions of FINI, PINI, CINI, and $\mathrm{CINI}_{\mathrm{ind}}$.

In a preparation phase, the design is first transferred into an XOR-AND Graph (XAG). Afterwards, the tool operates in two steps: (i) The tool identifies all non-linear and gates and replaces them with abstract gadget operations. In addition, it assigns the required random elements to the gadgets. This is done for all gadget types except for FINI, which does not require additional randomness and only results in a simple replication. (ii) Then, the tool replaces all operations with their secure implementation, i.e., replicating the logic and replacing the abstract gadget operations with their actual logic. All majority functions are realized via the median of a sorting network. Further, the tool adds additional input/output ports for shares, replication, and randomness to the design.

7 EVALUATION

We continue with a practical evaluation of the proposed designs. First, we provide implementation and verification results for the individual gadgets. Afterwards, we use these gadgets to construct

[4] Available at https://github.com/Chair-for-Security-Engineering/VERICA
[5] Available at https://github.com/Chair-for-Security-Engineering/SAIREDA

Table 2. Number of elements (without implementation of maj).

	$\mathsf{HPC}_1^{\mathsf{C}}$	$\mathsf{HPC}_1^{\mathsf{I}}$
and	$(2k+1)(d+1)^2$	$(2k+1)(d+1)^2$
xor	$3(2k+1)d(d+1)$	$(2k+1)^2d(d+1)$
reg	$(2k+1)(d+1)^2$	$(2k+1)(d+1)^2$
maj	$(2k+1)(d+1)^2$ *[input size: $2k+1$]*	$(2k+1)(d+1)^2$ *[input size: $2k+1$]*
rand	$d(d+1)$	$d(d+1)k$

cryptographic primitives (i.e., S-boxes) and analyze their security and implementation costs. Eventually, we instantiate a protected PRESENT S-box on an Field-Programmable Gate Array (FPGA) and confirm our theoretical approaches by practical measurements.

7.1 Gadget Verification

We start our evaluation by describing the different implementations and highlighting important details that need to be considered when implementing combined gadgets on hardware.

7.1.1 Implementation. We construct FINI gadgets according to Section 3, i.e., the target design is instantiated $k+1$ times for detection-based and $2k+1$ times for correction-based implementations. In addition, detection and correction gadgets follow the *independence property* with respect to redundancy domains by replicating the respective logic as well (cf. Algorithm 1).

For CINI and $\mathrm{CINI}_{\mathrm{ind}}$ linear functions are constructed by instantiating the function $(d+1)(2k+1)$ times, i.e., for each SRD one instance, while multiplications are implemented according to Algorithm 2, Algorithm 4, and Algorithm 5. Please note, that an implementation has to adhere to Remark 3 and Remark 4. In Table 2 we provide the number of logic elements, dependent on the order of fault and probing security, required to instantiate the multiplication gadgets $\mathsf{HPC}_1^{\mathsf{C}}$ and $\mathsf{HPC}_1^{\mathsf{I}}$. Those numbers are without considering the implementation of the majority function, as there is no closed formula for the related implementation cost. Please note that the gadgets have the same implementation cost except for required randomness and the number of xor gates.

7.1.2 Results. The implementations and verification results are shown in Table 3. We instantiated and analyzed a FINI and gadget equipped with a detection and a correction gadget for $k \in \{1,2,3,4\}$, respectively. All designs fulfill the FINI security notion while the evaluation of the correction gadget for $k=4$ requires a notably verification time of 6.24 h.

Next, we analyze the introduced CINI gadget from Algorithm 2 for $d \in \{1,2,3\}$ and $k \in \{1,2,3\}$ and report the required resources. We could verify the security for all gadgets except for the $(3,3)$-CINI gadget, as VERICA was not able to finish the analysis in a reasonable time. As already discussed in Section 4.2, we can construct for some parameters d and k a more tight CINI gadget applying Algorithm 4. Table 3 verifies the security for $d \in \{1,2\}$ and $k \in \{1,2\}$ while it reveals security flaws for a $(3,1)$ configuration. We confirm in Section 7.3 by a practical evaluation that well placed faults can lead to a reduced side-channel security order.

Eventually, we instantiated and verified the $\mathrm{CINI}_{\mathrm{ind}}$ gadgets for $d \in \{1,2,3\}$ and $k \in \{1,2,3\}$ (cf. Algorithm 5). The verification of these gadgets is more challenging since the number of faults does not reduce the number of probes. Therefore, VERICA is not able to verify several designs in a reasonable time (marked by ∞).

Table 3. Implementation and verification results for FINI, CINI, and $\mathrm{CINI}_{\mathrm{ind}}$ gadgets synthesized with the 45 nm Open Cell Library.

Gadget	Design						Verification		
	d	k	rand.	comb.	reg.	area [GE]	Def.	(d,k)	Time
Detect	–	1	0	3	0	4.7	FINI	$(0,1)^{\checkmark}$	0.387 s
	–	2	0	6	0	9		$(0,2)^{\checkmark}$	0.397 s
	–	3	0	13	0	15		$(0,3)^{\checkmark}$	0.429 s
	–	4	0	18	0	19.7		$(0,4)^{\checkmark}$	1.280 s
Correct	–	1	0	15	0	17	FINI	$(0,1)^{\checkmark}$	0.383 s
	–	2	0	75	0	98.3		$(0,2)^{\checkmark}$	0.445 s
	–	3	0	147	0	194.3		$(0,3)^{\checkmark}$	16.501 s
	–	4	0	297	0	390		$(0,4)^{\checkmark}$	6.24 h
$\mathsf{HPC}_1^{\mathsf{C}}$	1	1	2	78	24	238	CINI	$(1,1)^{\checkmark}$	0.409 s
	2	1	6	189	54	567		$(2,1)^{\checkmark}$	0.485 s
	3	1	12	356	96	1 032		$(3,1)^{\checkmark}$	39.544 s
	1	2	2	340	40	685		$(1,2)^{\checkmark}$	1.490 s
	2	2	6	795	90	1 595		$(2,2)^{\checkmark}$	6.321 s
	3	2	12	1420	160	2 860		$(3,2)^{\checkmark}$	4.662 min
	1	3	2	590	56	1 087		$(1,3)^{\checkmark}$	16.817 min
	2	3	6	1362	126	2 502		$(2,3)^{\checkmark}$	3.897 h
	3	3	12	2456	224	4 509		*	∞
$\mathsf{HPC}_2^{\mathsf{C}}$	1	1	1	66	36	294	CINI	$(1,1)^{\checkmark}$	0.389 s
	2	1	3	189	90	768		$(2,1)^{\checkmark}$	0.775 s
	1	2	1	210	60	640		$(1,2)^{\checkmark}$	0.804 s
	2	2	3	615	150	1 730		$(2,2)^{\checkmark}$	5.643 s
	3	1	6	372	168	1 460		$(3,1)^{\times}/(2,1)^{\checkmark}$	18.386 h
$\mathsf{HPC}_1^{\mathsf{I}}$	1	1	2	78	24	240	$\mathrm{CINI}_{\mathrm{ind}}$	$(1,1)^{\checkmark}$	0.397 s
	2	1	6	189	54	573		$(2,1)^{\checkmark}$	4.329 s
	3	1	12	356	96	1 044		*	∞
	1	2	4	360	40	728		$(1,2)^{\checkmark}$	7.153 s
	2	2	12	855	90	1 725		*	∞
	3	2	24	1540	160	3 120		*	∞
	1	3	6	646	56	1 203		$(1,3)^{\checkmark}$	4.743 h
	2	3	18	1530	126	2 852		*	∞
	3	3	36	2792	224	5 209		*	∞

* Due to the extensive amount of combinations, these gadgets could not be verified with VERICA.

7.2 Evaluation of Cryptographic Primitives

To assess the implementation impact of the proposed gadgets, we evaluate the efficiency in terms of area, required fresh randomness, and latency for three different cryptographic S-boxes. In particular, we chose S-box implementations from PRESENT [9], Keccak [6], and AES [7] with a minimal number of and gates. The results are given in Table 4, where we also report the respective numbers for HPC_1 and HPC_2 (PINI) for comparison. The probing secure implementations are not pipelined, i.e., we only add registers where required for probing security, and the area of the Random Number Generator (RNG) is not included.

Area. We report area in terms of Gate Equivalent (GE). As expected, FINI increases the area of each S-box by the replication factor. Even when sharing can be seen as some sort of replication, PINI has a significantly higher area demand than FINI, as PINI requires additional logic to securely connect different share domains and, more importantly, requires additional registers to prevent leakage from glitches. When considering combined-secure implementations, the impact for CINI is larger than the replication of the respective PINI gadget due to the additional correction done in each multiplication gadget. When comparing $\mathsf{HPC}_1^{\mathsf{C}}$ and $\mathsf{HPC}_2^{\mathsf{C}}$ we observe the same relation as between HPC_1 and HPC_2, namely

Table 4. Complexity of cryptographic primitives.

Gadget	Order			PRESENT [9]			Keccak [6]			AES [7]		
	d	k	prop.	area [GE]	rand.	lat.	area [GE]	rand.	lat.	area [GE]	rand.	lat.
	-	-		35	0	0	18	0	0	275	0	0
Replicates	-	1	FINI	112	0	0	53	0	0	823	0	0
	-	2		178	0	0	88	0	0	1382	0	0
	-	3		255	0	0	124	0	0	1901	0	0
HPC_1	1	-	PINI	266	8	3	277	10	2	2181	68	6
	2	-		551	24	3	603	30	2	4560	204	6
	3	-		940	48	3	1057	60	2	7803	408	6
HPC_2	1	-	PINI	390	4	3	432	5	2	3235	34	6
	2	-		919	12	3	1063	15	2	7688	102	6
	3	-		1673	24	3	1973	30	2	14048	204	6
$\mathsf{HPC}_1^{\mathsf{C}}$	1	1	CINI	1246	8	3	1270	10	2	9296	68	6
	2	1		2716	24	3	2965	30	2	21182	204	6
	2	2		6887	24	3	7892	30	2	55363	204	6
	3	1		4727	48	3	5337	60	2	37669	408	6
	3	2		12073	48	3	14137	60	2	98429	408	6
	3	3		19343	48	3	22843	60	2	158555	408	6
$\mathsf{HPC}_2^{\mathsf{C}}$	1	1	CINI	1462	4	3	1540	5	2	11132	34	6
	2	1		3496	12	3	3940	15	2	27857	102	6
	2	2		7435	12	3	8577	15	2	60204	102	6
$\mathsf{HPC}_1^{\mathsf{I}}$	1	1	$\mathsf{CINI}_{\mathsf{ind}}$	1246	8	3	1270	10	2	9296	68	6
	2	1		2716	24	3	2965	30	2	21182	204	6
	2	2		7367	36	3	8492	45	2	59443	306	6
	3	1		4727	48	3	5337	60	2	37669	408	6
	3	2		13033	72	3	15337	90	2	106589	612	6
	3	3		22031	96	3	26203	120	2	181403	816	6

that the HPC_2-based gadgets are always larger. As HPC_2 and $\mathsf{HPC}_2^{\mathsf{C}}$ capture the respective compositional notions more tightly, more registers are required to stop leakage from glitches, which has a high impact on the area. In general, $\mathsf{HPC}_1^{\mathsf{I}}$ is larger than $\mathsf{HPC}_1^{\mathsf{C}}$, except when $k = 1$, due to the additional logic required for the additional randomness. However, when comparing the actual security guarantees against combined attacks, we should compare (d, k)-$\mathsf{CINI}_{\mathsf{ind}}$ to $(d + k, k)$-CINI, in which case $\mathsf{HPC}_1^{\mathsf{I}}$ is more efficient.

Randomness. We report the required fresh randomness in bits. Additional randomness is only required to achieve probing security and, hence, no randomness is required for FINI. For CINI the number of randomness is only dependent on the order of probing security d and equal to the respective PINI implementation. For $\mathsf{CINI}_{\mathsf{ind}}$ the randomness also increases with the order of fault security k.

Latency. We report latency in terms of clock cycles, however, note that the output is not directly connected to a register (causing additional delay due to signal propagation). In general, additional registers are only required for probing security to stop leakage caused by glitches. Here $\mathsf{HPC}_1^{\mathsf{C}}$, $\mathsf{HPC}_2^{\mathsf{C}}$, and $\mathsf{HPC}_1^{\mathsf{I}}$ inherit the latency properties from the respective PINI gadgets.

7.3 Practical Evaluation

Besides verifying our designs with VERICA, we additionally performed practical measurements. More precisely, we synthesized a pipelined (2, 2)-CINI PRESENT S-box for the Sakura-G side-channel evaluation board which is equipped with a Xilinx Spartan 6 FPGA. As discussed above, we focus our practical evaluation on $\mathsf{HPC}_2^{\mathsf{C}}$ gadgets since they tightly fulfill Theorem 4.2. The design requires twelve bits of fresh randomness which is provided by a KECCAK core instantiated as Pseudorandom Number Generator (PRNG). The FPGA was supplied with a 4 MHz clock and the current was measured indirectly via the voltage drop over a shunt in the supply path. To acquire suitable power traces, we used a ZFL-2000GH+ Low Noise Amplifier (LNA) configured with a 25.5 dB gain and a Spectrum M4 oscilloscope (8 bit resolution) with a sample rate of 2.5 GS/s.

For the first experiment, we instantiated a fault-free design on the FPGA and evaluated the first three statistical moments using a univariate Welch's t-test as described by Schneider and Moradi [40]. More precisely, the absolute value of the t-test is commonly compared to a threshold of 4.5. If all t-values do not exceed this threshold for all sample points, the design is assumed to be secure with a confidence higher than 0.9999 against attacks exploiting the statistical order of the corresponding t-test. Figure 5 depicts the corresponding results while Figure 5a shows a sample trace of the S-box evaluation. As expected, the t-test only indicates leakage in the third statistical moment.

For the second experiment, we injected a persistent stuck-at-zero fault into one of the randomness gates. The corresponding measurement are shown in Figure 6. As formally noted in Definition 4.2, and confirmed by our measurements shown in Figure 6c, the one-bit fault reduces the side-channel security by one order.

Eventually, for our last experiment, we injected two persistent faults in the randomness gates. We expect that the two injected faults reduce the side-channel security by two orders which is confirmed in Figure 7.

Table 5. Comparison with state-of-the-art for an AES S-box.

Method	Order			Security		Performance			Approach
	d	k	indep.	SIFA	formal	area [GE]	rand.	lat.	
CAPA [35]	1	1	-	-	✓	112 878	64	5	Robust MPC
M&M [13]	1	1	-	-	-	6 500	116	6	Detection + Infection
This Work	1	1	✓	✓	✓	9 296	68	6	$\mathsf{CINI}_{\mathsf{ind}}$
CAPA [35]	2	2	-	-	✓	200 965	1 248	5	Robust MPC
M&M [13]	2	2	-	-	-	13 500	348	6	Detection + Infection
This Work	2	2	-	✓	✓	55 363	204	6	CINI
This Work	2	2	✓	✓	✓	59 443	306	6	$\mathsf{CINI}_{\mathsf{ind}}$

indep.: SCA order is independent of injected faults, SIFA: Protection against SIFA, formal: Provable secure in a formal security model.

8 RELATED WORK

Early works that combine SCA and FIA countermeasures considered both attacks in isolation and, hence, do not protect against combined attacks [25, 37, 41]. The first proposal protecting against CA is CAPA [35] which divides a chip architecture into different *tiles* and uses robust MPC for secure computation, using a combination of masking with a Message Authentication Code (MAC) for fault detection. The formal security argument is based on the well-established security guarantees of MPC and the introduced *Tile-Probe-and-Fault model*. In this model, the adversary can probe

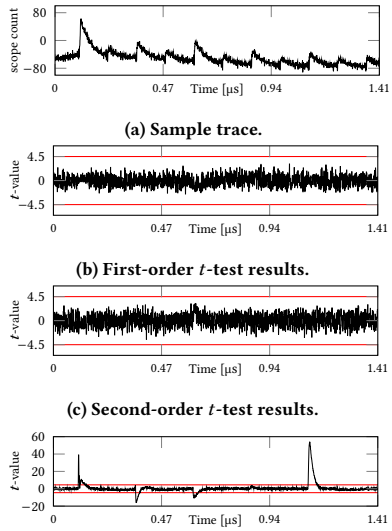

Figure 5. Measurement results of a fault free PRESENT S-box generated from $(2,2)$-CINI gadgets (100 Million traces).

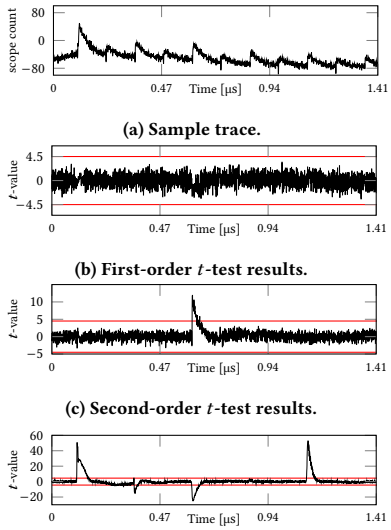

Figure 6. Measurement results of a PRESENT S-box generated from $(2,2)$-CINI gadgets with 1-bit fault injection (100 Million traces).

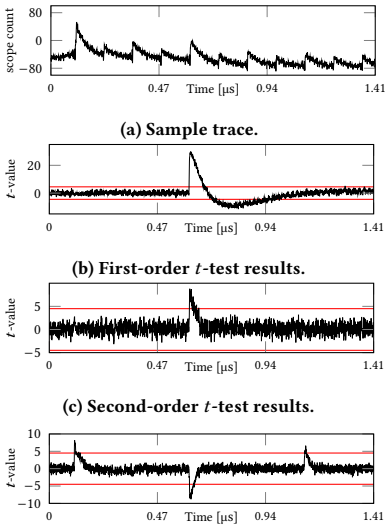

Figure 7. Measurement results of a PRESENT S-box generated from $(2,2)$-CINI gadgets with 2-bit fault injection (1 Million traces).

tiles (getting all intermediate values of that tile) or fault tiles (manipulating all intermediate values of that tile), as long as the sum of probed and faulted tiles does not exceed the order d. However, using MPC, CAPA suffers from a large area overhead. In contrast, M&M [13] combines masking with a MAC and infection quite efficiently, however, without any formal security guarantees for combined attacks. Similar to our work, they use the glitch-extended probing model in combination with precise fault-injection abilities and a gadget-based approach. As with CAPA, the sum of injected faults and placed probes is bounded by the probing order d. Both CAPA and M&M have a security parameter m that determines the unforgeability of the MAC, where the security increases with m. Neither CAPA nor M&M protects against Statistical Ineffective Fault Analysis (SIFA) [17]. However, we argue that SIFA is always possible in a CA setting, as the adversary can inject faults and perform enough executions for statistical evaluation. Dhooghe and Nikova [15] provide the work most similar to ours in that they give security definitions and gadgets based on the NI and SNI composability notion. However, their first software gadget (cf. Algorithm 2 in [15]) is not transferable to hardware as it uses an *abort* that is hard to implement in hardware, and their second gadget (cf. Algorithm 5 in [15]) was shown to be flawed by Richter-Brockmann et al. [36]. While it should be possible to fix the flaw, we expect the fix to be expensive in terms of area and randomness. We compare security guarantees and performance of our work with CAPA and M&M in Table 5. Here, the numbers for CAPA and M&M are given for $m = 1$, as done in the original papers. While M&M is more efficient in area than our proposal, it has a high cost in required fresh randomness and suffers from the lack of formal security guarantees in CA. Therefore, even when expensive, our work achieves a good balance between provided security guarantees and implementation costs.

9 CONCLUSION

In this work, we transferred the idea of domain isolation from PINI to both FIA and CA. Alongside, we introduce a formal treatment of replication and propose the first gadgets for combined security in hardware. Arguably, the proposed gadgets inflict a significant cost, however, also provide security guarantees against a powerful adversary, for which we provide formal proofs.

Future Work. While PINI-based compositions have proven more efficient than SNI-based compositions for many metrics [9, 10], SNI provides independence guarantees between inputs and outputs that have proven useful, in particular for optimization [21, 22]. Hence, other gadgets with other properties are a useful addition to our work. While $\mathsf{HPC}_2^{\mathsf{C}}$ captures the definition of CINI tightly for $d \leq 2$ and $k \leq 2$ a tight gadget for the general case remains an open challenge. We also focus entirely on replication to counteract the effect of faults, however, the literature knows a wide variety of linear codes for that purpose. In general, using linear codes poses new challenges for composition since it is not always possible to encode a single bit. However, using other linear codes than replication could lead to more efficient gadgets, reducing some of the discussed overhead for combined security. An interesting and open question is the option of hardware-implemented fault detection mechanisms in the context of combined security.

ACKNOWLEDGMENTS

The work described was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the projects VE-HEP (16KIS1345) and 6GEM (16KISK038) supported by the German Federal Ministry of Education and Research BMBF.

REFERENCES

[1] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. 2020. Impeccable Circuits. *IEEE Trans. Computers* 69, 3 (2020), 361–376.

[2] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. 2018. Private Circuits: A Modular Approach. In *CRYPTO*. 427–455.

[3] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. 2015. Verified Proofs of Higher-Order Masking. In *EUROCRYPT*. 457–485.

[4] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. 2016. Strong Non-Interference and Type-Directed Higher-Order Masking. In *SIGSAC*. 116–129.

[5] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. 2016. Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*. 23–39.

[6] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche. 2011. *The Keccak Reference*.

[7] Joan Boyar and René Peralta. 2012. A Small Depth-16 Circuit for the AES S-Box. In *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*. 287–298.

[8] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*. 136–145.

[9] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. 2021. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers* 70, 10 (2021), 1677–1690.

[10] Gaëtan Cassiers and François-Xavier Standaert. 2020. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 2542–2555.

[11] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO (Lecture Notes in Computer Science, Vol. 1666)*, Michael J. Wiener (Ed.). Springer, 398–412.

[12] Christophe Clavier. 2007. Secret External Encodings Do Not Prevent Transient Fault Analysis. In *CHES 2007 (Lecture Notes in Computer Science, Vol. 4727)*. Springer, 181–194.

[13] Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2019. M&M: Masks and Macs against Physical Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 1 (2019), 25–50.

[14] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. 2012. Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES. In *FDTC 2012*. IEEE Computer Society, 7–15.

[15] Siemen Dhooghe and Svetla Nikova. 2020. My Gadget Just Cares for Me - How NINA Can Prove Security Against Combined Attacks. In *CT-RSA (Lecture Notes in Computer Science, Vol. 12006)*. Springer, 35–55.

[16] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. 2018. Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures. In *ASIACRYPT (Lecture Notes in Computer Science, Vol. 11273)*. Springer, 315–342.

[17] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. 2018. SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 547–572.

[18] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. 2014. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*. 423–440.

[19] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine. 2019. Electromagnetic Fault Injection : How Faults Occur. In *FDTC 2019*. IEEE, 9–16.

[20] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. 2018. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 89–120.

[21] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. 2017. Amortizing Randomness Complexity in Private Circuits. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*. 781–810.

[22] Jakob Feldtkeller, David Knichel, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. 2022. Randomness Optimization for Gadget Compositions in Higher-Order Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 4 (2022), 188–227.

[23] Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. 2022. CINI MINIS: Domain Isolation for Fault and Combined Security. Cryptology ePrint Archive, Paper 2022/1131. *IACR Cryptol. ePrint Arch.* 2022 (2022). http://eprint.iacr.org/2022/1131

[24] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *CHES (Lecture Notes in Computer Science, Vol. 2162)*, Çetin Kaya Koç, David Naccache, and Christof Paar (Eds.). Springer, 251–261.

[25] Michael Gruber, Matthias Probst, Patrick Karl, Thomas Schamberger, Lars Tebelmann, Michael Tempelmeier, and Georg Sigl. 2021. DOMREP-An Orthogonal Countermeasure for Arbitrary Order Side-Channel and Fault Attack Protection. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 4321–4335.

[26] Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO (Lecture Notes in Computer Science, Vol. 2729)*, Dan Boneh (Ed.). Springer, 463–481.

[27] Adam M. Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, and Jonathan Bachrach. 2017. Reusability is FIRRTL Ground: Hardware Construction Languages, Compiler Frameworks, and Transformations. In *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*. 209–216.

[28] David Knichel and Amir Moradi. 2022. Low-Latency Hardware Private Circuits. *IACR Cryptol. ePrint Arch.* 2022 (2022). http://eprint.iacr.org/2022/507

[29] David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT (Lecture Notes in Computer Science, Vol. 12491)*. Springer, 787–816.

[30] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO (Lecture Notes in Computer Science, Vol. 1109)*, Neal Koblitz (Ed.). Springer, 104–113.

[31] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO (Lecture Notes in Computer Science, Vol. 1666)*, Michael J. Wiener (Ed.). Springer, 388–397.

[32] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques A. Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2021, Seoul, South Korea, February 27 - March 3, 2021*. 2–14.

[33] Tal Malkin, François-Xavier Standaert, and Moti Yung. 2006. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In *FDTC 2006 (Lecture Notes in Computer Science, Vol. 4236)*. Springer, 159–172.

[34] Ueli Maurer. 2011. Constructive Cryptography - A New Paradigm for Security Definitions and Proofs. In *TOSCA*. 33–56.

[35] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. 2018. CAPA: The Spirit of Beaver Against Physical Attacks. In *CRYPTO 2018 (Lecture Notes in Computer Science, Vol. 10991)*. Springer, 121–151.

[36] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. 2022. VERICA - Verification of Combined Attacks: Automated Formal Verification of Security Against Simultaneous Information Leakage and Tampering. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 4 (2022), 255–284.

[37] Jan Richter-Brockmann and Tim Güneysu. 2020. Improved Side-Channel Resistance by Dynamic Fault-Injection Countermeasures. In *31st IEEE International Conference on Application-specific Systems, Architectures and Processors , ASAP 2020, Manchester, United Kingdom, July 6-8, 2020*. 117–124.

[38] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. 2021. FIVER – Robust Verification of Countermeasures against Fault Injections. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 4 (Aug. 2021), 447–473.

[39] Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. 2022. Revisiting Fault Adversary Models - Hardware Faults in Theory and Practice. *IEEE Trans. Computers* (2022), 1 – 14.

[40] Tobias Schneider and Amir Moradi. 2015. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES (Lecture Notes in Computer Science, Vol. 9293)*. Springer, 495–513.

[41] Tobias Schneider, Amir Moradi, and Tim Güneysu. 2016. ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In *CRYPTO 2016 (Lecture Notes in Computer Science, Vol. 9815)*. Springer, 302–332.

[42] Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. 2020. Impeccable Circuits II. In *DAC 2020*. IEEE, 1–6.

[43] Sergei P. Skorobogatov and Ross J. Anderson. 2002. Optical Fault Induction Attacks. In *CHES 2002 (Lecture Notes in Computer Science, Vol. 2523)*. Springer, 2–12.

[44] Albert Spruyt, Alyssa Milburn, and Lukasz Chmielewski. 2021. Fault Injection as an Oscilloscope: Fault Correlation Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 1 (2021), 192–216.

[45] Loïc Zussa, Jean-Max Dutertre, Jessy Clédière, and Assia Tria. 2013. Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism. In *IOLTS 2013*. IEEE, 110–115.