VERICA - Verification of Combined Attacks

Automated formal verification of security against simultaneous information leakage and tampering

Jan Richter-Brockmann[1], Jakob Feldtkeller[1], Pascal Sasdrich[1] and Tim Güneysu[1,2]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
[2] DFKI, Bremen, Germany
firstname.lastname@rub.de

Abstract. Physical attacks, including passive Side-Channel Analysis and active Fault Injection Analysis, are considered among the most powerful threats against physical cryptographic implementations. These attacks are well known and research provides many specialized countermeasures to protect cryptographic implementations against them. Still, only a limited number of combined countermeasures, i.e., countermeasures that protect implementations against multiple attacks simultaneously, were proposed in the past. Due to increasing complexity and reciprocal effects, design of efficient and reliable combined countermeasures requires longstanding expertise in hardware design and security. With the help of formal security specifications and adversary models, automated verification can streamline development cycles, increase quality, and facilitate development of robust cryptographic implementations.
In this work, we revise and refine formal security notions for combined protection mechanisms and specifically embed them in the context of hardware implementations. Based on this, we present the first automated verification framework that can verify physical security properties of hardware circuits with respect to combined physical attacks. To this end, we conduct several case studies to demonstrate the capabilities and advantages of our framework, analyzing secure building blocks (gadgets), S-boxes build from Toffoli gates, and the ParTI scheme. For the first time, we reveal security flaws in analyzed structures due to reciprocal effects, highlighting the importance of continuously integrating security verification into modern design and development cycles.

Keywords: SCA · FIA · Formal Verification · BDD · Symbolic Simulation · Combined Analysis

1 Introduction

Physical Attacks and Countermeasures. While the theory for design and construction of secure cryptographic algorithms is a well-matured field of research [KR11], secure practical implementation and instantiation of cryptography still remains a challenging task. For instance, physical effects and characteristics of modern electronic devices, such as timing behavior [Koc96], instantaneous power consumption [KJJ99], or electromagnetic (EM) emanations [GMO01], can leak sensitive information on processed data. Since Side-Channel Analysis (SCA) is a well-known threat to cryptographic implementations, a wide range of protection mechanisms and countermeasures have been presented over time. Among all proposals, *masking* is a well-studied and promising approach, mainly due to its theoretically sound security foundations [CJRR99]. Besides, not only passive observation and analysis of electronic devices running cryptographic algorithms, but also active disturbance and manipulation of modern electronic devices, e.g., through clock glitches [DEG$^+$18], voltage glitches [ZDCT13], EM pulses [DDRT12, DLM19], or focused laser beams [SA02], can expose sensitive information to an adversary. Similarly, Fault Injection Analysis (FIA) is a well-known threat and several countermeasures, mostly relying on redundancy in terms of *time*, *area*, or *information*, are used to detect or correct injected transient faults.

Combined Hardware Security. However, although SCA and FIA are well-known and well-studied threats to secure implementations of cryptographic algorithms, both threats and potential security mechanisms are mostly addressed and evaluated in isolation. More precisely, while *masking* and *redundancy* provide strong security guarantees against SCA and FIA, respectively, the impact and threat of combining FIA and SCA has long been neglected and underestimated, hence, only a limited number of combined countermeasures can be found in literature.

As one of the first attempts, ParTI [SMG16], a first-order secure Threshold Implementation (TI) of LED, was protected against FIA based on a detection scheme using linear error codes for information redundancy. Based on the concepts presented in [SRM20], this TI can be extended to also provide a correction-based protection instead of only detecting faults. Subsequently, different approaches have been investigated, e.g., using concepts of Multi-Party Computation (MPC) [RMB$^+$18], Message Authentication Codes (MACs) [MAN$^+$19], orthogonal error correction [RSBG20], or transformation and encoding [SJR$^+$20], to build robust countermeasures against combined attacks. However, as it is well known, secure design and implementation of cryptographic algorithms require long-standing expertise and experience in the fields of hardware design and security, as the complexity, effort, and cost of these tasks increase dramatically with design complexity. For this, most recent approaches focus on the design and implementation of smaller building blocks, e.g., based on Toffoli gates [DDE$^+$20] or masked multiplication *gadgets* [DN20], to compose larger designs from provably secure components.

Formal Verification of Hardware Security. Due to the ever increasing and growing design complexity, formal verification nowadays is a fundamental part of Very Large Scale Integration (VLSI) design cycles in industry [BK18]. Regardless of this, existing formal verification mostly focuses on functional correctness (e.g., in form of model checking [CCGR99]) and is applied for example to entire Central Processing Units (CPUs) as shown in [SSR$^+$18] for RISC-V processors. Additionally, a widespread application of formal verification can be found in safety-critical environments, e.g., in the automotive industry [GLH18]. However, verification of secure implementation is often not considered. In light of this, recent hardware security research also stimulates progress and innovation for formal models of *active* and *passive* adversaries and the physical execution environments of modern electronic devices. Ideally, sound and accurate formal models can simplify and assist in verification of security and functional correctness of cryptographic implementations to shorten and streamline hardware development cycles. For this, in the context of *masking*, formal security verification is commonly performed in the abstract and elegant Ishai-Sahai-Wagner (ISW) threshold d-probing model [ISW03], modeling side-channel leakage in terms of d adversarial probes, providing access to intermediate values of a digital logic circuit during operation. Inspired by the sophistication of this formal security model, a consolidated fault adversary model was presented in [RBSG22]. However, again, formal models of physical attacks, consulted during security verification, are mostly developed in isolation but neglect the threat of combined attacks. In addition, as mentioned before, complexity of verification increases dramatically with increasing design size and model complexity, rendering manual security verification nearly infeasible.

Automated Tools for Formal Verification. These challenges inevitably led to the research and development of automated security reasoning and computer-aided security verification tools. In connection with SCA and the d-probing model, existing verification tools either focus on specific countermeasures [ANR18], direct security reasoning [BGI$^+$18, BBC$^+$19, KSM20, GHP$^+$21, HB21, BMRT21], or assume securely masked gadgets and verify the correct composition under secure composability notions [BGR18, BDM$^+$20, CGLS21]. In the context of FIA and active information tampering, state-of-the-art verification tools evaluate specific countermeasures [HPB21], algebraic vulnerabilities [KRH17], or use simulation to ensure the robustness of redundancy-based hardware countermeasures [AWMN20, RBRSS$^+$21]. As mentioned before, security threats from SCA and FIA are mainly considered in isolation and none of the existing security verification tools allow security reasoning under Combined Analysis (CA).

Contributions. We start this work by extending and proving formal models considering CA, by precisely specifying adversarial advantages when injecting faults in masked circuits. Specifically, we provide the first accurate definition of fault-free masked circuits required as fundamental construction for combined security. In addition, we refine existing software-only composability notions for CA (proposed in [DN20]) by specifically embedding them in a hardware context, considering gadgets with multiple-output functions, and, for the first time, give a formal security argument of the impact of faulty randomness. Thereby, in specifically considering reciprocal effects, we consolidate isolated security models and composability notions for SCA and FIA into a holistic system applicable for CA.

Given this, we present the first automated framework for formal security verification of hardware circuits under CA, reconciling and uniting previous concepts that only addressed each physical attack in isolation. More specifically, this framework automatically verifies security of digital logic circuits and secure composability of fundamental building blocks (gadgets) in the presence of combined attacks In light of this, we show that, due to unnoticed reciprocal effects, combined verification requires concepts and techniques beyond simple combination of SCA and FIA verification. Using these features, we conduct several case studies, i.e., analyzing the gadgets proposed in [DN20], dedicated Statistical Ineffective Fault Analysis (SIFA) countermeasure recently proposed in [DDE$^+$20], and the ParTI protection scheme [SMG16]. In particular, these case studies helped to reveal unknown security flaws in [DN20] which impact software and hardware implementations, likewise. Consequently, the automated verification framework is able to perform stand-alone SCA or FIA verification as well as CA and is publicly available on GitHub. We believe that our open-source tool VERICA sparks and supports new research on combined attacks and countermeasures.

Outline. In Section 2, we summarize our notations and introduce the fundamental hardware circuit model. Section 3 covers essential adversary models, security definitions, and composability notions for SCA, FIA, and CA. In Section 4, we combine stand-alone verification strategies into a combined verification methodology, for the first time considering reciprocal effects. Section 5 presents our case studies and extensively evaluates our proposed verification methodology using a variety of protected hardware circuits taken from literature. Eventually, we discuss and conclude our work in Section 6.

2 Preliminaries

In this section, we cover essential details required for the remainder of this work, i.e., we summarize our notations in Table 1 and specify our fundamental hardware circuit model.

Table 1. Notations used throughout this work.

Notation	Description
d	Security order of side-channel countermeasures.
s	Number of shares used by a side-channel countermeasure.
k	Security order of fault injection countermeasures.
f	Fault cardinality (number of simultaneously injected faults).
t	Fault type.
l	Fault location.
m	Number of bits of an uncoded message.
n	Number of bits of a codeword generated by a linear error code.
r	Number of redundant bits generated by a linear error code.
ℓ	Number of distinct messages.
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.

2.1 Circuit Model

We consider and model any deterministic digital logic circuit C, given in a gate-level netlist description, as a *Direct Acyclic Graph* $\mathcal{D} = \{\mathcal{V}, \mathcal{E}\}$. More specifically, the logic circuit C is decomposed into atomic components, denoted *gates* and modeled as nodes in the graph. Formally, we distinguish *combinational* gates, *randomness* gates and *memory* gates according to the following definitions.

Definition 1 (Combinational Gate)**.** A combinational gate $g \in \mathcal{G}_{\mathsf{c}}$ is a physical component in a digital logic circuit C that evaluates outputs as a pure Boolean function of its current inputs only, but without any dependency on the history of previous inputs.

For the sake of simplicity, but without loss of generality, we restrict the set of considered combinational gates to $\mathcal{G}_{\mathsf{c}} = \{\mathsf{not}, \mathsf{and}, \mathsf{nand}, \mathsf{or}, \mathsf{nor}, \mathsf{xor}, \mathsf{xnor}\}$.

Definition 2 (Memory Gate)**.** A memory gate $g \in \mathcal{G}_{\mathsf{m}}$ is a physical, clock-synchronized component in a digital logic circuit C for which the outputs not only depend on present inputs but also on the history of previous inputs.

Again, for the sake of simplicity, but without any loss of generality, we restrict the set of memory gates to registers, i.e., $\mathcal{G}_{\mathsf{m}} = \{\mathsf{reg}\}$. Each memory gate models a clock-dependent synchronization point within the circuit and can store a single Boolean variable $x \in \mathbb{F}_2$.

Definition 3 (Randomness Gate)**.** A randomness gate $g \in \mathcal{G}_{\mathsf{rand}}$ is a physical, clock-synchronized component in a digital logic circuit C without inputs. For each clock cycle the output is an independently and uniformly chosen random value.

In sum, the circuit model of a digital logic circuit C is defined according to Definition 4.

Definition 4 (Circuit Model)**.** A digital logic circuit C is modeled by a Direct Acyclic Graph (DAG) given as $\mathcal{D} = \{\mathcal{V}, \mathcal{E}\}$, with $\mathcal{V}$ denoting the set of vertices and $\mathcal{E}$ denoting the set of edges. Further, a single vertex $v \in \mathcal{V}$ represents a combinational, memory, or randomness gate $g \in \mathcal{G}$, with $\mathcal{G} = \mathcal{G}_{\mathsf{c}} \cup \mathcal{G}_{\mathsf{m}} \cup \mathcal{G}_{\mathsf{rand}}$, and each edge $e \in \mathcal{E}$ represents a wire connecting two gates carrying an element of the finite field $\mathbb{F}_2$.

Further, a circuit is called a *gadget* iff it implements a function under consideration of security and compositional properties. This could, for example, be a circuit that implements a multiplication in $\mathbb{F}_2$, such that it is secure against side-channel attacks and remains so under composition with other circuits.

3 Security Models

In this section, we briefly summarize existing security models for SCA and FIA and revisit corresponding secure composability notions. Eventually, we present an integrated adversary model and unified composability notions for CA, specifically embedded into the hardware context.

Basic Adversary Model. According to our basic adversary model, any adversary is given access to the circuit C and its physical structure. Then, the adversary can invoke C multiple times with constant secret inputs while potential randomness is drawn freshly for each invocation. Still, the adversary has no knowledge or control on the actual values that are processed, including input, intermediate, and output values [ISW03]. The specific view of the adversary, in addition to further capabilities, are discussed in the remainder of this section for different scenarios.

Security Proofs via Simulation. Simulation is a proof technique for security arguments that is in particular useful when statements about composability are required [Can01, Mau11]. In this setting, we define a *real* and an *ideal* game, where the ideal game is trivially secure (under some adversary model). A system in the real game is then said to be secure iff the view to the adversary is indistinguishable from the ideal game, i.e., there is no adversary who can distinguish the two games with a probability higher than $\frac{1}{2}$. The ideal game is defined by a probabilistic polynomial-time simulator that reproduces the view of the adversary without access to the secret. From this we can conclude, that the view of the adversary is indeed independent of the secret.

Statistical Independence. Two events A and B are *statistical independent* iff it holds that $Pr[B|A] = Pr[B]$.

3.1 Modeling Side-Channel Analysis

Let us first recall essential definitions for side-channel security from literature. Those are *passive* attacks in that the adversary only observes the behavior of some circuit without any active tampering during processing.

3.1.1 Adversary and Security Model

In the stateless ISW *d-probing model* [ISW03], an adversary is given the exact values of up to d wires of a circuit C (called probes) that can be freely chosen prior to each invocation. Consequently, probing security, in terms of *privacy*, is given iff for all observations the view of the adversary is independent of any processed secrets , i.e., all probes can be simulated without access to any wire of the circuit.

A well known source of additional side-channel leakage in hardware comes from physical defaults, e.g., glitches. A glitch occurs whenever there are timing differences in the propagation of signals through the logic, i.e., if the output of a gate evaluates for a short amount of time to a wrong value due to timing differences at the input. Then, for a short amount of time, a wire can carry a different secret-dependent value than the one it carries in an ideal execution. To model this kind of leakage, the *glitch-robust probing model* [FGP$^+$18] gives the adversary not only access to the exact value carried by a wire, but the exact value of all stable synchronization points, i.e., clocked registers, a probed wire depends on. Hence, a glitch-extended probe is a set of values instead of a single value. Throughout the paper, we consider registers as elements stopping the propagation of glitches, however, not defining a state in the sense of the stateful probing model [ISW03].

Further, we restrict our analysis to the glitch-robust probing model even when there are other physical defaults that can cause leakage [FGP+18].

3.1.2 Countermeasures and Protection Mechanisms

Boolean masking [CJRR99] is one of the most promising solutions to protect against d-probing adversaries, due to its sound theoretical foundations.

Definition 5 (Boolean Sharing)**.** A *Boolean sharing* of a value $x \in \mathbb{F}_2^m$ is a vector $\langle x_0, \ldots, x_{s-1} \rangle$ such that $x = \bigoplus_{i=0}^{s-1} x_i$, with $x_i \in \mathbb{F}_2^m$ uniform random and for all $\mathcal{X} \subsetneq \{x_0, ..., x_{s-1}\}$ the values $(x_i)_{i \in \mathcal{X}}$ are independent. Further, let $\mathsf{S} : \mathbb{F}_2^m \mapsto \mathbb{F}_2^{m \cdot s}$ be a probabilistic *share function* that outputs a valid Boolean sharing $\langle x_0, \ldots, x_{s-1} \rangle$ for some x. Similarly, $\mathsf{U} : \mathbb{F}_2^{m \cdot s} \mapsto \mathbb{F}_2^m$ is a deterministic *unshare function* that computes the original value x for a share vector $\langle x_0, \ldots, x_{s-1} \rangle$.

By abuse of the notation, we write $\mathsf{S}(x)$ for an $x \in \mathbb{F}_2^{m \cdot \ell}$ when the encode function is applied element-wise on ℓ different values $x^j \in \mathbb{F}_2^m$. Similarly, we write $\mathsf{U}(x)$ for an $x \in \mathbb{F}_2^{m \cdot s \cdot \ell}$ when the decode function is applied vector-wise on ℓ different share vectors.

We define a *shared circuit* as a digital logic circuit that operates on Boolean shares. Further, the sharing and unsharing functions are considered outside of the adversarial scope (as usually done in literature), as they inherently violate the definition of d-probing security [AIS18].

Definition 6 (Shared Circuit)**.** A *shared circuit* C_F^s for a function $\mathsf{F} : \mathbb{F}_2^{m \cdot \ell} \mapsto \mathbb{F}_2^{m \cdot \ell'}$ and a Boolean sharing scheme with s shares is a probabilistic circuit realizing a function $\mathsf{F}^\mathsf{C} : \mathbb{F}_2^{m \cdot s \cdot \ell} \mapsto \mathbb{F}_2^{m \cdot s \cdot \ell'}$, such that $\forall x \in \mathbb{F}_2^{m \cdot \ell}$ it holds that $\mathsf{F}(x) = \mathsf{U}(\mathsf{F}^\mathsf{C}(\mathsf{S}(x)))$ (*functional correctness*).

3.1.3 Composability Notions

The secure construction of shared circuits is a laborious and error-prone task and the notion of probing security does not support composition of circuits on its own. Therefore, research has focused on construction of gadgets, i.e., atomic building blocks, which retain certain security properties when combined.

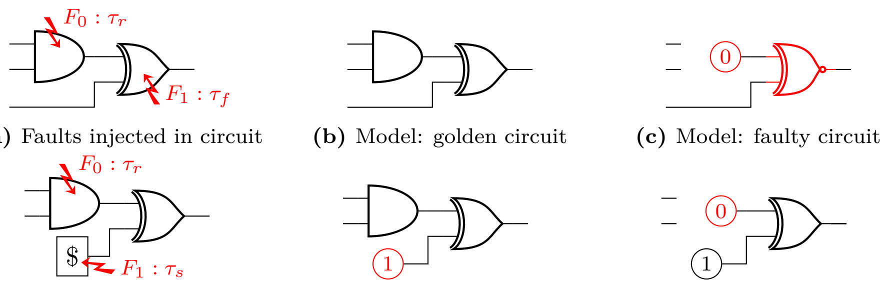
As a first notion for composition, Barthe et *al.* [BBD+15] proposed Probe Non-Interference (P-NI) which limits the amount of leakage for shared intermediates (in contrast to the conventional naming, we specify the type of non-interference to distinguish it from other non-interference definitions later in this work).

Definition 7 (Probe Non-Interference [BBD+15])**.** A shared gadget G is *d-P-NI* iff any set of $d' \leq d$ probes can be perfectly simulated with at most d' shares of each input.

As the composition of P-NI gadgets cannot guarantee probing secure circuits, P-NI on its own is not sufficient. To address this challenge, Probe Strong Non-Interference (P-SNI) [BBD+16] has been proposed as a more restrictive composability notion.

Definition 8 (Probe Strong Non-Interference [BBD+16])**.** A shared gadget G realizing a function $\mathsf{F} : \mathbb{F}_2^{m \cdot s \cdot \ell} \mapsto \mathbb{F}_2^{m \cdot s}$ is *d-P-SNI* iff for any set of probes, of which d_1 are internal probes and d_2 are output probes such that $d_1 + d_2 \leq d$, the probes can be perfectly simulated by d_1 shares of each input.

Both P-NI and P-SNI gadgets are inherently probing secure [BBD+15, BBD+16]. In addition, the composition of two P-SNI gadgets is again P-SNI [BBD+16]. However, a more efficient composition can be achieved by combining P-NI and P-SNI gadgets when following certain rules [BBD+16, BGR18].



(a) Faults injected in circuit **(b)** Model: golden circuit **(c)** Model: faulty circuit

(d) Faults injected in circuit **(e)** Model: golden circuit **(f)** Model: faulty circuit

Figure 1. Fault model and golden circuits for general circuits (a to c) and shared circuits in particular (d to f).

For large and complex circuits, the composition of P-NI and P-SNI gadgets can incur unnecessary overhead. For this Cassiers and Standaert proposed Probe-Isolating Non-Interference (PINI) [CS20], which enables more efficient constructions, e.g., a trivially masked XOR gadget.

3.2 Modeling Fault Injection Analysis

We continue with fundamental definitions in the context of fault injection security, which is considered as *active* attack as the adversary actively interferes with circuit execution. In addition, we make small adjustments (where necessary) in preparation for modeling combined attacks in Section 3.3.

3.2.1 Adversary and Security Model

Following the work in [RBSG22], an adversary is modeled by a function $\zeta(f, t, l)$. Here, f limits the number of simultaneous faults that can occur at the same time and t is used to precisely specify the fault and is selected from a set $\mathcal{T} = \{\tau_{sr}, \tau_s, \tau_r, \tau_{bf}, \tau_{fm}\}$. Particularly, τ_{bf} describes bit-flips, τ_s, τ_r, and τ_{sr} consider stuck-at-one (set), stuck-at-zero (reset), or both, respectively, while τ_{fm} is used to specify custom fault behaviors (e.g., given specific technology properties). Eventually, the fault location l can be selected from $\mathcal{L} = \{c_i, m, mc_i\}$ and restricts the fault injections to combinational gates $g \in \mathcal{G}_c$, to memory gates $g \in \mathcal{G}_m$, or both gate types $g \in \mathcal{G}_c \cup \mathcal{G}_m$, respectively. The index i for the selection of combinational gates is used to define a finer grained selection of combinational gates placed between two register stages. More precisely, $i = 0$ involves only these gates that have the longest Data Arrival Time (DAT) while $i = \infty$ involves all combinational gates. For more details, we refer the interested reader to the original work [RBSG22]. With this, specific faults are modeled by replacing the faulted gate with a different gate defined by the introduced fault. Afterwards, the fault propagation can be identified and effective and ineffective faults be distinguished by comparing the faulty circuit model to a fault-free circuit model, the so called *golden circuit*. We illustrate the used fault model in Figure 1(a-c).

Recently, Dhooghe and Nikova proposed a definition for *active security* [DN20] where an adversary is allowed to inject up to k faults into a circuit C at arbitrary – but prior to each invocation selected – locations. Then *active security*, in terms of correctness, is given iff for all fault combinations C either aborts or outputs a correct result, i.e., equivalence with the golden circuit. Here, the view of the adversary contains the (optional) abort signal and the correctness of the result. Please note, that to provide security against some attacks (e.g., SIFA [DEK+18]) no abort signal is allowed. We extend this definition with the adversary model from [RBSG22] and by explicitly require a protected decoding algorithm for error detection or correction[1].

When analyzing gadgets for secure composability in the presence of fault injections (e.g., gadgets from [DN20]), we further extend the locations $\mathcal{L}$ of valid fault injections to additionally consider faulty inputs. This includes faults in data inputs (due to faulty outputs of other gadgets) as well as randomness gates.

Definition 9 ((k, t, l)-Fault Security)**.** 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. A circuit C together with a decoding G^{D} is (k, t, l)-fault secure iff for any set of up to k faults of type t injected in gates of type l in C, the concatenation $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$ either aborts or outputs a result equal to the golden circuit of C.

The final decoding gadget is necessary for both practicality and security. It is practically impossible to guarantee functional correctness at the output with an unrestricted adversary, as they can always fault the output directly. For security, the output behavior must be independent of any sensitive information. In the following, we write k-fault security when we do not restrict the fault type and fault location.

3.2.2 Countermeasures and Protection Mechanisms

Countermeasures for fault security are mostly based on redundant information or computation to detect or correct transient faults. In this work, we focus on redundant information and consider data encoding using *binary linear codes*.

Definition 10 (Binary Linear Code)**.** An (n, m)*-linear code* $\mathcal{C}$ of length n, dimension m, and co-dimension $r = n - m$ is a m-dimensional subspace of $\mathbb{F}_2^n$. Further, let $\mathsf{E} : \mathbb{F}_2^m \mapsto \mathbb{F}_2^n$ be a deterministic *encode function* following the encoding mechanism of the linear code, and $\mathsf{D} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ be a deterministic *decode function* that receives a codeword c and computes the corresponding message m or aborts.

We define an *encoded circuit* as a digital circuit that operates on values secured by information redundancy of linear codes. In accordance with Definition 9 and similar to Definition 6 we do not consider the encoding and decoding as part of the circuit, as faulting them inherently violates the definition of fault security.

Definition 11 (Encoded Circuit)**.** An *encoded circuit* $\mathsf{C}_{\mathsf{F}}^{\mathcal{C}}$ for a function $\mathsf{F} : \mathbb{F}_2^{m\cdot\ell} \mapsto \mathbb{F}_2^{m\cdot\ell'}$ and a linear code $\mathcal{C}$ is a deterministic circuit realizing a function $\mathsf{F}^{\mathsf{C}} : \mathbb{F}_2^{n\cdot\ell} \mapsto \mathbb{F}_2^{n\cdot\ell'}$, such that $\forall x \in \mathbb{F}_2^{m\cdot\ell}$ it holds that $\mathsf{F}^{\mathsf{C}}(\mathsf{E}(x)) \in \mathcal{C}$ and $\mathsf{F}(x) = \mathsf{D}(\mathsf{F}^{\mathsf{C}}(\mathsf{E}(x)))$ (*functional correctness*).

3.2.3 Composability Notions

Dhooghe and Nikova additionally present security notions for composable injection-secure gadgets [DN20]. Inspired by the P-NI property, the Fault Non-Interference (F-NI) property ensures that each intermediate fault only propagates to at most a single output. However, this is relaxed by also allowing abortion of computations upon fault detection. Please note, this definition is equal to the definition of Non-Accumulation [DN20], however, with an explicit requirement for the existence of an appropriate decoding gadget.

Definition 12 (Fault Non-Interference [DN20])**.** A gadget G is k-F-NI iff for any set of $k' \leq k$ faults at inputs and injected on gates in G, the gadget either aborts or gives an output with at most k' bit faults and there exists a decoding gadget G^{D}, such that given an input with at most k faulty inputs and an abort signal, G^{D} either aborts or outputs a correct result.

[1] [DN20] implicitly requires a protected decode gadget as well, however, we explicitly include it into the definition to make this requirement more transparent.

Analogous to P-SNI tightening P-NI, the notion of Fault Strong Non-Interference (F-SNI) extends F-NI and ensures full composability in distinguishing input and intermediate faults. Again, this definition is equal to the definition of Strong Non-Accumulation [DN20], however, with an explicit requirement for the existence of an appropriate decoding gadget.

Definition 13 (Fault Strong Non-Interference [DN20])**.** A gadget G is k-F-SNI iff for any set of k_1 faulty inputs and every set of k_2 faults injected in gates of G, with $k_1 + k_2 \leq k$, the gadget either aborts or gives an output with at most k_2 bit faults and there exists a decoding gadget G^{D}, such that given an input with at most k faulty inputs and an abort signal, G^{D} either aborts or outputs a correct result.

Both k-F-NI and k-F-SNI imply k-fault security given a decoding gadget G^{D} that can detect or correct k faults. In addition, it holds that the composition of two F-SNI gadgets is F-SNI again [DN20].

3.3 Modeling Combined Analysis

In this section, we now focus on CA, i.e., considering adversaries that are able to simultaneously inject faults and place probes. In that, we extend the state of the art in [DN20] by refinement of circuit models in shared contexts and consideration of leakage caused by fault injections.

3.3.1 Combined Adversary and Security Model

Security in the combined adversary and security model has to amalgamate the definitions of probing security and fault security, while additionally considering any reciprocal effects. For this, we adopt and refine the formal notions in [DN20], considering the extended fault model in [RBSG22] and explicit decoding functions. Again, all probes and faults are selected prior to each invocation of the circuit and the view of the adversary is defined by the probed values, the (optional) abort signal, and the correctness of the result of the concatenation $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$, where G^{D} is a circuit realizing a error detection or error correction mechanism for up to k faults.

Definition 14 ((d, k, t, l)-Combined Security)**.** 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 corrected result. A circuit C together with a decoding G^{D} is (d, k, t, l)-*combined secure* iff for any set of up to k faults of type t injected on gates of type l in C, and any set of up to d probes placed on wires in C the following holds:

Privacy:	The abort signal and the probes can be simulated without access to any wire of C.
Correctness:	The concatenation $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$ either aborts or outputs a result equal to the golden circuit of C.

In the following, we write (d, k)-combined security when we do not restrict the fault type and location. Further, we emphasize the adversarial knowledge on faults, since injecting faults is strongly linked to placing probes [Cla07, SMC21].

Remark 1 (Adversarial Knowledge on Faults)*.* A fault is known iff the adversary exactly learns the targeted gate, and the fault type t, i.e., the misbehavior caused by the fault.

With this, an adversary additionally learns the following details: (i) the location of propagated faults, (ii) the distribution of the faults under a chosen input distribution, and (iii) the probability of fault occurrences under a chosen input distribution. Still, the adversary does not necessarily learn the exact values (but only distributions). This is also the case when the exact position of the fault injection is not modeled, e.g., in the case of faulty inputs (see below).

3.3.2 Countermeasures and Protection Mechanisms

One obvious countermeasure against combined attacks is the combination of Boolean sharing and linear codes. Since both rely on linear operations, we assume, without loss of generality, that the sharing is applied before the linear code. Hence, the initial transformation in the context of combined security is defined as $\mathsf{E} \circ \mathsf{S} : \mathbb{F}_2^m \mapsto \mathbb{F}_2^{n \cdot s}$ first applying Boolean sharing S and then an encoding of a linear-error code E on each created share. The corresponding reversed transformation is defined as $\mathsf{U} \circ \mathsf{D} : \mathbb{F}_2^{n \cdot s} \mapsto \mathbb{F}_2^m$ which receives a shared codeword and computes the corresponding unshared message.

Given this, we can define a *shared and encoded circuit* as follows. Again, to be consistent with Definition 6 and Definition 11, we keep the encoding and decoding separate from the actual circuit.

Definition 15 (Shared and Encoded Circuit)**.** A *shared and encoded circuit* $\mathsf{C}_{\mathsf{F}}^{(s,\mathcal{C})}$ for a function $\mathsf{F} : \mathbb{F}_2^{m \cdot \ell} \mapsto \mathbb{F}_2^{m \cdot \ell'}$, a Boolean-sharing scheme with s shares, and a linear code $\mathcal{C}$ is a probabilistic circuit realizing a function $\mathsf{F}^{\mathsf{C}} : \mathbb{F}_2^{n \cdot s \cdot \ell} \mapsto \mathbb{F}_2^{n \cdot s \cdot \ell'}$, such that $\forall x \in \mathbb{F}_2^{m \cdot \ell}$ it holds that $\mathsf{F}^{\mathsf{C}}(\mathsf{E}(\mathsf{S}(x))) \in \mathcal{C}$ and $\mathsf{F}(x) = \mathsf{U}(\mathsf{D}(\mathsf{F}^{\mathsf{C}}(\mathsf{E}(\mathsf{S}(x)))))$ (*functional correctness*).

3.3.3 Golden Circuit of Probabilistic Circuits

Accurately modeling CA requires a precise notion of fault propagation and misbehavior in a *shared circuit*. Unfortunately, this is a non-trivial task as, on the one hand, there are multiple valid Boolean sharings for one value, such that faults can be effective in the traditional meaning (i.e., have a different value than the fault-free circuit) but still be functionally correct, which is especially true for faults injected in generated randomness. On the other hand, if we have a faulty sharing, then it is hard to precisely determine the faulty shares, requiring a precise propagation of faults. As a consequence, we introduce an adjusted definition for the golden (fault-free) circuit of a shared circuit, where, if faulted, the randomness-generation gates are already replaced, as shown in Figure 1(d-f). Intuitively, this is correct as the actual value of randomness cannot have an impact on the functional correctness.

Definition 16 (Golden Circuit)**.** The *golden circuit* C_{golden} of a shared circuit C under a set of faults $\mathcal{F}$ is the circuit C transformed by the faults on generated randomness, i.e., $\{f \in \mathcal{F} \mid f \textit{ targets } g \in \mathcal{G}_{\mathsf{rand}}\}$.

In the following, we show that Definition 16 is meaningful and sound, since faults in randomness gates do not affect functional correctness and is at most equivalent to a probe for probing security of a circuit C.

Unaltered Randomness Distribution. We first start by showing that all faults in randomness gates that do not change the randomness distributions, e.g., bit-flips on uniformly drawn randomness, can be ignored without further impacting functional correctness, probing security, or simulation-based composability notions.

Theorem 1. *Faulting a gate $g \in \mathcal{G}_{\mathsf{rand}}$, generating some randomness r_g, has no effect on functional correctness, probing security, or simulation-based composability of C if the randomness distribution of r_g is not changed.*

Proof. Assume a circuit C with a gate $g \in \mathcal{G}_{\mathsf{rand}}$ generating a random bit r_g. Functional correctness of C cannot be affected by a fault in g as otherwise the functional correctness would always depend on the concrete value of r_g.

Further, assume there is a simulator S for the probing security or composability of C. The same simulator S can be used to simulate the respective probes with and without a fault injected in g for the following reasons. If the value at the output of g is observable from the selected probes, then the simulator must select some randomness to represent this value as otherwise the output distribution of the simulator differs from the distribution of the probes on the real circuit C. As the fault does not change the distribution of the output of g, the simulator can use the same randomness regardless of the injected fault in g. If the value at the output of g is not observable from the selected probes, then the fault does not influence the output distribution. □

Altered Randomness Distribution. Next, we show that faults injected in a randomness gate which alter the randomness distribution, e.g., set or reset faults, can be replaced by a probe without impacting functional correctness, probing security, or simulation-based composability notions. This further implies that propagation of faulty randomness does not affect probing security or composability (besides providing additional probes). For this we first prove a lemma claiming that a value with a biased distribution can not mask some other value, before we show the actual claim in Theorem 2.

Lemma 1. *Let $x, y \in \mathbb{F}_2$ where x has a biased distribution and the distributions of x and y are independent of each other. Further, let $\circ$ be a binary operator over $\mathbb{F}_2$. Then the distribution of $x \circ y$ and $y \circ x$ is dependent on the distribution of y.*

Proof. Let $x, y \in \mathbb{F}_2$ where x has a biased distribution, i.e., $Pr[x = 0] \neq Pr[x = 1]$, and the distributions of x and y are independent of each other. The operations in $\mathbb{F}_2$ are addition (xor) and multiplication (and). As both operations are commutative it is sufficient to look at $x \circ y$. We first consider addition and then multiplication.

Addition: It holds that $Pr[x+y = 0] = Pr[x = 0]Pr[y = 0] + Pr[x = 1]Pr[y = 1]$. This term can only be made independent of the distribution of y when $Pr[x = 0] = Pr[x = 1]$, however, as $Pr[x = 0] \neq Pr[x = 1]$ it holds that $Pr[x + y = 0]$ is dependent on the distribution of y. Further, $Pr[x + y = 1] = Pr[x = 0]Pr[y = 1] + Pr[x = 1]Pr[y = 0]$ is dependent on the distribution of y with the same argument.

Multiplication: It holds that $Pr[x \cdot y = 0] = Pr[x = 0]Pr[y = 0] + Pr[x = 0]Pr[y = 1] + Pr[x = 1]Pr[y = 0]$ and $Pr[x \cdot y = 1] = Pr[x = 1]Pr[y = 1]$. Both terms are always dependent on the distribution of y. □

Theorem 2. *Faulting a gate $g \in \mathcal{G}_{\mathsf{rand}}$, generating some randomness r_g, is equivalent to placing a probe on r_g with respect to functional correctness, probing security, or simulation-based composability of C, if the randomness distribution of r_g is changed.*

Proof. Assume a circuit C with a gate $g \in \mathcal{G}_{\mathsf{rand}}$ generating a random bit r_g. Again, functional correctness of C cannot be affected by a fault in g (for the same argument as before). Next, we prove equivalence in terms of probing security and simulation-based composability in both directions.

$\Rightarrow$ Assume there is a simulator S_0 capturing the probing security or composability of C when there is a fault injected in g causing a biased distribution of r_g. As the distribution of r_g is biased and independent of any input to C, r_g can not hide any input according to Lemma 1. Hence, there is a simulator S_1 with the same inputs as S_0 but with the additional output r_g, simulating an additional probe on r_g. The simulator S_1 is constructed in the same way as S_0, however, explicitly draws a value for r_g from the corresponding distribution if not already done by S_0. The correctness of S_1 follows from two facts: (i) for values dependent on the distribution of r_g the distribution already considered the biased distribution of r_g in S_0, and (ii) for values where the distribution of r_g is not required for simulation (either because they are functionally independent or there is a hiding later on) this is also true when leaking r_g, as r_g itself does not hide any values. With a similar argument, we can construct a simulator S_2 equivalent to S_1 but with a different distribution for r_g. Correctness of

S_2 follows from (i) simulation of values independent of r_g do not change by changing the distribution, and (ii) changes in the distribution of values dependent on r_g can be computed as the only difference to the distribution in S_1 is the changed and known distribution of r_g. In particular, we can chose an uniform random distribution for r_g. Now S_2 uses the same input shares as S_0 but simulates C with a probe placed on r_g instead of a fault injected in g.

$\Leftarrow$ Assume there is a simulator S'_0 capturing the probing security or composability of C when a probe is placed on r_g. Due to the probe on r_g the distribution of r_g gets fixed to $Pr[r_g = 0] = 1$ or $Pr[r_g = 1] = 1$ for each run of S'_0. Hence, the distribution of r_g is both independent to the distribution of all other inputs to C and biased for each individual invocation of S'_0. With Lemma 1 it follows that r_g does not hide any other value and, therefore, there exist a simulator S'_1 equivalent to S'_0 but with different distribution for r_g (same argument as the transition from S_1 to S_2 above). In particular, we can chose the distribution chosen by the adversary for a fault in g. Now, S'_1 uses the same input shares as S'_0 but simulates C with a fault injected in g instead of a probe placed on r_g.

This shows equivalence in terms of probing security or simulation-based composability by showing that we can always transform a simulator of one setting into a simulator of the other setting. □

Together, Theorems 1 and 2 show that faulting some randomness has no impact on correctness and affects security at most in equivalence to a probe. As probes (backwards) and fault (forwards) propagation are converse, fault and probing security is not impacted when defining the golden circuit in replacing/modifying randomness generating gates. Hence, our definition of the golden circuit is sound.

3.3.4 Composability Notions

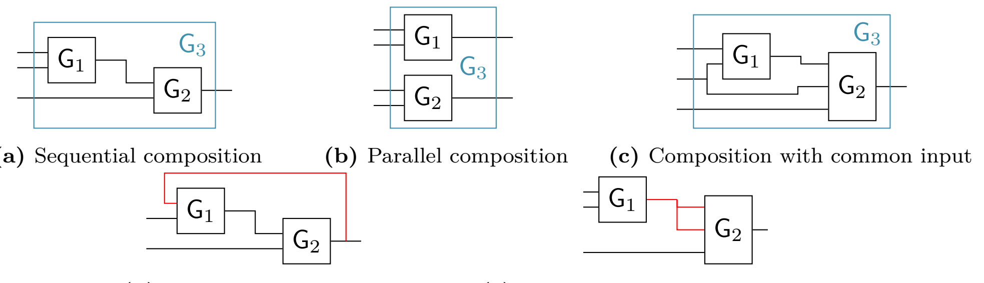
In the context of CA, it is again useful to define notions of secure composition to reduce the analysis complexity of gadgets. In this, we build upon the work of [DN20] by transferring their definitions to the glitch-extended probing model along with necessary refinements. To this end, we use Remark 1 to explicitly specify the information access of a simulator to mimic the view of the adversary. Similarly, we use Definition 16, to overcome the fact that faulting randomness can lead to more faults at the gadget output than allowed.

Definition 17 ((d, k)-Combined Non-Interference)**.** A gadget G is (d, k)-Combined Non-Interference (C-NI) if for any set of k_1 faulty inputs, k_2 faults injected on gates in G, and d' probes, such that $d' + k_1 + k_2 \leq d$ and $k_1 + k_2 \leq k$ the following holds:

Privacy: The probes and the abort signal can be simulated with $d' + k_2$ shares of each input and knowledge of the faults both injected and on inputs.

Correctness: The gadget either aborts or outputs a result with at most $k_1 + k_2$ bit faults compared to the golden circuit and there exists a decoding gadget G^D, such that given an input with at most k faulty inputs and an abort signal, G^D either aborts or outputs a correct result.

Specifically, C-NI is a combination of P-NI with F-NI, however, in contrast to the NINA definition in [DN20] we do not add k_1 to the number of allowed input shares for simulation, to clarify that faulty inputs must not reveal additional shares. Hence, we restrict the possible leakage through faulty inputs to the knowledge an adversary can have about those inputs according to Remark 1, i.e., the changed distribution under the faults. This ensures that a fault can only leak input shares locally and not by probe propagation from other gadgets. In addition, we refined the definition of NINA by giving a formal



(a) Sequential composition **(b)** Parallel composition **(c)** Composition with common input

(d) Composition with loop **(e)** Output of G_1 is used multiple times

Figure 2. Valid (a to c) and invalid (d and e) examples of gadget compositions according to Theorem 3 and Theorem 4. Example (d) is invalid as it contains a loop and (e) is invalid as the output of G_1 is used two times as input to G_2.

description of what a *correct* output is, specifically comparing the output of the gadget with the output of the golden circuit as defined in Definition 16. Those changes also apply to the other composablity notions, given below, in comparison with their counterpart from [DN20]. Please note, that knowledge about the faults according to Remark 1 is enough for simulation as a simulator requires only distributions and not concrete values.

Similar to the underlying definitions, C-NI is not sufficient for arbitrary compositions in the context of CA. Hence, Combined Strong Non-Interference (C-SNI) is defined as the combination of P-SNI and F-SNI (similar to SNINA in [DN20]).

Definition 18 ((d,k)-Combined Strong Non-Interference)**.** A gadget G is (d,k)-C-SNI iff for any set of k_1 faulty inputs, k_2 faults injected on gates in G, d_1 probes placed on intermediate values, and up to d_2 probes placed on shares of each output, such that $d_1 + d_2 + k_1 + k_2 \leq d$ and $k_1 + k_2 \leq k$, the following holds:

Privacy: The probes and the abort signal can be simulated with $d_1 + k_2$ shares of each input and knowledge of the faults both injected and on inputs.

Correctness: The gadget either aborts or outputs a result with at most k_2 bit faults compared to the golden circuit and there exists a decoding gadget G^D, such that given an input with at most k faulty inputs and an abort signal, G^D either aborts or outputs a correct result.

We now show that C-SNI gadgets, according to our refined definitions, still can be composed to larger circuits without degrading combined security. We give illustrative examples of valid and invalid compositions in Figure 2.

Theorem 3. *The composition of two (d,k)-C-SNI gadgets, that is loop-free and where no output of one gadget is used multiple times as input to the other gadget, is (d,k)-C-SNI.*

Proof. Let G_1 and G_2 be arbitrary (d,k)-C-SNI 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 assume no output of G_2 is connected to G_1 (as there is no loop). Further assume, no output of G_1 is connected to more than one input of G_2.

Let there be k_1 faulty inputs to G_3 of which k_1^1 go to G_1 and k_1^2 go to G_2. There may be some inputs shared between G_1 and G_2, however, it always holds that $k_1^1 \leq k_1$ and $k_1^2 \leq k_1$. 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. As the gadgets G_1 and G_2 are disjoint it holds that $k_2^1 + k_2^2 = k_2$. We chose k_1 and k_2 arbitrarily such that $k_1 + k_2 \leq k$. Similarly, let there be d_1 probes placed on intermediate values on G_3 of which d_1^1 are placed within G_1 and d_1^2 are placed within G_2. Further, let there be up to d_2 probes placed on shares of each output of G_3. We assume all probes placed on wires connecting G_1 and G_2 as output probes of G_3 if the wire is additionally connected to an output of G_3 and as internal probe to G_2 otherwise. Then it holds that $d_1^1 + d_1^2 = d_1$, as G_1 and G_2 are disjoint. We chose d_1 and d_2 such that $d_1 + d_2 + k_1 + k_2 \leq d$. 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 there are exactly k_1^1 faulty inputs and k_2^1 faults injected in gates of G_1. It holds that $k_1^1 + k_2^1 \leq k_1 + k_2 \leq k$. With C-SNI of G_1 it follows that G_1 either aborts or outputs a result with at most k_2^1 bit faults compared to the golden circuit of G_1.

Outputs of G_1 can be connected to inputs of G_2 and, hence, G_2 has at most $k_2^1 + k_1^2$ faulty inputs, and exactly k_2^2 faults injected in gates of G_2. It holds that $k_2^1 + k_1^2 + k_2^2 \leq k_1 + k_2 \leq k$. With C-SNI of G_2 it follows that G_2 either aborts or outputs a result with at most k_2^2 bit faults compared to the golden circuit of G_2.

Therefore, G_3 either aborts (if G_1 or G_2 aborts) or outputs a result with at most $k_2^1 + k_2^2 = k_2$ bit faults compared to the golden circuit of G_3. This concludes *correctness* of G_3.

Privacy: As all outputs of G_2 are also outputs of G_3 there are up to d_2 probes placed on shares of each output of G_2. There are also d_1^2 internal probes placed in G_2 as well as $k_1^2 + k_2^1$ faulty input and k_2^2 internal faults. It holds that $d_1^2 + d_2 + k_1^2 + k_2^1 + k_2^2 \leq d_1 + d_2 + k_1 + k_2 \leq d$. With C-SNI of G_2 it follows, that the abort signal and the probes related to G_2 can be simulated with $d_1^2 + k_2^2$ shares of each input and knowledge of the faults (Please note, that knowledge of the faulty inputs dependent on G_1 can be gained by propagating the corresponding faults through G_1). We call the corresponding simulator S_2.

The outputs of G_1 can be connected to the inputs of G_2. Therefore, the simulator for G_1 not only has to simulate the probes related to G_1 but also the input shares required for S_2. We do this by assuming additional probes at those input shares and, hence, there are up to $d_2 + d_1^2 + k_2^2$ probed shares of each output of G_1. In addition, there are d_1^1 internal probes, k_1^1 faulty inputs, and k_2^1 internal faults related to G_1. It holds that $d_1^1 + d_1^2 + d_2 + k_1^1 + k_2^1 + k_2^2 \leq d_1 + d_2 + k_1 + k_2 \leq d$. With C-SNI of G_1 it follows that the abort signal and the probes related to G_1 can be simulated with $d_1^1 + k_2^1$ shares of each input and knowledge of the faults. We call this simulator S_1.

By combining S_1 and S_2 we get a simulator for G_3 that requires at most $d_1^1 + k_2^1 + d_1^2 + k_2^2 = d_1 + k_2$ shares of each input (as an input can go to both G_1 and G_2) and knowledge of the faults (in the sense of Remark 1) to simulate the abort signal and all the probes. With this we have shown *privacy* of G_3 and conclude the proof. □

While C-SNI supports arbitrary composition, as shown by Theorem 3, the maximum number of allowed probes and faults is not independent. Independent Combined Strong Non-Interference (C-SNI$_{\text{ind}}$), however, is designed to allow independent security levels for probing and injection attacks, without losing the notion of composition (similar to SININA in [DN20]).

Definition 19 ((d, k)-Independent Combined Strong Non-Interference)**.** A gadget G is (d, k)-C-SNI$_{\text{ind}}$ if for any set of k_1 faulty inputs, k_2 faults injected on gates in G, d_1 probes placed on intermediate values, and up to d_2 probes placed on shares of each output, such that $d_1 + d_2 \leq d$ and $k_1 + k_2 \leq k$, the following holds:

Privacy:	The probes can be simulated with d_1 shares of each input and knowledge of the faults both injected and on inputs.
Correctness:	The gadget outputs a result with at most k_2 bit faults compared to the golden circuit.

Theorem 4. *The composition of two (d, k)-C-SNI$_{ind}$ gadgets, that is loop-free and where no output of one gadget is used multiple times as input to the other gadget, is (d, k)-C-SNI$_{ind}$.*

The proof of Theorem 4 follows the same structure as the proof of Theorem 3 with only marginal changes. For *correctness* it is exactly the same argumentation without an abort signal. For *privacy* we remove all k_i and k_i^j from the discussion.

Proof. Let G_1 and G_2 be arbitrary (d, k)-C-SNI$_{\text{ind}}$ 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 assume no output of G_2 is connected to G_1 (as there is no loop). Further assume, no output of G_1 is connected to more than one input of G_2.

Let there be k_1 faulty inputs to G_3 of which k_1^1 go to G_1 and k_1^2 go to G_2. There may be some inputs shared between G_1 and G_2, however, it always holds that $k_1^1 \leq k_1$ and $k_1^2 \leq k_1$. 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. As the gadgets G_1 and G_2 are disjoint it holds that $k_2^1 + k_2^2 = k_2$. We chose k_1 and k_2 arbitrarily such that $k_1 + k_2 \leq k$. Similarly, let there be d_1 probes placed on intermediate values on G_3 of which d_1^1 are placed within G_1 and d_1^2 are placed within G_2. Further, let there be up to d_2 probes placed on shares of each output of G_3. We assume all probes placed on wires connecting G_1 and G_2 as output probes of G_3 if the wire is additionally connected to an output of G_3 and as internal probe to G_2 otherwise. Then it holds that $d_1^1 + d_1^2 = d_1$, as G_1 and G_2 are disjoint. We chose d_1 and d_2 such that $d_1 + d_2 \leq d$. 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 there are exactly k_1^1 faulty inputs and k_2^1 faults injected in gates of G_1. It holds that $k_1^1 + k_2^1 \leq k_1 + k_2 \leq k$. With C-SNI$_{\text{ind}}$ of G_1 it follows that G_1 outputs a result with at most k_2^1 bit faults compared to the golden circuit of G_1.

Outputs of G_1 can be connected to inputs of G_2 and, hence, G_2 has at most $k_2^1 + k_1^2$ faulty inputs, and exactly k_2^2 faults injected in gates of G_2. It holds that $k_2^1 + k_1^2 + k_2^2 \leq k_1 + k_2 \leq k$. With C-SNI$_{\text{ind}}$ of G_2 it follows that G_2 either aborts or outputs a result with at most k_2^2 bit faults compared to the golden circuit of G_2.

Therefore, G_3 outputs a result with at most $k_2^1 + k_2^2 = k_2$ bit faults compared to the golden circuit of G_3. This concludes *correctness* of G_3.

Privacy: As all outputs of G_2 are also outputs of G_3 there are up to d_2 probes placed on shares of each output of G_2. There are also d_1^2 internal probes placed in G_2. It holds that $d_1^2 + d_2 \leq d_1 + d_2 \leq d$. With C-SNI$_{\text{ind}}$ of G_2 it follows, that the probes related to G_2 can be simulated with d_1^2 shares of each input and knowledge of the faults (Please note, that knowledge of the faulty inputs dependent on G_1 can be gained by propagating the corresponding faults through G_1). We call the corresponding simulator S_2.

The outputs of G_1 can be connected to the inputs of G_2. Therefore, the simulator for G_1 not only has to simulate the probes related to G_1 but also the input shares required for S_2. We do this by assuming additional probes at those input shares and, hence, there are up to $d_2 + d_1^2$ probed shares of each output of G_1. In addition, there are d_1^1 internal probes to G_1. It holds that $d_1^1 + d_1^2 + d_2 \leq d_1 + d_2 \leq d$. With C-SNI$_{\text{ind}}$ of G_1 it follows that the probes related to G_1 can be simulated with d_1^1 shares of each input and knowledge of the faults. We call this simulator S_1.

By combining S_1 and S_2 we get a simulator for G_3 that requires at most $d_1^1 + d_1^2 = d_1$ shares of each input (as an input can go to both G_1 and G_2) and knowledge of the faults (in the sense of Remark 1) to simulate the abort signal and all the probes. With this we have shown *privacy* of G_3 and conclude the proof. $\square$

4 Verification Concept

In this section, we first introduce an appropriate data structure to represent the circuit model (used for the symbolic simulation). Afterwards, we separately discuss our verification strategies to validate side-channel security and fault injection resistance. Eventually, we combine those strategies into a novel combined verification approach.

4.1 Circuit Model

Our verification approach relies on essential ideas of [KSM20] and [RBRSS+21], such that we also assume that the target logic circuit C is given as a (Verilog) gate-level netlist. The gate-level netlist is transformed into a DAG $\mathcal{D} = \{\mathcal{V}, \mathcal{E}\}$ providing a perfectly suited data structure to efficiently analyze the given design. More precisely, for symbolic simulation we adopt the application of Binary Decision Diagrams (BDDs) according to [KSM20] and [RBRSS+21]. Specifically, each node $d \in \mathcal{V}$ is associated with a BDD (based on its represented Boolean function). For more details, we refer the interested readers to [KSM20, RBRSS+21].

4.2 Side-Channel Analysis

The analysis and verification of resistance against SCA of a target circuit C closely follows the realization of SILVER originally presented in [KSM20]. SILVER is a formal verification framework that utilizes BDDs to verify the probing security and composability of digital logic circuits given as (Verilog) gate-level netlist. Due to the data structure of BDDs, the tool can efficiently check the statistical independence of two Boolean functions.

Hence, as already introduced in Section 3.1, this perfectly enables verification in the glitch-extended d-probing model. Additionally, for gadgets solely designed against a SCA attacker, we consider the P-NI and P-SNI composability notions as recapitulated in Definition 7 and Definition 8, respectively. Besides, our composability verification approach also covers the analysis of the recently introduced PINI notation [CS20], closely following the approach of [KSM20].

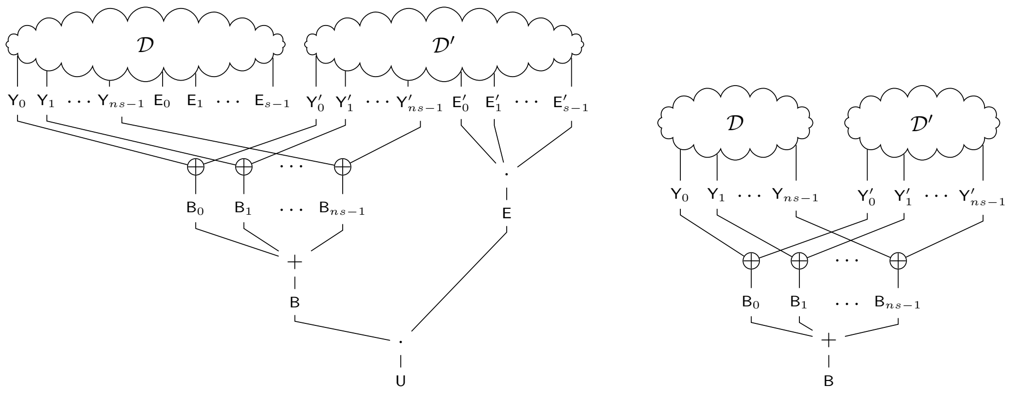
4.3 Fault Injection Analysis

Our verification for resistance against (stand-alone) FIA primarily adapts the concepts of [RBSG22, RBRSS+21]. The authors of [RBRSS+21] presented a formal verification framework called FIVER that verifies the resistance of digital logic circuits against fault injection attacks. Similar to SILVER, FIVER utilizes BDDs as underlying data structure to evaluate the Boolean functions of the circuits under test. Consequently, FIVER can evaluate countermeasures that are based on detection and correction schemes by incorporating the fault model presented in [RBSG22]. We adopt these features and further introduce extended verification methods in the following section. More specifically, we discuss fault diagnosis strategies to analyze detection or correction countermeasures (slightly adapted compared to [RBRSS+21] to cover shared circuits as well), approaches to evaluate resistance against statistical (ineffective) fault attacks, as well as composability of gadgets according to F-NI and F-SNI, as defined in Definition 12 and Definition 13, respectively.

(Share-wise) Detection. Even though the primary goal of this work is the verification of combined countermeasures, our framework still supports stand-alone FIA verification for detection-based countermeasures.

Given a faulty circuit model $\mathcal{D}'$, fault-free operation is indicated by $\mathsf{E} = \prod_{i=0}^{s-1} \mathsf{E}'_i$, assuming that each (share-wise) error flag $\mathsf{E}'_i = 1$ indicates a correct operation (see Figure 3a). Additionally, given the same input assignments, mismatching circuit outputs are derived as $\mathsf{B} = \sum_{i=0}^{ns-1} (\mathsf{Y}_i \oplus \mathsf{Y}'_i)$. Eventually, the number of effective faults, i.e., undetected faults visible at the circuit's output, is derived by counting satisfying assignments of $\mathsf{U} = \mathsf{E} \cdot \mathsf{B}$

(Share-wise) Correction. For pure correction-based countermeasures, usually coming without dedicated error detection flags, this procedure is simplified to only counting satisfying assignments of $\mathsf{B} = \sum_{i=0}^{ns-1} (\mathsf{Y}_i \oplus \mathsf{Y}'_i)$ (see Figure 3b).


$\mathcal{D}$
$\mathcal{D}'$
Y_0 Y_1 $\cdots$ Y_{ns-1} E_0 E_1 $\cdots$ E_{s-1}
Y'_0 Y'_1 $\cdots$ Y'_{ns-1} E'_0 E'_1 $\cdots$ E'_{s-1}
B_0 B_1 $\cdots$ B_{ns-1}
E
B
U
$\mathcal{D}$
$\mathcal{D}'$
Y_0 Y_1 $\cdots$ Y_{ns-1} Y'_0 Y'_1 $\cdots$ Y'_{ns-1}
B_0 B_1 $\cdots$ B_{ns-1}
B

(a) Detection-based. **(b)** Correction-based.

Figure 3. Evaluation strategies for detection- and correction-based countermeasures.

Statistical Ineffective Fault Analysis. SIFA exploits statistical dependencies between fault injections and input values that alter the output distribution of the correct unshared data [DEK+18, DEG+18, DDE+20]. According to Hadžić et *al.* [HPB21], absence of vulnerabilities against SIFA can be proven through statistical independence of detection values and processed secrets. In [HPB21], this is verified indirectly via *Boolean dependency analysis*, *factorization*, and *properties of masked computations*. However, as our framework naturally supports verification of statistical independence (due to the integration of analysis techniques provided by SILVER [KSM20]) and injections of faults in all available gates (features from FIVER [RBRSS+21]), we opted to implement direct verification of SIFA.

Consequently, the fundamental step of the verification procedure is the derivation of the fault detection values. While for detection-based countermeasures this is implicitly given as $\mathsf{E} = \prod_{i=0}^{s-1} \mathsf{E}'_i$, for correction-based countermeasures this has to be explicitly constructed according to $\mathsf{E} = \prod_{i=0}^{ns-1} \overline{(\mathsf{Y}_i \oplus \mathsf{Y}'_i)}$. Eventually, in both cases, we verify statistical independence of E and the processed secrets to reason about resistance against SIFA[2].

Composability Notions. For secure composition of detection-based or correction-based gadgets, we specifically verify the notions of F-NI (see Definition 12) and F-SNI (see Definition 13). For this, we ensure that at most k or $k_2 = k - k_1$ circuit outputs differ, i.e., $\mathsf{Y}_i \oplus \mathsf{Y}'_i = 1$, for all input assignments in case of F-NI or F-SNI, respectively.

For each verification that analyzes the F-NI and F-SNI notions, we additionally consider faults in primary gadget inputs and randomness gates $g \in \mathcal{G}_{\mathsf{rand}}$. This is independent of the location parameter l of the fault model $\zeta(f, t, l)$. Further, considering the effects of faulty randomness, as discussed in Theorem 1 and Theorem 2, we adjust the golden circuit model according to Definition 16 and proceed with the fault diagnosis as before. More precisely, a fault injection in a randomness gate $g \in \mathcal{G}_{\mathsf{rand}}$ could lead to more than k errors at the output of the target gadget G if the outputs are plainly compared to the fault-free gadget. We address these special cases in our verification process by tracking if a current fault injection includes faults in randomness gates. In case there is at least one randomness gate faulted, we apply the corresponding modification to the golden circuit model as well (i.e., altering the randomness gates in the golden circuit model according to the faulty circuit model), and compare the adapted golden circuit model with the faulty circuit model (cf. Figure 1d to 1f). This allows us to apply the same strategy of counting the satisfied BDDs B_i as explained above and compare the result to the threshold k or k_2 for F-NI or F-SNI, respectively. The influence of this procedure on the side-channel verification is discussed in the next section.

[2]For verification of Statistical Fault Attack (SFA), the same concepts can be applied.

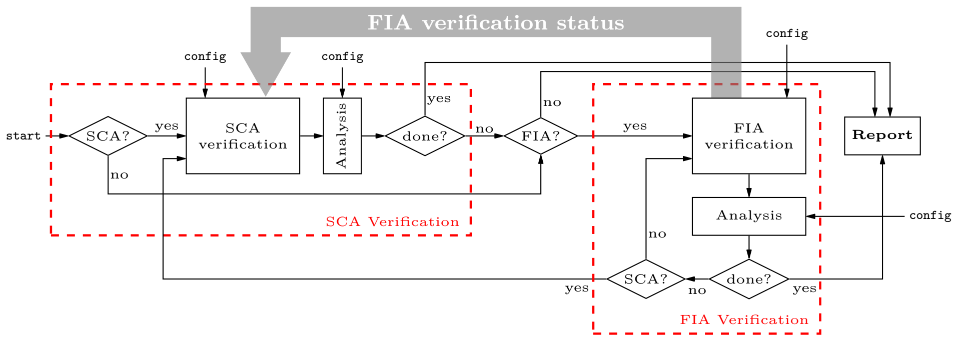


Figure 4. Concept of our combined verification approach.

4.4 Combined Verification

After the introduction of stand-alone side-channel verification and fault injection verification, we now introduce our approach to perform combined verification. For this, we again rely on the glitch-extended d-probing model, now combined with the $\zeta(f, t, l)$ fault model. More specifically, our combined framework enables to verify (d, k, t, l)-combined security as well as secure composition under the C-NI, C-SNI, and C-SNI$_{\text{ind}}$ notions. Figure 4 visualizes the general verification concept, allowing to perform (stand-alone) SCA and FIA, as well as CA verification.

(d, k)-Combined Security Verification. Combined security verification always starts with verification of glitch-extended d-probing security, ensuring the secure implementation of the countermeasure in the absence of any faults. Afterwards, symbolic fault injection, fault diagnosis, and passive security verification is performed in an incremental *active-then-passive* approach, i.e., continuously increasing number of faults while in turn performing fault diagnosis and SCA verification on the faulty circuit model. Hence, combined verification requires that the statistical independence checks for the SCA verification can also be checked on the faulty circuit model and not just on the golden circuit model. However, in a purely combined security verification scenario, the current state of the fault injection verification does not influence the checks for the probing security. More precisely, both verification techniques are applied independently and the number of faults does not reduce to number of probes.

Composability Verification. Verification of combined composability notions, i.e., C-NI, C-SNI, and C-SNI$_{\text{ind}}$, again starts with verification in the absence of faults, hence is reduced to verification of P-NI or P-SNI. In the presence of faults, verification of (d, k)-C-NI and (d, k)-C-SNI requires the reduction of available adversarial probes to $d' = d - k_1 - k_2$ as introduced in Section 3.2. Additionally, the attacker is allowed to learn $d - k + k_1 = d - k_2$ shares, given k_1 input faults, assuming that input faults provide additional probes. This interaction between the FIA verification and the SCA verification is highlighted in Figure 4 by the gray arrow and is very important for the *combined verification* of *combined composability notions.* Next, our framework checks P-NI and F-NI or P-SNI and F-SNI (under these modifications) for C-NI or C-SNI verification, respectively. As discussed above, we apply Definition 16 (i.e., adapt the golden circuit model in case randomness gates are faulted) in order to verify the F-NI and F-SNI notions. Note, this modification ensures that the detection or correction capabilities of the gadget under test are not exceeded. However, we do not automatically assume that such a fault does not influence the side-channel security. This is separately verified by the SCA verification and therefore ensures that faults in randomness gates do not violate the SCA security assumptions. Eventually, for verification of C-SNI$_{\text{ind}}$, the F-SNI and P-SNI properties are verified independently without any modifications on probes and shares.

Optimizations. Adopting reduction strategies from [RBRSS$^+$21], we reduce the combined verification complexity through *incremental probing-security verification*. Particularly, we only consider altered probe combinations, i.e., probing the fault propagation path, during combined verification. In more detail, we compute all probe combinations that include at least one gate that was either altered by the fault injection or lies in one of the propagation paths of the altered gates. Hence, all probe combinations that can solely be created by gates that were not effected by the preceding fault injection can be neglected in the probing verification process leading to an increased verification performance.

Soundness of our Verification Approach. In this paragraph we briefly discuss our verification approach and its soundness. All verification techniques utilized for our strategies are based on the d-probing model [ISW03] and the fault model presented in [RBSG22]. Hence, our verification can at most be as precise as the abstractions made by these two models.

Nevertheless, we verify a circuit under test against these models in an *active-then-passive* approach as explained above. Since we always verify stand-alone SCA security on a fault-free model first, following an active-then-passive approach can be justified by Theorem 5.

Theorem 5. *Let C be an arbitrary circuit and G^{D} 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 corrected result. Then C is combined secure iff the* active-then-passive *verification approach with preceding* passive *verification of $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$, such that no fault or probe targets G^{D}, does not find an attack.*

Proof. Let C be an arbitrary circuit and G^{D} 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 corrected result.

- $\Rightarrow$ Assume C is combined secure. From *privacy* of C follows that for all sets of up to k faults injected on gates in C together with all sets of up to d probes placed on wires of C the (optional) abort signal and the probes can be simulated without access to any wire of C. Hence, for all sets of faults, the abort signal and all sets of probes are statistical independent from any wire of C and in particular from any sensitive input [KSM20]. As this includes statistical independence of the abort signal and all probes for the empty fault set the initial passive verification does not find any attack. In addition, the statistical independence holds for all non-empty fault sets, i.e., all manipulations of C with up to k faults. This ensures that the active-then-passive verification step does not find any attack. As a result, all passive verification steps find no attack. Now, from *correctness* of C follows that for all sets of up to k faults the concatenation of $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$ either aborts or outputs a result equal to the golden circuit of C. Hence, no active verification step finds an attack.
- $\Leftarrow$ Assume the active-then-passive verification approach with preceding passive verification of $\mathsf{G}^{\mathsf{D}}(\mathsf{C}(\cdot))$ does not find any attack when no fault or probe targets G^{D}. From the initial passive verification it follows that the (potential) abort signal and all sets of up to d probes are statistical independent of any input to C. Hence, the abort signal and probes can be simulated without accessing any wire of C [KSM20]. In addition, as the active-then-passive verification approach does not find any attack, we have statistical independence of the abort signal and probes from the inputs for all sets of up to k faults, by enumerating all possible sets of faults and manipulating

C accordingly. Again, this means the abort signal can be simulated without accessing any wire of C [KSM20]. Together, this proves privacy of C. Correctness of C follows from the enumeration of all sets of up to k faults and the fact that the corresponding active verification does not find any attack.

□

With a similar argument we can prove the soundness of the verification approaches for C-NI, C-SNI, and C-SNI$_{\text{ind}}$. However, in those proofs we allow certain dependencies to inputs, where the number of dependent inputs is also determined by the set of injected faults (in accordance with the respective definitions).

5 Evaluations and Experiments

In this section, we present verification results for various case studies using our proposed framework VERICA. For this, we first analyze combined gadgets proposed in [DN20]. The gadgets were originally designed to fulfill the combined security notions NINA, SNINA, and SINIA [DN20] which are the foundation of our security notions presented in Section 3.3. Hence, an analysis with VERICA is interesting and important for practical designs since these gadgets could be used to construct combined protected cryptographic primitives and entire circuits.

Next, we demonstrate that our framework is able to validate protection against SIFA checking directly the independence of the secrets and the error detection flag as introduced in Section 4.3. This strategy is not supported by FIVER [RBRSS+21] and is an additional feature provided by VERICA.

Eventually, VERICA verifies a 4-bit S-box protected by the ParTI scheme which is one of the first countermeasures providing protection against SCA and FIA independently. Even though ParTI does not claim protection against *combined attacks*, we use the scheme to demonstrate that VERICA is able to check (d, k)-combined security (cf. Section 4.4) and that the mere combination of SCA and FIA countermeasures does not automatically result in combined security.

5.1 Verification of Gadgets against Combined Analysis

The first set of case studies is extracted from the gadget descriptions provided in [DN20]. The algorithms are originally designed for software implementations such that we added register stages at critical locations to stop glitches. Gruber et *al.* [GPK+21] presented a similar approach where they combined Domain-Oriented Masking (DOM) gates with repetition codes. However, their work does not target the protection of gadgets but rather the protection of an entire cipher.

Designs. Before we present and discuss evaluation results, we briefly summarize and explain the different gadget variants and their design and security properties. Here, we adapt the naming NINA, SNINA, and SININA from [DN20] to describe the different gadget types.

(d, k)-NINA: NINA corresponds to our C-NI security definition from Definition 17. We can construct a (d, k)-NINA gadget by implementing a shared xor gate which is d-order secure with respect to the threshold d-probing model. In order to fulfill the F-NI security notion, the shared xor gate is replicated k times such that no additional detection or correction mechanisms are required.

Table 2. Combined verification results for different gadget variants according to [DN20].

Gadget	Design					SCA			FIA			Combined		
	d	k	rand.	comb.	memory	P-NI	P-SNI	Time	F-NI	F-SNI	Time		(d,k)	Time
NINA	1	1	0	4	0	1✓	–	0.460 s	1✓	–	0.429 s	C-NI	(1, 1)✓	0.430 s
NINA	1	2	0	6	0	1✓	–	0.455 s	2✓	–	0.445 s	C-NI	(1, 2)✓	0.492 s
NINA	2	1	0	6	0	2✓	–	0.471 s	1✓	–	0.451 s	C-NI	(2,1)✓	0.436 s
NINA	2	2	0	9	0	2✓	–	0.442 s	2✓	–	0.444 s	C-NI	(2,2)✓	0.442 s
SNINA	1	1	1	22	16	–	1✓	0.476 s	–	1✓	0.449 s	C-SNI	(1,1)✓	0.473 s
SNINA	1	2	1	38	26	–	1✓	0.451 s	–	2✓	0.500 s	C-SNI	(1, 2)✓	0.519 s
SNINA	2	1	3	57	33	–	2✓	0.566 s	–	1✓	0.456 s	C-SNI	(2,1)✗/(1,1)✓	0.592 s
SNINA	2	2	3	96	54	–	2✓	0.821 s	–	2✓	0.673 s	C-SNI	(2,2)✗/(1,1)✓	1.062 s
SININA	1	1	2	90	30	–	1✓	0.450 s	–	1✓	0.461 s	C-SNI$_{\text{ind}}$	(1,1)✗/(0,0)✓	0.456 s
SININA	1	2	3	360	50	–	1✓	0.555 s	–	2✓	1.395 s	C-SNI$_{\text{ind}}$	(1,2)✗/(0,0)✓	17.985 s
SININA	2	1	6	207	63	–	2✓	1.334 s	–	1✓	0.511 s	C-SNI$_{\text{ind}}$	(2,1)✗/(0,0)✓	73.574 s
SININA*	2	2	9	825	105	–	2✓	76.030 s	–	2✓	5.300 s	C-SNI$_{\text{ind}}$	(2,2)✗/(0,0)✓	>2.7 h

* Due to the high verification complexity, we interrupted the combined analysis after testing (2, 1)-C-SNI$_{\text{ind}}$ where VERICA already reported a failure.

(d,k)-SNINA. [DN20, Algorithm 2] presents a detection-based design for the protected multiplication of duplicated shared values. As mentioned above, we opted to implement the gadgets in hardware while adding necessary registers on intermediate multiplication results (i.e., $u_{i,j,l}$) to ensure the security in the glitch-extended d-probing model.

(d,k)-SININA. [DN20, Algorithm 5] constructs a protected multiplication gadget for duplicated shared values, relying on error correction instead of error detection. Again, we opted to implement Algorithm 5 in hardware and inserted additional registers where necessary to stop propagation of glitches.

All the different gadget variants, implemented in Verilog, are provided on GitHub and have been synthesized using Synopsys Design Compiler using a subset of cells in the NanGate 45 nm Open Cell Library (OCL). In addition, each detection-based gadget has been modified to provide a separate error detection flag per output share instead of returning a null-output, as suggested in [DN20].

Verification. Combined verification of the gadgets is performed for an adversary who is able to precisely inject (at most) two independent set/reset faults into arbitrary gates of the circuits. As noted in [IPSW06], this fault model is more powerful for CA than the commonly employed bit-flip fault model. More specifically, each fault in the set/reset model can be modeled as an additional probe since an adversary is able to precisely inject values into the circuit, hence learning information on the processed data. In contrast to this, the commonly-used bit-flip model certainly maximizes the number of effective faults, however, an injected fault does not reveal information on processed data of the circuit and, hence, cannot be modeled as additional probe. Further, for CA, each gadget instance has been analyzed with respect to composability under the stand-alone P-NI, P-SNI, F-NI, and F-SNI security notions, first. Afterwards, we verify combined composability notions, i.e., C-NI, C-SNI, or C-SNI$_{\text{ind}}$.

Results. All verification results provided in Table 2 were generated under a 64-bit Linux Operating System (OS) environment on an Intel Xeon E5-1660v4 CPU with 16 cores, a clock frequency of 3.20 GHz, and 128 GB of RAM. More precisely, each gadget variant has been instantiated for all combinations of $d \in \{1,2\}$ and $k \in \{1,2\}$.

Starting with the combined analysis of the proposed NINA gadgets, Table 2 reports the expected security under the C-NI notion for all four gadgets. Note, however, according to Definition 17, the number of fault injections is limited by the side-channel security order d. Nevertheless, we can construct gadgets achieving a higher protection against faults (by

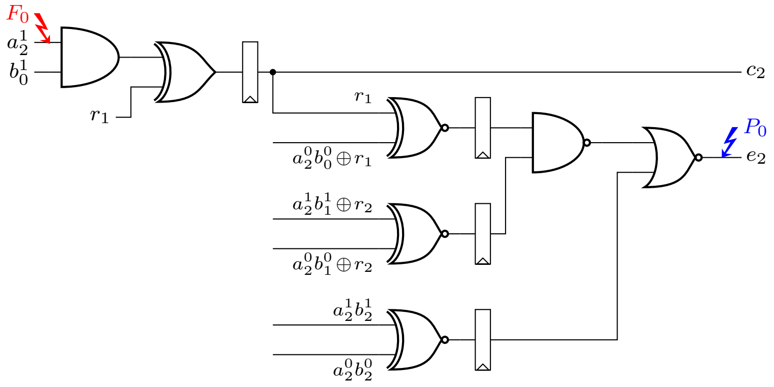


Figure 5. Combined attack for the $(2, 1)$-SNINA gadget proposed in [DN20] considering a set/reset fault F_0.

introducing more duplications) than against probes as shown for the $(1, 2)$-NINA gadget in Table 2. Hence, once the number of fault injections is equal or greater d, the gadget does not provide any protection against SCA but is still protected against fault injections. VERICA can handle these cases and therefore verifies $(1, 2)$-C-NI security.

The analysis of the SNINA gadgets shows that the $(1, 1)$ and $(1, 2)$ gadgets are secure under the C-SNI notion. However, the remaining two gadgets are vulnerable against combined attacks. For example, the definition of the $(2, 1)$-SNINA gadget allows to inject one single-bit fault while providing probing security up to the first order. However, injecting one precise fault at the input of this gadget leads to information leakage at the corresponding error flag. This is visualized in Figure 5 for the detection path of output c_2. It is assumed that the attacker injects a set/reset fault at input a_2^1 which is the third share of a belonging to the second duplicate of the and gate. Without any loss of generality, we assume the attacker injects a set fault which leads to a propagation of the random input r_1 to gate g_2. The second input of g_2 provides the same multiplication result but from the second instantiation. Hence, the randomness r_1 is canceled out in g_2 and only $a_2 b_0$ is sampled in the subsequent register. Since all other data paths are fault-free, the remaining registers only store a logical 0 which eventual leads to leakage of the shares a_2 and b_0 at the output of the gadget which violates the P-SNI property. Note, that this phenomena is due to the difficulties to implement appropriate error detection signals in hardware (it is not caused by flaws in the definitions of [DN20, Algorithm 2] and does not occur in software implementations). Since the same data is processed in the different duplications and this data is merged in the detection paths, a violation of the C-SNI property is expected and hard to prevent.

Eventually, all SININA gadgets are insecure under the C-SNI$_{\text{ind}}$ security notions. Figure 6 shows a schematic of a $(1, 1)$-SININA gadget as suggested in [DN20] with the required modifications to secure it against hardware glitches. By Definition 19, a (d, k) gadget should be secure even if an attacker injects up to k faults and uses d probes. However, in case an attacker injects a fault F_0 in one of the registers containing the partial products that were refreshed in the multiplication module (the most left modules in Figure 6), the probe P_0 can be used to observe the corresponding output after the compression step (xor gates in Figure 6). It can clearly be seen that the attacker is able to learn one share of a and one share of b which violates the P-SNI property (cf. Definition 8). Hence, simultaneous protection against side-channel attacks and fault injection attacks is not guaranteed. Even though we slightly adapted the C-SNI$_{\text{ind}}$ definition and the gadget implementation (i.e., we explicitly consider faults in randomness gates in the security notion and add register to the hardware implementation) compared to the original proposals from [DN20], the same flaws occur and can be transferred to the original work (i.e., the

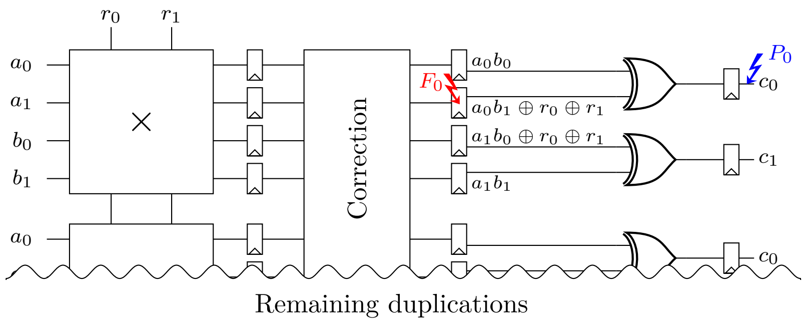


Figure 6. Combined attack for the $(1,1)$-SININA gadget proposed in [DN20] considering a set/reset fault F_0.

flaws also occur for software implementations). Hence, using VERICA, we were able to detect flaws in [DN20, Algorithm 5] which does not provide security against combined attacks.

Note, VERICA is not able to finish the verification of the $(2,2)$-SININA gadget since the design is complex to be analyzed by the proposed algorithms. However, within 2.7 h VERICA already failed checking the properties of $(2,1)$-C-SNI$_{\text{ind}}$, such that $(2,2)$-C-SNI$_{\text{ind}}$ cannot be fulfilled.

5.2 SIFA Constructions

While most countermeasures against SIFA are based on correction mechanisms [SJR+20, GPK+21], Daemen et *al.* proposed to use incomplete sub-circuits to avoid that effective faults become ineffective [DDE+20]. More precisely, the protected circuits are constructed from three basic circuits, i.e., a Toffoli gate $p_T(a,b,c) \mapsto \{a \oplus b \odot c, b, c\}$ [Tof80], a modified Toffoli gate $p_\chi(a,b,c) \mapsto \{\bar{a} \oplus b \odot c, b, c\}$, and a simple xor-gate.

To achieve the desired security, the basic circuits are masked and used as building blocks to construct cryptographic primitives. The masked circuits are given by

$$p_{TS}(a_0,a_1,b_0,b_1,c_0,c_1) \mapsto \{a_0 \oplus (b_0 \cdot c_1) \oplus (b_0 \cdot c_0), a_1 \oplus (b_1 \cdot c_1) \oplus (b_1 \cdot c_0), b_0, b_1, c_0, c_1\}$$
$$p_{\chi S}(a_0,a_1,b_0,b_1,c_0,c_1) \mapsto \{a_0 \oplus (\overline{b_0} \cdot c_1) \oplus (\overline{b_0} \cdot c_0), a_1 \oplus (b_1 \cdot c_1) \oplus (b_1 \cdot c_0), b_0, b_1, c_0, c_1\}$$

while necessary registers are added to achieve glitch-extended probing security. In the following, we first analyze these building blocks, before proceeding with the 3-bit permutation in Xoodoo [DHAK18], the 5-bit S-box in Keccak [BDPA13], and the AES S-box presented in [HPB21]. This case study should demonstrate the functionality of the SIFA extension introduced in Section 4.3. As discussed above, this strategy was not implemented in FIVER [RBRSS+21]. Due to the combination of statistical independence checking (as presented in SILVER) and fault injection (as presented in FIVER), the verification support of SIFA-based countermeasures can be realized by checking the statistical independence of the secrets and the error detection flag.

Masked Toffoli Gates. The masked Toffoli gate p_{TS} consists of four simple Toffoli gates that process the two shares of the masked input data. We first verify the security against side-channel attacks in the glitch extended d-probing model and no fault injections, i.e., $\zeta(0,\tau_{sr},cm)$. As expected and shown in Table 3, the design is secure against first-order side-channel attacks. Next, we perform an analysis with enabled fault injections using $\zeta(1,\tau_{sr},cm)$ as fault model and the SIFA strategy presented in Section 4.3. As claimed by the authors, the fault detection value is independent of the secret input values, i.e., the unshared input data $a = a_0 \oplus a_1$, $b = b_0 \oplus b_1$, and $c = c_0 \oplus c_1$. Hence, the design is secure against single-bit SIFA attacks. Interestingly, the design is still first-order secure

Table 3. Verification results for designs based on Toffoli gates [DDE+20, HPB21].

Implementation	**Design**		$\zeta(0, \tau_{sr}, cm)$		$\zeta(1, \tau_{sr}, cm)$		$\zeta(2, \tau_{sr}, cm)$	
	comb.	mem.	SIFA	Prob.	SIFA	Prob.	SIFA	Prob.
p_{TS}	8	6	–	$1^{✓}$[0.47 s]	$1^{✓}$[0.45 s]	$1^{✓}$[0.45 s]	$1^{✗}$[0.46 s]	$1^{✓}$[0.44 s]
$p_{\chi S}$	10	6	–	$1^{✓}$[0.45 s]	$1^{✓}$[0.44 s]	$1^{✓}$[0.45 s]	$1^{✗}$[0.46 s]	$1^{✓}$[0.45 s]
χ_3	30	30	–	$1^{✓}$[0.43 s]	$1^{✓}$[0.46 s]	$0^{✗}$[0.46 s]	$1^{✗}$[0.46 s]	$0^{✗}$[0.49 s]
χ_5	52	42	–	$1^{✓}$[0.44 s]	$1^{✓}$[0.48 s]	$0^{✗}$[0.44 s]	$1^{✗}$[0.48 s]	$0^{✗}$[0.54 s]
AES S-box, g_{104} [HPB21]	631	0	–	$0^{✗}$[13.80 s]	$0^{✗}$[194.89 s]	$0^{✗}$[191.93 s]	[∞]	[∞]
AES S-box, full [HPB21]	634	0	–	$0^{✗}$[13.90 s]	$1^{✓}$[194.58 s]	$0^{✗}$[194.70 s]	[∞]	[∞]

in the glitch extended d-probing model even in the presence of single-bit faults. Note, we do not fault input gates in this experiment such that the attacker cannot gain any additional information from the injected fault, as faults on gates do not provide additional information to the attacker. Eventually, we also evaluate the masked Toffoli gate under the fault model $\zeta(2, \tau_{sr}, cm)$. The design is still first-order secure (with the same argument as above), however, the tool reports only SIFA security under single-bit faults (which is expected).

We performed the same experiment for the adapted Toffoli gate $p_{\chi S}$. The verification results are similar to the results for p_{TS}.

Xoodoo 3-bit S-box. VERICA confirms the first-order probing security of the 3-bit S-box used on Xoodoo, implemented with the masked Toffoli gates p_{TS} and $p_{\chi S}$. However, as the countermeasure was not designed to provide $(1, 1)$-combined security, single-bit faults lead to successful first-order probing attacks.

Keccak 5-bit S-box. Similarly, the 5-bit S-box used in Keccak, again implemented with Toffoli gates, is probing secure in the glitch-extended d-probing model, but does not provide (d, k)-combined security.

AES S-box. Eventually, we examine two different implementations of the AES S-box presented in [HPB21][3] which is also build from Toffoli gates. The first implementation is automatically optimized to reuse the intermediate results g_{104} in order to demonstrate a flaw in the SIFA protection [HPB21], which is confirmed by VERICA. In contrast to this, the (non-optimized) full AES S-box design in [HPB21] is secure against single-bit SIFA attacks. Interestingly, both designs are not secure in the glitch extended d-probing model, since no register stages have been added to stop potential glitches.

Unfortunately, due to the increasing complexity and number of fault combinations, VERICA is not able to analyze the S-boxes for two simultaneous injected faults.

5.3 ParTI Verification

In 2016, ParTI presented a first-order secure TI with linear Error-Correcting Codes (ECCs) to provide protection against SCA and FIA [SMG16]. As a case study, the authors applied their approach to the lightweight cipher LED [GPPR11]. Even though the design is not secure against combined attacks, we implemented the scheme to demonstrate that VERICA is able to check (d, k)-combined security (cf. Section 4.4).

Designs. In this work, we implement a first-order secure LED S-box based on the TI scheme. We create two designs that are additionally protected against FIA by applying ECCs. The first design uses the error detection capabilities of the $[8, 4, 4]$-code as was

[3]The source files are publicly available at `https://extgit.iaik.tugraz.at/scos/danira`

Table 4. Verification results for an LED S-box TI with error detection or correction [SMG16].

Implementation	**Design**		$\zeta(0, \tau_{sr}, cm)$		$\zeta(1, \tau_{sr}, cm)$	
	comb.	memory	Det./Corr.	Prob.	Det./Corr.	Prob.
ParTI S-box (Detection)	678	78	–	1✓[0.866 s]	1✓[1.010 s]	0✗[1.950 s]
ParTI S-box (Correction)	2063	72	–	1✓[4.103 s]	1✓[3.677 s]	0✗[336.239 s]

done for ParTI. Hence, this design is expected to be secure against first-order SCA and up to 3-bit faults should be detectable (Hamming distance of the code is four). In the second design, we utilized the error correcting capabilities of the linear code such that the implementation is able to correct single-bit faults (and, again, secure against first-order SCA without fault injections). For both implementations, we satisfy the *independent property* introduced in [AMR+20, SRM20]. In the following, we present the verification results provided by VERICA.

Verification. Table 4 shows the verification results for both S-boxes. As expected, the first design provides the claimed probing security (during fault-free operation). Similarly, all faults in the $\zeta(1, \tau_{sr}, cm)$ model are detected by the linear ECC. As expected, the design is not $(1, 1)$-combined secure which was also not claimed in [SMG16] For verification of correction capabilities, we excluded the final correction stage from the analysis since injected faults cannot be detected or corrected there. Nevertheless, the verification results are similar to the design that only uses detection. However, due to the increased number of gates, the verification complexity increases significantly.

6 Conclusion

The main contribution of this work is twofold. First, we revise and extend the state-of-the-art formal modeling of CA which leads to the refinement of composability notions of hardware gadgets that are protected against SCA and FIA. Second, we introduce and present the first formal verification framework that can validate side-channel security and fault injection resistance as well as the protection against combined attacks. We demonstrate the functionality and advantages of VERICA in three extensive case studies where we analyze combined gadgets, protection mechanisms against SIFA based on Toffoli gates, and entire S-box implementations according to the ParTI protection scheme.

Limitations. However, even though VERICA can assist the designer in creating secure hardware implementations of cryptographic primitives, it has some limitations. As shown in our case studies, CA becomes more difficult with increasing number of gates in the design under test. This is naturally expected since the number of valid fault injections drastically increases (especially for multi-bit fault injections) while for each valid fault injection a separate verification of the side-channel security is conducted (for which the complexity also increases with the number of gates and security order). Note, even more powerful computers (i.e., using more cores and more memory) could not analyze these large circuits since the problem gets too complex. More precisely, the number of valid fault combinations and valid probe combinations increases exponentially with the number of gates and the corresponding order (i.e., with the number of simultaneous injected faults and the probing threshold, respectively).

Correctness of VERICA VERICA relies on the theoretical foundation of the security notions and their corresponding proofs presented in Section 3. We transferred these notions to software and verified them by (smaller) hand-verified examples. These hand-verified examples are further used in test strategies to ensure the correct functionality of different methods used in VERICA. However, we cannot fully guarantee the correctness of our source code due to the huge size of the project. Therefore, we additionally rely on the scrutiny of the community by releasing the source code and results of the case studies to ensure correct functionality and implementations.

Future Work. In our case studies we reveal some security flaws in the hardware implementations of combined gadgets. Hence, an interesting question is how these gadgets need to be adapted and implemented on hardware such that they resist combined attacks.

Furthermore, in this work we concentrated our investigations on composability notions based on the P-NI/P-SNI and F-NI/F-SNI security notions. We assume that the composability notions of C-NI, C-SNI, and C-SNI$_{\text{ind}}$ can be extended to a combined composability notion that includes the ideas of the PINI security notion.

Eventually, VERICA reports that the shared implementations of the Toffoli gates from Section 5.2 are even first-order secure against SCA in the presence of single-bit and two-bit faults. However, using the shared Toffoli gates to construct larger circuits (e.g., S-boxes), leads to implementations that are not protected against SCA in the presence of faults. This observation could be used in the future to formulate composability notions for gadgets that are protected against SIFA.

Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the project VE-HEP (16KIS1345) supported by the German Federal Ministry of Education and Research BMBF.

References

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private Circuits: A Modular Approach. In *CRYPTO*, pages 427–455, 2018.

[AMR$^+$20] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. Impeccable Circuits. *IEEE Trans. Computers*, 69(3):361–376, 2020.

[ANR18] Victor Arribas, Svetla Nikova, and Vincent Rijmen. VerMI: Verification Tool for Masked Implementations. In *ICECS*, pages 381–384. IEEE, 2018.

[AWMN20] Victor Arribas, Felix Wegener, Amir Moradi, and Svetla Nikova. Cryptographic Fault Diagnosis using VerFi. In *HOST 2020*, pages 229–240. IEEE, 2020.

[BBC$^+$19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.

[BBD$^+$15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT*, pages 457–485, 2015.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *SIGSAC*, pages 116–129, 2016.

[BDM$^+$20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.

[BGI$^+$18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In *EUROCRYPT*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *ASIACRYPT*, pages 343–372, 2018.

[BK18] Raik Brinkmann and Dave Kelf. *Formal system verification*, chapter Formal Verification - The Industrial Perspective, pages 155–182. Springer, 2018.

[BMRT21] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. IronMask: Versatile Verification of Masking Security. *IACR Cryptol. ePrint Arch.*, 2021.

[Can01] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*, pages 136–145, 2001.

[CCGR99] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A new symbolic model verifier. In *International conference on computer aided verification*, pages 495–499. Springer, 1999.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[Cla07] Christophe Clavier. Secret External Encodings Do Not Prevent Transient Fault Analysis. In *CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[DDE$^+$20] Joan Daemen, Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Florian Mendel, and Robert Primas. Protecting against Statistical Ineffective Fault Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):508–543, 2020.

[DDRT12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES. In *FDTC 2012*, pages 7–15. IEEE Computer Society, 2012.

[DEG$^+$18] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures. In *ASIACRYPT*, volume 11273 of *Lecture Notes in Computer Science*, pages 315–342. Springer, 2018.

[DEK$^+$18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.

[DHAK18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of Xoodoo and Xoofff. *IACR Trans. Symmetric Cryptol.*, 2018(4):1–38, 2018.

[DLM19] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine. Electromagnetic Fault Injection : How Faults Occur. In *FDTC 2019*, pages 9–16. IEEE, 2019.

[DN20] Siemen Dhooghe and Svetla Nikova. My Gadget Just Cares for Me - How NINA Can Prove Security Against Combined Attacks. In *CT-RSA*, volume 12006 of *Lecture Notes in Computer Science*, pages 35–55. Springer, 2020.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[GHP$^+$21] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. In Michael Bailey and Rachel Greenstadt, editors, *USENIX*, pages 1469–1468. USENIX Association, 2021.

[GLH18] Tomás Grimm, Djones Lettnin, and Michael Hübner. A survey on formal verification techniques for safety-critical systems-on-chip. *Electronics*, 7(6):81, 2018.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[GPK$^+$21] Michael Gruber, Matthias Probst, Patrick Karl, Thomas Schamberger, Lars Tebelmann, Michael Tempelmeier, and Georg Sigl. DOMREP-An Orthogonal Countermeasure for Arbitrary Order Side-Channel and Fault Attack Protection. *IEEE Trans. Inf. Forensics Secur.*, 16:4321–4335, 2021.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[HB21] Vedad Hadzic and Roderick Bloem. COCOALMA: A Versatile Masking Verifier. In *FMCAD*, pages 1–10. IEEE, 2021.

[HPB21] Vedad Hadzic, Robert Primas, and Roderick Bloem. Proving SIFA protection of masked redundant circuits. In *Automated Technology for Verification and Analysis*, volume 12971 of *Lecture Notes in Computer Science*, pages 249–265. Springer, 2021.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In *EUROCRYPT*, volume 4004 of *LNCS*, pages 308–327. Springer, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KR11] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion.* Information Security and Cryptography. Springer, 2011.

[KRH17] Punit Khanna, Chester Rebeiro, and Aritra Hazra. XFC: A Framework for eXploitable Fault Characterization in block ciphers. In *DAC 2017*, pages 8:1–8:6. ACM, 2017.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

[MAN$^+$19] Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. M&M: Masks and Macs against Physical Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):25–50, 2019.

[Mau11] Ueli Maurer. Constructive Cryptography - A New Paradigm for Security Definitions and Proofs. In *TOSCA*, pages 33–56, 2011.

[RBRSS$^+$21] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. FIVER – Robust Verification of Countermeasures against Fault Injections. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):447–473, Aug. 2021.

[RBSG22] Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. Revisiting Fault Adversary Models – Hardware Faults in Theory and Practice. *IEEE Transactions on Computers*, pages 1–1, 2022.

[RMB$^+$18] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. CAPA: The Spirit of Beaver Against Physical Attacks. In *CRYPTO 2018*, volume 10991 of *Lecture Notes in Computer Science*, pages 121–151. Springer, 2018.

[RSBG20] Jan Richter-Brockmann, Pascal Sasdrich, Florian Bache, and Tim Güneysu. Concurrent Error Detection Revisited: Hardware Protection against Fault and Side-channel Attacks. In *ARES 2020*, pages 20:1–20:11. ACM, 2020.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[SJR$^+$20] Sayandeep Saha, Dirmanto Jap, Debapriya Basu Roy, Avik Chakraborty, Shivam Bhasin, and Debdeep Mukhopadhyay. A Framework to Counter Statistical Ineffective Fault Analysis of Block Ciphers Using Domain Transformation and Error Correction. *IEEE Trans. Inf. Forensics Secur.*, 15:1905–1919, 2020.

[SMC21] Albert Spruyt, Alyssa Milburn, and Lukasz Chmielewski. Fault injection as an oscilloscope: Fault correlation analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):192–216, 2021.

[SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In *CRYPTO 2016*, volume 9815 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2016.

[SRM20] Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. Impeccable Circuits II. In *DAC 2020*, pages 1–6. IEEE, 2020.

[SSR$^+$18] Pasquale Davide Schiavone, Ernesto Sánchez, Annachiara Ruospo, Francesco Minervini, Florian Zaruba, Germain Haugou, and Luca Benini. An Open-Source Verification Framework for Open-Source Cores: A RISC-V Case Study. In *IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2018, Verona, Italy, October 8-10, 2018*, pages 43–48. IEEE, 2018.

[Tof80] Tommaso Toffoli. Reversible computing. In *International colloquium on automata, languages, and programming*, pages 632–644. Springer, 1980.

[ZDCT13] Loïc Zussa, Jean-Max Dutertre, Jessy Clédière, and Assia Tria. Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism. In *IOLTS 2013*, pages 110–115. IEEE, 2013.