SILVER – Statistical Independence and Leakage Verification

David Knichel*[0000−0002−2510−8881], Pascal Sasdrich*[0000−0002−5443−626X], and Amir Moradi[0000−0002−4032−7433]

Ruhr University Bochum,
Horst Görtz Institute for IT Security,
Bochum, Germany
{firstname.lastname}@rub.de

Abstract. Implementing cryptographic functions securely in the presence of physical adversaries is still a challenge although a lion's share of research in the physical security domain has been put in development of countermeasures. Among several protection schemes, *masking* has absorbed the most attention of research in both academic and industrial communities, due to its theoretical foundation allowing to provide proofs or model the achieved security level. In return, masking schemes are difficult to implement as the implementation process often is manual, complex, and error-prone. This motivated the need for formal verification tools that allow the designers and engineers to analyze and verify the designs before manufacturing.
In this work, we present a new framework to analyze and verify masked implementations against various security notions using different security models as reference. In particular, our framework – which directly processes the resulting gate-level netlist of a hardware synthesis – particularly relies on Reduced Ordered Binary Decision Diagrams (ROBDDs) and the concept of statistical independence of probability distributions. Compared to existing tools, our framework captivates due to its simplicity, accuracy, and functionality while still having a reasonable efficiency for many applications and common use-cases.

Keywords: Verification · Side-Channel Analysis · Probing Security · Reduced Ordered Binary Decision Diagram · Statistical Independence · Probability Distribution.

1 Introduction

Even after two decades of research since the seminal description of Side-Channel Analysis (SCA) as a threat to cryptographic implementations [32,33], secure implementation of cryptographically strong algorithms is still a challenging and open problem. In particular, those decades of research have shown that SCA on cryptographic implementations can be performed by observing various physical

* These authors contributed equally to the work.

sources and effects, such as timing [32], power consumption [33], electromagnetic (EM) emanations [27], or temperature and heat dissipation [30]. Eventually, observing the physical characteristics of an electronic device during security-critical cryptographic operations can reveal secret and sensitive information to any observer and adversary. As a consequence, a wide range of protection mechanisms and countermeasures have been proposed to prevent or mitigate any side-channel leakage.

Among all candidates, *masking* (based on the concepts of *secret sharing*) is one of the most promising countermeasures against SCA due to its formal and sound security foundation [18] . As a consequence, many different schemes and variants have been introduced and proposed over the years [31,43,39,42,29,28] to address different implementation and security requirements. Unfortunately, not a few of those schemes have been shown to be insecure due to design flaws or inaccurate models or assumptions [36]. As a result, all these examples confirm that design and implementation of protection mechanisms and countermeasures against SCA is a mostly manual, complex, and error-prone process which requires good understanding of the execution environment and careful consideration of physical and security models.

To this end, an entirely new branch of research started to focus on the development of formal models for adversaries and physical execution environments to simplify and assist in formal verification [31,23,5,26]. Ideally, strong theoretical foundations in security models can assist and help to simplify the design, implementation, and verification of cryptographic implementations and appropriate security mechanisms. In the context of *masking*, formal verification often is conducted in the simple and abstract Ishai-Sahai-Wagner (ISW) d-probing security model [31] (under some basic assumptions on noise and independence of inputs), which allows an adversary to probe (observe) up to d intermediate values during the processing of sensitive information.

Due to its conceptual simplicity and level of abstraction, the d-probing model was rapidly and widely adopted for formal verification [38,6,24,3,4,11,20,44,2]. Indeed, the introduction of this simple but effective security model propelled the automation of formal verification, allowing to reduce the combinatorial complexity of security proofs for masking schemes and their implementations. In fact, development of automated formal verification tools also – in return – stimulated the research and progress on masking schemes, e.g., reducing the cost in terms of randomness [9] or solving the problem of secure composition of masked circuits and gadgets [3,4,17].

However, in its basic manifestation, the d-probing model does not consider specific physical defaults, such as *glitches*, *transitions*, or *couplings* [26], that may occur during the processing of sensitive information on a physical device. In fact, many schemes proven to be secure in the d-probing model, eventually fail in security analyzes when concretely implemented. That is mainly due to undesired and unintentional physical defaults that particularly violate the assumption on the independence of inputs. In particular for hardware implementations, *glitches* are well-known to be an issue and concern for masking schemes [39], wherefore

Bloem *et al.* [11] and Faust *et al.* [26] independently proposed an extension of the basic ISW d-probing model considering glitches for hardware implementations of masking schemes.

In addition, Bloem *et al.* used the concept of Fourier coefficient estimation to implement an automated tool formally verifying the security of masking schemes and their implementations against the basic and glitch-extended d-probing model. However, due to computational limitations based on the estimation of Fourier coefficients, this tool primarily applies to the security analysis of the first-order setting without consideration of advanced notions such as Non-Interference (NI), Strong Non-Interference (SNI), and Probe-Isolating Non-Interference (PINI). In contrast to this, Barthe *et al.* [2] recently presented a language-based formal verification tool called `maskVerif` which uses the probabilistic information flow to assess the security of masking schemes and their implementations. In particular, using conservative heuristics and an optimistic sampling method, `maskVerif` executes more efficiently than the tool by Bloem *et al.*, while minimizing but still accepting false negatives for non-linear cases.

Contributions. In this work, we present and introduce an efficient methodology to analyze and verify the security of masked circuits and implementations under various security notions. Due to a symbolic and exhaustive analysis of probability distributions and statistical independence of joint distributions, we can avoid false negatives and overly conservative decisions. In particular, by means of ROBDDs, a well-known concept and methodology for Integrated Circuit (IC) testing and verification, we formally analyze and verify masked circuits in the ISW d-probing model even in the presence of *glitches* as physical defaults.

In addition, based on the seminal work of De Meyer *et al.* [21], we reformulate the security notions of d-probing security, d-Non-Interference, d-Strong Non-Interference, and, for the first time, d-Probe-Isolating Non-Interference based on statistical independence which can be efficiently checked and verified by our tool. Hence, for the first time, state-of-the-art security notions for masked circuits can be analyzed exhaustively without false negatives. Eventually, this contribution is even extended further by efficient verification methods to check and verify the uniformity for output sharings of arbitrary masked circuits.

Outline. While Section 2 briefly summarizes our notations and introduces preliminary concepts and notions, including ROBDDs, our circuit model, and security notions, Section 3 is dedicated to a conception and discussion of our verification approach. Besides, Section 3 outlines our leakage models and discusses the main ideas of our verification concept, particularly sketching the application of ROBDDs to check and verify security notions. In Section 4, we then provide formal proofs for all security notions and our leakage verification concept based on statistical independence checks. Before we present details on practical evaluations and experiments in Section 6, we briefly discuss and compare our approach and concept to essential related work in Section 5. Eventually, we conclude our work in Section 7.

2 Background

2.1 Notation

We use upper-case characters to denote random variables, bold ones for sets of random variables, and subscripts for elements within a set of variables. Further, let us denote $\mathbf{X}_{\bar{i}}$ as the set of variables $\mathbf{X} \setminus X_i$. Accordingly, we use lower-case characters to denote values a random variable can take, bold ones for sets of values, and subscripts for elements within the set of values. Again, let us denote $\mathbf{x}_{\bar{i}}$ as the set of values $\mathbf{x} \setminus x_i$. In addition, we use $Pr[X = x]$ for the probability that a random variable X takes a value x, while $Pr[\mathbf{X} = \mathbf{x}]$ denotes the joint probability that each $X_i \in \mathbf{X}$ takes the value $x_i \in \mathbf{x}$. Accordingly, the conditional probability for $X = x$ given $Y = y$ is written as $Pr[X = x|Y = y]$. Hence, $Pr[\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y}]$ denotes the conditional probability that each $X_i \in \mathbf{X}$ takes the value $x_i \in \mathbf{x}$, if each $Y_i \in \mathbf{Y}$ takes the value $y_i \in \mathbf{y}$. Moreover, the joint distribution over the set $\mathbf{X}$ is denoted as $Pr[\mathbf{X}]$, while $Pr[\mathbf{X}|\mathbf{Y}] = Pr[\mathbf{X}]$ is simply equivalent to $Pr[\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y}] = Pr[\mathbf{X} = \mathbf{x}]$ for all possible combination of $\mathbf{x}$ and $\mathbf{y}$. Extending this notation, $Pr[\mathbf{X}|\mathbf{Y}] = Pr[\mathbf{X}|\mathbf{Z}]$ is the same as $Pr[\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y}] = Pr[\mathbf{X} = \mathbf{x}|\mathbf{Z} = \mathbf{z}]$ for all possible combination of $\mathbf{x}$,$\mathbf{y}$, and $\mathbf{z}$.

Further, functions are denoted using sans-serif fonts. Handling masked functions, we denote the s-th share of the a variable as X^s. Hence, the set of all unshared inputs of a function f is denoted as $\mathbf{X} = (X_0, \ldots, X_{n-1})$ while the set containing all t shares of each variable in $\mathbf{X}$ is denoted as $Sh(\mathbf{X}) = (X_0^0, X_0^1, \ldots, X_0^{t-1}, X_1^0, \ldots, X_{n-1}^0, \ldots, X_{n-1}^{t-1})$. Similarly, the set containing all shares of $X_i \in \mathbf{X}$ is denoted as $Sh(X_i)$. Eventually, for a set of indices $\mathbf{I} \subseteq [0, \ldots, t-1]$, $Sh(\mathbf{X})^{\mathbf{I}}$ denotes the set containing all shares X_i^s with $0 \leq i < n$ and $s \in \mathbf{I}$.

2.2 Reduced Ordered Binary Decision Diagrams (ROBDDs)

Binary Decision Diagrams (BDDs) are a basic structure in discrete mathematics and computer science introduced by Akers [1] and refined by Bryant (introducing variable ordering) [15]. In particular, many applications in computer-aided IC design and verification make use of (reduced, ordered) BDDs.

In general, BDDs are concise and unique (i.e., canonical) graph-based representations of Boolean functions $\mathbb{F}_2^n \to \mathbb{F}_2$ with a single root node and two terminal nodes (leaves) $\{\mathbf{T}, \mathbf{F}\}$. The formal definition of ROBDDs, given in the following paragraphs, is divided into a purely *syntactical* definition, describing the structure based on Directed Acyclic Graphs (DAGs), before providing a *semantical* definition, clarifying the representation of Boolean functions as ROBDDs.

Syntactical Definition of ROBDDs. Before providing a syntactical definition for ROBDDs, we first recall the (syntactical) definition of Ordered Binary Decision Diagrams (OBDDs).

Definition 1 (OBDD Syntax). *An* Ordered Binary Decision Diagram *is a pair* $(\pi, \mathcal{G})$*, where* π *denotes the variable ordering of the OBDD and* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is a finite DAG with vertices* $\mathcal{V}$*, edges* $\mathcal{E}$*, and the following properties:*

(1) There is a single root node and each node $v \in \mathcal{V}$ *is either a non-terminal node or one of two terminal nodes* $\{\mathbf{T}, \mathbf{F}\}$*.*
(2) Each non-terminal node v *is labeled with a variable in* $\mathbf{X}$*, with* $|\mathbf{X}| = n$*, denoted as* $\mathsf{var}(v)$*, and has exactly two child nodes in* $\mathcal{V}$ *which are denoted as* $\mathsf{then}(v)$ *and* $\mathsf{else}(v)$*.*
(3) For each path from the root node to a terminal node, the variables in $\mathbf{X}$ *are encountered at most once and in the same order defined by the variable ordering* π*. More precisely, the variable ordering* π *of an OBDD is a bijection* $\pi : \{1, 2, \ldots, n\} \to \mathbf{X}$*.*

Furthermore, assuming the following two restrictions ensures a concise and canonical representation (under a given variable ordering π), defined as ROBDD.

Definition 2 (ROBDD Syntax). *An OBDD is called* Reduced Ordered Binary Decision Diagram*, if and only if there is no node* $v \in \mathcal{V}$ *such that* $\mathsf{then}(v) = \mathsf{else}(v)$ *and there are no duplicate nodes, i.e., two nodes* $\{v, v'\} \in \mathcal{V}$ *such that* $\mathsf{var}(v) = \mathsf{var}(v')$*,* $\mathsf{then}(v) = \mathsf{then}(v')$*, and* $\mathsf{else}(v) = \mathsf{else}(v')$*.*

Semantical Definition of ROBDDs. Each ROBDD with root $v \in \mathcal{V}$ recursively defines a Boolean function $\mathsf{f} : \mathbb{F}_2^n \to \mathbb{F}_2$ according to the following definition.

Definition 3 (ROBDD Semantics). *An* ROBDD *over* $\mathbf{X}$ *represents a Boolean function* f *recursively carried out at each node and defined as follows:*

(1) If v *is the terminal node* $\mathbf{F}$*, then* $\mathsf{f}_v|_x = 0$*, otherwise, if* v *is the terminal node* $\mathbf{T}$*, then* $\mathsf{f}_v|_x = 1$*.*
(2) If v *is a non-terminal node and* $\mathsf{var}(v) = x_i$*, then* f_v *is defined by the Shannon decomposition* $\mathsf{f}_v = x_i \cdot \mathsf{f}_{\mathsf{then}(v)}|_{x_i=1} + \overline{x_i} \cdot \mathsf{f}_{\mathsf{else}(v)}|_{x_i=0}$*.*

Boolean Operations over ROBDDs. Given the syntactical and semantical definitions for ROBDDs, we now can define arbitrary Boolean operations over Boolean functions f_{v_1} and f_{v_2} represented by two ROBDDs with root nodes v_1 and v_2. In particular, let $\mathsf{f} = \mathsf{f}_{v_1} \circ \mathsf{f}_{v_2}$ where $\circ$ denotes any binary Boolean operation, then the ROBDD for f can be derived and composed recursively as:

$$\begin{aligned} \mathsf{f} &= x_i \cdot \mathsf{f}|_{x_i=1} + \overline{x_i} \cdot \mathsf{f}|_{x_i=0} \\ &= x_i \cdot (\mathsf{f}_{v_1} \circ \mathsf{f}_{v_2})|_{x_i=1} + \overline{x_i} \cdot (\mathsf{f}_{v_1} \circ \mathsf{f}_{v_2})|_{x_i=0} \\ &= x_i \cdot (\mathsf{f}_{v_1}|_{x_i=1} \circ \mathsf{f}_{v_2}|_{x_i=1}) + \overline{x_i} \cdot (\mathsf{f}_{v_1}|_{x_i=0} \circ \mathsf{f}_{v_2}|_{x_i=0}) \end{aligned} \tag{1}$$

2.3 Circuit Model

For the remainder of this work, we consider and model a deterministic circuit C as a DAG, where the vertices are combinational gates and edges are wires carrying elements from the finite field $\mathbb{F}_2$.

Physical Model. Without loss of generality, the physical structure of a deterministic circuit C at gate level is modeled using the set of combinational gates $\{\mathsf{not}, \mathsf{and}, \mathsf{nand}, \mathsf{or}, \mathsf{nor}, \mathsf{xor}, \mathsf{xnor}\}$ (with fan-in at most 2) while all sequential memory gates reg model a clock-dependent synchronization point. Further, each Boolean input variable is associated with a single in gate (with fan-in 0), while the output and result of a Boolean function is associated with a single out gate (with fan-out 0). Eventually, ref are special-purpose gates with fan-in 0 that introduce a *independently and identically distributed* (i.i.d) random element from the finite field $\mathbb{F}_2$.

Functional Model. Each deterministic circuit C realizes an $n \times m$ vectorial Boolean function $\mathsf{F} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ given its *coordinate functions* $\mathsf{f}_1, \ldots, \mathsf{f}_m$ defined over $\mathbf{X} \in \mathbb{F}_2^n$. In particular, each $X_i \in \mathbf{X}$ is assumed to be *independently and identically distributed* (i.i.d) and associated with a single in gate, while each f_i is associated with a single out gate.

Further, the function of each gate within the circuit model C is derived recursively over the functions of its fan-in gates by means of Boolean operations over ROBDDs. Hence, each gate in the circuit model itself can be considered as a Boolean function over (a subset of) the inputs $\mathbf{X} \in \mathbb{F}_2^n$ and we can introduce a functional abstraction layer to the physical circuit model using ROBDDs to canonically represent and store the derived Boolean functions.

Security Model. Eventually, security critical circuits handling a sensitive secret X are associated with a security order d and protected (masked) based on Boolean sharing. This means, each security critical and sensitive secret X is shared with at least $d+1$ shares such that $X = \bigoplus_0^d X^i$. Similarly, sensitive and security critical outputs of a masked circuit are shared using Boolean sharing, such that $\mathsf{F}(\mathbf{X}) = \bigoplus_0^d \mathsf{F}^i(\mathbf{X})$.

2.4 Security Notions

Before introducing our verification approach and methodology to analyze an arbitrary circuit under various security notions, we first introduce the definitions of all necessary security notions. In particular, our security definitions are based on the work in [21] while we reformulate the definitions in order to provide generalizations from circuits with $d+1$ input shares to circuits with an arbitrary number of shares when examining dth-order security. In addition, we extend the definitions from single sensitive and secret variable to a set of arbitrary number of secret variables.

Probing Security. Probing security is defined as the probes being statistically independent of any sensitive input. More precisely, the joint distribution of the considered set of probes has to be independent of the joint distribution of all sensitive inputs. This can be formally defined as:

Definition 4 (d-Probing Security.). *A circuit* C *with secret input set* $\mathbf{X} \in \mathbb{F}_2^n$ *is d-probing secure, if and only if for any observation set* $\mathbf{Q}$ *containing d wires,* $\mathbf{X}$ *is statistically independent of the observation set, i.e., the following condition holds:*

$$Pr[\mathbf{Q}|\mathbf{X}] = Pr[\mathbf{Q}] \tag{2}$$

Non-Interference. The notion of *Non-Interference* allows partial information on the sensitive inputs becoming available to the adversary through probing the circuit. In particular, if the observed circuit is d-NI, the adversary is not able to successfully distinguish the circuit result from a simulator working on partial information knowing, i.e., using at most d shares of each input.

More formally, each adversarial probe set should be perfectly simulatable knowing only a subset of all shares of each input. Let $\mathbf{S}$ be a set over arbitrary input shares X_i^j, i.e., $\mathbf{S} \subset Sh(\mathbf{X})$, and $|\mathbf{S}|_i$ denote the number of shares in $\mathbf{S}$ that correspond to input X_i. In order to guarantee d-NI, there exist a simulation set $\mathbf{S}$ with $|\mathbf{S}|_{\forall i} \leq d$ for which a probe in $\mathbf{Q}$ is perfectly simulatable, i.e., an attacker is not able to distinguish between a simulation of C using only elements in $\mathbf{S}$ from the observations of C even when knowing all input shares. This can be directly translated into the condition that there has to exist a simulation set $\mathbf{S}$ with $|\mathbf{S}|_{\forall i} \leq d$, for which the distributions $Pr[\mathbf{Q}|\mathbf{S}]$ and $Pr[\mathbf{Q}|Sh(\mathbf{X})]$ are equal.

Definition 5 (d-Non-Interference). *A circuit* C *with secret input set* $\mathbf{X} \in \mathbb{F}_2^n$ *provides d-Non-Interference if and only if for any observation set of $t \leq d$ wires* $\mathbf{Q}$ *there exists a set* $\mathbf{S}$ *of input shares with* $|\mathbf{S}|_{\forall i} \leq t$ *such that*

$$Pr[\mathbf{Q}|\mathbf{S}] = Pr[\mathbf{Q}|Sh(\mathbf{X})]. \tag{3}$$

Strong Non-Interference. The notion of SNI has been introduced as extension to NI correcting deficiencies in terms of composability of secure gadgets within a circuit. In contrast to NI, any probe on a circuit output (also, through composition, considered as input to a subsequent gadget) is not allowed to give information about any share in the input. This means, each probe on an output wire must be perfectly simulatable without knowledge of any input shares in order to stop the flow of sensitive information between composed gadgets.

More formally, each adversarial probe set again should be perfectly simulatable knowing only a subset of all shares of each input. However, for a set $\mathbf{Q}$ of d probes with t_1 probes on internal wires and t_2 probes on output wires while $t_1 + t_2 \leq d$, the size of the set $\mathbf{S}$ is bounded by the internal probes only, i.e., $|\mathbf{S}|_{\forall i} \leq t_1$. This directly translates into the following definition and condition.

Definition 6 (d-Strong Non-Interference). *A circuit* C *with secret input set* $\mathbf{X} \in \mathbb{F}_2^n$ *provides d-Strong Non-Interference if and only if for any observation set of $t = t_1 + t_2 \leq d$ wires* $\mathbf{Q}$ *of which t_1 are internal wires and t_2 are output wires, there exists a simulation set* $\mathbf{S}$ *of input shares with* $|\mathbf{S}|_{\forall i} \leq t_1$ *such that Equation* (3) *holds.*

Probe-Isolating Non-Interference. Unfortunately, the security notion of SNI in practice is often very conservative and inefficient as it introduces more area and randomness than necessary to achieve certain security goals. To address this issue, Cassiers *et al.* introduced the notion of Probe-Isolating Non-Interference [17] which offers trivial composition of any gadgets inspired by the trivial composition of linear functions and the concept of sharing domain separations as introduced in [29].

More formally, each circuit input and output is assigned a unique circuit share index (i.e., a share domain) and any probe set on these wires should be perfectly simulatable knowing only the set of inputs that are assigned to the same circuit share index. Further, any additional probe on internal wires gives the adversary access to at most one additional circuit share, i.e., must be perfectly simulatable knowing only the according set of inputs assigned to these circuit shares. Eventually, this translates to the following definition.

Definition 7 (d-Probe-Isolating Non-Interference). *Let* $\mathbf{P}$ *be the set of internal probes with* $|\mathbf{P}| = t_1$*. Let further* $\mathbf{I_O}$ *be the index set assigned to the probed output wires* $\mathbf{O}$ *with* $|\mathbf{I_O}| = t_2$*.*

A circuit C *with secret input set* $\mathbf{X} \in \mathbb{F}_2^n$ *provides d-Probe-Isolating Non-Interference if and only if for every* $\mathbf{P}$ *and* $\mathbf{O}$ *with* $t_1 + t_2 \leq d$ *there exists a set* $\mathbf{I_I}$ *of circuit indices with* $|\mathbf{I_I}| \leq t_1$ *such that* $\mathbf{Q} = \mathbf{P} \cup \mathbf{O}$ *can be perfectly simulated by* $\mathbf{S} = Sh(\mathbf{X})^{\mathbf{I_I} \cup \mathbf{I_O}}$*, i.e., Equation* (3) *holds.*

Uniformity. The security of (Boolean) masking schemes relies on a fundamental assumption: *uniform sharing*. For that, the initial sharing of any secret variable X using $d+1$ shares, such that $X = \bigoplus_0^d X^i$, can be done by assigning random values to $X^0, \ldots, X^{d-1}$ and deriving $X^d = X \oplus \bigoplus_0^{d-1} X^i$. Such a sharing then is uniform if all random variables $X^0, \ldots, X^{d-1}$ are independent of each other and have a uniform distribution over $\mathbb{F}_2$.

In practice, the uniformity of the output sharing of gadgets has been defined as a fundamental requirement for Threshold Implementations (TIs), particularly for secure composition of gadgets [39]. Otherwise, a non-uniform output sharing of a gadget becomes the non-uniform input sharing of another gadget, hence violating the essential assumption of uniformity for (Boolean) masking schemes. Note, however, that a gadget can be probing secure, but it is not necessarily uniform. Likewise, a uniform gadget does not automatically lead to a probing-secure construction. This has been handled specifically in NI and SNI gadgets, e.g., by injecting fresh randomness at the output thereby refreshing the output sharing, i.e., achieving uniformity.

Definition 8 formalizes the notion of a uniform sharing as it states that for every unshared value, each valid sharing has to occur with the same probability.

Definition 8 (Uniform Sharing). *Let* $\mathbf{Y}$ *be a set of binary random variable and* $Sh(\mathbf{Y})$ *its corresponding Boolean sharing. Then* $Sh(\mathbf{Y})$ *is said to be a uni-*

form sharing iff for some constant p

$$Pr\big[Sh(\mathbf{Y}) = \mathbf{y}^* | \mathbf{Y} = \mathbf{y}\big] = \begin{cases} p & \textit{if } \mathbf{y}^* \textit{ is a valid sharing for } \mathbf{y} \\ 0 & else \end{cases}. \quad (4)$$

3 Verification Concept

This section briefly elaborates our main idea and concept for our verification model and approach.

3.1 Leakage Models

For verification, we additionally consider each security notion under two different leakage models denoted *standard* respectively *robust* leakage model.

Standard Leakage Model. For our standard leakage model following the concept of the traditional ISW d-probing model [31], we assume an ideal circuit without any physical defaults such as *glitches* or *transitions*. In practice, this leakage model relates to a software scenario where each result of an operation (i.e., a gate in a circuit C) is stored in a register before it is used by subsequent operations (gates). Note that in this model, the implementation platform's specific effects like pipelines are entirely ignored. In this model, an adversarial probe provides access to the field element carried on the probed wire. More precisely, the adversary gains full access to the Boolean function represented by the driving gate in order to derive the field element.

Robust Leakage Model. In contrast to our standard leakage model, for our robust model following the leakage model in [26], we also take physical default in terms of glitches into consideration, hence, in practice this model relates to a hardware scenario. Since only circuit inputs and memory elements are assumed to switch synchronous to a circuit clock, glitches will propagate through all combinational gates between two synchronization points. Therefore, by probing a wire, the adversary not only gains access to the field element of the driving gate but also can access all stable field elements of the last synchronization points which drive the probed gate (having a path to the driving gate in the circuit graph). More precisely, the adversary gains full access to the set of these field elements (and any subset) through so called *glitch-extended* probes.

3.2 Verification Approach

Based on some fundamental observation, this section outlines our basic concept and explains our main approach to verify different security notions starting from a circuit model given as gate-level netlist.

Random Variables with Binary Events. According to our circuit model, each edge in the circuit graph models a wire and carries an element from the field $\mathbb{F}_2$ with two elements. Thus, we first observe each wire, and its associated field element can be modeled as a binary random variable defined over the sample space $\Omega = \{0, 1\}$ of two basic events given the assumption that all primary circuit inputs are *independent and identically distributed* (i.i.d.). Based on this observation, we can use the probability distributions of all random variables in order to analyze and verify a circuit model against the security notions defined in Section 2.4.

Probability Distribution and Satisfiability. In general, the probability of an event is defined by the sum of the probabilities of all *outcomes* that satisfy the event. In the context of our circuit model, an *outcome* can be considered as a variable assignment to the primary circuit inputs that leads to the desired element of the sample space at the observed random variable. For this, computing the probability density function of a random variable associated with a circuit wire reduces to enumerating and counting the primary input variable assignments that satisfy the corresponding basic events for the observed random variable.

Symbolic Simulation using ROBDDs. As the naive approach of exhaustive and literal simulation of the circuit model expeditiously becomes infeasible with increasing circuit complexity and number of primary circuit inputs, symbolic simulation and analysis is necessary to maintain the generation of all probability distributions practicable even for large and complex circuit models. More precisely, each gate in the circuit model represents a sub-circuit and is associated with a Boolean function given as ROBDD that computes the gate output over the set of primary circuit inputs. Since ROBDDs are concise and canonical representations of Boolean functions, counting the number of *cubes*, i.e., the satisfying variable assignments, for each basic binary event can be done efficiently using symbolic analysis. Knowing the total number of possible variable assignments, computing the probability distribution for each random variable remains feasible even for large and complex circuits.

Standard and Glitch-Extended Probes. Considering our two different leakage models, we also have to differentiate the capabilities and knowledge of the adversary. Firstly, for the standard model we thus assume that an adversarial probe gives access to the probability distribution of a field element carried on an arbitrary wire observed by the adversarial probe. More precisely, the adversary in this case learns the Boolean sub-function associated with the driving gate in order to compute the field element and its probability distribution as function of the primary circuit inputs. Secondly, in contrast to the standard model, a robust or *glitch-extended* probe extends the capabilities and knowledge of the adversary as it also provides access to the joint distribution of all hindmost contributing

synchronization points (memory elements or primary inputs). Hence, in order to model physical defaults and in particular glitches, the adversary also learns the Boolean sub-functions associated with the corresponding synchronization elements.

Statistical Independence and Security Checks. Eventually, depending on the targeted security order d, an adversarial *observation* can consist of up to d independently placed adversarial probes and the adversary is allowed to combine the information and knowledge of all probes to learn details of the secret. In order to verify security under the given security notions as defined in Section 2.4, we perform an exhaustive exploration and check of all possible adversarial observations $\mathbf{Q}$ combining up to d probes. For this, the following section is dedicated to a detail description and verification of our performed security checks.

4 Statistical Independence and Security Checks

Before formally analysis and verification of the correctness of our security checks, we briefly recap the definitions of (joint) probability mass functions and statistical independence for sets of random variables.

4.1 Statistical Independence

The probability mass function provides the probability of all possible values for a set of discrete random variables based on their probability distribution.

Definition 9 (Probability Mass Function). *Let* $\mathbf{X}$ *be a set of discrete random variables. The* probability mass function $p_{\mathbf{X}}(\mathbf{x})$ *is defined as:*

$$p_{\mathbf{X}}(\mathbf{x}) = Pr[\mathbf{X} = \mathbf{x}].$$

Based on this, given two arbitrary sets of discrete random variables, the joint probability mass function between these two variable sets then is defined as follows.

Definition 10 (Joint Probability Mass Function). *Let* $\mathbf{X}, \mathbf{Y}$ *be two sets of discrete random variables. The* joint probability mass function $p_{\mathbf{X},\mathbf{Y}}(\mathbf{x}, \mathbf{y})$ *is defined as:*

$$p_{\mathbf{X},\mathbf{Y}}(\mathbf{x}, \mathbf{y}) = Pr[\mathbf{X} = \mathbf{x} \text{ and } \mathbf{Y} = \mathbf{y}].$$

Using the definitions of probability mass function and joint probability mass function, we can express the notion of statistical independence of two sets of discrete random variables according to the following definition.

Definition 11 (Statistical Independence). *Let* $\mathbf{X}, \mathbf{Y}$ *be two sets of discrete random variables.* $\mathbf{X}, \mathbf{Y}$ *are statistically independent if and only if the joint probability mass function for* $\forall \mathbf{x}$ *and* $\forall \mathbf{y}$ *satisfies*

$$p_{\mathbf{X},\mathbf{Y}}(\mathbf{x}, \mathbf{y}) = p_{\mathbf{X}}(\mathbf{x}) \cdot p_{\mathbf{Y}}(\mathbf{y}).$$

Statistical Independence of Binary Random Variables. In the context of our security notions, we are mainly interested in statistical independence of *binary random variables*. As any binary random variable can only take two different events, Theorem 1 states that checking statistical independence for one event implies statistical independence for both events, even extending to the case of sets of binary random variables.

Theorem 1. *Let* $\mathbf{X}, \mathbf{Y}$ *be two sets of binary random variables. Then* $\mathbf{X}$ *and* $\mathbf{Y}$ *are statistically independent, if and only if* $p_{\mathbf{X}',\mathbf{Y}'}(\mathbf{a},\mathbf{b}) = p_{\mathbf{X}'}(\mathbf{a}) \cdot p_{\mathbf{Y}'}(\mathbf{b})$ *for any fixed values* $\mathbf{a}$ *and* $\mathbf{b}$ *and every possible combination of* $\mathbf{X}' \subseteq \mathbf{X}$ *and* $\mathbf{Y}' \subseteq \mathbf{Y}$*.*

In order to proof correctness of Theorem 1, we start with the basic case of two binary random variables (i.e., sets of cardinality one).

Lemma 1. *Let* $X, Y \in \mathbb{F}_2$ *be two binary random variables. Then, a necessary and sufficient condition for* X *to be statistically independent of* Y *is that, for any fixed values* $a, b \in \{0,1\}$*, it holds*

$$p_{X,Y}(a,b) = p_X(a) \cdot p_Y(b).$$

Proof. According to Definition 11, the necessity of this proposition is obvious, hence, the proof focuses on the sufficiency. Without loss of generality, we now assume Lemma 1 is true for $a = b = 1$, i.e., $p_{X,Y}(1,1) = p_X(1) \cdot p_Y(1)$. Since $X = 0$ and $X = 1$ are counter events for binary variables, both the fact $p_X(0) + p_X(1) = 1$ and the fact $p_{X,Y}(0,1) + p_{X,Y}(1,1) = p_Y(1)$ hold, and we have

$$\begin{aligned} & p_{X,Y}(0,1) + p_{X,Y}(1,1) = p_Y(1) \\ \Leftrightarrow\ & p_{X,Y}(0,1) + p_X(1) \cdot p_Y(1) = p_Y(1) \\ \Leftrightarrow\ & p_{X,Y}(0,1) = (1 - p_X(1)) \cdot p_Y(1) \\ \Leftrightarrow\ & p_{X,Y}(0,1) = p_X(0) \cdot p_Y(1) \end{aligned}$$

Proving the cases for $a = 1, b = 0$ and $a = b = 0$ is trivial as it follows the same approach, hence is left out for brevity. □

In a next step, we extend the basic case through mathematical induction in order to prove statistical independence between a single random binary variable and a set of random binary variables.

Lemma 2. *Let* X *be a binary random variable and* $\mathbf{Y}$ *a set of* n *random binary variables. Further, let* X *be statistically independent of the joint distribution of* $\mathbf{Y}$*. Now, the joint distribution* $\mathbf{Y}^+$*, with* $\mathbf{Y} \subset \mathbf{Y}^+$ *and* $|\mathbf{Y}^+| = n+1$ *is statistically independent of* X*, if and only if* $p_{X,\mathbf{Y}^+}(a,\mathbf{b}^+) = p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+)$ *for any fixed values* $a, \mathbf{b}^+$*.*

Proof. We now give a formal proof using mathematical induction on n.

Base case: We first show that Lemma 2 holds for $n = 0$.

Clearly, if $n = 0$, $\mathbf{Y}$ is the empty set while $\mathbf{Y}^+$ is a single binary random variable. Then, according to Lemma 1, X and $\mathbf{Y}^+$ are statistically independent if and only if for any fixed values a, b it holds that $p_{X,\mathbf{Y}^+}(a,b) = p_X(a) \cdot p_{\mathbf{Y}^+}(b)$. □

Induction: If Lemma 2 holds for $n = k$, it also holds for $n = k+1$ with $k \geq 0$.

For this, we first show that, without loss of generality, for $X, \mathbf{Y}^+$ the following fact $p_{X,\mathbf{Y}^+}(a, \mathbf{b}_{\bar{i}}^+) = p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}_{\bar{i}}^+)$ with $\mathbf{b}_{\bar{i}}^+ = \{y_0, y_1, \ldots, \overline{y_i}, \ldots, y_k, y_{k+1}\}$ holds, if:

(i) $p_{X,\mathbf{Y}}(a, \mathbf{b}) = p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b})$ with $\mathbf{b} = \{y_0, y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k, y_{k+1}\}$,
(ii) $p_{X,\mathbf{Y}^+}(a, \mathbf{b}^+) = p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+)$ with $\mathbf{b}^+ = \{y_0, y_1, \ldots, y_i, \ldots, y_k, y_{k+1}\}$

Further, we note that for binary random variables it always holds that:

$$p_{X,\mathbf{Y}}(a, \mathbf{b}) = p_{X,\mathbf{Y}^+}(a, \mathbf{b}^+) + p_{X,\mathbf{Y}^+}(a, \mathbf{b}_{\bar{i}}^+)$$

Given that (i), (ii) are conditions for Lemma 2, we can state the following:

$$\begin{aligned}
& p_{X,\mathbf{Y}}(a, \mathbf{b}) = p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) \\
\Leftrightarrow\ & p_{X,\mathbf{Y}^+}(a, \mathbf{b}^+) + p_{X,\mathbf{Y}^+}(a, \mathbf{b}_{\bar{i}}^+) = p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) \\
\Leftrightarrow\ & p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+) + p_{X,\mathbf{Y}^+}(a, \mathbf{b}_{\bar{i}}^+) = p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) \\
\Leftrightarrow\ & p_{X,\mathbf{Y}^+}(a, \mathbf{b}_{\bar{i}}^+) = p_X(a) \cdot p_{\mathbf{Y}}(\mathbf{b}) - p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}^+) \\
\Leftrightarrow\ & p_{X,\mathbf{Y}^+}(a, \mathbf{b}_{\bar{i}}^+) = p_X(a) \cdot [p_{\mathbf{Y}}(\mathbf{b}) - p_{\mathbf{Y}^+}(\mathbf{b}^+)] \\
\Leftrightarrow\ & p_{X,\mathbf{Y}^+}(a, \mathbf{b}_{\bar{i}}^+) = p_X(a) \cdot p_{\mathbf{Y}^+}(\mathbf{b}_{\bar{i}}^+)
\end{aligned}$$

As the sorting of variables in $\mathbf{Y}^+$ is not fixed, this approach also extends to inversion of any other event and therefore can easily be extended to show statistical independence for every combination of events. □

Eventually, this also proves Theorem 1. In particular, knowing that $\mathbf{X}, \mathbf{Y}$ are statistically independent, we can argue that $\mathbf{X}^+, \mathbf{Y}$ with $\mathbf{X} \subset \mathbf{X}^+$, $|\mathbf{X}| = n$, and $|\mathbf{X}^+| = n + 1$ must be statistically independent, if and only if $p_{\mathbf{X}^+,\mathbf{Y}}(\mathbf{a}^+, \mathbf{b}) = p_{\mathbf{X}^+}(\mathbf{a}^+) \cdot p_{\mathbf{Y}}(\mathbf{b})$ using the same approach as for Lemma 2.

4.2 *d*-Probing Security

Checking d-probing security according to Definition 4 requires to verify statistical independence of the set of secret variables and any observation of at most d probes. This section presents an exploration algorithm that allows to find and verify the maximum security order of a given circuit with secret variables $\mathbf{X}$. Eventually, the algorithm will return the first set of $d + 1$ probes that is not statistically independent of the secret variables.

Algorithm 1: Explore d-Probing Security.

```
   Input  : X – Set of n secret variables.
   Output: Q – Set of d + 1 successful probes.
 1 d ← 1
 2 while true do
 3 |   foreach probing set Q with |Q| = d do
 4 |   |   for t = 1 to n do
 5 |   |   |   foreach X' ⊆ X with X' = t do
 6 |   |   |   |   if p_{Q,X'}(1,1) ≠ p_Q(1) · p_{X'}(1) then
 7 |   |   |   |   |   return Q
 8 |   |   |   |   end
 9 |   |   |   end
10 |   |   end
11 |   end
12 |   d ← d + 1
13 end
```

Algorithmic Verification Approach. Algorithm 1 presents our algorithmic approach to explore and verify d-probing security of a Circuit Under Test (CUT). In general, the algorithm is initialized with $d = 1$, i.e., starts to explore and verify first-order security before extending verification to higher orders. Since for $|\mathbf{Q}| = 1$ each observation set contains only a single binary variable (observed by a single probe placed on a wire within the circuit C), according to Theorem 1 it is sufficient to verify:

$$p_{\mathbf{Q},\mathbf{X}'}(\mathbf{1},\mathbf{1}) \stackrel{?}{=} p_{\mathbf{Q}}(\mathbf{1}) \cdot p_{\mathbf{X}'}(\mathbf{1}) \tag{5}$$

for all possible combinations of secret variables $\mathbf{X}' \subseteq \mathbf{X}$. If any of those checks fails, the current observation is not statistically independent of the secret variables and Algorithm 1 terminates with returning the current set of observation indicating the security of the CUT to be at most $d = |\mathbf{Q}| - 1$.

If probing security is verified for all joint distributions of d probes, the algorithm continues with all combinations of $d+1$ probes. However, for independence of the current set of probes $\mathbf{Q}$, it is still sufficient to check Equation (5) for all combinations of secret variables (but only the current combination of probes), since any subset of probes has already been verified during previous iterations (for smaller d).

Eventually, all verification and statistical independence checks are performed based on ROBDDs in order to generate all (joint) probability mass functions $p_{\mathbf{Q}}$, $p_{\mathbf{X}}$, and $p_{\mathbf{Q},\mathbf{X}}$. In particular, evaluation of the probability mass functions for $\mathbf{1}$ is very efficient for ROBDD-based representations, usually implemented as *satisfiability-check*.

Algorithm 2: Explore d-Non-Interference Security.

Input : $Sh(\mathbf{X})$ – Set of all shares of n secret variables $\mathbf{X}$.
Output: $\mathbf{Q}$ – Set of $d+1$ successful probes.

```
1  d ← 1
2  while true do
3  |  foreach probing set Q with |Q| = d do
4  |  |  simulatable ← true
5  |  |  for t = 0 to d do
6  |  |  |  foreach S ⊆ Sh(X) with |S|_∀i = t do
7  |  |  |  |  if p_{Q,Sh(X)}(1,1) ≠ p_{Q,S}(1,1) · p_{S̄}(1) then
8  |  |  |  |  |  simulatable ← false
9  |  |  |  |  end
10 |  |  |  end
11 |  |  end
12 |  |  if not simulatable then
13 |  |  |  return Q
14 |  |  end
15 |  end
16 |  d ← d + 1
17 end
```

4.3 d-Non-Interference

Checking d-NI security using Definition 5 requires to verify Equation (3) that every set of at most d probes $\mathbf{Q}$ on a circuit C has to be perfectly simulatable using only a subset $\mathbf{S}$ of all shares of the secret variables $\mathbf{X}$. Using the concept of statistical independence of two sets of random binary variables, we can express NI using the following theorem.

Theorem 2. *Let $\overline{\mathbf{S}} := Sh(\mathbf{X}) \setminus \mathbf{S}$. Since all input shares are i.i.d., Equation (3) simplifies to:*

$$p_{\mathbf{Q},Sh(\mathbf{X})}(\mathbf{q},\mathbf{x}) = p_{\mathbf{Q},\mathbf{S}}(\mathbf{q},\mathbf{s}) \cdot p_{\overline{\mathbf{S}}}(\overline{\mathbf{s}}). \tag{6}$$

In particular, since $Sh(\mathbf{X}) = \mathbf{S} \cup \overline{\mathbf{S}}$, we can simply verify statistical independence of $\mathbf{Q} \cup \mathbf{S}$ and $\overline{\mathbf{S}}$ (with $\mathbf{x} = \mathbf{s} \cup \overline{\mathbf{s}}$).

Proof.

$$\begin{aligned}
& Pr[\mathbf{Q}|\mathbf{S}] = Pr[\mathbf{Q}|Sh(\mathbf{X})] \\
\Leftrightarrow\ & Pr[\mathbf{Q},\mathbf{S}] \cdot Pr[Sh(\mathbf{X})] = Pr[\mathbf{Q},Sh(\mathbf{X})] \cdot Pr[\mathbf{S}] \\
\overset{\text{i.i.d.}\,Sh(\mathbf{X})}{\Leftrightarrow}\ & Pr[\mathbf{Q},\mathbf{S}] \cdot Pr[\overline{\mathbf{S}}] = Pr[\mathbf{Q},Sh(\mathbf{X})] \\
\Leftrightarrow\ & p_{\mathbf{Q},\mathbf{S}}(\mathbf{q},\mathbf{s}) \cdot p_{\overline{\mathbf{S}}}(\overline{\mathbf{s}}) = p_{\mathbf{Q},Sh(\mathbf{X})}(\mathbf{q},\mathbf{x})
\end{aligned}$$

□

Algorithm 3: Explore d-Strong Non-Interference Security.

Input : $Sh(\mathbf{X})$ – Set of all shares of n secret variables $\mathbf{X}$.
Output: $\mathbf{Q}$ – Set of $d+1$ successful probes.

1 $d \leftarrow 1$
2 **while** *true* **do**
3 **foreach** *probing set* $\mathbf{Q}$ *with* $|\mathbf{Q}| = t_1 + t_2 \leq d$ *(t_1 internal, t_2 output probes)* **do**
4 $simulatable \leftarrow true$
5 **for** $t = 0$ *to* t_1 **do**
6 **foreach** $\mathbf{S} \subseteq Sh(\mathbf{X})$ *with* $|\mathbf{S}|_{\forall i} = t$ **do**
7 **if** $p_{\mathbf{Q},Sh(\mathbf{X})}(\mathbf{1},\mathbf{1}) \neq p_{\mathbf{Q},\mathbf{S}}(\mathbf{1},\mathbf{1}) \cdot p_{\overline{\mathbf{S}}}(\mathbf{1})$ **then**
8 $simulatable \leftarrow false$
9 **end**
10 **end**
11 **end**
12 **if** *not simulatable* **then**
13 **return** $\mathbf{Q}$
14 **end**
15 **end**
16 $d \leftarrow d + 1$
17 **end**

Algorithmic Verification Approach. Algorithm 2 explores and verifies d-NI for increasing d and all possible observations $\mathbf{Q}$ of at most d probes. In particular, the algorithm proceeds as soon as a successful simulation set $\mathbf{S}$ of input shares is found for the current set of probes $\mathbf{Q}$, such that $\mathbf{Q}$ is perfectly simulatable using $\mathbf{S}$. However, if the algorithm encounters a set of probes $\mathbf{Q}$ with $|\mathbf{Q}| = d+1$ which is not simulatable for set of input shares (according to the definition of NI), the algorithm terminates and returns the current set of probes indicating d-NI with $d = |\mathbf{Q}| - 1$.

4.4 *d*-Strong Non-Interference

Checking d-SNI is very similar to checking d-NI, except for stronger constraints on the simulation set $\mathbf{S}$ due to stronger distinction between internal and output probes.

Algorithmic Verification Approach. In contrast to NI, for SNI the number of shares per input in each simulation set is bounded by the number of internal probes (instead of the number of all probes). Hence, except for minor difference, the algorithmic verification approach given in Algorithm 3 (notation matching the one given in Definition 6) has the same structure as the approach for NI, but enforcing stronger constraints on the selection of shares (lines 5 and 6) for the simulation set $\mathbf{S}$.

Algorithm 4: Explore d-Probe-Isolating Non-Interference Security.

Input : $Sh(\mathbf{X})$ – Set of all shares of n secret variables $\mathbf{X}$.
Output: $\mathbf{Q}$ – Set of $d+1$ successful probes.

```
d ← 1
while true do
    foreach probing set Q with |Q| = d do
        simulatable ← true
        foreach S ⊆ Sh(X)^{I_O ∪ I_I} do
            if p_{Q,Sh(X)}(1,1) ≠ p_{Q,S}(1,1) · p_{S̄}(1) then
                simulatable ← false
            end
        end
        if not simulatable then
            return Q
        end
    end
    d ← d + 1
end
```

4.5 d-Probe-Isolating Non-Interference

For the notion of PINI, the index of any input or output wires correspond to the associated circuit share. In contrast to NI and SNI, the concept of PINI constrains the simulation set not by the number of (internal) probes, but according to the associated circuit shares.

Verification Approach. The algorithmic verification approach in order to explore and verify the security notion of PINI for increasing security order d is given in Algorithm 4. Again, the algorithm is based on the concept of perfect simulatability of every $\mathbf{Q}$ with a set $\mathbf{S}$, in conformity with the notions in Definition 7.

4.6 Uniformity

In order to examine the uniformity of the output sharing of a gadget, we start with the following observation.

Lemma 3. *Assume the function* f *with single output* $Y \in \mathbb{F}_2$ *whose shared version is realized by a gadget with* $d+1$ *output shares* $Sh(Y) = (Y^0, \ldots, Y^d)$*. The gadget's output sharing is uniform iff any selection of* d *output shares make a balanced function.*

Proof. We start with $d = 1$, i.e., 2 output shares. Let us denote the joint probability of the output shares by $\rho_{0,0} = Pr[Y^0 = 0, Y^1 = 0]$, $\rho_{0,1} = Pr[Y^0 = 0, Y^1 = 1]$, $\rho_{1,0} = Pr[Y^0 = 1, Y^1 = 0]$, and $\rho_{1,1} = Pr[Y^0 = 1, Y^1 = 1]$, assuming

that the gadget's input is uniformly distributed, which is true since the gadget's input sharing should be uniform (essential assumption of Boolean masking, see Section 2.4). Hence, the probability of the output shares can be written as

$$\begin{aligned} Pr[Y^0 = 0] &= \rho_{0,0} + \rho_{0,1}, & Pr[Y^0 = 1] &= \rho_{1,0} + \rho_{1,1}, \\ Pr[Y^1 = 0] &= \rho_{0,0} + \rho_{1,0}, & Pr[Y^1 = 1] &= \rho_{0,1} + \rho_{1,1}. \end{aligned}$$

1) If $Sh(Y)$ is uniform, $(Y^0 = 0, Y^1 = 0)$ and $(Y^0 = 1, Y^1 = 1)$ are equally likely to occur, i.e., $\rho_{0,0} = \rho_{1,1}$. The same holds for $(Y^0 = 0, Y^1 = 1)$ and $(Y^0 = 1, Y^1 = 0)$, i.e., $\rho_{0,1} = \rho_{1,0}$. Therefore, $Pr[Y^0 = 0] = \rho_{0,0} + \rho_{0,1} = \rho_{1,1} + \rho_{1,0} = Pr[Y^0 = 1]$, i.e., the gadget's coordinate function f^0 with output Y^0 is balanced. The same trivially holds for Y^1. Hence, individual balancedness of each output share is essential for uniformity.
2) If the coordinate functions of Y^0 and Y^1 are balanced, we can write

$$\begin{aligned} Pr[Y^0 = 0] = Pr[Y^0 = 1] \quad &\Longleftrightarrow \quad \rho_{0,0} + \rho_{0,1} = \rho_{1,0} + \rho_{1,1}, \\ Pr[Y^1 = 0] = Pr[Y^1 = 1] \quad &\Longleftrightarrow \quad \rho_{0,0} + \rho_{1,0} = \rho_{0,1} + \rho_{1,1}. \end{aligned}$$

These two equations directly translate into $\rho_{0,0} = \rho_{1,1}$ and $\rho_{0,1} = \rho_{1,0}$. This means that $(Y^0 = 0, Y^1 = 0)$ and $(Y^0 = 1, Y^1 = 1)$ are equally likely to occur. The same holds for $(Y^0 = 0, Y^1 = 1)$ and $(Y^0 = 1, Y^1 = 0)$, i.e., $Sh(Y)$ is a uniform sharing. Hence, individual balancedness of each output share is also a sufficient condition for uniformity.

For $d = 2$, we have $Sh(Y) = (Y^0, Y^1, Y^2)$. Assuming a uniform sharing for the gadget's input, similar to the above case for $d = 1$, we denote the joint probability of the output shares by $\rho_{y^0,y^1,y^2} = Pr[Y^0 = y^0, Y^1 = y^1, Y^2 = y^2]$, e.g., $\rho_{1,0,1} = Pr[Y^0 = 1, Y^1 = 0, Y^2 = 1]$. Exemplary, the joint probability of two output shares (Y^0, Y^1) can be derived as

$$\begin{aligned} Pr[Y^0 = 0, Y^1 = 0] &= \rho_{0,0,0} + \rho_{0,0,1}, & Pr[Y^0 = 0, Y^1 = 1] &= \rho_{0,1,0} + \rho_{0,1,1}, \\ Pr[Y^0 = 1, Y^1 = 0] &= \rho_{1,0,0} + \rho_{1,0,1}, & Pr[Y^0 = 1, Y^1 = 1] &= \rho_{1,1,0} + \rho_{1,1,1}. \end{aligned} \tag{7}$$

1) In case $Sh(Y)$ is uniform, we have

$$\rho_{0,0,0} = \rho_{0,1,1} = \rho_{1,0,1} = \rho_{1,1,0}, \qquad \rho_{0,0,1} = \rho_{0,1,0} = \rho_{1,0,0} = \rho_{1,1,1}. \tag{8}$$

Hence, it can be trivially seen that

$$Pr[Y^0 = 0, Y^1 = 0] = Pr[Y^0 = 0, Y^1 = 1] = Pr[Y^0 = 1, Y^1 = 0] = Pr[Y^0 = 1, Y^1 = 1],$$

meaning that (Y^0, Y^1) are jointly balanced. The same holds for other output shares (Y^0, Y^2) and (Y^1, Y^2).
2) If (Y^0, Y^1) are jointly balanced, all probabilities given in Equation (7) are the same, i.e.,

$$\rho_{0,0,0} + \rho_{0,0,1} = \rho_{0,1,0} + \rho_{0,1,1} = \rho_{1,0,0} + \rho_{1,0,1} = \rho_{1,1,0} + \rho_{1,1,1}.$$

The same can be written for (Y^0, Y^2) and (Y^1, Y^2) as

$$\rho_{0,0,0} + \rho_{0,1,0} = \rho_{0,0,1} + \rho_{0,1,1} = \rho_{1,0,0} + \rho_{1,1,0} = \rho_{1,0,1} + \rho_{1,1,1},$$
$$\rho_{0,0,0} + \rho_{1,0,0} = \rho_{0,0,1} + \rho_{1,0,1} = \rho_{0,1,0} + \rho_{1,1,0} = \rho_{0,1,1} + \rho_{1,1,1}.$$

Combination of these equations leads to the expressions given in Equation (8), indicating the uniformity of $Sh(Y)$.

The same procedure can be followed to trivially verify Lemma 3 for $d > 2$. □

Lemma 4. *Assume the function* f *with n-bit output* $\mathbf{Y} = (Y_0, \ldots, Y_{n-1}) \in \mathbb{F}_2^n$ *whose shared version is realized by a gadget with* $d+1$ *output shares* $Sh(\mathbf{Y}) = (\mathbf{Y^0}, \ldots, \mathbf{Y^d})$*. The gadget's output sharing is uniform iff any selection of up to* $n \cdot d$ *output shares is balanced excluding the cases where all* $d+1$ *shares of the same output are involved in the selection.*

Proof. For $n = 1$ it is the same as Lemma 3. Hence, we start with $n = 2$ and minimum number of output shares, $d + 1 = 2$. Assuming a uniform sharing for the gadget's input, we denote the joint probability of the output shares by $\rho_{(y_0^0, y_0^1),(y_1^0, y_1^1)} = Pr[Y_0^0 = y_0^0, Y_0^1 = y_0^1, Y_1^0 = y_1^0, Y_1^1 = y_1^1]$, e.g., $\rho_{(1,0),(1,1)} = Pr[Y_0^0 = 1, Y_0^1 = 0, Y_1^0 = 1, Y_1^1 = 1]$.
Exemplary, the joint probability of two output shares (Y_0^0, Y_1^1) is written as

$$\begin{aligned}
Pr[Y_0^0 = 0, Y_1^1 = 0] &= \rho_{(0,0),(0,0)} + \rho_{(0,0),(1,0)} + \rho_{(0,1),(0,0)} + \rho_{(0,1),(1,0)},\\
Pr[Y_0^0 = 0, Y_1^1 = 1] &= \rho_{(0,0),(0,1)} + \rho_{(0,0),(1,1)} + \rho_{(0,1),(0,1)} + \rho_{(0,1),(1,1)},\\
Pr[Y_0^0 = 1, Y_1^1 = 0] &= \rho_{(1,0),(0,0)} + \rho_{(1,0),(1,0)} + \rho_{(1,1),(0,0)} + \rho_{(1,1),(1,0)},\\
Pr[Y_0^0 = 1, Y_1^1 = 1] &= \rho_{(1,0),(0,1)} + \rho_{(1,0),(1,1)} + \rho_{(1,1),(0,1)} + \rho_{(1,1),(1,1)}.
\end{aligned} \tag{9}$$

1) If $Sh(\mathbf{Y})$ is uniform, we have

$$\begin{aligned}
\rho_{(0,0),(0,0)} &= \rho_{(0,0),(1,1)} = \rho_{(1,1),(0,0)} = \rho_{(1,1),(1,1)},\\
\rho_{(0,0),(0,1)} &= \rho_{(0,0),(1,0)} = \rho_{(1,1),(0,1)} = \rho_{(1,1),(1,0)},\\
\rho_{(0,1),(0,0)} &= \rho_{(0,1),(1,1)} = \rho_{(1,0),(0,0)} = \rho_{(1,0),(1,1)},\\
\rho_{(0,1),(0,1)} &= \rho_{(0,1),(1,0)} = \rho_{(1,0),(0,1)} = \rho_{(1,0),(1,0)}.
\end{aligned}$$

This results in equal probabilities for all probabilities given in Equation (9), such that

$$Pr[Y_0^0 = 0, Y_1^1 = 0] = Pr[Y_0^0 = 0, Y_1^1 = 1] = Pr[Y_0^0 = 1, Y_1^1 = 0] = Pr[Y_0^0 = 1, Y_1^1 = 1],$$

i.e., the two output shares (Y_0^0, Y_1^1) are jointly balanced. The same can be similarly verified all other combinations (Y_0^0, Y_1^0), (Y_0^1, Y_1^0), and (Y_0^1, Y_1^1).
2) If (Y_0^0, Y_1^1) are jointly balanced, based on Equation (9) we have

$$\begin{aligned}
\rho_{(0,0),(0,0)} + \rho_{(0,0),(1,0)} + \rho_{(0,1),(0,0)} + \rho_{(0,1),(1,0)} &=\\
\rho_{(0,0),(0,1)} + \rho_{(0,0),(1,1)} + \rho_{(0,1),(0,1)} + \rho_{(0,1),(1,1)} &=\\
\rho_{(1,0),(0,0)} + \rho_{(1,0),(1,0)} + \rho_{(1,1),(0,0)} + \rho_{(1,1),(1,0)} &=\\
\rho_{(1,0),(0,1)} + \rho_{(1,0),(1,1)} + \rho_{(1,1),(0,1)} + \rho_{(1,1),(1,1)} &= \quad 1/4.
\end{aligned} \tag{10}$$

This leads to

$$\begin{aligned}
&\rho_{(0,0),(0,0)} + \rho_{(0,0),(1,0)} + \rho_{(0,1),(0,0)} + \rho_{(0,1),(1,0)} \\
&\quad + \rho_{(0,0),(0,1)} + \rho_{(0,0),(1,1)} + \rho_{(0,1),(0,1)} + \rho_{(0,1),(1,1)} = \\
&\rho_{(1,0),(0,0)} + \rho_{(1,0),(1,0)} + \rho_{(1,1),(0,0)} + \rho_{(1,1),(1,0)} \\
&\quad + \rho_{(1,0),(0,1)} + \rho_{(1,0),(1,1)} + \rho_{(1,1),(0,1)} + \rho_{(1,1),(1,1)} = \quad 1/2,
\end{aligned} \tag{11}$$

meaning that $Pr[Y_0^0 = 0] = Pr[Y_0^0 = 1]$, i.e., it is balanced. The same can be written for $Pr[Y_1^1 = 0] = Pr[Y_1^1 = 1]$, and similarly for (Y_0^0, Y_1^0), (Y_0^1, Y_1^0), and (Y_0^1, Y_1^1). In short, every single output bit is balanced, hence, according to Lemma 3, the sharing of every output is individually uniform. Note that, in general when a function with d output bits is balanced, any combination of $d' < d$ output bits also makes a balanced function [34, §12.1.2].

According to Equation (9) and Equation (10), we exemplary write

$$Pr[Y_0^0 = 0, Y_1^1 = 0] = 1/4.$$

On the other hand, according to Equation (11) we have

$$Pr[Y_0^0 = 0] = 1/2, \qquad Pr[Y_1^1 = 0] = 1/2,$$

which implies $Pr[Y_0^0 = 0, Y_1^1 = 0] = Pr[Y_0^0 = 0] \cdot Pr[Y_1^1 = 0]$. The same can similarly be seen for $(Y_0^0, Y_1^1) = (0,1)$, $(1,0)$ and $(1,1)$, meaning that

$$Pr[Y_0^0, Y_1^1] = Pr[Y_0^0] \cdot Pr[Y_1^1]. \tag{12}$$

In other words, Y_0^0 and Y_1^1 are statistically independent. In a similar way, statistical independence of (Y_0^0, Y_1^0), (Y_0^1, Y_1^0), and (Y_0^1, Y_1^1) can be shown.

Now, let us denote conditional probability $Pr[Y_0^0 = y_0^0, Y_0^1 = y_0^1, Y_1^0 = y_1^0, Y_1^1 = y_1^1 | Y_0 = y_0, Y_1 = y_1]$ by $\rho_{(y_0^0,y_0^1),(y_1^0,y_1^1)|(y_0,y_1)}$. For the sharing $Sh(\mathbf{Y})$ to be uniform, according to Equation (4) and exemplary for $\mathbf{Y} = (0,0)$ we should have

$$\rho_{(0,0),(0,0)|(0,0)} = \rho_{(0,0),(1,1)|(0,0)} = \rho_{(1,1),(0,0)|(0,0)} = \rho_{(1,1),(1,1)|(0,0)} = 1/4. \tag{13}$$

The same should hold for other values of $\mathbf{Y} = (0,1)$, $(1,0)$, and $(1,1)$. Considering the statistical independence of (Y_0^1, Y_1^0) explained above, We can write

$$\begin{aligned}
\rho_{(0,0),(0,0)|(0,0)} &= Pr\left[Y_0^0{=}0, Y_0^1{=}0, Y_1^0{=}0, Y_1^1{=}0 \middle| Y_0{=}0, Y_1{=}0\right] \\
&= Pr\left[Y_0^0{=}0, Y_1^1{=}0 \middle| Y_0^1{=}0, Y_1^0{=}0, Y_0{=}0, Y_1{=}0\right] \cdot Pr\left[Y_0^1{=}0, Y_1^0{=}0 \middle| Y_0{=}0, Y_1{=}0\right] \\
&= 1 \cdot Pr\left[Y_0^1{=}0 \middle| Y_0{=}0, Y_1{=}0\right] \cdot Pr\left[Y_1^0{=}0 \middle| Y_0{=}0, Y_1{=}0\right]
\end{aligned}$$

Due to the balancedness of every individual output, we have

$$Pr\left[Y_0^1 = 0 \middle| Y_0 = 0, Y_1 = 0\right] = Pr\left[Y_0^1 = 0 \middle| Y_0 = 0, Y_1 = 0\right] = 1/2.$$

This leads to $\rho_{(0,0),(0,0)|(0,0)} = 1/4$. The same can be shown for $\rho_{(0,0),(1,1)|(0,0)}$, $\rho_{(1,1),(0,0)|(0,0)}$, and $\rho_{(1,1),(1,1)|(0,0)}$, satisfying Equation (13). The same can be similarly verified for $\mathbf{Y} = (0,1)$, $(1,0)$, and $(1,1)$, hence the uniformity of $Sh(\mathbf{Y})$.

The same procedure can be followed to verify Lemma 4 for $n > 2$ and $d > 2$. □

Indeed, for a given circuit netlist we efficiently perform balancedness checks directly based on the ROBDDs of the circuit.

5 Related Work

For formal verification of masked implementations, both in software and hardware, several tools and frameworks have been proposed, each following a different methodology and verification approach.

Formal Verification of Software Implementations. For automated masking of software implementations, the work of Moss *et al.* [38] was first to consider a type-based methodology for security annotation while dynamically repairing the masked implementation based on heuristics if leakage was detected at some point in the program flow. As any type-based approach inevitably results in an overly conservative verification, logic-based methods have been proposed as an alternative approach. Here, the work by Byrak *et al.* Byrak [6] translates verification to a set of Boolean satisfiability problems which can then be solved by a SAT solver. Nonetheless, both approaches only consider verification of masking against first-order attacks.

Later, an SMT-solver-based method for formally verifying even higher-order security has been introduced in [24]. As for [6], this verification method is also based on the notion of *perfect masking* as presented in [13]. Similarly, in [44] another method for verifying perfect masking was introduced, this time aiming to optimize the trade-off between accuracy (as offered by logic-based approaches) and efficiency (as given in type-based verification). Eventually, a composition-based verification approach in direct conformity with d-probing security (i.e., without any false negatives) is given by Belaïd *et al.* in [9].

Formal Verification of Hardware Implementations. Considering hardware designs, the work of Bloem *et al.* [11,12] resulted in a seminal tool enabling formal verification even in the presence of glitches, but with restriction to verification of probing security only. Most recently, the work of Cassiers *et al.* [16] proposes a composition-based approach of verifying probing security of a concrete implementation composed of so-called *Hardware Private Circuits*.

Besides, the latest version of `maskVerif` – as presented in [2] – supports efficient verification of d-probing security, d-NI, and d-SNI for arbitrary orders for both software and hardware designs, even in the presence of glitches. Currently, `maskVerif` is the state-of-the-art tool offering the widest-ranging verification features which is not composition-based, hence, in the following we provide a more detailed discussion and comparison to our developed tool.

5.1 Comparison to `maskVerif`

In general, `maskVerif` offers an efficient approach to verify security of masked software and hardware implementations. In contrast to our approach, `maskVerif`

utilizes a symbolic representation of leakage defined by a given syntax and semantic. For verifying security, the tool first assigns a symbolic leakage set to every instruction. Depending on the security order, each combination of symbolic leakage sets, i.e., each possible observation, is exploited afterwards and tested for the absence of secret dependency through performing a syntactical check and applying semantic-preserving transformation on the sets.

Due to its language-based verification approach, security checks in `maskVerif` follow a very conservative approach for particular designs. More precisely, it may falsely reject some secure designs because the checks are not based on explicit statistical properties in conformity with the actual definition of the security notions. Due to these limitations of a purely syntactical verification, it more likely fails to provide correct verification of probing security if an output of a masked circuit is not non-complete (as used for TIs) but also does not rely on fresh randomness. In other words, its computation is a result of all input shares of at least one input without using any fresh randomness for blinding purposes. In particular, a computation using all input shares not necessarily implies statistical dependency on the corresponding input (e.g., due to blinding with shares of different inputs). Nonetheless, since the verification approach of `maskVerif` is mainly based on syntactical checks, it may falsely categorize the design as not being probing secure although it is (i.e., resulting in false negative).

Examples for False Negatives in `maskVerif`. One small example is a shared version of the 4-bit bijection quadratic class $\mathcal{Q}_{12}^4$ (based on the classification given in [10]), utilizing two shares per input, as presented in Appendix of [42]. Using `maskVerif`, this design is falsely categorized as not being first-order probing secure although all possible probes are statistically independent of the secrets. Hence, according to `maskVerif`, in order to gain successful verification, one possible solution would be to introduce additional randomness $r \in \mathbb{F}_2$ into the design, such that:

$$
\begin{aligned}
x_1 &= F_1(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_1 \\
x_2 &= F_2(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_2 \\
y_1 &= G_1(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_1c_1 \oplus b_1 \oplus r & \bar{x}_1 &= x_1 \\
y_2 &= G_2(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_1c_2 & \bar{x}_2 &= x_2 \\
y_3 &= G_3(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_2c_1 \oplus b_2 \oplus r & \bar{y}_1 &= y_1 \oplus y_2 \\
y_4 &= G_4(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_2c_2 & \bar{y}_2 &= y_3 \oplus y_4 \\
z_1 &= H_1(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_1b_1 \oplus a_1c_1 \oplus c_1 & \bar{z}_1 &= z_1 \oplus z_2 \\
z_2 &= H_2(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_1b_2 \oplus a_1c_2 \oplus r & \bar{z}_2 &= z_3 \oplus z_4 \\
z_3 &= H_3(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_2b_1 \oplus a_2c_1 \oplus r & \bar{t}_1 &= t_1 \\
z_4 &= H_4(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = a_2b_2 \oplus a_2c_2 \oplus c_2 & \bar{t}_2 &= t_2 \\
t_1 &= K_1(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = d_1 \\
t_2 &= K_2(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = d_2
\end{aligned}
$$

This new realization of $\mathcal{Q}_{12}^4$ is now correctly verified by `maskVerif` as being first-order probing secure. However, introducing randomness is costly and not necessary to gain independence of the secret input, i.e., fulfilling first-order probing security.

However, this given example based on $\mathcal{Q}_{12}^4$ is only a small design. For larger and more complex circuits, this inaccurate determination of the security level will lead to significantly more overhead being introduced during the design process. An example for a more complex design, which is falsely classified as not begin first-order probing secure, is the `PRESENT` S-box realized as a TI utilizing three shares for every output and input bit as presented in [40].

In fact, in order to achieve a sufficient security level while only introducing marginal overhead into the design, it is thus necessary to be in conformity with the security notions. As our verification is based on actual statistical properties between probes and inputs, i.e., in accordance with the formal definitions of the security notions, we actually meet this need and completely avoid false negatives. This eventually is expected to result in less overhead in terms of area and randomness when designing and implementing masked implementations. Moreover, and in addition to features in `maskVerif`, our tool is extended to verify dth-order PINI and the output uniformity of a given design while also returning the first probe combination found which is not in conformity with the respective security notion.

Hence, despite being slower and slightly less efficient for larger design compared to a type-based approach, as used for instance in `maskVerif`, our tool is assumed to close the gap between accuracy and efficiency by providing a complete and sound verification framework for the security and composability of both software and hardware designs.

6 Experiments and Evaluations

This section presents implementation, evaluation, and performance results of our proposed tool for formal verification of masked circuits.

Implementation. For a practical evaluation of our proposed concepts and methodologies, we opted to implement a formal verification tool using `Sylvan` [22], a state-of-the-art BDD high-performance, multi-core decision diagram package implemented in C/C++. Further, we also customized and extended the native instructions of `Sylvan` in order to provide and support dedicated operations computing $p_{\mathbf{X}}(\mathbf{1})$ and $p_{\mathbf{X},\mathbf{Y}}(\mathbf{1},\mathbf{1})$ based on [35], i.e., without formal construction of new BDDs each time these operations are executed. Eventually, our framework implements all verification algorithms presented in Section 4 for both, standard and robust probing model, and is running in a 64-bit Linux Operating System (OS) environment on an Intel Xeon E5-1660v4 CPU with a clock frequency of 3.20 GHz and 128 GB of Random-Access Memory (RAM).

Our tool process a netlist file as the specification of the CUT. The user can either make such a netlist manually, e.g., for software applications or a

Table 1. Verification of Various Masked Circuits and Security Notions.

Scheme	Pos.†	d	Probing		NI		SNI		PINI		Unif.
			std.	rob.	std.	rob.	std.	rob.	std.	rob.	
Gadgets											
DOM1 [29]	19	1	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
DOM2 [29]	42	2	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
DOM3 [29]	74	3	✓[0.2 s]	✓[1.2 s]	✓[2.5 s]	✓[24.4 s]	✓[3.7 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
DOM1 SNI	21	1	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
DOM2 SNI	45	2	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
DOM3 SNI	78	3	✓[0.1 s]	✓[1.5 s]	✓[2.4 s]	✓[39.4 s]	✓[3.7 s]	✓[39.4 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
PARA1 [5]	22	1	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
PARA2 [5]	45	2	✓[0.0 s]	✓[0.0 s]	✓[0.1 s]	✓[0.1 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
PARA3 [5]	68	3	✓[0.1 s]	✓[0.5 s]	✓[1.6 s]	✓[12.1 s]	✗/✓[0.8 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
PARA3 SNI [5]	82	3	✓[0.2 s]	✓[1.2 s]	✓[2.8 s]	✓[33.0 s]	✓[4.1 s]	✓[38.7 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
PINI1 [17]	21	1	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]
PINI2 [17]	51	2	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]
CMS3 [36]	104	3	✗/✓[0.2 s]	✗/✓[0.4 s]	✗/✓[1.2 s]	✗/✓[2.9 s]	✗/✓[1.7 s]	✗/✓[4.6 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
UMA2 [36]	81	2	✗/✓[0.0 s]	✗/✓[0.0 s]	✗/✓[0.0 s]	✗/✓[0.0 s]	✗/✓[0.0 s]	✗/✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
DOM2 DEP‡ [36]	56	2	✓[0.0 s]	✗/✓[0.0 s]	✓[0.0 s]	✗/✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✓[0.0 s]	✗/✓[0.0 s]	✓[0.0 s]
S-boxes											
$\text{PRESENT}_{\text{TI}}$ [40]	177	2	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.1 s]
$\text{PRESENT}_{\text{TI}}$ [25]	377	2	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.1 s]
$\text{PRESENT}_{\text{TI}}$ [25]	161	2	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]
$\text{PRINCE}_{\text{TI}}$ [37]	150	2	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.1 s]
$\text{PRINCE}_{\text{CMS}}$ [14]	261	1	✓[3.1 s]	✓[0.1 s]	✓[0.0 s]	✓[2.6 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
$\text{SKINNY8}_{\text{TI}}$ [7]	240	2	✓[16.4 s]	✓[46.8 s]	✗[2 min]	✗[0.4 s]	✗[2 min]	✗[0.4 s]	✗[16.3 s]	✗[0.1 s]	✓[27.9 s]
$\text{SKINNY8}_{\text{CMS}}$ [8]	192	1	✓[0.1 s]	✓[0.1 s]	✗[0.3 s]	✗[0.0 s]	✗[0.2 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]
AES_{DOM} [29]	885	1	✓[3.1 s]	✓[21 min]	✗[0.8 s]	✗[0.3 s]	✗[0.6 s]	✗[0.2 s]	✗[0.1 s]	✗[0.0 s]	✓[18.8 s]
AES_{CMS} [19]	938	1	✓[4.5 s]	✓[2.9 h]	✗[0.9 s]	✗[0.3 s]	✗[0.6 s]	✗[0.2 s]	✗[0.1 s]	✗[0.0 s]	✓[86.1 s]
Functions											
A_{in} [37]	18	1	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]
A_{m} [37]	20	1	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]
A_{out} [37]	20	1	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]	✓[0.0 s]	✓[0.0 s]
$\mathcal{Q}_{12}^4$ [42]	48	1	✓[0.0 s]	✓[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✗[0.0 s]	✓[0.0 s]

† Number of possible probe positions, i.e., output wires of gates. ‡ Assuming identical inputs, i.e., $a = b$.

sequence of operations, or can provide a verilog file as the result of a hardware synthesis, e.g., Design Compiler or Yosys[1], using a restricted library (defined in Section 2.3). It is beneficial to directly evaluate the circuit's netlist as any user-originated mistakes or flaws (e.g., not keeping design hierarchy, hence violating non-completeness [39]) can be detected.

Experiments and Benchmarks. In Table 1, we summarize verification and performance results for our tool using various different examples as a benchmark. For this, the number d indicates the masking order of the circuit design (i.e., the number of input shares given as $d + 1$), while the number next to the tick indicates the maximum security order found by our tool during security check and verification (i.e., the number of probes that did not lead to a failing check).

[1] `http://www.clifford.at/yosys/`

For all designs, we provide analysis results for the security notions of d-probing, NI, SNI, PINI, and uniformity of the output sharing. Except for uniformity, all security checks are performed for the standard (i.e., without physical defaults in terms of *glitches*) and robust (i.e., with *glitches*) leakage models as presented in Section 3. Eventually, along with the number of potential probe positions, i.e., the number of distinct wires determined by the number of gates in the circuit, the security parameter d yields the verification complexity in terms of possible observations $\mathcal{O} = \sum_{i=1}^{d} \binom{pos}{i}$.

Examples. In Table 1, we list verification results for three different categories of masked circuits. In the first category, denoted as **Gadgets**, we analyze different variants to implement a masked field multiplication for $\mathbb{F}_2$. Note, that for the SNI variant of Domain-Oriented Masking (DOM) multiplier [29], we simply added additional registers at the output to achieve an SNI-secure circuit. Interestingly, PARA1 [5] and PARA2 gadgets are up to d-SNI secure in both models, but higher-order variants cannot achieve full security, and need design modifications instead (although still SNI for smaller d). We should stress that `maskVerif` reports PARA3 to be not SNI, while it is up to 2-SNI, which is correctly reported by our tool. Also, our tool could identify and report all flaws described in [36] including the probes as identified by the authors. Our second category lists different masked **S-boxes** of lightweight and standard block ciphers implemented following the concepts of Consolidating Masking Schemes (CMS) [42], TI [39], or DOM [29]. Eventually, our last category **Functions** lists arbitrary masked functions with linear or quadratic algebraic complexity.

Interestingly, non of the designs, except the linear functions, are secure in the robust, glitch-extended probing model under the notion of PINI. Since PINI gadgets [17] are even not robust probing secure, they are mainly useful in software applications (i.e., standard probing model). Indeed, a question is raised whether it is generally possible to build a non-linear gadget which is PINI-secure considering robust probing model. Also, besides $\mathcal{Q}_{12}^4$ we also analyzed other quadratic functions provided in [42] and our tool revealed that the implementation of $\mathcal{Q}_{300}^4$ as given by the authors is not uniform.

Verification Complexity. In contrast to the language-based verification approach of `maskVerif`, our framework heavily relies on statistical independence verification of probability distributions in order to avoid false negatives. Therefore, the overall run time of our verification approach is mainly governed by construction of intermediate ROBDDs representing the logical conjunctions as part of the statistical independence checks for the security notions. As already shown in [41], the complexity of constructing ROBDDs increases mainly by the number of product terms occurring in the minimal Disjunctive Normal Form (DNF) of the represented Boolean function.

Generally speaking, when considering higher-order security verification, we have to test for statistical independence of larger sets of random variables with possible non-linear dependence on many of the inputs. As our test of statistical

independence is based on logical conjunctions of sets of random variables (and every possible subset), this leads to a high number of product terms occurring in the resulting DNF, and hence to an increased complexity of the constructed ROBDDs. As a result, verification speed of our framework is mainly influenced by the complexity, i.e., input dependencies of wires, and the maximum security order of the CUT.

Further, with increasing security order, the combinatorial complexity $\mathcal{O}$ of constructing all possible observations grows exponentially. However, as we opted for accurate security verification without relying on heuristics, reducing the number of probe combinations is not trivial, but instead we have to check and verify all of them. Although some joint distributions might be similar for different probe combinations[2], we still have to analyze most combinations which is rather time consuming for higher security orders and larger circuits. It is worth to mention that if any of the combinations leads to a negative statistical independence, the tool stops and reports the found leaking probes. Hence, the maximum run time is taken only if the CUT passes all desired security checks.

7 Conclusion

In this work, we developed and presented a sound and accurate framework to verify the security and composability of masked gate-level netlists and circuits directly resulting from hardware logic synthesis processes. In particular, our approach enables formal verification of all pertinent security notions in the domain of physical security and is applicable to both, software and hardware designs, even considering physical defaults in terms of *glitches*. More concretely, it supports sound, accurate, and immediate verification whether a masked implementation provides probing security, Non-Interference, Strong Non-Interference, and Probe-Isolating Non-Interference – even for higher security orders. In addition, we proposed and integrated a novel methodology of verifying uniformity of the output sharing of a masked gadget. Eventually, if verification fails, it reports the failing set of probes being in non-conformity with the corresponding security notion.

In contrast to common type-based methods, our approach is based on formal verification of statistical properties in direct conformity with the fundamental definitions of the security notions. As a result, our approach completely avoids overly conservative decisions when falsely declaring designs as not being secure (false negatives), ultimately leading to a reduction in design overhead as otherwise introduced by additional (and expensive) fresh randomness. For this, all verification checks of statistical properties are executed efficiently by reducing statistical independence checks on joint distributions over multiple binary random variables to checks of distributions over single binary random variables, which can be efficiently done utilizing the concepts of ROBDDs. Eventually, this

[2] This case is caught by the internal caching scheme of the `Sylvan` BDD package which first checks if the current operation has been performed and cached recently before executing the actual operation in case no cache entry was found.

results in a framework exceeding comparable tools in accuracy and functionality while still being reasonable efficient for most applications and common use cases.

The current version of our tool is mainly beneficial to evaluate gadgets, particularly at higher orders, although we have given its capability to examine the entire S-boxes (see Table 1). For future work, we will focus on extending capabilities and improving efficiency of our tool, mainly with respect to larger and more complex circuits and implementations and higher security orders. For this, distinguishing univariate and multivariate leakages would be interesting, as it would allow *divide-and-conquer* approaches based on partitioning complex circuits along register stages while security analysis then would be performed an smaller circuits automatically. Certainly, verification then can be performed more efficiently, even for large and complex designs and higher-orders as long as the design is not entirely combinational but contains register stages. The future version of our tool should receive the netlist of a complete cipher implementation, unroll the loops, divide it into separate gadgets, and conduct security evaluation respectively.

Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and through the project 393207943 "Security for Internet of Things with Low Energy and Low Power Consumption (GreenSec).

References

1. Akers, S.B.: Binary Decision Diagrams. IEEE Trans. Computers **27**(6), 509–516 (1978)
2. Barthe, G., Belaïd, S., Cassiers, G., Fouque, P., Grégoire, B., Standaert, F.: maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In: Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11735, pp. 300–318. Springer (2019)
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P.: Verified Proofs of Higher-Order Masking. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 457–485. Springer (2015)
4. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 116–129. ACM (2016)
5. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., Strub, P.: Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10210, pp. 535–566 (2017)

6. Bayrak, A.G., Regazzoni, F., Novo, D., Ienne, P.: Sleuth: Automated Verification of Software Power Analysis Countermeasures. In: Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Proceedings. Lecture Notes in Computer Science, vol. 8086, pp. 293–310. Springer (2013)
7. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)
8. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: SKINNY-AEAD and SKINNY-Hash. IACR Trans. Symmetric Cryptol. **2020**(special), 1–44 (2020)
9. Belaïd, S., Goudarzi, D., Rivain, M.: Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In: Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 343–372. Springer (2018)
10. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N.N., Vitkup, V.: Threshold implementations of small s-boxes. Cryptogr. Commun. **7**(1), 3–33 (2015)
11. Bloem, R., Groß, H., Iusupov, R., Könighofer, B., Mangard, S., Winter, J.: Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In: Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 321–353. Springer (2018)
12. Bloem, R., Groß, H., Iusupov, R., Krenn, M., Mangard, S.: Sharing independence & relabeling: Efficient formal verification of higher-order masking. IACR Cryptol. ePrint Arch. **2018**, 1031 (2018), `https://eprint.iacr.org/2018/1031`
13. Blömer, J., Guajardo, J., Krummel, V.: Provably secure masking of AES. IACR Cryptol. ePrint Arch. **2004**, 101 (2004), `http://eprint.iacr.org/2004/101`
14. Bozilov, D., Knezevic, M., Nikov, V.: Optimized threshold implementations: Minimizing the latency of secure cryptographic accelerators. In: Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11833, pp. 20–39. Springer (2019)
15. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986)
16. Cassiers, G., Grégoire, B., Levi, I., Standaert, F.: Hardware private circuits: From trivial composition to full verification. IACR Cryptol. ePrint Arch. **2020**, 185 (2020), `https://eprint.iacr.org/2020/185`
17. Cassiers, G., Standaert, F.: Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. IEEE Trans. Information Forensics and Security **15**, 2542–2555 (2020)
18. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999)
19. Cnudde, T.D., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with d+1 shares in hardware. In: Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Proceedings. Lecture Notes in Computer Science, vol. 9813, pp. 194–212. Springer (2016)

20. Coron, J.: Formal Verification of Side-Channel Countermeasures via Elementary Circuit Transformations. In: Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Proceedings. Lecture Notes in Computer Science, vol. 10892, pp. 65–82. Springer (2018)
21. De Meyer, L., Bilgin, B., Reparaz, O.: Consolidating Security Notions in Hardware Masking. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(3), 119–147 (2019)
22. van Dijk, T.: Sylvan: multi-core decision diagrams. Ph.D. thesis, University of Twente, Enschede, Netherlands (2016), `http://purl.utwente.nl/publications/100676`
23. Duc, A., Dziembowski, S., Faust, S.: Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In: Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 423–440. Springer (2014)
24. Eldib, H., Wang, C., Schaumont, P.: Formal Verification of Software Countermeasures against Side-Channel Attacks. ACM Trans. Softw. Eng. Methodol. **24**(2), 11:1–11:24 (2014)
25. Ender, M., Ghandali, S., Moradi, A., Paar, C.: The first thorough side-channel hardware trojan. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 755–780. Springer (2017)
26. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 89–120 (2018)
27. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 251–261. Springer (2001)
28. Groß, H., Mangard, S.: A Unified Masking Approach. J. Cryptographic Engineering **8**(2), 109–124 (2018)
29. Groß, H., Mangard, S., Korak, T.: An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In: Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10159, pp. 95–112. Springer (2017)
30. Hutter, M., Schmidt, J.: The Temperature Side Channel and Heating Fault Attacks. In: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8419, pp. 219–235. Springer (2013)
31. Ishai, Y., Sahai, A., Wagner, D.A.: Private Circuits: Securing Hardware against Probing Attacks. In: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer (2003)
32. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Proceedings. Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer (1996)
33. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Pro-

ceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999)
34. Mesnager, S.: Bent Functions - Fundamentals and Results. Springer (2016)
35. Miller, D.M.: An improved method for computing a generalized spectral coefficient. IEEE Trans. on CAD of Integrated Circuits and Systems **17**(3), 233–238 (1998)
36. Moos, T., Moradi, A., Schneider, T., Standaert, F.: Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 256–292 (2019)
37. Moradi, A., Schneider, T.: Side-Channel Analysis Protection and Low-Latency in Action - - Case Study of PRINCE and Midori -. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 517–547 (2016)
38. Moss, A., Oswald, E., Page, D., Tunstall, M.: Compiler Assisted Masking. In: Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Proceedings. Lecture Notes in Computer Science, vol. 7428, pp. 58–75. Springer (2012)
39. Nikova, S., Rijmen, V., Schläffer, M.: Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. J. Cryptology **24**(2), 292–321 (2011)
40. Poschmann, A., Moradi, A., Khoo, K., Lim, C., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2, 300 GE. J. Cryptology **24**(2), 322–345 (2011)
41. Raseen, M., Prasad, P.W.C., Assi, A.: An efficient estimation of the ROBDD's complexity. Integr. **39**(3), 211–228 (2006)
42. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating Masking Schemes. In: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 764–783. Springer (2015)
43. Trichina, E.: Combinational Logic Design for AES SubByte Transformation on Masked Data. IACR Cryptol. ePrint Arch. **2003**, 236 (2003), `http://eprint.iacr.org/2003/236`
44. Zhang, J., Gao, P., Song, F., Wang, C.: SCInfer: Refinement-Based Verification of Software Countermeasures Against Side-Channel Attacks. In: Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10982, pp. 157–177. Springer (2018)