Formal Verification of Masked Hardware Implementations in the Presence of Glitches

Roderick Bloem([✉]), Hannes Gross, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{roderick.bloem,hannes.gross,rinat.iusupov,bettina.konighofer,
stefan.mangard,johannes.winter}@iaik.tugraz.at

Abstract. Masking provides a high level of resistance against side-channel analysis. However, in practice there are many possible pitfalls when masking schemes are applied, and implementation flaws are easily overlooked. Over the recent years, the formal verification of masked software implementations has made substantial progress. In contrast to software implementations, hardware implementations are inherently susceptible to glitches. Therefore, the same methods tailored for software implementations are not readily applicable.

In this work, we introduce a method to formally verify the security of masked hardware implementations that takes glitches into account. Our approach does not require any intermediate modeling steps of the targeted implementation. The verification is performed directly on the circuit's netlist in the probing model with glitches and covers also higher-order flaws. For this purpose, a sound but conservative estimation of the Fourier coefficients of each gate in the netlist is calculated, which characterize statistical dependence of the gates on the inputs and thus allow to predict possible leakages. In contrast to existing practical evaluations, like t-tests, this formal verification approach makes security statements beyond specific measurement methods, the number of evaluated leakage traces, and the evaluated devices. Furthermore, flaws detected by the verifier are automatically localized. We have implemented our method on the basis of a SAT solver and demonstrate the suitability on a range of correctly and incorrectly protected circuits of different masking schemes and for different protection orders. Our verifier is efficient enough to prove the security of a full masked first-order AES S-box, and of the Keccak S-box up to the third protection order.

Keywords: Masking · Formal verification
Threshold implementations · Hardware security
Side-channel analysis · Private circuits

J. B. Nielsen and V. Rijmen (Eds.): EUROCRYPT 2018, LNCS 10821, pp. 321–353, 2018.
https://doi.org/10.1007/978-3-319-78375-8_11

1 Introduction

Security critical embedded systems rely on the protection of sensitive information against exposure. While the transmission of sensitive information over conventional communication channels can be protected by means of strong cryptography, the protection against unintentionally created side channels, like power consumption [28] or electromagnetic emanation [33], requires additional countermeasures.

Since the risk of side-channel analysis (SCA) is inevitable in many applications, different countermeasures have been proposed in the past. One of the best researched and most effective countermeasures against SCA is masking. Many different masking schemes have been introduced [24,26,31,35,37] over the years. The history of masking, however, is also a history of failure and learning. Some of the first masking schemes were shown to be insecure in practical implementations because glitches in the combinatorial logic (temporary logic states caused by propagation time differences of the driving signals) were not considered in the formal models [26,37]. The first provably secure masking scheme with inherent resistance to glitches was the threshold implementation (TI) scheme of Nikova et al. [31]. Over the last years the TI scheme has been extended further by Bilgin et al. [12], and other schemes have been proposed like the consolidated masking scheme (CMS) of Reparaz et al. [35], and the domain-oriented masking scheme (DOM and the low-randomness variant UMA) of Gross et al. [23,24].

Even if the used masking scheme is secure, this is not automatically true for the implementations that rely on this scheme. One problem that thus still remains is the verification of masked implementations. There are basically two approaches that are used in practice to verify the resistance against SCA, namely formal verification and empirical leakage assessment. The predominant approach for the verification of masked hardware implementations is still the empirical leakage assessment in form of statistical significance tests [21] or by attacking the devices by using state-of-the-art side-channel analysis techniques. However, such practical evaluations are never complete, in a sense that if no flaw is found it remains uncertain whether or not the implementation could be broken with a better measurement setup or more leakage traces.

Recently there has been some substantial development towards the formal verification for masked software implementations [3,7,17]. However, these verification methods are tailored to software and do not take glitches into account. Therefore, they cannot readily be applied to hardware implementations. In terms of non-empirical verification of hardware implementations, there exist tools to test for leakage by either modeling of the circuit in software [34] or approaches that simulate possible leakages by assuming a specific power model [9]. To the best of our knowledge there exist no formal tools that take glitches into account and directly prove the security of masked hardware implementations on its netlist.

Our contribution. In this work, we introduce a method to formally prove the security of masked hardware implementations in the presence of glitches. In contrast to existing formal or non-empirical verification approaches for hardware designs, the introduced approach does not require any additional modeling of the circuit or the leakage source and proves the security of a circuit directly on its netlist. Compared to empirical verification methods based on the statistical analysis of leakage traces, our formal approach allows direct localization of the detected flaws, and gives conclusive security statements that are independent of device- or measurement-specific conditions, or the amount of gathered leakage information.

We base our approach on the probing model of Ishai et al. [26] and take the effects of glitches into account. We introduce a circuit verification method that performs a conservative estimate of the data that an attacker can learn by probing different gates and wires. The verification works directly on the gate-level representation of the circuit. It uses the Fourier expansion (or Walsh expansion) of the functions that are computed and uses the fact that a non-zero Fourier coefficient for a linear combination of variables indicates a correlation between the function and these variables (cf. [10]). A correlation with a linear combination of variables that contains secrets but no uniformly distributed masking variables corresponds to an information leak. By only keeping track of whether coefficients are zero or not, we circumvent the complexity of a full Fourier representation of all functions computed by all gates of the circuit, at the cost of a loss of precision that may lead to false alarms. The information of whether a coefficient is zero or not can be encoded as a propositional logic formula whose size is linear in the size of the circuit and vulnerability can be computed efficiently by a SAT solver.

To show the practicality of this approach, we check a variety of masked circuits that originate from different masking schemes. We focus on acyclic (feedback free) pipelined masked circuits, like the S-boxes of symmetric primitives which are usually the parts of a circuit that are the hardest to protect in practice and therefore most susceptible to flaws. The security of the linear circuits parts, on the other hand, can be established and verified more easily in practice, for instance by ensuring that only one masked value or mask of one variable is used inside each of the linear circuit parts [24]. For the same reason multiple cipher rounds and S-box lookups can be analyzed separately, as long as it is ensured that the masked outputs of the nonlinear parts are always independently and freshly masked (which is the case for most masking schemes).

We ran our tool on a set of example circuits including the S-boxes of Keccak, Fides and AES. Our verifier is efficient enough to formally prove the resistance of a full first-order masked AES S-box. Because of the circuit size of the AES S-box, which consumes about 40% of the entire AES area [24], the parallelized evaluation takes about 10 h. We also prove a Keccak S-box up to order three, and the $GF(2)$ multipliers of DOM up to order four. Furthermore, we show that our approach correctly detects flaws in masked circuits that are known to be flawed in the presence of glitches e.g. [26,37]. The implementation of our tool and some example circuits are available on github [27].

This paper is organized as follows. In Sect. 2 we give a short overview of existing verification approaches and discuss the differences to our approach. In Sect. 3, we introduce the used notation and the Fourier expansion. We give an introduction to masked circuits and the probing model in Sect. 4, and show how to leverage the Fourier expansion to test for probing security. We start the introduction of our verification approach in Sect. 5, at first without taking signal timings into account. Before we complete the description of our general verification approach in Sect. 7, we first discuss in Sect. 6 how we model timing effects i.e. glitches. A concrete instantiation of our verification approach based on a SAT solver is then introduced in Sect. 8. Evaluation results for various masked circuits are discussed in Sect. 9. We conclude this work in Sect. 10.

2 Related Work

Automated verification of masked implementations has been intensively researched over the last years and recently many works targeting this topic have been published [1,6,7,9,16–18,30]. Most of the existing work, however, targets software based masking which does not include the effects of glitches.

Verification of masked software. One of the most groundbreaking works towards the efficient verification of masked software implementations is the work of Barthe et al. [3]. Instead of proving the security of a whole implementation at once, this work introduces the notion of strong non-interference (SNI). SNI is an extension to the more general non-interference (NI) notion introduced in [2]. The SNI notion allows to prove the security of smaller code sequences (called gadgets) in terms of composability with other code parts. Gadgets fulfilling this SNI notion can be freely composed with other gadgets without interfering with their SCA resistance.

Verification of algorithms that fulfill this notion scale much better than other approaches but, on the other hand, not all masking algorithms that are secure are also directly composable. As a matter of fact the most efficient software masking algorithms in terms of randomness of Belaid et al. [7,8], Barthe et al. [4], and Gross et al. [23], for example, do not achieve SNI directly.

In contrast to Barthe et al.'s work on SNI [3], our approach does not check for composability and is therefore less restrictive to the circuits and masking schemes that can be proven (similar to the NI approach of Barthe et al.' [2]). Since Barthe et al.'s work is designed to prove masked software implementations it does not cover glitches. In our work we introduce the necessary formal groundwork for the verification of masked circuits and in particular the propagation of glitches. Our approach is thereby not bound to our SAT realization but is also compatible with existing tools like easycrypt which is developed by Barthe et al. [5].

Most recently another formal verification approach by Coron [14] was introduced that builds on the work of Barthe et al.. Essentially two approaches are discussed in this work. The first approach is basically the same as the approach in [2] but written in Common Lisp language. The second approach is quite different and works by using elementary transformations in order to make the targeted program verifiable using the NI and SNI properties. The work of Coron targets again only software based masking and does not take glitches into account.

Eldib et al. [17] present an approach to verify masked software implementations. Similar to our approach, the verification problem is encoded into SMT and verified by checking the constraints for individual nodes (operations) inside the program. This approach allows direct localization of the vulnerable code parts.

However, their approach targets software and therefore does not cover glitches. It also produces SMT formulas that are exponential in the number of secret variables, whereas the formulas that are produced by our approach are only linear.

Bhasin et al. [10] also use Fourier transforms to estimate the side channel attack resistance of circuits. Their approach uses a SAT solver to construct low-weight functions of a certain resistance order. They have not used their approach to evaluate existing implementations of cryptographic functions, and they do not take glitching behavior into account.

Verification of masked hardware. Similar to our approach, Bertoni et al. [9] address verification of masked hardware implementations in the presence of glitches. In this work all possible transients at the input of a circuit are considered and all resulting glitches that could occur at the gates are modeled. However, this approach focuses on first-order masking of purely combinatorial logic and uses a rather simple power model to measure the impact (transitions from 0 to 1 result in the same power consumption as transitions from 1 to 0). We note that focusing on combinatorial logic only, leaves out most of the existing hardware-based masking schemes such as [23,24,31,35]. Bertoni et al. demonstrated their approach on a masked implementation of Keccak based on a masking scheme that is known to be insecure in the presence of glitches.

In contrast to Bertoni et al.'s work, our approach considers combinatorial logic as well as sequential gates (registers), covers also higher-order leakages, and is not restricted to circuits with only one output bit.

In the work of Reparaz [34], a leakage assessment approach is introduced that works by simulating leakages of a targeted hardware implementation in software. At first, a high-level model of the hardware implementation is created, and the verification then works by simulating the model with different inputs and extracting leakage traces. The verification result is gathered by applying statistical significance tests (t-tests) to the simulated leakage traces. Compared to our approach, the leakage detection approach of Reparaz does not perform a formal verification but an empirical leakage assessment. Furthermore, the verification is not directly performed on the targeted hardware implementation but requires to model its (leakage) behavior in software.

Most recently a paper by Faust et al. [19] was published that introduces the so-called robust-probing model as extension to the original probing model with regard to glitches. They build upon the work of Barthe et al. [3] and target the verification of the SNI notion in their extended probing model. In contrast to Faust et al.'s approach, our formal verification approach does not strife for the verification of the SNI notion which trades higher randomness and implementations costs against faster verification. Furthermore, to the best of our knowledge, there exists no implementation of their verification approach in form of a tool to check real hardware circuits.

3 Preliminaries

In the following we make extensive use of the usual set notation, where $S \triangle T = S \setminus T \cup T \setminus S$ denotes the symmetric difference of S and T and for two sets of sets $\mathcal{S}$ and $\mathcal{T}$, we define $\mathcal{S} \triangle \mathcal{T} = \{S \triangle T \mid S \in \mathcal{S}, T \in \mathcal{T}\}$ to be the pointwise set difference of all elements. We write $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ for the set of Booleans. For a set X of Boolean variables, we identify an assignment $f : X \to \mathbb{B}$ with the set of variables x for which $f(x) = \mathsf{true}$. For a Boolean function $f(X, Y)$ and an assignment $x \subseteq X$, we write $f|_x$ to denote the function $f|_x(y) = f(x, y)$.

Fourier expansion of Boolean functions. There is a close connection between statistical dependence and the Fourier expansion of Boolean functions. First we formally define statistical independence.

Definition 1 (Statistical independence). *Let X, Y, and Z be sets of Boolean variables and let $f : 2^X \times 2^Y \to 2^Z$. We say that* f *is statistically independent of X if for all z there is a c such that for all x we have $|\{y \mid f(x, y) = z\}| = c$.*

Lemma 2. *Let $F : \mathbb{B}^X \times \mathbb{B}^Y \to \mathbb{B}^Z$. Function* F *is statistically independent of X iff for all functions $f : \mathbb{B}^Z \to \mathbb{B}$ we have that $f \circ F$ is statistically independent of X.*

Please find the proof in Appendix A. To define Fourier expansions, we follow the exposition of [32] and associate true with -1 and false with 1. We can then represent a Boolean function as a multilinear polynomial over the rationals.

Definition 3 (Fourier expansion). *A Boolean function $f : \{-1, 1\}^n \to \{-1, 1\}$ can be* uniquely *expressed as a multilinear polynomial in the n-tuple of variables $X = (x_1, x_2, \ldots, x_n)$ with $x_i \in \{\pm 1\}$, i.e., the multilinear polynomial of f is a linear combination of monomials, called* Fourier characters*, of the form $\chi_T(X) = \prod_{x_i \in T} x_i$ for every subset $T \subseteq X$. The coefficient of $\chi_T \in \mathbb{Q}$ is called the* Fourier coefficient $\widehat{f}(T)$ *of the subset* T*. Thus we have the* Fourier representation *of f:*

$$f(X) = \sum_{T \subseteq X} \widehat{f}(T) \chi_T(X) = \sum_{T \subseteq X} \widehat{f}(T) \prod_{x_i \in T} x_i.$$

The Fourier characters $\chi_T : \{-1, 1\}^n \to \{-1, 1\}$ form an orthonormal basis for the vector space of functions in $f : \{-1, 1\}^n \to \{-1, 1\}$. The Fourier coefficients are given by the projection of the function to its basis, i.e., for $f : \{-1, 1\}^n \to \{-1, 1\}$ and $T \subseteq X = (x_1, x_2, \ldots, x_n)$, the coefficient $\widehat{f}(T)$ is given by $\widehat{f}(T) = 1/2^n \cdot \sum_{X \in \{\pm 1\}^n} (f(X) \cdot \chi_T(X))$. In order to prevent confusion between multiplication and addition on rationals and conjuction and XOR on Booleans, we write $\cdot$ and $+$ for the former and $\wedge$ and $\oplus$ for the latter.

As an example, the Fourier expansion of $x \wedge y$ is

$$1/2 + 1/2 \cdot x + 1/2 \cdot y - 1/2 \cdot x \cdot y. \tag{1}$$

If $x = \mathsf{false} = 1$ and $y = \mathsf{true} = -1$, for example, the polynomial evaluates to $1/2 + 1/2 - 1/2 + 1/2 = 1 = \mathsf{false}$ as expected for an AND function.

Let us note some simple facts. (1) the Fourier expansion uses the exclusive or of variables as the basis: $x \oplus y = x \cdot y$. (2) $f^2 = 1$ for the Fourier expansion of any Boolean function f [32]. (3) there are two linear functions of two arguments: $f = x \cdot y$ (XOR) and $f = -(x \cdot y)$ (XNOR). All other functions f are nonlinear and for them, each of $\widehat{f}(\emptyset)$, $\widehat{f}(\{x\})$, $\widehat{f}(\{y\})$, and $\widehat{f}(\{x, y\})$ is nonzero. (We are ignoring the constant and unary functions.) (4) the statistical dependence of the functions can be read off directly from the Fourier expansion: the conjunction has a constant bias, positively correlates with x and y, and negatively with its $x \oplus y$. This last fact can be generalized to the following lemma.

Lemma 4 (Xiao-Massey [39]). *A Boolean function $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$ is statistically independent of a set of variables $X' \subseteq X$ iff $\forall T \subseteq X'$ it holds that if $T \neq \emptyset$ then $\widehat{f}(T) = 0$.*

4 Masking and the Probing Model

The intention of masking is to harden side-channel analysis attacks (like differential power analysis or electromagnetic emanation analysis) by making side-channel information independent of the underlying security sensitive information. This independence is achieved through the randomization of the representation of security sensitive variables inside the circuit. For this purpose, randomly produced and uniformly distributed masks are added (XOR) to the security sensitive variables on beforehand of a security critical computation. The number of used masks depends on the used masking scheme and is a function of the security order.

As a simple example, we consider the security sensitive 1-bit variable s in Eq. 2 that is protected by adding a uniformly random mask m_s, resulting in the masked representation s_m.

$$s_m = s \oplus m_s. \tag{2}$$

The masked value s_m is again uniformly distributed and statistically independent of s, i.e., it has the same probability to be 0 or 1 regardless of the value of s. Any operation that is performed only on s_m is statistically independent of s and thus also the produced side-channel information. Since the mask m_s is randomly produced, operations on the mask are uncritical. However, the combination of side-channel information on s_m and m_s can reveal information on s. The independence achieved through masking is thus only given up to a certain degree (the number of fresh masks used for masking s), and it is important to ensure this degree of independence throughout the entire circuit. The degree of independence is usually refereed to as the protection order d.

Masked circuits. For the remainder of the paper, let us fix an ordered set $X = \{x_0, \ldots, x_n\}$ of input variables. We partition the input variables X into three categories:

- $S = \{s_1, \ldots s_j\}$ are security sensitive variables such as key material and intermediate values of cryptographic algorithms that must be protected against an attacker by means of masking.

- $M = \{m_1, \dots m_k\}$ are masks that are used to break the statistical dependency between the secrets S and the information carried on the wires and gates. Masks are assumed to be fresh random variables with uniform distribution and with no statistical dependency to any other variable of the circuit.
- $P = \{p_1, \dots p_l\}$ are all other variables including publicly known constants, control signals, et cetera. Unlike secret variables, these signals do not need to be protected by masks and are unsuitable to protect secret variables.

We define a circuit $C = (\mathcal{G}, \mathcal{W}, R, f, \mathcal{I})$, where $(\mathcal{G}, \mathcal{W})$ is an acyclic directed graph with vertices $\mathcal{G}$ (gates) and edges $\mathcal{W} \subseteq \mathcal{G} \times \mathcal{G}$ (wires). Gates with indegree zero are called inputs I, gates with outdegree zero are called outputs O. Furthermore, $R \subseteq \mathcal{G}$ is a set of registers, f is a function that associates with any gate $g \in \mathcal{G} \setminus I$ with indegree k a function $f(g) : \mathbb{B}^k \to \mathbb{B}$, and $\mathcal{I} : I \to (2^X \to \mathbb{B})$ associates an externally computed Boolean function over X to each input. We require that registers have indegree one and that the associated function is the identity. In the following, we assume, wlog, that all gates, except inputs and registers, have indegree 2 and we partition these gates into a set L of linear gates (XOR, XNOR) and a set N of nonlinear gates (AND, NAND, OR, NOR, the two implications and their negations). We also require that for any gate g, any path from some input to g has the same number of registers.

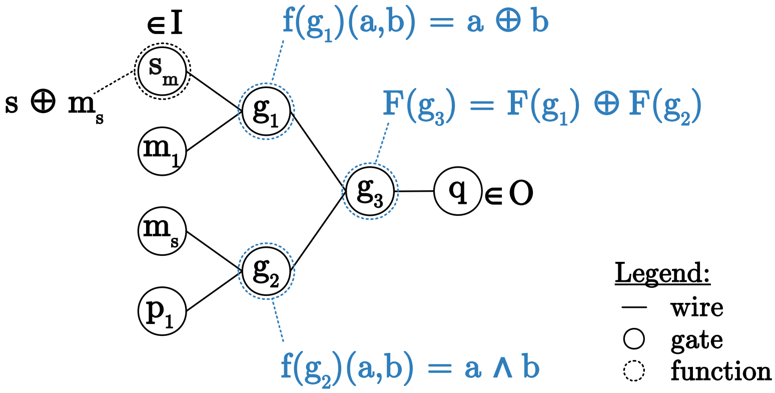

Fig. 1. Circuit graph of circuit in Fig. 2

The intuitive meaning of f is the local function computed by a gate. For instance, if g is an AND gate, $f(g)(x, y) = x \wedge y$. We associate with every gate another function $F(g) : 2^X \to \mathbb{B}$, which defines the function computed by the output of the gates in terms of the circuit inputs. The function $F(g)$ is defined by the functions of the predecessor gates and $f(g)$ in the obvious way. Given a sequence of gates $(g_1, \dots, g_d)$, we extend F pointwise to $F(g_1, \dots, g_d) : 2^X \to \mathbb{B}^d$: $F(g_1, \dots, g_d)(x) = (g_1(x), \dots, g_d(x))$. We often identify a gate with its function.

As an example, consider the circuit graph in Fig. 1 (which corresponds to the circuit depicted in Fig. 2). We have $f(g_3)(a, b) = a \oplus b$ and $F(g_3) = (s_m \oplus m_1) \oplus (m_s \wedge p_1)$.

For a circuit C, a sequence of gates $G = (g_1, \ldots, g_n)$, and a sequence of functions $F = (f_1, \ldots, f_n)$ with $f_i \in \mathbb{B}^2 \to \mathbb{B}$, we write $C[G \mapsto F]$ for the circuit C in which gate g_i is replaced by a gate with the Boolean function f_i.

Security of masked circuits. The security of various masking schemes is often analyzed in the so-called probing model that was introduced by Ishai et al. [26]. It was shown by Faust et al. [20] and Rivain et al. [36] that the probing model is indeed suitable to model side-channel attacks and to describe the resistance of an implementation in relation to the protection order d. As it was shown by Chari et al. [13], there is an exponential relation between d and the number of leakage traces required to exploit the side-channel information.

In the probing model, an attacker is bound to d probing needles which can be freely placed on arbitrary circuit gates (or wires). Probes are placed permanently on these gates and monitor all signals states and signal transitions that occur at the probed circuit gate from the circuit reset onwards. Thus one probe records the probed signals at all time instances. The probing model quantifies the level of side-channel resistance of a circuit over the minimum number of probing needles an attacker requires to extract any secret information. More specifically, a circuit is secure in the probing model if an attacker cannot combine the information gathered from d probes over all points in time in an arbitrary function F such that F statistically depends on any of the secret variables in S. We model a probe as the ability to read the Boolean function produced by the probed gate or its associated wire. Since we assume that the masking variables are uniformly distributed, and the public variables are known, the circuit leaks information iff F is statistically dependent on S regardless of the values that the public variables take.

Definition 5 (secure functions). *A function $f : 2^X \to \mathbb{B}^d$ is secure if* f *is for any assignment $p \subseteq P$ to the public variables, $f|_p$ is statistically independent of* S.

Definition 6 (d-probing security [26]). *A circuit $C = (\mathcal{G}, \mathcal{W}, f, \mathcal{I})$ is order d probing secure (d-probing secure) iff for any gates $g_1, \ldots, g_d \in \mathcal{G}$, $F(g_1, \ldots, g_d)$ is secure.*

Verification example using the Fourier expansion. According to Lemma 4, we can decide whether the values computed by a circuit are secure by computing the Fourier expansion of all its gates and checking whether there is a coefficient that contains only secret variables without a mask (and with or without public variables). Formally we check that $\emptyset \neq S' \subseteq S \cup P$ such that $\widehat{F}(g)(S') \neq 0$. The first-order security of a circuit can thus be verified using the probing model by calculating the Fourier expansion of the whole circuit. As an example consider the Fourier expansion of the circuit in Fig. 2 for which we have:

$$\begin{aligned}
F(g_1) &= s_m \cdot m_1, \\
F(g_2) &= 1/2 + 1/2 \cdot m_s + 1/2 \cdot p_1 - 1/2 \cdot m_s p_1, \text{ and} \\
F(g_3) &= F(g_1) \cdot F(g_2) \\
&= 1/2 \cdot s_m m_1 + 1/2 \cdot m_s s_m m_1 + 1/2 \cdot p_1 s_m m_1 - 1/2 \cdot m_s p_1 s_m m_1.
\end{aligned}$$

Assuming that $s_m = s \oplus m_s$ and using the properties of the Fourier expansion this implies that

$$F(g_3) = 1/2 \cdot s m_s m_1 + 1/2 \cdot s m_1 + 1/2 \cdot s p_1 m_s m_1 - 1/2 \cdot s p_1 m_1. \qquad (3)$$

For the example circuit in Fig. 2, if s is a secret and m_1 is a uniformly distributed random mask, then g_3 in Eq. 3 computes a function that does not reveal any secret information. This follows from the fact that in $F(g_3)$ there are only (non-zero) Fourier coefficients for terms that contain s and at least one masked value.

Since the exact computation of Fourier coefficients is very expensive and the extension to higher-order probing security nontrivial, in the following we develop a method to estimate the Fourier coefficients of each gate and to check for higher-order security.

5 Our Verification Approach for Stable Signals

In this section, we present a sound verification method for (d-)probing security for the steady-state of a digital circuit. It is assumed that the signals at the circuit input are fixed to a certain value and that all intermediate signals at the gates and the circuit output have reached their final (stable) state. This approach is later on extended in Sects. 6 and 7 to cover transient signals and glitches.

Since the formal verification of the security order of masked circuits has proven to be a non-trivial problem in practice, the intuition behind a circuit verifier is to have a method that correctly classifies a wide range of practically relevant and securely masked circuits but rejects all insecure circuits. Any circuit that is not secure according to Definition 6 is rejected. Our verification approach can be subdivided into three parts: (1) the labeling system, (2) the propagation rules, and (3) the actual verification.

5.1 Labeling

In order to check the security of a circuit we introduce a labeling over the set of input variables X for the stable signals $\mathcal{S} : \mathcal{G} \rightarrow 2^{2^X}$ that associates a set of sets of variables to every gate. This labeling system is based on the Fourier representation of Boolean functions (see Sect. 3) and intuitively, a label contains at least those sets $X' \subseteq X$ for which $\widehat{f}(X') \neq 0$ (the sets that correlate with the Boolean functions).

The initial labeling is derived from $\mathcal{I}$. For an input g which is fed by function $f_g = \mathcal{I}(g)$, we have $\mathcal{S}(g) = \{X' \subset X \mid \widehat{f_g}(X') \neq 0\}$. In practice, the initial labeling of the circuits is easy to determine as inputs are typically provided with either a single variable m or a masked secret $x \oplus m$. An example for the labeling of an example circuit is shown in Fig. 2 (blue). Inputs containing security sensitive variables contain a single set listing all security sensitive variables and masks that protect this sensitive variables. For the masked signal $s_m = s \oplus m_s$, for example, the initial label is $\mathcal{S}(s_m) = \{\{s, m_s\}\}$. The meaning of the label is that by probing this input the attacker does not learn anything about s. In order to reveal any information on s, also some information on m_s needs to be combined with this wire in, either by the circuit itself (which would be a first-order flaw) or by the attacker by probing an according wire. If the attacker is assumed to be restricted to a single probing needle ($d = 1$) the signal s_m is secure against first-order attacks. Finally, the masked inputs m_s and m_1 in Fig. 2 contain only the mask variables. Formally, for inputs $g \in I$ with function $\mathcal{I}(g) = f(X)$, we set $\mathcal{S}(g) = \{X' | X' = X\}$.

5.2 Propagation Rules

To estimate the information that an attacker can learn by probing the output of a gate, we propagate the input labels through the circuit. For the verification we conservatively estimate which coefficients of the Fourier representation are different from zero and correlate with the variables. We prove at the end of this section that our estimation is sufficient to point out all security relevant information.

Nonlinear gates. To generate the labels for the outputs of each gate of the circuit, we introduce the *nonlinear gate* rule. The nonlinear gate rule corresponds to a worst-case estimation of the concrete Fourier spectrum of the signals and trivially catches all flaws. The labeling for the output of the nonlinear gate $g \in N$, with inputs g_a and g_b is:

$$\mathcal{S}(g) = \{\emptyset\} \cup \mathcal{S}(g_a) \cup \mathcal{S}(g_b) \cup \mathcal{S}(g_a) \triangle \mathcal{S}(g_b).$$

See gate g_2 in Fig. 2 for a simple example of an AND gate calculating $m_s \wedge p_1$. The resulting labels denote the information that can be learned by probing this gate which could be either m_s or p_1 alone, or together. The labeling reflects the Fourier spectrum of the AND gate (see Eq. 1). In particular the labeling shows all terms of the Fourier polynomial which coefficients are different from zero and are therefore statistical dependent.

Linear gate rule. By following the Definition 3 of the Fourier expansions further we can also model linear gates which have a reduced spectrum compared to nonlinear gates. We model this circumstance by introducing a new rule for labeling a linear gate $g \in L$ with inputs g_a and g_b:

$$\mathcal{S}(g) = \mathcal{S}(g_a) \triangle \mathcal{S}(g_b).$$

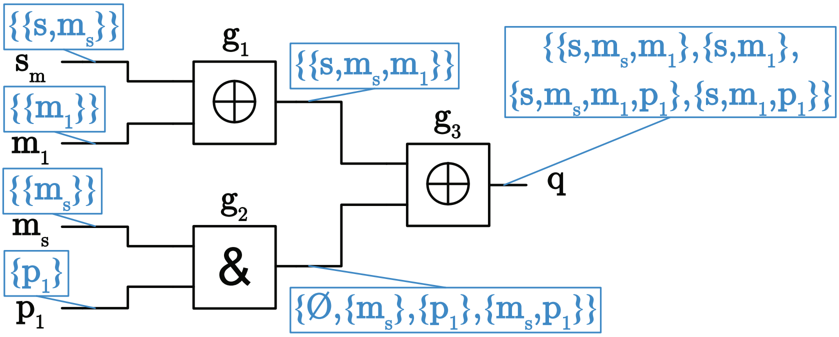

Fig. 2. Masked circuit example with according labels after the propagation step (Color figure online)

Combined example. To demonstrate how the propagation step works in practice, we applied the propagation rules (summarized in Table 1) to an example circuit. The result is shown in Fig. 2. The AND gate g_2 is a nonlinear gate, and the propagation rules are then iteratively applied to the gates g_1 to g_3. The output labeling of g_1 indicates that the security critical variable s is here not only protected by m_s but also by the mask m_1. Combining the public signal p_1 with the mask m_s in the nonlinear gate results in a nonuniform output signal which is indicated by the $\{\emptyset\}$ label at the output of g_2. For the calculation of the labels of g_3, the linear rule is used on the output labels of g_1 and g_2 which results in a labeling that indicates that s is even in the worst-case still protected by m_s, or m_1, or both.

5.3 Verification

For the verification step, in the first-order case, the circuit verifier checks if any of the sublabels created in the propagation step contains one or more secret variables without any masking variables (public variables are ignored since they are unable to mask secret data). If this is the case, the verifier rejects the circuit. In the example circuit in Fig. 2, all of the labels that contain s also contain m_1 or m_s and therefore the circuit is accepted by the verifier.

Table 1. Propagation rules for the stable set $\mathcal{S}(g)$ connected to the gates g_a and g_b

Gate type of g	Stable set rule
Input $\mathcal{I}(g) = f(X)$	$\mathcal{S}(g) = \{X' \mid X' = X\}$
Nonlinear gate	$\mathcal{S}(g) = \{\emptyset\} \cup \mathcal{S}(g_a) \cup \mathcal{S}(g_b) \cup \mathcal{S}(g_a) \triangle \mathcal{S}(g_b)$
Linear gate	$\mathcal{S}(g) = \mathcal{S}(g_a) \triangle \mathcal{S}(g_b)$
Register	$\mathcal{S}(g) = \mathcal{S}(g_a)$

Higher-order verification. For the generalization to d-order verification it is quite tempting to model the attackers abilities by letting the attacker pick multiple labels from any gate and combining them in an arbitrary manner. However, we note that the labeling does not reflect the relation of the probed information among each other and thus does not give a suitable approximation of what can be learned when multiple gates are probed. As a trivial example consider a circuit that calculates $q = (a \wedge b) \oplus c$ where all inputs are uniformly distributed. The labeling of the output q after the propagation step consists of the labels $\{c\}$, $\{a, c\}$, $\{b, c\}$, and $\{a, b, c\}$ for all of which an attacker probing q would indeed see a correlation. If an attacker restricted to two probes would probe q with the first probe, she obviously would not learn anything more by probing q a second time. In other words, if we would model a higher-order attacker by the ability to combine multiple labels, she could combine the label $\{c\}$ with any other label of q, e.g. $\{a, b, c\}$, in order to get information on a or b which is of course not the case.

Instead of modeling higher-order verification by the straight-forward combination of labels, we instead check the nonlinear combination of any tuple of d gates. An attacker can thus pick any number of up to d gates and combines them in an arbitrary function. We then need to check that even the worst case function over the gates could never contain a secret variable without a mask. This causes an obvious combinatorial blowup. In Sect. 8, we show how to harness a SAT solver to combat this problem. A proof for the correctness of the verification without glitches is provided in Appendix B.

In the next two sections we extend the verifier to cover glitches which shows that the example circuit is actually insecure.

6 Modeling Transient Timing Effects

So far, we have only considered the circuit's stable signals. We now discuss signal timing effects inside one clock cycle i.e. glitches and formalize how we model glitches in the probing model. Subsequently, we discuss how we model information that is collected from multiple clock cycles.

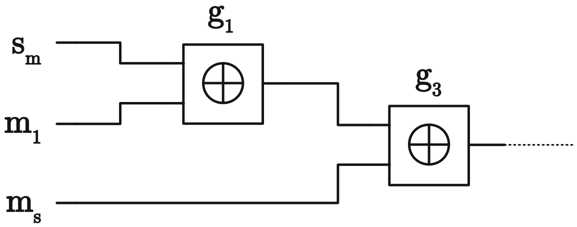

Fig. 3. Masked circuit example, insecure due to glitches

6.1 Glitches

As an example of what can go wrong when differences in the signal propagation times are not taken into account [29], consider the circuit in Fig. 3. The depicted circuit is secure in the original probing model as introduced in [26].

The information on the outputs of the XOR gates is ($s_m = s \oplus m_s$):

$$\begin{aligned} g_1 &= s_m \oplus m_1 = s \oplus m_s \oplus m_1 \text{ and} \\ g_3 &= s_m \oplus m_1 \oplus m_s = s \oplus m_1. \end{aligned}$$

Since the other circuit gates (input terminals are modeled as gates) only carry information on the masked value s_m or the masks m_s and m_1, a single probe on any parts of the circuit does not reveal s and the circuit is thus first-order secure in the original probing model.

However, if we assume that in a subsequent clock cycle (Cycle 2 in Fig. 4) a different secret s' is processed, the circuit inputs change accordingly from s_m, m_s, and m_1 to s'_m, m'_s, and m'_1, respectively. Figure 4 shows an example on how these changes propagate through the circuit. Due to signal timing variance caused by physical circumstances, like different wire lengths or different driving strengths of transistors, so-called glitches arise. As a result of this timing variance m_1 changes its value later (t_2) than the other inputs (t_1) thus creating a temporary information leak (glitch). An attacker who places one probe on the output of g_3 firsts observes the original value $s \oplus m_1$ (at time t_0) and then $s' \oplus m_1$ (between t_1 and t_2). By combining the information the attacker obtains the information $(s \oplus m_1) \oplus (s' \oplus m_1)$ which is equivalent to $s \oplus s'$. Thus, she learns the relation of two secret bits. This information could not be obtained by combining the stable signals in the two clock cycles. Indeed, the leakage critically depends on the temporary information provided by the glitch in the circuit. To verify the security of a circuit in the probing model with glitches, all possible signal combinations that could arise because of propagation delays of signals need to be considered.

6.2 Formalization of Probing Security with Glitches

To formalize the probing model with glitches in the first-order case, the attacker's abilities are extended as follows: The attacker can first replace any number of gates (except for registers) by a gate that computes an arbitrary Boolean function from the gate's original inputs, and may then place one probe on any wire such that there is no register between any replaced gate and the probe.

For higher-order attacks with $d > 1$, the formalization is a little more cumbersome. Intuitively, the attacker should be able to modify the behavior of arbitrary gates, but this effect should disappear when the signal passes through a register. We model this by copying the combinational parts of the circuit and allowing

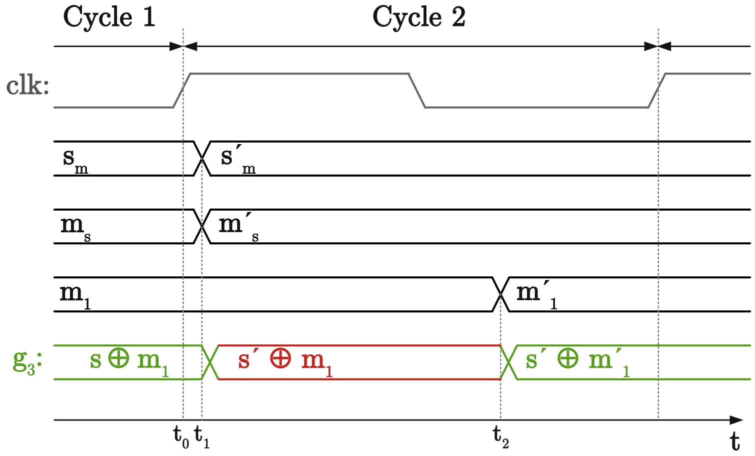

Fig. 4. Waveform example for the circuit in Fig. 3, showing security critical glitch (red) (Color figure online)

the attacker to change gates in the copy, whereas the original, unmodified signals are propagated by the unmodified gates. Figure 5 illustrates an example for the modeling of the glitches. The copied gates, which the attacker may modify, are drawn in blue. Note in particular that gate g_7 feeds into register g_8, but the copy g_7' becomes a new primary output.

Formally, given a circuit $C = (\mathcal{G}, \mathcal{W}, R, f, \mathcal{I})$, we do the following.

(1) We define a circuit $C' = (\mathcal{G}', \mathcal{W}', R, f', \mathcal{I})$. We copy all the gates except inputs and registers: $\mathcal{G}' = \mathcal{G} \cup \{g' \mid g \in \mathcal{G} \setminus R \setminus I\}$. We introduce wires from the inputs and registers to the copied gates and introduce wires between the copied gates: $\mathcal{W}' = \mathcal{W} \cup \{(g, h') \mid (g, h) \in \mathcal{W}, g \in I \cup R\} \cup \{(g', h') \mid (g, h) \in \mathcal{W}, g \notin I \cup R, h \notin R\}$. Finally, the functions of the copied gates are the same as those of the originals: $f'(g') = f(g)$ for $g \in \mathcal{G}' \setminus \mathcal{G}$.

(2) The attacker may replace any gate copy g' by a gate that computes an arbitrary Boolean function. We model this by defining a set of circuits, one for any set of gates that the attacker may modify:

$$\mathcal{C}_{\text{glitch}}(C) = \{C'[(g_1', \ldots, g_n') \mapsto (f_1, \ldots, f_n)] \mid (g_1, \ldots, g_n) \in \mathcal{G}^n, \forall i. f_i : \mathbb{B}^2 \to \mathbb{B}\}.$$

Definition 7 (*d*-probing security with glitches). *A circuit C is order d probing secure with glitches iff for any $C_{glitch} = (\mathcal{G}', \mathcal{W}', R, f', \mathcal{I}) \in \mathcal{C}_{glitch}$ and any gates $g_1, \ldots, g_d \in \mathcal{G}'$, $F(g_1, \ldots, g_d)$ is secure.*

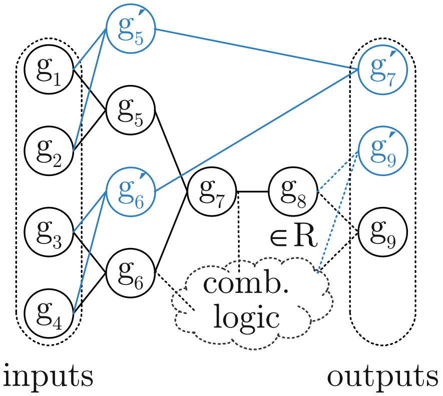

Fig. 5. Example for modeling of glitches of a circuit C (without blue parts) in C' (Color figure online)

6.3 Modeling Information from Multiple Clock Cycles

The verification of higher-order probing security requires to model information that is obtained and combined over different clock cycles. In our verification approach we consider dependencies between variables rather than concrete instantiation of these variables. The way we model glitches allows an attacker to exploit the worst case dependencies between the variables in between two register stages. We now state assumptions on masked circuit that ensure that the worst case dependencies are the same in each clock cycle.

Assumptions on masked circuits. Without loss of relevance for masked circuits we make the following assumptions which are inspired by practical masked circuits: (1) We assume that the values on the inputs remain the same throughout a clock cycle, they toggle only once at the beginning of a new clock cycle (registered inputs). (2) The class of the variables that are used in the input functions and the functions themselves do not change over time. For the circuit in Fig. 3, for example, the input s_m always contains a variable $s \in S$ and the associated mask $m_s \in M$ even though in each clock cycle the variables may change (e.g. from s to s'). (3) Mask variables are fresh random and uniform distributed at each clock cycle. (4) The circuits are feedback free and loop free, except the inherent feedback loops of registers. (5) The register depth (number of registers passed, counting from the input of the circuit) for each variable combined in a gate function is the same. No information resulting from different clock cycles is thus combined apart from the effects of delays and glitches which may temporarily combine information from two successive clock cycles. This assumption is motivated by the fact that most of the masked hardware designs, e.g. common S-box designs, are designed in a pipelined way.

From these assumptions it follows that all variables change in each cycle (e.g. from s to s', and so on), however, at varying times and in an arbitrary order. The variable classes and functions remain the same, and as a result from the assumptions 4 and 5 it is ensured that only variables that are fed into the circuit at the same cycle or from the cycle before are combined. It is therefore enough to consider the propagation of dependencies instead of concrete instantiation of variables.

7 Extending the Verification Approach to Transient Signals

In this section we use the modeling of the transient timing effects from the previous section to complete our verification approach. We take glitches into account by extending the propagation rules accordingly. The modeling of information from different clock cycles, on the other hand, does not require any changes in the verification approach from Sect. 5.

The nonlinear gate rule in Table 1 already inherently covers glitches by propagating the labels of the inputs and all possible combinations of these labels directly to the output. To hinder the propagation of glitches, circuit designers use registers that propagate their input only on a specific clock event, and thus isolate the register input from the output during the evaluation phase. We model the glitching behavior of a circuit by introducing an additional transient set of labels $\mathcal{T}$ per gate. Each gate thus has two associated sets: $\mathcal{S}$ carries the information of the stable state of the circuit as before, and the transient set $\mathcal{T}$ describes the transient information that is only accessible to an attacker in between two registers (or an input and a register, or a register and an output). In between two registers we also apply the nonlinear gate rule to linear gates to ensure we cover all possible effects of glitches.

Figure 6 illustrates the new linear gate rule for the stable (blue) and the transient (red) set of labels. The stable and transient sets of the inputs are equal at the beginning because the inputs are either circuit inputs or outputs of a register. When the signals propagate through the linear XOR gate, the transient set is calculated by applying the linear rule from Table 2 and the stable set with the linear rule from Table 1. After the signal passes the register, only the stable information remains and the transient set carries thus the same information as the stable set. Table 2 summarizes the rules for creating the transient-set labels $\mathcal{T}(g)$. Please note that introducing the transient set and the transient gate rules corresponds to the modeling of glitches from Sect. 6 as depicted in Fig. 5 (blue), where the gates in between two registers are copied and their function can be changed in an arbitrary manner by the attacker. Replacing the transient labels with the stable labels at a register corresponds to connecting the copied gates to the circuit output to hinder the propagation of glitches.

Aside from the introduction of the transient set and the according propagation rules, the verification work as described in Sect. 5. The circuit inputs are initially labeled according to their input variables where both the stable and transient sets hold the same labels. Then for all possible combinations of up to d gates the propagation of the labels is performed according to the stable and transient propagation rules. The circuit is order-d probing secure if for no combination of gates produces a label that only consists of secrets and public variable

Table 2. Propagation rules for the transient set $\mathcal{T}(g)$ fed by the gates g_a and g_b

Gate type of g	Transient set rule
Input	$\mathcal{T}(g) = \mathcal{S}(g)$
Nonlinear gate	$\mathcal{T}(g) = \{\emptyset\} \cup \mathcal{T}(g_a) \cup \mathcal{T}(g_b) \cup \mathcal{T}(g_a) \triangle \mathcal{T}(g_b)$
Linear gate	$\mathcal{T}(g) = \{\emptyset\} \cup \mathcal{T}(g_a) \cup \mathcal{T}(g_b) \cup \mathcal{T}(g_a) \triangle \mathcal{T}(g_b)$
Register	$\mathcal{T}(g) = \mathcal{S}(g_a)$

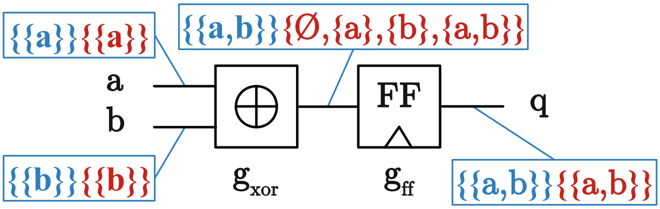

Fig. 6. XOR gate rules for stable (blue) and transient (red) signal sets (Color figure online)

without masks. A proof for the verification approach for transient signals is provided in Appendix C.

Example. The transient labels $\mathcal{T}$ of the circuit in Fig. 2 are shown in Fig. 7 (the stable sets are omitted since they do not carry any additional information). Due to the transient set propagation rules, the functionality of the gates g_1 and g_3, which are linear gates in the underlying circuit in Fig. 2, are replaced with nonlinear gates. As can be observed at the output of the circuit, the verification under the consideration of glitches leads to a rejection of the circuit because the s variable (black labels) is in the output labeling without being masked by either m_s or m_1.

To make it clear that the circuit is indeed insecure, we assume that $p_1 = \mathsf{true}$ and that s_m and m_s change their values to s'_m and m'_s, resp., but the value of m_1 and p_1 temporarily remains unchanged. Then, g_1 transitions from $s \oplus m_s \oplus m_1$ to $s' \oplus m'_s \oplus m_1$ and as a result g_3 transitions from $s \oplus m_1$ to $s' \oplus m_1$, thus leaking information about the relation of s and s'. (Cf. Fig. 4). The flaw can be easily repaired by adding a register after g_1 which ensures that s_m is always remasked before m_s is combined with s_m in g_3, and the same labels as in Fig. 2 for g_1 would thus be propagated.

8 SAT Based Circuit Verification

In this section, we introduce one concrete instantiation of our verification approach based on a SAT solver. The verification approach introduced in the previous sections is thus encoded as formulas in propositional logic. We start with the stable set rules and verification before we extend the encoding to the transient set rules.

Verification of stable signals. The SAT based verification works as follows. Intuitively, for every gate g, we pick one set $X' \subseteq \mathcal{S}(g)$, i.e., we pick one Fourier character with a possibly nonempty coefficient. We then encode the rules for the linear and nonlinear gates of Tables 2 and 1, respectively. To check for higher-order security we connect an XOR gate (checking gate) to any possible subset of up to d gates and check that the label of this gate does not contain a label with just secrets and no masks.

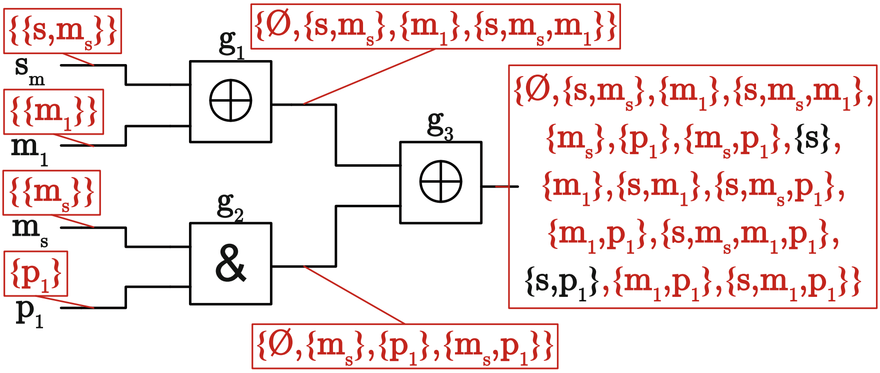
{{s,m_s}}
s_m
{{m_1}}
m_1
{{m_s}}
m_s
{p_1}
p_1
g_1
⊕
g_2
&
g_3
⊕
{∅,{s,m_s},{m_1},{s,m_s,m_1}}
{∅,{s,m_s},{m_1},{s,m_s,m_1},
{m_s},{p_1},{m_s,p_1},{s},
{m_1},{s,m_1},{s,m_s,p_1},
{m_1,p_1},{s,m_s,m_1,p_1},
{s,p_1},{m_1,p_1},{s,m_1,p_1}}
{∅,{m_s},{p_1},{m_s,p_1}}

Fig. 7. Masked circuit example from Fig. 2 reevaluated with the transient rules (red) which leads to a flaw due to glitches (black labels) (Color figure online)

Let $C = (\mathcal{G}, \mathcal{W}, R, f, \mathcal{I})$ be a circuit. For each gate g we introduce a set of Boolean variables $X_g = \{x_g \mid x \in X\}$ and a Boolean *activation variable* a_g. For a checking gate g_c we introduce a set of Boolean variables X_{g_c}. We define a formula Ψ to check whether the masking scheme is secure. Recall that L and N are the sets of linear gates and nonlinear gates, resp. Formula Ψ consist of multiple parts:

$$\Psi = \Psi_{\text{gates}} \wedge \Psi_{\text{unsafe}}, \text{ where}$$

$$\Psi_{\text{gates}} = \bigwedge_{g \in I} \Psi_{\text{inp}}(g) \wedge \bigwedge_{g \in N} \Psi_{\text{nl}}(g) \wedge \bigwedge_{g \in L} \Psi_{\text{lin}}(g) \wedge \bigwedge_{g \in R} \Psi_{\text{reg}}(g).$$

The labeling of the inputs is determined by $\mathcal{I}$. For $X' \subseteq X$, we define

$$\psi_g(X') = \bigwedge_{x \in X} \begin{cases} x_g(X') & \text{if } x \in X', \\ \neg x_g(X') & \text{if } x \notin X', \text{ and} \end{cases}$$

$$\Psi_{\text{inp}}(g) = \bigvee_{X' \subseteq X : \widehat{\mathcal{I}(g)}(X') \neq 0} \psi_g(X').$$

To define the behavior of linear and nonlinear gates we define the following auxiliary formulas using the rules from Table 1, where $T = (t_1, \ldots, t_n)$,

$U = (u_1, \ldots, u_n)$, and $V = (v_1, \ldots, v_n)$ are ordered sets of variables, and define $\leftrightarrow$ to denote equality.

$$\varPsi_{\text{empty}}(T) = \bigwedge_i \neg t_i,$$

$$\varPsi_{\text{copy}}(T, U) = \bigwedge_i (t_i \leftrightarrow u_i), \text{ and}$$

$$\varPsi_{\text{lin}}(T, U, V) = \bigwedge_i (t_i \leftrightarrow (u_i \oplus v_i)).$$

For a linear gate g with inputs g' and g'', we use the formula

$$\varPsi_{\text{lin}}(g) = \varPsi_{\text{lin}}(X_g, X_{g'}, X_{g''}),$$

for a nonlinear gate g with inputs g' and g'', we use the formula

$$\varPsi_{\text{nl}}(g) = \varPsi_{\text{empty}}(X_g) \vee \varPsi_{\text{copy}}(X_g, X_{g'}) \vee \varPsi_{\text{copy}}(X_g, X_{g''}) \vee \varPsi_{\text{lin}}(X_g, X_{g'}, X_{g''}), \text{ and}$$

for a register g with input g', we simply have $\varPsi_{\text{reg}}(g) = \varPsi_{\text{copy}}(X_g, X'_g)$. Also we introduce an integer variable a_{sum}, and bound it to the attack order d:

$$a_{sum} = \sum_g \text{ite}(a_g, 1, 0)$$

$$a_{sum} \leq d.$$

The function $Ite(a_g, 1, 0)$ (if-then-else) converts a Boolean variable to Integer.

For the checking gate we xor the corresponding inputs:

$$\varPsi(g_c) = \bigwedge_{x \in X} x_{g_c} \leftrightarrow \bigoplus\nolimits_{g \in \mathcal{G}} a_g \wedge x_g.$$

Finally, for the checking gate g_c we define a constraint to check whether security is violated, that is, whether there is a non-zero Fourier coefficient which contains secrets and no masks:

$$\varPsi_{\text{unsafe}}(g_c) = \bigvee_{s \in S} s_g \wedge \bigwedge_{m \in M} \neg m_g.$$

Formula $\varPsi$ contains $|X| \cdot |\mathcal{G}|$ propositional variables and $O(|X| \cdot |\mathcal{G}|)$ constraints.

An example for the SAT encoding is provided in Appendix F along with a proof for its correctness in Appendix D.

Extension to transient signals. The encoding for the transient rules follows the exposition in Sect. 7 and in particular the rules from Table 2. We introduce a second set of variables $X'_g = \{x'_g \mid x \in X\}$, which represent the Fourier characters on the "copied" gates in Definition 7. We introduce a slightly modified set of constraints, where we write Φ' to denote a formula Φ in which each variable x_g has been replaced by x'_g.

$$\begin{aligned}\Phi &= \Phi_{\text{gates}} \wedge \Phi'_{\text{unsafe}}, \text{ where}\\ \Phi_{\text{gates}} &= \bigwedge_{g\in I} (\Psi_{\text{inp}}(g) \wedge \Psi'_{\text{inp}}(g)) \wedge \bigwedge_{g\in N} (\Psi_{\text{nl}}(g) \wedge \Psi'_{\text{nl}}(g)) \wedge\\ &\quad \bigwedge_{g\in L} (\Psi_{\text{lin}}(g) \wedge \Psi'_{\text{nl}}(g)) \wedge \bigwedge_{g\in R} \Phi_{\text{reg}}(g),\end{aligned}$$

where for a register g with input g', we copy only the original (glitch-free) signals:

$$\Phi_{\text{reg}}(g) = \Psi_{\text{copy}}(X_g, X_{g'}) \wedge \Psi_{\text{copy}}(X_g, X'_{g'}).$$

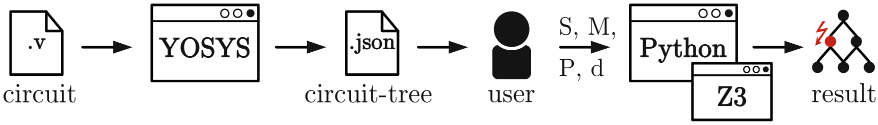

Fig. 8. Illustration of the verification flow

Note the use of the constraint for nonlinear gates for the copy of linear gates, which corresponds to the attacker's ability to replace such a gate by any other gate in $\mathcal{C}_{\text{glitch}}(C)$. Finally, we check for leakage only on the gate copies:

$$\Phi'_{\text{unsafe}}(g_c) = \bigvee_{s\in S} s'_{g_c} \wedge \bigwedge_{m\in M} \neg m'_{g_c}.$$

Formula Φ contains $2\cdot|X|\cdot|\mathcal{G}|$ propositional variables and $O(|X|\cdot|\mathcal{G}|)$ constraints. A proof for the correctness of the encoding for transient signals is given in Appendix E.

9 Practical Results

Figure 8 illustrates the implemented verification flow that is used to gather the results presented in this section. At first the circuit description is parsed using Yosys 0.7 [38] open synthesis suite. The resulting circuit tree is stored in JavaScript Object Notation (JSON). The user then needs to provide the circuit's input labels by telling the JSON parser (written in Python) which signals are secrets (S), masks (M), or other signals (P), and for which security order (d) the circuit needs to be tested. The construction of the SAT formulas is then performed in about 1,000 lines of Python code, and checked by the Z3 Theorem

Prover 4.5.1 [15] (initial experiments with other SAT solvers, including CryptoMinisat 5.0.1 were not encouraging). All results are gathered on a Intel Xeon E5-2699v4 CPU with a clock frequency of 3.6 GHz and 512 GB of RAM running in a 64-bit Linux OS environment (Debian 9).

Optimizations. There are a two simple optimizations that we use to speed up the verification. First, we can treat public variables at the inputs as constants. We can easily prove by induction that if $P' \cup S' \cup M' \in \mathcal{S}(q)$ for some gate g and $P' \subseteq P$, $S' \subseteq S$, and $M' \subseteq M$ and we compute a new labeling $\mathcal{S}'$ by treating the public variables as constants, then $S' \cup M' \in \mathcal{S}'(q)$ and thus, if $\mathcal{S}(q)$ is insecure, so is $\mathcal{S}'(q)$. A similar argument holds for $\mathcal{T}$ and for combinations of signals.

Second, we can treat secret bits one at a time, treating the other secret bits as constants. The argument is much the same as for the first optimization, if a function is insecure then it has a label with at least one secret bit and no masks. Removing any other secret bits from the label does not affect whether the label is seen as secure or not. This optimization allows for a significant speedup on larger examples as it allows us to run the checks for each of the secret bits in parallel.

Table 3. Overview of masked circuits the first order verification results

Name	Gates			Variables			Part	w/o glitches		w/glitches	
	Linear	Nonlin	Reg	Secret	Mask	Pub		Time	Result	Time	Result
Trichina gate [37]	4	4	0	2	3	1		$\leq$1 s	✗	$\leq$2 s	✗
ISW AND [26]	4	4	0	2	3	0		$\leq$1 s	✓	$\leq$2 s	✗
TI AND [31]	6	9	0	2	4	0		$\leq$1 s	✓	$\leq$3 s	✓
DOM AND [24]	4	4	2	2	3	1		$\leq$1 s	✓	$\leq$2 s	✓
DOM Keccak S-box [25]	30	20	10	5	10	1		$\leq$1 s	✓	$\leq$20 s	✓
DOM AES S-box [24]	392	144	208	8	26	1	1–8	$\leq$30 s	✓	$\leq$5–10 h	✓
TI Fides-160 S-box [11]	128	60	0	5	15	0	1–4	$\leq$1–2 s	✓	$\leq$1–3 s	✓
TI Fides-192 APN [11]	4,035	3,046	0	6	24	0					
	134	44	0	0	24	0	1	$\leq$2 s	✓	$\leq$5 s	✓
	649	314	0	6	24	0	2	$\leq$1 m	✓	$\leq$15 m	✓
	1,697	1,098	0	6	24	0	3	$\leq$20 m	✓	$\leq$2 h	✓
	1,186	1,086	0	6	24	0	4	$\leq$10 m	✓	$\leq$40 m	✓
	369	504	0	6	24	0	5	$\leq$2 m	✓	$\leq$3 m	✓

Evaluation. An overview of the experiments is given in Table 3. The table states the number of linear and nonlinear gates of the circuits as well as the number of variables classified as secret, mask, and public, resp. Furthermore, the verification results are given for the stable set (without glitches) and transient set (with glitches) verification separately. The table states whether the circuit is secure in the given model (✓ for secure and ✗ for insecure) and the time needed for checking and generation of the constraints.

The selection of masked circuits cover different masked $GF(2)$ multiplier constructions (masked AND gates) of the Trichina gate [37], the ISW scheme [26], the threshold implementation (TI) scheme [31], and the domain-oriented masking scheme (DOM) [24]. We also check larger circuits including the AES S-box constructions of Gross et al. [24] using the domain-oriented masking (DOM) scheme. Furthermore, we verify a FIDES S-box implementation by Bilgin et al. [11], and a Keccak S-box by Gross et al. [25].

9.1 Verification of First-Order Masked Implementations

Table 3 shows the verification results of the first-order masked hardware implementations. For larger circuits, like the AES S-box, we checked each of the secret bits separately. If multiple CPU's are available, these verifications can run simultaneously and we thus split the verification up into multiple parts.

Masked AND gates. The first masked AND we verify is the so-called Trichina gate which was originally designed to resist first-order attacks. Equation 4 shows the underlying logic function. The Trichina gate was designed without considering the effect of glitches. As a result, if the inputs are correctly and independently masked ($a_m = a \oplus m_a$ and $b_m = b \oplus m_b$), the stable state of the output of the circuit is also correctly masked.

$$q = a_m \wedge b_m \oplus a_m \wedge m_b \oplus m_a \wedge b_m \oplus m_a \wedge m_b \oplus m_q \tag{4}$$

However, due to timing differences in the propagation of the signals, glitches may occur in one of the XOR gates. This makes the design vulnerable unless additional measures are taken which is also indicated by our verification results. Interestingly also the result of the stable verification already shows the vulnerability of the Trichina gate. This is due to timing optimizations of the synthesizer that change the sequence in which the AND gate outputs are XORed together which is a common problem in masked circuit designs and is easily overseen.

The masked AND gate from the ISW scheme is similar to the Trichina gate but scalable to any protection order. This gate suffers from the same vulnerability to glitches, which makes any straightforward implementation of the original proposed circuit construction insecure against first-order attacks. This time the flaw is not detected in the stable set because the gates are arranged in a way that the secrets are always masked in the stable analysis of the circuit. However, the circuit is nevertheless vulnerable to glitches which is shown in the transient analysis of the circuit.

To overcome the issue of glitches, the threshold implementation (TI) scheme proposed a masking with two fresh masks per sensitive variable (e.g., $a_m = a \oplus m_{a0} \oplus m_{a1}$). The resistance to glitches is then achieved by ensuring that in no part of the circuit the masked value and all its masks come together.

A different approach, which requires fewer masks is provided e.g. by the domain-oriented masking AND. For the security of this masked AND a separation of the terms is required by using a register for the combination of the terms with a fresh random mask. The verifier also correctly labels the DOM AND to be secure for the stable and in the transient verification. The verification without glitches takes less than a second for all masked AND gate constructions, and less than three seconds for the verification with glitches.

Verification of masked S-box circuits and permutations. For the remaining circuits, we either used the source code which is available online [22] for the DOM Keccak S-box and the DOM AES S-box, or in case of Fides kindly received the circuit design from the designers. In order to check the circuits in a more efficient manner, we used different optimizations. For the TI S-box of the Fides-160 design, we checked the individual S-box functions in parallel but for a fixed assignment of the secrets and masks. The result for all Fides-160 TI functions is computed in less than three seconds with and without glitches. For the TI Fides-192 design not only the S-Box but the whole APN permutation is split into five functions. Again we assumed a fixed masking and checked the functions individually, which makes the verification of the first TI function very fast (because no secrets are fed into this part). For the other circuit parts, the verification takes between 20 min for the verification without glitches verification and 2 h for the verification with glitches. Please note that the differences in the verification timings for the different parts of Fides-192 result from the varying gate counts. All circuit parts are labeled to be secure. Finally, we also checked a DOM AES S-box design for which we checked the whole circuit for the eight individual secret bits separately. The stable set verification takes less than 30 s, and the verification of the transient sets between 5 and 10 h for each part. Again the verification result indicates a securely masked first-order protected circuit.

9.2 Verification of Higher-Order Masked Implementations

To evaluate the performance of our verification approach for higher-order masked circuits, we run our tool on the generically masked DOM AND gate [24] and the Keccak S-box [25]. The results are shown in Table 4 where the protection order of the circuit and the verification order are always set equal and are summarized in a single column (order).

The verification of the second-order masked DOM AND takes less than a second. For the fourth-order protected AND the verification time increases to about 7 min. The influence of the protection order at varying verification orders is depicted in Fig. 9. We evaluated each masked DOM AND from first-order up to its claimed protection order plus one additional order. This figure underlines the intuition that finding a flaw takes less time than ensuring that the circuit is free from flaws.

For the Keccak S-box circuit we again split the verification for the five secrets into five separate verificaiton runs. The verification for the second order than takes about 10 s per verification run without glitches and about 40 s when glitches are considered. For the third-order verification the times increase to 4 min and 25 min, respectively.

Table 4. Overview of masked circuits and the higher order verification results

Name	Order	Gates			Variables			w/o glitches		w/glitches	
		Linear	Nonlin	Reg	Secret	Mask	Pub	Time	Result	Time	Result
DOM AND [24]	2	12	9	9	2	11	1	≤1 s	✓	≤1 s	✓
	3	24	16	16	2	17	1	≤4 s	✓	≤20 s	✓
	4	40	25	25	2	24	1	≤2 m	✓	≤7 m	✓
Keccak S-box[a] [25]	2	75	45	45	5	35	7	≤10 s	✓	≤40 s	✓
	3	140	80	80	5	60	7	≤4 m	✓	≤25 m	✓

[a] For the Keccak S-box we performed the verification for the five secrets separately.

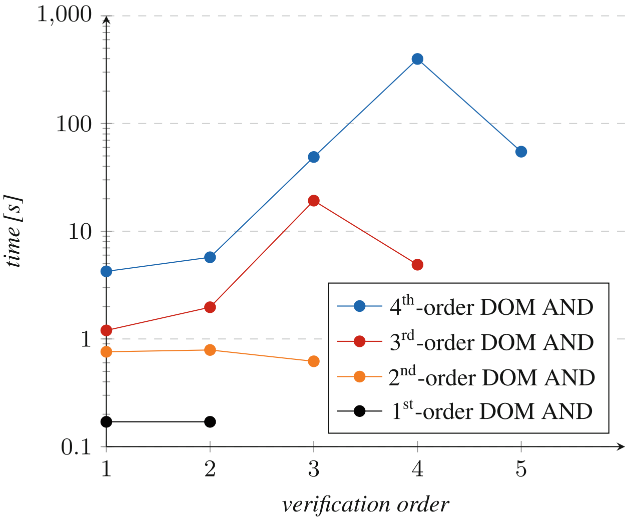

Fig. 9. Verification time for the DOM ANDs with varying protection and verification order

10 Conclusions

In this paper we introduced the formal groundwork for the verification of masked hardware implementations in the presence of glitches. We built upon the probing model of Ishai et al. and presented a method to conservatively estimate the security of circuits under this model for the worst case signal timings. Our approach is based on an estimation of the non-zero Fourier coefficients of the functions computed by the circuit, and we have provided a proof of its correctness. To demonstrate the practicality, we have implemented our formal approach on top of the Z3 theorem prover to verify the masking properties of a side-channel protected hardware implementation directly on the gate-level netlist. We have shown the suitability of our approach to verify masked circuits on practical examples from different masking schemes and different sources.

The advantages of this approach are evident. Circuits deemed secure do not leak secret information under any possible signal timings, which includes the effects of glitches in the combinatorial logic of the circuit, and even for higher-order attacks. If a circuit is rejected, we can pinpoint the gate that causes the potential leakage, which makes checking and fixing of the flaw much easier than by conventional approaches. Furthermore, the verifier can be used at different development stages of the masked circuit or for testing new masking schemes. This makes it a useful method for both practical applications as well as for research purposes.

Acknowledgements. The work has been supported in part by the Austrian Science Fund (FWF) through project P26494-N15, project S114-06, and project W1255-N23. Furthermore this work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402), and from the European Commission (grant agreement No 644905).

A Proof of Lemma 2

1. Suppose that $f \circ F$ is statistically dependent on X. Then by Definition 1 there are $x, x' \in 2^X$ such that $|\{y \mid (f \circ F)(x,y) = 1\}| \neq |\{y \mid (f \circ F)(x',y) = 1\}|$. Let $Z' = f^{-1}(1)$ be the assignments of Z that are mapped to true. Since $|\{y \mid (f \circ F)(x,y) = 1\}| = \sum_{z \in Z} |\{y \mid F(x,y) = z\}|$, there must be at least one $z \in Z'$ such that $|\{y \mid F(x,y) = z\}| \neq |\{y \mid F(x',y) = z\}|$.
2. Suppose F is statistically dependent on X. Then there is a $z \in 2^Z$ and $x, x' \in 2^X$ such that $|\{y \mid F(x,y) = z\}| \neq |\{y \mid F(x',y) = z\}|$. Let $f(z') = 1$ iff $z' = z$, then $f \circ F$ is statistically dependent on X.

□

B Proof of the Stable Verification Approach

Lemma 8. *For any circuit C, any gate* g *and any $T \subseteq X$, if the Fourier coefficient $\widehat{g}(T) \neq 0$, then $T \in \mathcal{S}(g)$.*

Proof. We prove the lemma by induction on the depth k of the circuit.

Base. $k = 0$. For an input g the lemma holds by the definition of the initial label.

Inductive step. Let $k \geq 1$ and suppose that a gate g at depth k is the output of a gate with two input gates u and v, with Fourier representations $u(X) = \sum_{T \subseteq X} \widehat{u}(T)\chi_T(X)$, and $v(X) = \sum_{T \subseteq X} \widehat{v}(T)\chi_T(X)$.

We distinguish two cases: (1) g is linear or (2) g is nonlinear. (The case of registers is trivial when we do not consider glitching.)

Case 1: $g = u \oplus v$. The Fourier representation of g is

$$g(X) = u(X) \cdot v(X) = \sum_{T \subseteq X} \widehat{g}(T) \cdot \chi_T(X),$$

where

$$\widehat{g}(T) = \sum_{T_1 \subseteq X} \widehat{u}(T_1) \cdot \widehat{v}(T \triangle T_1).$$

Assume that $\widehat{g}(T) \neq 0$. If $\widehat{g}(T) \neq 0$, then there exists a set $T_1 \subseteq X$ such that $\widehat{u}(T_1) \cdot \widehat{v}(T \triangle T_1) \neq 0$. Therefore, $\widehat{u}(T_1) \neq 0$ and $\widehat{v}(T \triangle T_1) \neq 0$. By the inductive hypothesis, it holds that $T_1 \in \mathcal{S}(u)$ and $T \triangle T_1 \in \mathcal{S}(v)$, which, by the linear rule means that $T_1 \triangle (T \triangle T_1) = T \in \mathcal{S}(g)$.

Case 2: g is a nonlinear gate. In this case, the Fourier representation of g is

$$g(X) = \alpha_{00} + \alpha_{01} \cdot u(X) + \alpha_{10} \cdot v(X) + \alpha_{11} \cdot u(X) \cdot v(X)$$

for some α_{ij}. Consequently, $\widehat{g}(T) \neq 0$ implies that either (1) $\widehat{u}(T) \neq 0$, (2) $\widehat{v}(T) \neq 0$, or (3) $\exists T' \subseteq X . \widehat{u}(T) \neq 0$ and $\widehat{v}(T' \triangle T) \neq 0$. (The converse does not hold.) In each of these three conditions, $T \in \mathcal{S}(w)$. □

The next lemma shows that an arbitrary function for d gates corresponds to a generalization of the non-linear rule from Table 1. (Note the use of $\triangle_{d \in D} T_d$ to denote the symmetric set difference of all T_ds).

Lemma 9. *Let $F_1, \dots, F_d : \mathbb{B}^X \rightarrow \mathbb{B}$, let $f : \mathbb{B}^d \rightarrow \mathbb{B}$, and let $F(x) = f(F_1(x), \dots, F_d(x))$. For any $T \subseteq X$, we have that $\widehat{F}(T) \neq 0$ implies that there is a $D \subseteq \{1, \dots, d\}$ and $T_1 \dots T_d \subseteq X$ such that $T = \triangle_{i \in D} T_i$ and for all i, $\widehat{F}_i(T_i) \neq 0$.*

Proof. Let $f(a_1, \dots, a_d) = \sum_{D \subseteq \{1,\dots,d\}} \alpha_D \prod_{i \in D} a_i$ be the Fourier expansion of f. We have that

$$\begin{aligned}
F(x) &= \sum_{D \subseteq \{1,\dots,d\}} \alpha_D \prod_{i \in D} F_i(x) \\
&= \sum_{D \subseteq \{1,\dots,d\}} \alpha_D \prod_{i \in D} \sum_{T \subseteq X} \widehat{F}_i(T) \chi_T(X) \\
&= \sum_{D \subseteq \{1,\dots,d\}} \alpha_D \sum_{\substack{T_1 \subseteq X \\ \vdots \\ T_d \subseteq X}} \prod_{i \in D} \widehat{F}_i(T_i) \chi_{T_i}(X) \\
&= \sum_{D \subseteq \{1,\dots,d\}} \alpha_D \sum_{\substack{T_1 \subseteq X \\ \vdots \\ T_d \subseteq X}} \chi_{\triangle_{i \in D} T_i}(X) \prod_{i \in D} \widehat{F}_i(T_i).
\end{aligned}$$

Consequently, if $\widehat{F}(T) \neq 0$, then there is a $D \subseteq \{1, \dots, d\}$ and $T_1 \dots T_d \subseteq X$ such that for all $i \in D$ $\widehat{F}_i(T_i) \neq 0$ and $T = \triangle_{i \in K} T_i$. □

Note that for $d = 2$, the lemma specializes to $\widehat{F}(T) \neq 0$ implies $T = \emptyset$, $\widehat{F_1(T)} \neq 0$, $\widehat{F_2(T)} \neq 0$, or $T = T_1 \triangle T_2$, $\widehat{F_1(T_1)} \neq 0$, and $\widehat{F_2(T_2)} \neq 0$, which reflects the nonlinear rule.

Theorem 10. *Let C be a circuit. If for all $S' \subseteq S \cup P$ such that $S' \cap S \neq \emptyset$, for all sets $g_1, \ldots, g_d$ of gates, for all $D \subseteq \{1, \ldots, d\}$, for all $T_1 \ldots T_d \subseteq X$ such that $\triangle_{d \in D} T_d = S'$, and for all $i \in D$, we have that $T_i \notin \mathcal{S}(g_i)$, then C is order d secure.*

Proof. (By contradiction.) Suppose that C is not order d secure. By Definitions 5 and 6, and Lemma 2, this implies that there are gates $g_1, \ldots, g_d$, a function$f$ and an assignment $p \subseteq P$ such that $f \circ F|_p(g_1, \ldots, g_d)$ is statistically dependent on S. By Lemma 4, this implies that for some $S' \subseteq S$, $S' \neq \emptyset$, $\widehat{f \circ F|_p}(g)(S') \neq \emptyset$. The Fourier expansion of $F|_p$ is obtained from the expansion of p by substituting -1 or 1 for each public variable, so if $\widehat{F|_p}(S') \neq 0$, then $\widehat{F}(S' \cup p') \neq 0$ for some $p' \subseteq p$. By Lemmas 8 and 9, this means that there is a $D \subseteq \{1, \ldots, d\}$, $T_1 \ldots T_d \subseteq X$ such that $\triangle_{d \in D} T_k = S'$, and $i \in K$, such that that $T_i \in \mathcal{S}(g_i)$. □

Note that for the first order attacks, we just need to consider the labels of all gates (see Lemma 8. For order d attacks, conceptually we connect a d-ary AND gate to any set of d gates and check that the label of this gate does not contain a label with secrets and no masks.

It is worth pointing out that the converse of the theorem does not hold. As an example of imprecision, one can construct $a \oplus b$ as a combination of three nonlinear gates. In this case, the output would be labeled $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ although the Fourier expansion of the circuit is $a \cdot b$.

C Proof of the Transient Verification Approach

Lemma 11. *For any gate* g *of C and any $T \subseteq X$, if for any $C_{glitch} \in \mathcal{C}_{glitch}$, either $\widehat{g}(T) \neq 0$ or $\widehat{g}'(T) \neq 0$, then $T \in \mathcal{T}(g)$.*

Proof. The proof follows that of Lemma 8 with the modification that regardless of the function of a gate g', if g' has inputs g_1 and g_2, then $\widehat{g'}(T) \neq 0$ implies that either (1) $\widehat{g_1}(T) \neq 0$, (2) $\widehat{g_2}(T) \neq 0$, or (3) $\exists T' \subseteq X . \widehat{g_1}(T') \neq 0$ and $\widehat{g_2}(T' \triangle T) \neq 0$. Thus, we can overapproximate the set of non-zero Fourier coefficients for a copied gate g' with inputs g_1 and g_2 by the set that consists of the union of the nonzero coefficients $\mathcal{T}(g_1)$ of g_1, the nonzero coefficients $\mathcal{T}(g_2)$ of g_2, and the set $\mathcal{T}(g_1) \triangle \mathcal{T}(g_2)$. □

Theorem 12. *Let C be a circuit. If for all $S' \subseteq S \cup P$ such that $S' \cap S \neq \emptyset$, for all sets $g_1, \ldots, g_d$ of gates, for all $D \subseteq \{1, \ldots, d\}$, for all $T_1 \ldots T_d \subseteq X$ such that $\triangle_{d \in D} T_d = S'$, and for all $i \in D$, we have that $T_i \notin \mathcal{T}(g_i)$, then C is order d secure.*

Proof. The proof proceeds along the lines Theorem 12 by using Lemma 11 instead of Lemma 8. □

D Proof of the SAT Based Verification for Stable Signals

Lemma 13. *For any gate $g \in \mathcal{G}$ and any $X' \subseteq X$, if $\widehat{F}(g)(X') \neq 0$ then there is a satisfying assignment χ of Ψ_{gates} with $\chi(x_g) = \texttt{true}$ iff $x \in X'$.*

Proof. The Fourier expansion of any gate g can be obtained recursively from the Fourier expansion of its inputs, where the coefficient of every Fourier character is obtained by multiplying out the coefficients of the inputs. For a linear gate, this is a simple multiplication. A nonlinear gate results in more Fourier coefficients: those for each input separately, and those that result from the multiplication of the inputs. We can represent the way that each Fourier character is derived as a subgraph of the circuit.

Suppose that a coefficient $\widehat{f}(g)(X')$ is nonzero. Then there is a set of wires W' and a set of gates G' that fulfills the following constraints. (1) $g \in G'$. (2) If $g \in G'$ and g is a linear gate, then both incoming wires are in W'; if g' in nonlinear, than 0, 1, or 2 of the incoming wires are in W'. (3) if $w \in W'$ then the gate g' that feeds w is in G'. (4) X' is the symmetric set difference of the nonzero Fourier coefficients of the inputs that feed into an *odd* number of paths to g. The latter observation follows from the fact that in Fourier representation $f^2 = 1$ for any Boolean function f.

The choice of W' and G' is dictated by the choices made at the nonlinear gates, which corresponds to the disjuncts in the definition of $\Psi_{\text{nl}}(g)$. The satisfying assignments of the formula then follow the paths described above, where the cancellation of coefficients that occur an even number of times is ensured by the XORs in the formulas for linear and nonlinear gates. □

Theorem 14. *If C is not order d secure without glitches then Ψ is satisfiable.*

Proof. The theorem follows easily from the Lemma 13 by the fact that information leakage occurs iff there is a gate which is statistically dependent on a set of secret variables (Lemma 4, Definition 6). □

Note if the formula Ψ is satisfiable, i.e. the circuit is not secure, we can easily see what gates the solver picked for the probes. The activation variables for those gates are equal to *true*.

E Proof of the SAT Based Approach for Transient Signals

Lemma 15. *For any circuit $C' \in \mathcal{C}_{glitch}(C)$, gate* g *in C', and $X' \subseteq X$, if $\widehat{F}(g)(X') \neq 0$, then there is a satisfying assignment χ of Φ_{gates} with $\chi(x'_g) = \texttt{true}$ iff $x \in X'$.*

Proof. The proof follows that of Lemma 13 with the modification that for a gate g, an assignment to the variables $\{x'_g \mid x \in X\}$ is part of a satisfying assignment if the corresponding Fourier coefficient is non-zero for *any* $C' \in \mathcal{C}_{\text{glitch}}(G)$. Intuitively, a nonlinear gate presents the "worst-case" scenario that subsumes the behavior of an arbitrary gate. □

Theorem 16. *If C is not order d secure with glitches then Φ is satisfiable.*

Proof. The proof follows from Lemmas 4, 15, and Definition 7. □

F Example for the SAT Encoding

To illustrate the encoding let us consider the example on the Fig. 2. For this circuit we have one secret variable s and two mask variables m_s and m_1. Since input s_m is driven by the function $\mathcal{I}(s_m) = s \oplus m_s = s \cdot m_s$, we have the constraint

$$\psi_{\text{inp}}(s_m) = s_{s_m} \wedge m_{s,s_m} \wedge \neg m_{1,s_m} \wedge \neg p_{1,s_m}.$$

For the other inputs we have:

$$\psi_{\text{inp}}(m_1) = \neg s_{m_1} \wedge \neg m_{s,m_1} \wedge m_{1,m_1} \wedge \neg p_{1,m_1}$$

$$\psi_{\text{inp}}(m_s) = \neg s_{m_s} \wedge m_{s,m_s} \wedge \neg m_{1,m_s} \wedge \neg p_{1,m_s}$$

$$\psi_{\text{inp}}(p_1) = \neg s_{p_1} \wedge \neg m_{s,p_1} \wedge \neg m_{1,p_1} \wedge p_{1,p_1}.$$

For the linear gates g_1 and g_3 we use the linear rule

$$\begin{aligned}\Psi_{\text{lin}}(g_1) = (s_{g_1} \leftrightarrow (s_{s_m} \oplus s_{m_1})) \wedge (m_{s,g_1} \leftrightarrow (m_{s,s_m} \oplus m_{s,m_1}))\\ \wedge (m_{1,g_1} \leftrightarrow (m_{1,s_m} \oplus m_{1,m_1}) \wedge (p_{1,g_1} \leftrightarrow (p_{1,s_m} \oplus p_{1,m_1}))\end{aligned}$$

$$\begin{aligned}\Psi_{\text{lin}}(g_3) = (s_{g_3} \leftrightarrow (s_{g_2} \oplus s_{g_1})) \wedge (m_{s,g_3} \leftrightarrow (m_{s,g_1} \oplus m_{s,g_2}))\\ \wedge (m_{1,g_3} \leftrightarrow (m_{1,g_1} \oplus m_{1,g_2}) \wedge (p_{1,g_3} \leftrightarrow (p_{1,g_1} \oplus p_{1,g_2})).\end{aligned}$$

For the non-linear gate g_2 we use the non-linear rule:

$$\begin{aligned}\Psi_{nl}(g_2) = (\neg s_{g_2} \wedge \neg m_{s,g_2} \wedge \neg m_{1,g_2} \wedge \neg p_{1,g_2})\\ \vee (s_{g_2} \leftrightarrow s_{m_s} \wedge m_{s,g_2} \leftrightarrow m_{s,m_s} \wedge m_{1,g_2} \leftrightarrow m_{1,m_s} \wedge p_{1,g_2} \wedge p_{1,m_s})\\ \vee (s_{g_2} \leftrightarrow s_{p_1} \wedge m_{s,g_2} \leftrightarrow m_{s,p_1} \wedge m_{1,g_2} \leftrightarrow m_{1,p_1} \wedge p_{1,g_2} \wedge p_{1,p_1})\\ \vee (s_{g_2} \leftrightarrow (s_{m_s} \oplus s_{p_1})) \wedge (m_{s,g_2} \leftrightarrow (m_{s,m_s} \oplus m_{s,p_1}))\\ \wedge (m_{1,g_2} \leftrightarrow (m_{1,m_s} \oplus m_{1,p_1}) \wedge (p_{1,g_2} \leftrightarrow (p_{1,m_s} \oplus p_{1,p_1})).\end{aligned}$$

For the checking gate we have:

$$\begin{aligned}\Psi(g_c) = (s_{g_c} \leftrightarrow (a_{s_m} \wedge s_{s_m} \oplus a_{m_1} \wedge s_{m_1} \oplus a_{m_s} \wedge s_{m_s} \oplus a_{p_1} \wedge s_{p_1}))\\ \wedge (m_{1,g_c} \leftrightarrow (a_{s_m} \wedge m_{1,s_m} \oplus a_{m_1} \wedge m_{1,m_1} \oplus a_{m_s} \wedge m_{1,m_s} \oplus a_{p_1} \wedge m_{1,p_1}))\\ \wedge (m_{s,g_c} \leftrightarrow (a_{s_m} \wedge m_{s,s_m} \oplus a_{m_1} \wedge m_{s,m_1} \oplus a_{m_s} \wedge m_{s,m_s} \oplus a_{p_1} \wedge m_{s,p_1}))\\ \wedge (p_{1,g_c} \leftrightarrow (a_{s_m} \wedge p_{1,s_m} \oplus a_{m_1} \wedge p_{1,m_1} \oplus a_{m_s} \wedge p_{1,m_s} \oplus a_{p_1} \wedge p_{m,p_1})).\end{aligned}$$

To check first order security we bound a_{sum} to 1:

$$a_{sum} = Ite(a_{s_m}, 1, 0) + Ite(a_{m_1}, 1, 0) + Ite(a_{m_s}, 1, 0) + Ite(a_{p_1}, 1, 0) \\ + Ite(a_{g_1}, 1, 0) + Ite(a_{g_2}, 1, 0) + Ite(a_{g_3}, 1, 0)$$

The unsafety constraint is

$$\Psi_{\text{unsafe}}(g_c) = s_{g_c} \land \neg m_{s,g_c} \land \neg m_{1,g_c}.$$

In this example the formula Ψ for the entire circuit is unsatisfiable meaning that it is first secure in the probing model without glitches.

References

1. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B.: Compositional verification of higher-order masking: application to a verifying masking compiler. IACR Cryptology ePrint Archive, 2015:506 (2015)
2. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 457–485. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_18
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Proceedings of the 2016 ACM SIGSAC CCS, Vienna, Austria, 24–28 October 2016, pp. 116–129 (2016)
4. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., Strub, P.: Parallel implementations of masking schemes and the bounded moment leakage model. IACR Cryptology ePrint Archive, 2016:912 (2016)
5. Barthe, G., Dupressoir, F., Grégoire, B., Stoughton, A., Strub, P.: EasyCrypt: Computer-Aided Cryptographic Proofs (2017). https://github.com/EasyCrypt/easycrypt
6. Bayrak, A.G., Regazzoni, F., Novo, D., Ienne, P.: Sleuth: automated verification of software power analysis countermeasures. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 293–310. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_17
7. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Randomness complexity of private circuits for multiplication. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 616–648. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_22
8. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 397–426. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_14
9. Bertoni, G., Martinoli, M.: A methodology for the characterisation of leakages in combinatorial logic. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 363–382. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_21

10. Bhasin, S., Carlet, C., Guilley, S.: Theory of masking with codewords in hardware: low-weight dth-order correlation-immune boolean functions. IACR Cryptology ePrint Archive, 2013:303 (2013)
11. Bilgin, B., Bogdanov, A., Knežević, M., Mendel, F., Wang, Q.: Fides: lightweight authenticated cipher with side-channel resistance for constrained hardware. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 142–158. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_9
12. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_18
13. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26
14. Coron, J.-S.: Formal verification of side-channel countermeasures via elementary circuit transformations. Cryptology ePrint Archive, Report 2017/879
15. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
16. Eldib, H., Wang, C.: Synthesis of masking countermeasures against side channel attacks. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 114–130. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_8
17. Eldib, H., Wang, C., Schaumont, P.: SMT-based verification of software countermeasures against side-channel attacks. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 62–77. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_5
18. Eldib, H., Wang, C., Taha, M.M.I., Schaumont, P.: QMS: evaluating the side-channel resistance of masked software from source code. In: DAC 2014, San Francisco, CA, USA, 1–5 June 2014, pp. 209:1–209:6 (2014)
19. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable masking schemes in the presence of physical defaults and the robust probing model. IACR Cryptology ePrint Archive, 2017:711 (2017)
20. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting circuits from leakage: the computationally-bounded and noisy cases. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 135–156. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_7
21. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A Testing Methodology for Side-Channel Resistance Validation. In: NIST Non-Invasive Attack Testing Workshop (2011)
22. Gross, H.: Collection of protected hardware implementations. https://github.com/hgrosz
23. Gross, H., Mangard, S.: Reconciling d+1 masking in hardware and software. In: CHES 2017 (2017)
24. Gross, H., Mangard, S., Korak, T.: An efficient side-channel protected AES implementation with arbitrary protection order. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 95–112. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_6
25. Gross, H., Schaffenrath, D., Mangard, S.: Higher-order side-channel protected implementations of keccak. Cryptology ePrint Archive, Report 2017/395

26. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
27. Iusupov, R.: REBECCA - Masking verification tool. https://github.com/riusupov/rebecca
28. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
29. Mangard, S., Schramm, K.: Pinpointing the side-channel leakage of masked AES hardware implementations. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 76–90. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_7
30. Moss, A., Oswald, E., Page, D., Tunstall, M.: Compiler assisted masking. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 58–75. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_4
31. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (2006). https://doi.org/10.1007/11935308_38
32. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press, Cambridge (2014)
33. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45418-7_17
34. Reparaz, O.: Detecting flawed masking schemes with leakage detection tests. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 204–222. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_11
35. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_37
36. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_28
37. Trichina, E.: Combinational logic design for AES subbyte transformation on masked data. IACR Cryptology ePrint Archive (2003)
38. Wolf, C., Glaser, J.: Yosys - a free verilog synthesis suite. In: Proceedings of Austrochip 2013 (2013)
39. Xiao, G., Massey, J.L.: A spectral characterization of correlation-immune combining functions. IEEE Trans. Inf. Theory **34**(3), 569–571 (1988)