

# 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  
`{firstname.lastname}@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 and is not bound to a certain leakage model. The verification is performed directly on the circuit's netlist, and covers also higher-order and multivariate flaws. Therefore, 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 an 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 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

## 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 [26] or electromagnetic emanation [31], 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 [23, 25, 29, 33, 36] 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 [25, 36]. The first provably secure masking scheme with inherent resistance to glitches was the threshold implementation (TI) scheme of Nikova et al. [29]. 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. [33], and the domain-oriented masking scheme (DOM, and the low-randomness variant UMA) of Gross et al. [22, 23].

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 [20] or by attacking the devices by using state-of-the-art side-channel analysis techniques. Such practical evaluations, however, have several severe limitations: (1) they are time consuming, (2) they can hardly be applied during the design phase, (3) they do not allow for general security statements that remain valid for other instances of the implementation (different FPGAs or ASIC chips) or for different environmental conditions than for which they were evaluated, (4) when a flaw is detected, localization must be performed manually, (5) they are never conclusive, meaning that if no flaw is found it remains uncertain whether 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 [32] 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 d-probing model of Ishai et al. [25] 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 [23]. 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 [23], the parallelized evaluation takes about 10 hours. 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. [25, 36]. The implementation of our tool and some example circuits are available on github [21].

## 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, 28]. Most of the existing work, however, targets software based masking and provides only heuristic security without formal proofs.

*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 randomness efficient software masking algorithms of Belaid et al. [7, 8], Barthe et al. [4], and Gross et al. [22], for example, do not achieve SNI directly but require additional randomness through remasking.

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 to cover glitches in the verification of masked circuits. Our approach is thereby not bound to our SAT realization but could also be integrated in similar form in 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. Therefore, 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 [22, 23, 29, 33]. Bertoni et

al. demonstrated their approach on a masked implementation of Keccak based on a masking scheme that is known to be not resistant to glitches.

In contrast to Bertoni et al.’s work, our approach considers combinatorial logic as well as sequential gates (registers), covers also higher-order and multivariate (different points in time) leakages, and is not restricted to circuits that produce only one output bit. In terms of efficiency, Bertoni et al.’s approach of enumerating all possible glitches requires double exponential effort, while our SAT based approach has only exponential effort.

In the work of Reparaz [32], a leakage assessment approach is introduced that works by simulating leakages of a targeted hardware implementation in software. Therefore, 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.

### 3 Preliminaries

In the following we will extensively make use of the usual set notation, where  $S \Delta T = S \setminus T \cup T \setminus S$  denotes the symmetric difference of  $S$  and  $T$  and for two sets of sets  $\mathcal{S}$  and  $\mathcal{T}$ , we define  $\mathcal{S} \Delta \mathcal{T} = \{S \Delta T \mid S \in \mathcal{S}, T \in \mathcal{T}\}$  to be the pointwise set difference of all elements.

We write  $\mathbb{B} = \{\text{true}, \text{false}\}$  for the set of Booleans. For a set  $X$  of Boolean variables, we will identify an assignment  $f : X \rightarrow \mathbb{B}$  with the set of variables  $x$  for which  $f(x) = \text{true}$ . 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. We will first formally define statistical independence.

**Definition 1 (Statistical independence)** *Let  $X, Y$ , and  $Z$  be sets of Boolean variables and let  $f : 2^X \times 2^Y \rightarrow 2^Z$ . 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  $F : \mathbb{B}^X \times \mathbb{B}^Y \rightarrow \mathbb{B}^Z$ . Function  $F$  is statistically independent of  $X$  iff for all functions  $f : \mathbb{B}^Z \rightarrow \mathbb{B}$  we have that  $f \circ F$  is statistically independent of  $X$ .*

*Proof.* 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  $|\{y \mid F(x, y) = z\}| \neq |\{y \mid F(x', y) = z\}|$ .

**Table 1.** Truth table of AND and XOR gates for two input signals  $x$  and  $y$

Inputs		Outputs	
$x$	$y$	$x \wedge y$	$x \oplus y$
0	0	0	0
	1	0	1
1	0	0	1
	1	1	0

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  $X$ . □

As an example for statistical dependency, consider the truth table in Table 1 for an AND and an XOR gate for two uniformly distributed input signals  $x$  and  $y$ . By observing the output signal of an AND gate calculating  $x \wedge y$  information about both  $x$  and  $y$  can be obtained, because if  $x \wedge y$  is 1 then both  $x$  and  $y$  are 1. The Boolean function calculated by an AND gate is therefore statistically dependent of  $x$  and  $y$ .

For the XOR gate, the information that can be learned at the output differs as can be seen again in Table 1. One can not obtain information about  $x$  or  $y$  from  $x \oplus y$ , because for both cases  $x \oplus y = 0$  and  $x \oplus y = 1$  all values for  $x$  (and  $y$ ) could be either 0 or 1. However, what can be determined is whether  $x$  and  $y$  are equal. In this case  $x$  “masks”  $y$  and vice versa because the determination of  $x$  requires a priori knowledge about  $y$ . The XOR function is in this case statistically independent of  $x$  and  $y$ .

To define Fourier expansions, we will follow the exposition of [30] 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  $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$  can be uniquely expressed as a multilinear polynomial in the  $n$ -tuple of variables  $X = (x_1, x_2, \dots, x_n)$  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  $\hat{f}(T)$  of the subset  $T$ . Thus we have the Fourier representation of  $f$ :

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

The Fourier characters  $\chi_T : \{-1, 1\}^n \rightarrow \{-1, 1\}$  form an orthonormal basis for the vector space of functions in  $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$ . The Fourier coefficients are given by the projection of the function to its basis, i.e., for  $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$  and  $T \subseteq X = (x_1, x_2, \dots, x_n)$ , the coefficient  $\hat{f}(T)$  is given by  $\hat{f}(T) = 1/2^n \cdot \sum_{X \in \{\pm 1\}^n} (f(X) \cdot \chi_T(X))$ . In order to prevent confusion between

multiplication and addition on rationals and conjunction and XOR on Booleans, we will write  $\cdot$  and  $+$  for the former and  $\wedge$  and  $\oplus$  for the latter.

As an example, the Fourier expansion of  $x \wedge y$  in Table 1 is

$$1/2 + 1/2 \cdot x + 1/2 \cdot y - 1/2 \cdot x \cdot y. \quad (1)$$

If  $x = \text{false} = 1$  and  $y = \text{true} = -1$ , for example, the polynomial evaluates to  $1/2 + 1/2 - 1/2 + 1/2 = 1 = \text{false}$  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 any Boolean function  $f$  [30]. (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  $\hat{f}(\emptyset)$ ,  $\hat{f}(\{x\})$ ,  $\hat{f}(\{y\})$ , and  $\hat{f}(\{x, y\})$  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 [38])** *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  $\hat{f}(T) = 0$ .*

For the AND gate in Table 1 the Fourier coefficients in Equation 1 reveal that there is a positive correlation with  $x$  and  $y$  because the Fourier coefficients in both cases are  $1/2$ . There is also a negative correlation with  $x \cdot y$  indicated by the coefficient  $-1/2$ . For the XOR gate on the other hand there is only one nonzero coefficient for  $x \cdot y$ . There is thus no direct correlation with  $x$  or  $y$  but there is one for  $x \cdot y$ .

## 4 Masking and the d-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 Equation 2 that is protected by adding a uniformly random mask  $m_s$ , resulting in the masked representation  $s_m$ .

$$s_m = s \oplus m_s. \quad (2)$$

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 referred to as the protection order  $t$ .

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

- $S = \{s_1, \dots, s_j\}$  are security sensitive variables such as key material 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 but are also 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  $f(g) : \mathbb{B}^k \rightarrow \mathbb{B}$ , and  $\mathcal{I} : I \rightarrow (2^X \rightarrow \mathbb{B})$  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 will assume, wlog, that all gates, except inputs and registers, have indegree 2 and we will 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.

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  $F(g) : 2^X \rightarrow \mathbb{B}$ , 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  $F(g_1, \dots, g_d) : 2^X \rightarrow \mathbb{B}^d: F(g_1, \dots, g_d)(x) = (g_1(x), \dots, g_d(x))$ . We will often identify a gate with its function.

As an example, consider the circuit on Figure 1 showing a masked circuit and the according circuit graph in Figure 2. We have  $f(g_3)(a, b) = a \oplus b$  and  $F(g_3) = (s_m \oplus m_1) \oplus (m_s \wedge p_1)$ . Please note that we use the example in Figure 1 throughout the whole paper to give examples for the introduced concepts (sometimes in reduced or slightly modified form).

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



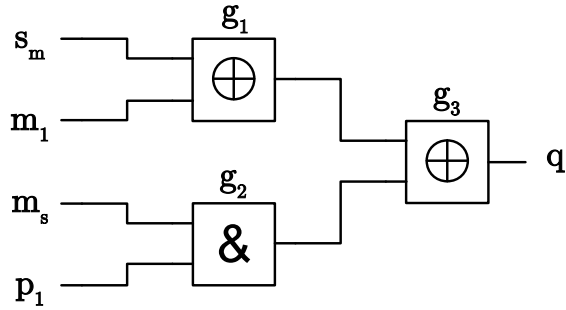


Fig. 1. Masked circuit example

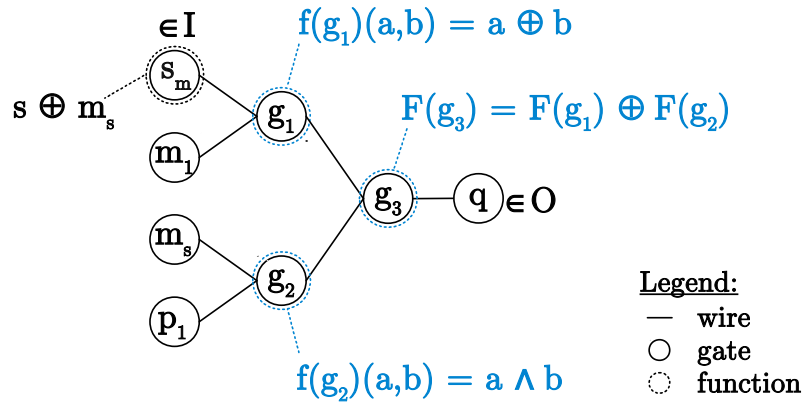


Fig. 2. Circuit graph of circuit in Figure 1

**Security of masked circuits.** The security of various masking schemes is often analyzed in the so-called  $d$ -probing model that was introduced by Ishai et al. [25]. It was shown by Faust et al. [19] and Rivain et al. [34] that the  $d$ -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  $d$ -probing model, an attacker is bound to  $d$  probing needles which can be freely placed on arbitrary circuit gates (or wires) to reveal intermediate results during the computation of security critical operations. The  $d$ -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  $d$ -probing model if an attacker cannot combine the information gathered from  $d$  probes in a function  $F$  that is statistically dependent on 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, but 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. Note that this allows the attacker to control the public variables (but not to change them during a single clock cycle).

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

**Definition 6 ( $t$ -probing security [25])** A circuit  $C = (\mathcal{G}, \mathcal{W}, f, \mathcal{I})$  is order  $d$  probing secure ( $d$ -probing secure) iff for any gates  $g_1, \dots, g_d \in \mathcal{G}$ ,  $F(g_1, \dots, g_d)$  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 an  $\emptyset \neq S' \subseteq S \cup P$  such that  $\widehat{F}(g)(S') \neq 0$ . The first-order security of a circuit can thus be verified using the  $d$ -probing model by calculating the Fourier expansion of the whole circuit. As an example consider the Fourier expansion of the circuit in Figure 1 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

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

For the example circuit in Figure 1, if  $s$  is a secret and  $m_1$  is a uniformly distributed random mask, then  $g_3$  in Equation 3 computes a function that does not reveal any secret information: any Fourier coefficient that contains  $s$  also contains  $m_1$ . However, if e.g.  $m_1$  would be a public value, then  $g_3$  would reveal a correlation with a secret as evidenced by the coefficient  $-1/2 \cdot s p_1 m_1$ .

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

## 5 Verification in the $d$ -Probing Model without Glitches

In this section, we will present a sound verification method for  $d$ -probing security that for the moment neglects signal propagation times. Therefore, it is assumed that the signals at the circuit input are fixed and do not change anymore, and that all intermediate signals at the gates and the circuit output have reached their final state. This approach will be later on extended in Sections 6 and 7 to cover also signal transitions and timing effects.

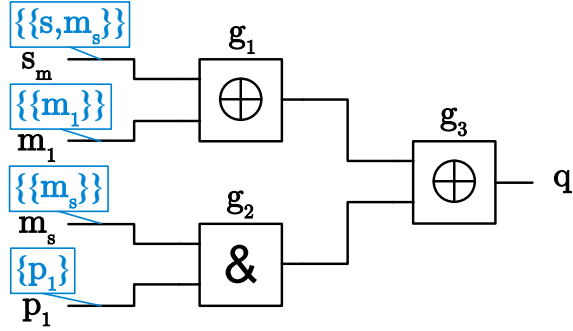


Fig. 3. Masked circuit example with initial labeling (blue)

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  $\mathcal{S} : \mathcal{G} \rightarrow 2^{2^X}$  that associates a set of sets of variables to every gate. This labeling system is based on the Fourier representation of Boolean functions (see Section 3) and intuitively, a label contains at least those sets  $X' \subseteq X$  for which  $\hat{f}(X') \neq 0$  (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  $\mathcal{S}(g) = \{X' \subset X \mid \hat{f}_g(X') \neq 0\}$ . 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 Figure 3 (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 flow) 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 Figure 3 contain only the mask variables. Formally, for inputs  $g \in I$  with function  $\mathcal{I}(g) = f(X)$ , we set  $\mathcal{S}(g) = \{X' \mid X' = X\}$ .

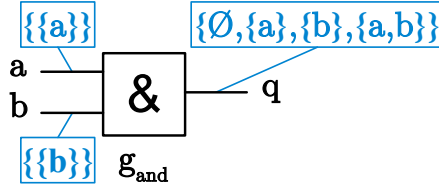


Fig. 4. Nonlinear gate and for an AND example, labels denoted in blue

## 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 :

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

Figure 4 shows a simple example for a AND gate calculating  $a \wedge b$ . The resulting labels denote the information that can be learned by probing this gate which could be either  $a$  or  $b$  alone, or together. The labeling reflects the Fourier spectrum of the AND gate (see Equation 1). In particular the labeling shows all terms of the Fourier polynomial which coefficients are different from zero and are therefore statistical dependent.

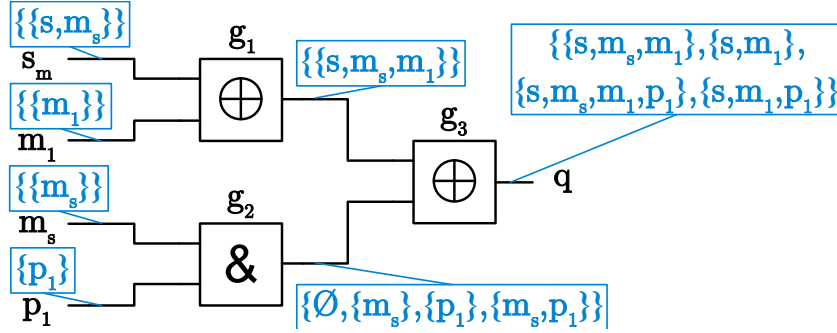
**Linear gate rule.** By following the Definition 3 of the Fourier expansions further we can also model linear gates which have a reduced spectrum compared to nonlinear gates. We model this circumstance by introducing a new rule for labeling a linear gate  $g \in L$  with inputs  $g_a$  and  $g_b$ :

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

**Combined example.** To demonstrate how the propagation step works in practice, we applied the propagation rules (summarized in Table 2) to the example circuit in Figure 3. The result is shown in Figure 5. 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 biased signal  $p_1$  with the mask  $m_s$  in the nonlinear gate biases the 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.

**Table 2.** 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	$\mathcal{S}(g) = \{\emptyset\} \cup \mathcal{S}(g_a) \cup \mathcal{S}(g_b) \cup \mathcal{S}(g_a) \Delta \mathcal{S}(g_b)$
Linear gate	$\mathcal{S}(g) = \mathcal{S}(g_a) \Delta \mathcal{S}(g_b)$
Register	$\mathcal{S}(g) = \mathcal{S}(g_a)$



**Fig. 5.** Masked circuit example from Figure 3 after the propagation step

### 5.3 Verification and Proof of Correctness

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 Figure 5, all of the labels that contains  $s$  also contain  $m_1$  or  $m_s$  and therefore the circuit is accepted by the verifier. For higher order attacks, we must check that the non-linear combination of any tuple of  $d$  gates does not contain a secret variable without a mask. This causes an obvious combinatorial blowup. In Section 8, we will show how to harness a SAT solver to combat this problem.

In the next section we will extend the verifier to cover glitches, which will show that the example circuit is actually insecure when signal timings are considered.

**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 combine 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. On 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 Section 8, we will show how to harness a SAT solver to combat this problem.

We will now prove the correctness of our approach formally. The first lemma shows that the labels that we compute consist of all non-zero Fourier coefficients.

**Lemma 7** *For any circuit  $C$ , any gate  $g$  and any  $T \subseteq X$ , if the Fourier coefficient  $\hat{g}(T) \neq 0$ , 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  $u(X) = \sum_{T \subseteq X} \hat{u}(T) \chi_T(X)$ , and  $v(X) = \sum_{T \subseteq X} \hat{v}(T) \chi_T(X)$ .

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

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

where

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

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

*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  $\alpha_{ij}$ . Consequently,  $\hat{g}(T) \neq 0$  implies that either (1)  $\hat{u}(T) \neq 0$ , (2)  $\hat{v}(T) \neq 0$ , or (3)  $\exists T' \subseteq X. \hat{u}(T') \neq 0$  and  $\hat{v}(T' \Delta T) \neq 0$ . (The converse does not hold.) In each of these three conditions,  $T \in \mathcal{S}(w)$ .  $\square$

The next lemma shows that a arbitrary function fo  $d$  gates corresponds to a generalization of the non-linear rule from Table 2. (Note the use of  $\Delta_{d \in D} T_d$  to denote the symmetric set difference of all  $T_d$ s.)

**Lemma 8** 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  $T = \Delta_{i \in D} T_i$  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

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

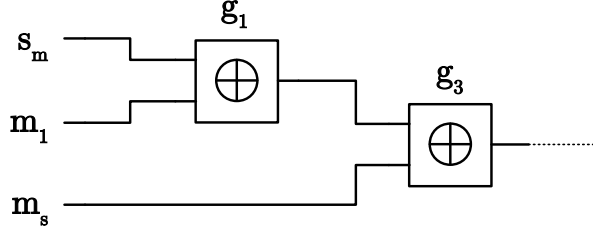
Consequently, if  $\widehat{F}(T) \neq 0$ , then there is a  $D \subseteq \{1, \dots, d\}$  and  $T_1 \dots T_d \subseteq X$  such that for all  $i \in D$   $\widehat{F}_i(T_i) \neq 0$  and  $T = \Delta_{i \in D} T_i$ .  $\square$

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  $T = T_1 \Delta T_2$ ,  $\widehat{F}_1(T_1) \neq 0$ , and  $\widehat{F}_2(T_2) \neq 0$ , which reflects the nonlinear rule.

**Theorem 9** Let  $C$  be a circuit. If for all  $S' \subseteq S \cup P$  such that  $S' \cap S \neq \emptyset$ , for all sets  $g_1, \dots, g_d$  of gates, for all  $D \subseteq \{1, \dots, d\}$ , for all  $T_1 \dots T_d \subseteq X$  such that  $\Delta_{i \in D} T_i = S'$ , 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 Definition 5 and 6, and Lemma 2, this implies that there are gates  $g_1, \dots, g_d$ , a function  $f$  and an assignment  $p \subseteq P$  such that  $f \circ F|_p(g_1, \dots, g_d)$  is statistically dependent on  $S$ . By Lemma 4, this implies that for some  $S' \subseteq S$ ,  $S' \neq \emptyset$ ,  $f \circ \widehat{F|_p}(g)(S') \neq \emptyset$ . 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 Lemma 7 and 8, this means that there is a  $D \subseteq \{1, \dots, d\}$ ,  $T_1 \dots T_d \subseteq X$  such that  $\Delta_{i \in D} T_i = S'$ , and  $i \in D$ , such that  $T_i \in \mathcal{S}(g_i)$ .  $\square$

Note that for the first order attacks, we just need to consider the labels of all gates (see Lemma 7. 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.



**Fig. 6.** Masked circuit example, insecure due to glitches

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  $a \cdot b$ .

## 6 The d-Probing Model in the Presence of Glitches

Like in the original d-probing model, the effects of timing differences of the signals have been neglected so far. As an example of what can go wrong when signal timings are not taken into account [27], consider the circuit in Figure 6. The depicted circuit is secure in the original d-probing model as introduced in [25].

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

$$g_1 = s_m \oplus m_1 = s \oplus m_s \oplus m_1 \text{ and}$$

$$g_3 = s_m \oplus m_1 \oplus m_s = s \oplus m_1.$$

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 d-probing model.

However, if we assume that in a subsequent clock cycle (Cycle 2 in Figure 7) 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 7 shows an example on how these changes may 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



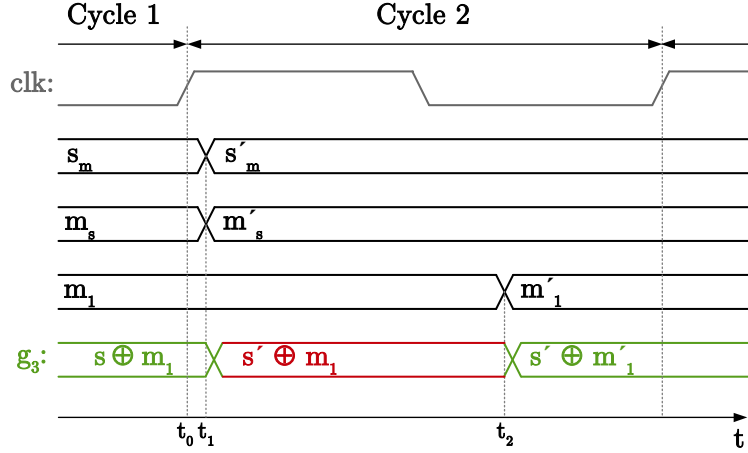


Fig. 7. Waveform example for the circuit in Figure 6, showing security critical glitch (red)

of a circuit in the d-probing model with glitches, all possible signal combinations that could arise because of propagation delays of signals need to be considered.

Please note, that probing the same circuit position at different points in time (so-called multivariate probing) does not interfere with the protection order  $d$  because still only one probe is used. Multivariate probing models the fact that leakage traces like power traces reflect signal transitions rather than signal states. This may not be confused with multivariate attacks of a side-channel attacker that combines different points in time in a power trace to perform higher-order analysis. Higher-order multivariate attacks are, however, covered by our formal verification approach that is introduced in the next section. To make our approach useful but still efficient enough to verify practical circuits we first specify the class of circuits that we consider.

**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. (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 Figure 6, 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.

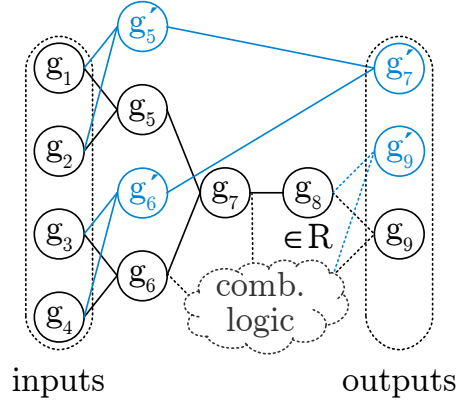
**Reducing the number of variables to model multivariate probing.** Based on the aforementioned assumptions on circuits, we can reduce the number of consecutive clock cycles that need to be considered to model multivariate probing and therefore the number of instances per input variable. From these assumptions 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 uncontrollable 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. We will show that it is therefore enough to consider the present state of a circuit (before the next clock cycle begins) and the transition to the next state to make statements on the security of such a circuit in the presence of glitches over arbitrary number of clock cycles.

Therefore we consider the following two situations: 1) The circuit in Figure 6 is considered after the reset phase, such that all inputs are at first set to 0. After the reset phase the first input variables  $s_m$ ,  $m_1$ , and  $m_s$  are applied to the circuit and the inputs change from 0 to the according variables in an unpredictable order. For example if  $s_m$  transits before  $m_1$  then an attacker that probes at gate  $g_1$  could at first probe  $s_m$  and later on  $s_m \oplus m_1$ . 2) In the next cycle the variables change from  $s_m$ ,  $m_1$ , and  $m_s$  to  $s'_m$ ,  $m'_1$  and  $m'_s$ , respectively. If again  $s_m$  changes at first to  $s'_m$  the attacker can observe the change from  $s_m \oplus m_1$  to  $s'_m \oplus m_1$  at  $g_1$  and learns the relation between  $s_m \oplus s'_m$ . Since the attacker knows  $s_m$  from the last cycle she can also determine  $s'_m$ .

Comparing these two situations, it shows that under the stated assumptions in a multivariate setting it does not make a difference for the security analysis of the circuit which situation we consider. An attacker can always observe differences from the current state to the state before and gain the same information (only from a different cycle) as if the later transiting signals are set to 0. Instead of considering multiple instances of the same variables over different clock cycles we can thus use aliases for all of these variable instances, and write e.g.  $s_m$  instead of  $s'_m$  and so on, to describe the effects of multivariate probing. This will reduce the amount of possible labels in the next section where we extend the introduced set of propagation rules to also cover glitches.

**Formalization.** To formalize the d-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. We also extend this approach to higher protection orders in the following.

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 the attacker to change gates in the copy, whereas the original, unmodified signals are propagated by the unmodified gates. Figure 8 illustrates an example for the modeling of the glitches. The copied gates,



**Fig. 8.** Example for modeling of glitches of a circuit  $C$  (without blue parts) in  $C'$

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:

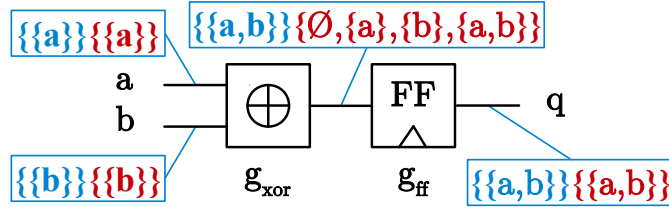
$$\mathcal{C}_{\text{glitch}}(C) = \{C'[(g'_1, \dots, g'_n) \mapsto (f_1, \dots, f_n)] \mid (g_1, \dots, g_n) \in \mathcal{G}^n, \forall i. f_i : \mathbb{B}^2 \rightarrow \mathbb{B}\}.$$

**Definition 10** (*d-probing security with glitches*) *A circuit  $C$  is order  $d$  probing secure with glitches iff for any  $C_{\text{glitch}} = (\mathcal{G}', \mathcal{W}', R, f', \mathcal{I}) \in \mathcal{C}_{\text{glitch}}$  and any gates  $g_1, \dots, g_d \in \mathcal{G}'$ ,  $F(g_1, \dots, g_d)$  is secure.*

## 7 Verification in the t-Probing Model with Glitches

Up to now, we have only considered the stable output of gates in our verification approach. In real circuits, signals and gates have different timings because of various circuit effects like gate delays, differences in wire dimensions, different input capacitance of gates, clock skew, et cetera. These timing differences lead to ripple effects that could toggle the output of a gate several times before it finally reaches a stable state (see Section 6, glitches).

The nonlinear gate rule 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



**Fig. 9.** XOR gate rules for stable (blue) and transient (red) signal sets

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 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.

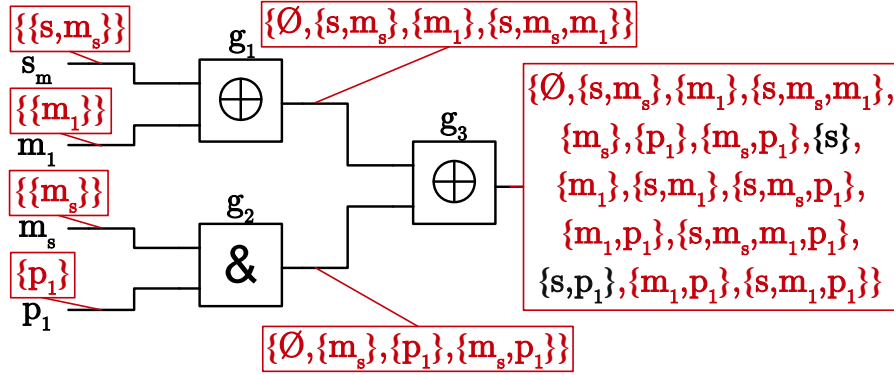
Figure 9 illustrates the new linear gate rule for the stable (blue) and the transient (red) set of labels. The stable and transient sets of the inputs are equal at the beginning because the inputs are either circuit inputs or outputs of a register. When the signals propagate through the linear XOR gate, the transient set is calculated by applying the linear rule from Table 3 and the stable set with the linear rule from Table 2. After the signal passes the register, only the stable information remains and the transient label carries thus the same information as the stable set.

Table 3 summarizes the rules for creating the transient-set labels  $\mathcal{T}(g)$ . Together with the stable rules in Table 2 this forms the basis of the circuit verifier.

**Table 3.** 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	$\mathcal{T}(g) = \{\emptyset\} \cup \mathcal{T}(g_a) \cup \mathcal{T}(g_b) \cup \mathcal{T}(g_a) \Delta \mathcal{T}(g_b)$
Linear gate	$\mathcal{T}(g) = \{\emptyset\} \cup \mathcal{T}(g_a) \cup \mathcal{T}(g_b) \cup \mathcal{T}(g_a) \Delta \mathcal{T}(g_b)$
Register	$\mathcal{T}(g) = \mathcal{S}(g_a)$

**Example.** The transient labels  $\mathcal{T}$  of the circuit in Figure 3 are shown in Figure 10 (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 Figure 3, are replaced with nonlinear gates. As can be observed at the output of the circuit, the verification under the consideration of glitches



**Fig. 10.** Masked circuit example from Figure 5 reevaluated with the transient rules (red) which leads to a flaw due to glitches (black labels)

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  $p_1 = \text{true}$  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,  $g_1$  will transition 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. Figure 7).

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 Figure 5 for  $g_1$  would thus be propagated.

In the following we proof the correctness of the verification in the d-probing model with glitches.

**Lemma 11** *For any gate  $g$  of  $C$  and any  $T \subseteq X$ , if for any  $C_{\text{glitch}} \in \mathcal{C}_{\text{glitch}}$ , 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 7 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  $\widehat{g}_2(T' \Delta T) \neq 0$ . 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  $\mathcal{T}(g_1) \Delta \mathcal{T}(g_2)$ .  $\square$

**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  $g_1, \dots, g_d$  of gates, for all  $D \subseteq \{1, \dots, d\}$ , for all  $T_1 \dots T_d \subseteq X$  such that  $\Delta_{d \in D} T_d = S'$ , 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 7.  $\square$

Note that for the first order security, we just need to check the label of each gate.

## 8 SAT Encoding

In this section, we will encode the labeling system introduced in the previous section as a formula in propositional logic. We will start with the  $d$ -probing model without glitches and then generalize to glitches. Intuitively, for every gate  $g$ , we will pick one set  $X' \subseteq \mathcal{S}(g)$ , i.e., we pick one Fourier character with a possibly nonempty coefficient. We can then encode the rules for the linear and nonlinear gates that we have seen above propositionally. To check higher-order security we connect a  $d$ -ary XOR 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. Since we allow the SAT solver pick any set of  $d$  gates it is enough to use a linear gate. If the leakage is in individual input of the checking gate we can catch it with smaller value of  $d$ .

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  $\Psi$  to check whether the masking scheme is secure. Recall that  $L$  and  $N$  are the sets of linear gates and nonlinear gates, resp. Formula  $\Psi$  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 2, where  $T = (t_1, \dots, t_n)$ ,  $U = (u_1, \dots, u_n)$ , and  $V = (v_1, \dots, v_n)$  are ordered sets of variables, and define  $\leftrightarrow$  to denote equality.

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

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

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

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

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

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

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

for a register  $g$  with input  $g'$ , we simply have  $\Psi_{\text{reg}}(g) = \Psi_{\text{copy}}(X_g, X'_g)$ . Also we introduce an integer variable  $a_{sum}$ , and bound it to the attack order  $t$ :

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

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

The function  $\text{Ite}(a_g, 1, 0)$  converts a Boolean variable to Integer.

For the checking gate we xor the corresponding inputs:

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

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:

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

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

*Example* To illustrate the encoding let us consider the example on the Figure 5 . 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 linear rule

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

$$\wedge (m_{1,g_1} \leftrightarrow (m_{1,s_m} \oplus m_{1,m_1})) \wedge (p_{1,g_1} \leftrightarrow (p_{1,s_m} \oplus p_{1,m_1}))$$

$$\Psi_{\text{lin}}(g_3) = (s_{g_3} \leftrightarrow (s_{g_2} \oplus s_{g_1})) \wedge (m_{s,g_3} \leftrightarrow (m_{s,g_1} \oplus m_{s,g_2}))$$

$$\wedge (m_{1,g_3} \leftrightarrow (m_{1,g_1} \oplus m_{1,g_2})) \wedge (p_{1,g_3} \leftrightarrow (p_{1,g_1} \oplus p_{1,g_2})).$$

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

$$\Psi_{\text{nl}}(g_2) = (\neg s_{g_2} \wedge \neg m_{s,g_2} \wedge \neg m_{1,g_2} \wedge \neg p_{1,g_2})$$

$$\vee (s_{g_2} \leftrightarrow s_{m_s} \wedge m_{s,g_2} \leftrightarrow m_{s,m_s} \wedge m_{1,g_2} \leftrightarrow m_{1,m_s} \wedge p_{1,g_2} \wedge p_{1,m_s})$$

$$\vee (s_{g_2} \leftrightarrow s_{p_1} \wedge m_{s,g_2} \leftrightarrow m_{s,p_1} \wedge m_{1,g_2} \leftrightarrow m_{1,p_1} \wedge p_{1,g_2} \wedge p_{1,p_1})$$

$$\vee (s_{g_2} \leftrightarrow (s_{m_s} \oplus s_{p_1})) \wedge (m_{s,g_2} \leftrightarrow (m_{s,m_s} \oplus m_{s,p_1}))$$

$$\wedge (m_{1,g_2} \leftrightarrow (m_{1,m_s} \oplus m_{1,p_1})) \wedge (p_{1,g_2} \leftrightarrow (p_{1,m_s} \oplus p_{1,p_1})).$$

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:

$$\begin{aligned} a_{sum} = & Ite(a_{s_m}, 1, 0) + Ite(a_{m_1}, 1, 0) + Ite(a_{m_s}, 1, 0) + Ite(a_{p_1}, 1, 0) \\ & + Ite(a_{g_1}, 1, 0) + Ite(a_{g_2}, 1, 0) + Ite(a_{g_3}, 1, 0) \\ & a_{sum} \leq 1. \end{aligned}$$

The unsafety constraint is

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

In this example the formula  $\Psi$  for the entire circuit is unsatisfiable meaning that it is first secure in the d-probing model without glitches.

**Lemma 13** *For any gate  $g \in \mathcal{G}$  and any  $X' \subseteq X$ , if  $\hat{F}(g)(X') \neq 0$  then there is a satisfying assignment  $\chi$  of  $\Psi_{gates}$  with  $\chi(x_g) = \text{true}$  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  $\hat{f}(g)(X')$  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  $\Psi_{nl}(g)$ . 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.  $\square$

**Theorem 14** *If  $C$  is not order  $d$  secure without glitches then  $\Psi$  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).  $\square$



Note if the formula  $\Psi$  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*.

*Encoding for the model with glitches* The encoding for the d-probing model with glitches follows the exposition in Section 7 and particularly rules from Table 3. 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 10. We introduce a slightly modified set of constraints, where we write  $\Phi'$  to denote a formula  $\Phi$  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:

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

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:

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

Formula  $\Phi$  contains  $2 \cdot |X| \cdot |\mathcal{G}|$  propositional variables and  $O(|X| \cdot |\mathcal{G}|)$  constraints.

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

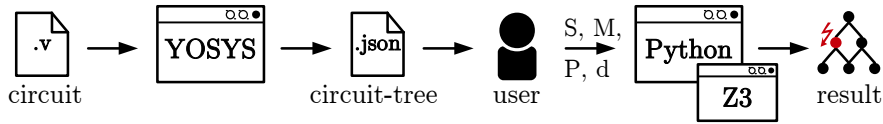
*Proof.* The proof follows that of Lemma 13 with the modification that for a gate  $g$ , an assignment to the variables  $\{x'_g \mid x \in X\}$  is part of a satisfying assignment if the corresponding Fourier coefficient is non-zero for *any*  $C' \in \mathcal{C}_{\text{glitch}}(G)$ . Intuitively, a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

**Theorem 16** *If  $C$  is not order  $t$  secure with glitches then  $\Phi$  is satisfiable.*

*Proof.* The proof follows from Lemmas 4, 15, and Definition 10.  $\square$

## 9 Practical Results

Figure 11 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 [37]



**Fig. 11.** Illustration of the verification flow

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 1000 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 [35] were not encouraging). If a flaw is found a subcircuit tree with highlighted source of a flaw is returned in DOT format which makes localization of the flaw very comfortable. 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  $S'$  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 will 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.

**Evaluation.** An overview of the experiments is given in Table 4. 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 [36], the ISW scheme [25], the threshold implementation (TI) scheme [29], and the domain-oriented masking scheme (DOM) [23]. We also check larger circuits including the AES S-box constructions of Gross et al. [23] 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. [24].

**Table 4.** 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 [36]	4	4	0	2	3	1		≤ 1 s	✗	≤ 2 s	✗
ISW AND [25]	4	4	0	2	3	0		≤ 1 s	✓	≤ 2 s	✗
TI AND [29]	6	9	0	2	4	0		≤ 1 s	✓	≤ 3 s	✓
DOM AND [23]	4	4	2	2	3	1		≤ 1 s	✓	≤ 2 s	✓
DOM Keccak S-box [24]	30	20	10	5	10	1		≤ 1 s	✓	≤ 20 s	✓
DOM AES S-box [23]	392	144	208	8	26	1					
	392	144	208	1	26	1	1	≤ 30 s	✓	≤ 9h	✓
	392	144	208	1	26	1	2	≤ 30 s	✓	≤ 10h	✓
	392	144	208	1	26	1	3	≤ 30 s	✓	≤ 8h	✓
	392	144	208	1	26	1	4	≤ 30 s	✓	≤ 8h	✓
	392	144	208	1	26	1	5	≤ 30 s	✓	≤ 7h	✓
	392	144	208	1	26	1	6	≤ 30 s	✓	≤ 8h	✓
	392	144	208	1	26	1	7	≤ 30 s	✓	≤ 9h	✓
	392	144	208	1	26	1	8	≤ 30 s	✓	≤ 6h	✓
TI Fides-160 S-box [11]	128	60	0	5	15	0					
	34	10	0	5	15	0	1	≤ 1 s	✓	≤ 2 s	✓
	48	30	0	5	15	0	2	≤ 2 s	✓	≤ 3 s	✓
	41	20	0	5	15	0	3	≤ 2 s	✓	≤ 3 s	✓
	5	0	0	5	15	0	4	≤ 1 s	✓	≤ 1 s	✓
TI Fides-192 APN [11]	4,035	3,046	0	6	24	0					
	134	44	0	0	24	0	1	≤ 2 s	✓	≤ 5 s	✓
	649	314	0	6	24	0	2	≤ 1 m	✓	≤ 15 m	✓
	1,697	1,098	0	6	24	0	3	≤ 20 m	✓	≤ 2 h	✓
	1,186	1,086	0	6	24	0	4	≤ 10 m	✓	≤ 40 m	✓
	369	504	0	6	24	0	5	≤ 2 m	✓	≤ 3 m	✓

## 9.1 Verification of First-Order Masked Implementations

Table 4 shows the verification results of the first-order masked hardware implementations. We verified each circuit both in the original d-probing model (without glitches) and the d-probing model with glitches. For larger circuits, 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 multipliers.** Figure 12 shows the so-called Trichina gate which is a masked finite-field multiplier 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.

$$q = a_m \wedge b_m \oplus a_m \wedge m_b \oplus m_a \wedge b_m \oplus m_a \wedge m_b \oplus m_q \quad (4)$$

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. For example the subgraph marked in red bring the masked signal  $b_m = \{b, m_b\}$  and its mask  $\{m_b\}$  together. The circuit verifier correctly detects the first-order flaw because a single sensitive variable  $\{b\}$  remains in label of the output  $q$  and the circuit is therefore labeled insecure. 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. The first-order masked gate corresponds to a 1-bit finite-field multiplier and can be described by Equation 5.

$$\begin{aligned} q_m &= m_a \wedge m_b \oplus m_0 \oplus a_m \wedge m_b \oplus b_m \wedge m_a \\ m_q &= a_m \wedge b_m \oplus m_0 \end{aligned} \quad (5)$$

Outputs  $q_m$  and  $m_q$  together again generate masked result which is masked by the fresh random mask  $m_0$ . While the equation of  $m_q$  is secure, a flaw is detected when  $q_m$  is calculated because again the masked value and associated masks of  $a$  and  $b$  could come together in one of the XOR gates. Interestingly, the construction is even insecure in the stable case. This results from the fact that Equation 5 is only secure if the terms are evaluated from left to right. In hardware design this equation is mapped it to a circuit hierarchy that is beneficial in terms of the delay behavior. In the resulting netlist creates a direct logic path between the masked values and the masks which leads to a flaw that is already visible in the stable behavior of the circuit. 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.

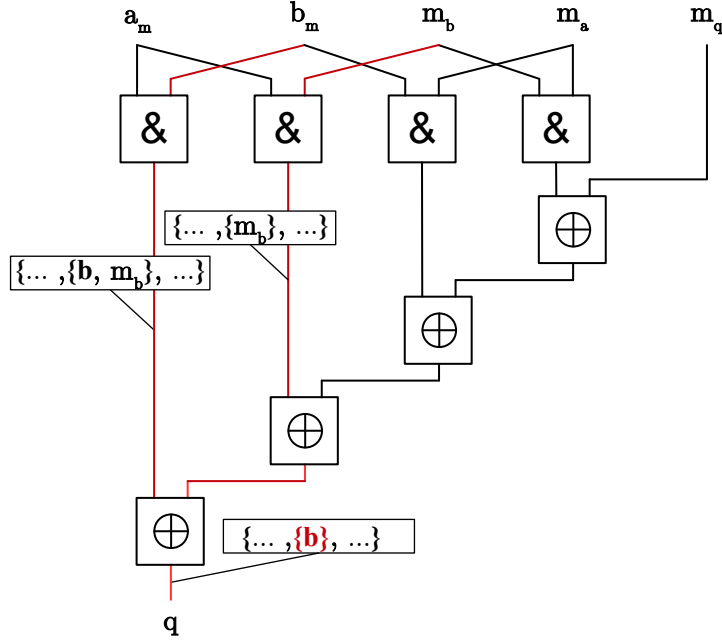


Fig. 12. Trichina gate with insecure signal path example (red)

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. This is done by making the calculation of  $q_{m0}$  independent of  $a_m$  and  $b_m$ , the calculation of  $q_{m1}$  independent of  $m_{a0}$  and  $m_{b0}$ , and the calculation of  $q_{m2}$  independent of  $m_{a1}$  and  $m_{b1}$  in Equation 6 taken from [29].

$$\begin{aligned}
 q_{m0} &= m_{a0} \wedge m_{b0} \oplus m_{a0} \wedge m_{b1} \oplus m_{a1} \wedge m_{b0} \\
 q_{m1} &= m_{a1} \wedge m_{b1} \oplus a_m \wedge m_{b1} \oplus m_{a1} \wedge b_m \\
 q_{m2} &= a_m \wedge b_m \oplus a_m m_{b0} \oplus m_{a0} \wedge b_m
 \end{aligned} \tag{6}$$

A different approach, which requires fewer masks is provided e.g. by the domain-oriented masking multiplier which implements Equation 7. For the security of this multiplier a separation of the terms is required by using a register (indicated by the parentheses) for the combination of the multiplication terms with a fresh random mask  $m_0$ .

$$\begin{aligned}
 q_m &= a_m \wedge b_m \oplus (a_m \wedge m_b \oplus m_0) \\
 m_q &= m_a \wedge m_b \oplus (m_a \wedge b_m \oplus m_0)
 \end{aligned} \tag{7}$$

The verifier also correctly labels the DOM multiplier circuit 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 [21] 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 minutes for the verification without glitches verification and 2 hours 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 but for the eight individual secret bits in parallel. The stable set verification takes less than 30 seconds, and the verification of the transient sets between 5 and 10 hours. 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 [23] and the Keccak S-box [24]. The results are shown in Table 5 where the protection order of the circuit and the verification order are always set equal, and are summarized in the second column.

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 minutes. The influence of the protection order at varying verification orders is depicted in Figure 13. 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.

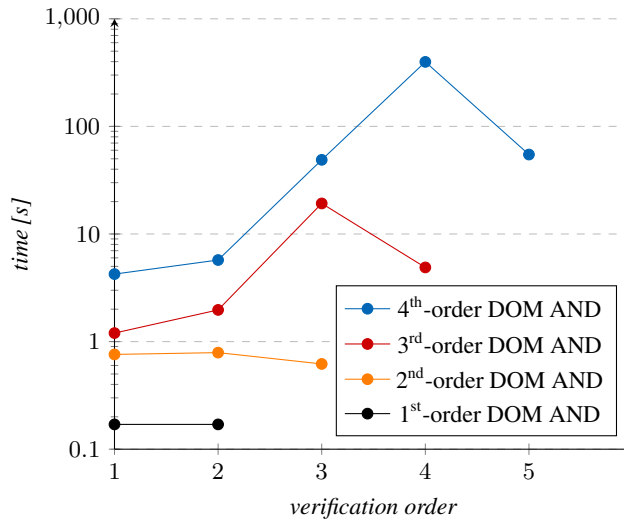
For the Keccak S-box circuit it shows that even without glitches, the third-order verification already takes about 51 minutes. When taking glitches into account the verification time increases to 17.5 hours.

## 10 Conclusions

In this paper we introduced the formal groundwork for the verification of masked hardware implementations in the presence of glitches. Therefore, we built upon the d-probing model of Ishai et al. and presented a method to conservatively estimate the

**Table 5.** 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 [23]	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 [24]	2	75	45	45	5	35	7	≤ 1 m	✓	≤ 2 m	✓
	3	140	80	80	5	60	7	≤ 51 m	✓	≤ 17.5 h	✓



**Fig. 13.** Verification time for the DOM ANDs with varying protection and verification order

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 and multivariate 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 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).



## References

1. G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, and B. Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. *IACR Cryptology ePrint Archive*, 2015:506, 2015.
2. G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, and P. Strub. Verified proofs of higher-order masking. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
3. G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, P. Strub, and R. Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129, 2016.
4. G. Barthe, F. Dupressoir, S. Faust, B. Grégoire, F. Standaert, and P. Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. *IACR Cryptology ePrint Archive*, 2016:912, 2016.
5. G. Barthe, F. Dupressoir, B. Grégoire, A. Stoughton, and P. Strub. EasyCrypt: Computer-Aided Cryptographic Proofs. <https://github.com/EasyCrypt/easycrypt>, 2017.
6. A. G. Bayrak, F. Regazzoni, D. Novo, and P. Ienne. Sleuth: Automated Verification of Software Power Analysis Countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, pages 293–310, 2013.
7. S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud. Randomness Complexity of Private Circuits for Multiplication. In *EUROCRYPT 2016*, 2016.
8. S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud. Private multiplication over finite fields. In *CRYPTO (3)*, volume 10403 of *Lecture Notes in Computer Science*, pages 397–426. Springer, 2017.



9. G. Bertoni and M. Martinoli. A methodology for the characterisation of leakages in combinatorial logic. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, pages 363–382, 2016.
10. S. Bhasin, C. Carlet, and S. Guilley. Theory of masking with codewords in hardware: low-weight  $d$ th-order correlation-immune boolean functions. *IACR Cryptology ePrint Archive*, 2013:303, 2013.
11. B. Bilgin, A. Bogdanov, M. Knezevic, F. Mendel, and Q. Wang. Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, pages 142–158, 2013.
12. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-Order Threshold Implementations. In *ASIACRYPT 2014*, volume 8874 of *LNCS*, 2014.
13. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO 99*, volume 1666 of *LNCS*, 1999.
14. J.-S. Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. *Cryptology ePrint Archive*, Report 2017/879, 2017. <http://eprint.iacr.org/2017/879>.
15. L. de Moura and N. Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
16. H. Eldib and C. Wang. Synthesis of Masking Countermeasures against Side Channel Attacks. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 114–130, 2014.
17. H. Eldib, C. Wang, and P. Schaumont. SMT-Based Verification of Software Countermeasures against Side-Channel Attacks. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, pages 62–77, 2014.
18. H. Eldib, C. Wang, M. M. I. Taha, and P. Schaumont. QMS: Evaluating the Side-Channel Resistance of Masked Software from Source Code. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*, pages 209:1–209:6, 2014.
19. S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT 2010*, volume 6110 of *LNCS*, 2010.
20. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A Testing Methodology for Side-Channel Resistance Validation. In *NIST Non-Invasive Attack Testing Workshop*, 2011.
21. H. Gross. Collection DOM protected hardware implementations. <https://github.com/hgrosz>, 2017.
22. H. Groß and S. Mangard. Reconciling  $d+1$ m asking in hardware and software. *CHES 2017*, 2017, 2017.
23. H. Gross, S. Mangard, and T. Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In H. Handschuh, editor, *Topics in Cryptology – CT-RSA 2017: The Cryptographers’ Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings*, pages 95–112, Cham, 2017. Springer International Publishing.

24. H. Gross, D. Schaffenrath, and S. Mangard. Higher-order side-channel protected implementations of keccak. Cryptology ePrint Archive, Report 2017/395, 2017. <http://eprint.iacr.org/2017/395>.
25. Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO 2003*, volume 2729 of *LNCS*, 2003.
26. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO '99*, London, UK, 1999. Springer-Verlag.
27. S. Mangard and K. Schramm. Pinpointing the side-channel leakage of masked aes hardware implementations. In *CHES 2006*, volume 4249 of *LNCS*, 2006.
28. A. Moss, E. Oswald, D. Page, and M. Tunstall. Compiler Assisted Masking. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 58–75, 2012.
29. S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security*, volume 4307 of *LNCS*, 2006.
30. R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.
31. J.-J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and countermeasures for smart cards. In *Smart Card Programming and Security*, volume 2140 of *LNCS*, 2001.
32. O. Reparaz. Detecting flawed masking schemes with leakage detection tests. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 204–222, 2016.
33. O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, 2015.
34. M. Rivain and E. Prouff. Provably secure higher-order masking of aes. In *CHES 2010*, volume 6225 of *LNCS*, 2010.
35. M. Soos. Cryptominisat 2.5.0. solver description, 2010.
36. E. Trichina. Combinational logic design for AES subbyte transformation on masked data. *IACR Cryptology ePrint Archive*, 2003, 2003.
37. C. Wolf and J. Glaser. Yosys - a free verilog synthesis suite. In *Proceedings of Austrochip 2013*, 2013.
38. G. Xiao and J. L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Trans. Information Theory*, 34(3):569–571, 1988.