



# PROLEAD

## A Probing-Based Hardware Leakage Detection Tool

Nicolai Müller<sup>1</sup>  and Amir Moradi<sup>2</sup> 

<sup>1</sup> Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany  
[firstname.lastname@rub.de](mailto:firstname.lastname@rub.de)

<sup>2</sup> University of Cologne, Institute for Computer Science, Cologne, Germany  
[firstname.lastname@uni-koeln.de](mailto:firstname.lastname@uni-koeln.de)

**Abstract.** Even today, Side-Channel Analysis attacks pose a serious threat to the security of cryptographic implementations fabricated with low-power and nano-scale feature technologies. Fortunately, the masking countermeasures offer reliable protection against such attacks based on simple security assumptions. However, the practical application of masking to a cryptographic algorithm is not trivial, and the designer may overlook possible security flaws, especially when masking a complex circuit. Moreover, abstract models like probing security allow formal verification tools to evaluate masked implementations. However, this is computationally too expensive when dealing with circuits that are not based on composable gadgets. Unfortunately, using composable gadgets comes at some area overhead. As a result, such tools can only evaluate subcircuits, not their compositions, which can become the Achilles' heel of such masked implementations.

In this work, we apply logic simulations to evaluate the security of masked implementations which are not necessarily based on composable gadgets. We developed PROLEAD, an automated tool analyzing the statistical independence of simulated intermediates probed by a robust probing adversary. Compared to the state of the art, our approach (1) does not require any power model as only the state of a gate-level netlist is simulated, (2) can handle masked full cipher implementations, and (3) can detect flaws related to the combined occurrence of glitches and transitions as well as higher-order multivariate leakages. With PROLEAD, we can evaluate masked implementations that are too complex for existing formal verification tools while being in line with the robust probing model. Through PROLEAD, we have detected security flaws in several publicly-available masked implementations, which have been claimed to be robust probing secure.

**Keywords:** Side-Channel Analysis · Leakage Detection · Hardware

## 1 Introduction

Since Kocher et al. reported the first Side-Channel Analysis (SCA) attacks as a threat to the security of cryptographic hardware [Koc96, KJJ99], protecting concrete implementations of cryptographic algorithms has been attracting the researchers' attention. Based on this groundbreaking work, the past twenty years of research have shown successful SCA attacks [KJJ99, BCO04, GBTP08] based on traces obtained by measuring one out of potentially various physical characteristics of a device [Koc96, KJJ99, GMO01, HS13, GST14]. In particular, if a designer does not consider SCA as a serious attack vector, an adversary may exploit the dependency between a physical characteristic of the device and the processed data revealing sensitive information efficiently.

As there has been an urgent need to mitigate information leakage, numerous countermeasures trying to protect cryptographic devices from any SCA have been proposed during

the past decades. So far, *masking*, based on the concept of secret sharing [Sha79], is the most popular proposal [CJRR99]. Masking reaches its popularity due to its basic security assumptions simplifying the design and verification of concrete masking schemes such as the ones given in [ISW03, Tri03, NRS11, RBN<sup>+</sup>15, GMK16, GMK17, GIB18, GM18]. If the masking scheme satisfies that all input sharings are drawn uniformly and if the noise level is sufficient, conceptual simple adversary models abstract the SCA security of a masking scheme [DDF14]. A basic and efficient – and therefore widely used – adversary model is the  $d$ -probing model [ISW03]. Due to its simplicity, the verification under the  $d$ -probing model is efficient but does not cover physical shortcomings. Consequently, the more advanced robust probing model has been built on top of the  $d$ -probing model to cover different types of physical defaults [FGMDP<sup>+</sup>18] including glitches, transitions, and couplings.

However, designing, implementing, and verifying a masked implementation of a cryptographic algorithm is often a manual and error-prone task. Consequently, some masking schemes are shown to be insecure because of, so far, unnoticed design flaws or inaccurate formal modeling of the adversary [MMSS19].

A common approach to verify the experimental security of a masked implementation is to collect power/electro-magnetic traces from a prototype and perform a leakage assessment based on statistical hypothesis testing, i.e.  $t$ -test [GJJR11] or  $\chi^2$ -test [MRSS18]. Leakage assessment takes place after fabricating a prototype of the device. If the leakage assessment reports a leakage, the design needs to be analyzed to avoid the flaw leading to the fabrication of a new device under test. This procedure is repeated until no further leakage is detected.

The fabrication can be a new design configured on an FPGA with a significantly short time to market. However, in the case of ASIC design procedures, refabrication is time-consuming and expensive; hence leakage evaluation techniques at early design stages, i.e., before fabrication of the chip, are getting more and more popular. In particular, formal verification and leakage simulation have become promising research fields. Nevertheless, their concrete instantiations, given as automated verification frameworks, come with restrictions. Formal verification tools can efficiently verify the robust probing security of a tiny circuit, e.g., a small S-Box, but the complete formal verification of a larger design, like a masked round-based implementation of a cipher, becomes computationally infeasible. While leakage simulation tools can handle larger circuits, they mainly check resistance against a particular Differential Power Analysis (DPA) or Correlation Power Analysis (CPA) attack, where assumptions about the leakage and power models as well as targeted intermediate values are required [HSZ13, FGBR20, NPH<sup>+</sup>20, ZPTF21]. More formally, they cannot evaluate the given circuits based on the robust probing model. As a shortcoming, they may report an implementation as secure, while an undetected design flaw (caused by not being probing secure) is exploitable by another attack not covered by the evaluations.

To build provably-secure masked implementations of large designs, small and securely composable masked circuits, so-called gadgets, are applied as building blocks of larger circuits. Therefore, the security properties of such gadgets can be individually verified by formal verification tools, e.g., [BBC<sup>+</sup>19, KSM20]. However, formal verification of a masked full cipher is still not feasible if the implementation is made by not-necessarily composable gadgets. These implementations are potentially beneficial and usually more efficient compared to their gadget-based variants. For example, [KMMS22] provides a comparison of different masked byte-serial implementations of the Advanced Encryption Standard (AES). However, their probing security remains unproven. We must rely on experimental leakage assessments that might be inaccurate, incomplete, setup-dependent, and therefore not always trustworthy. In short, we can evaluate the leakage of large designs in an experimental setting, but we cannot see this as a security proof. Even if we detect no weaknesses by a leakage assessment, we can never be sure that no security flaw exists.

## 1.1 PROLEAD

We introduce PROLEAD, a novel leakage detection tool for hardware. PROLEAD performs logic simulations at gate level to evaluate the  $d$ -probing security of a circuit even in presence of glitches and transitions. To clarify the benefits and limitations of PROLEAD compared to the existing tools from literature, we give a detailed comparison.

**Compare with Formal Verification Tools.** All formal verification tools presented in Section 3 are restricted to the verification of small building blocks, e.g., gadgets or small S-Boxes. Hence, it is infeasible to verify large circuits, e.g., a masked implementation of a complete cipher, with formal verification tools. Due to its underlying simulation-based approach, PROLEAD can evaluate large circuits even at higher security orders. For example, a masked first-order round-based implementation of AES can be fully examined in an hour, while a flawed variant is identified in a couple of seconds. In Section 5, we evaluate some examples with PROLEAD that cannot be verified by the tools from Section 3. As a result, by means of PROLEAD we have identified a high number of security flaws in the implementations which are supposed to be not only secure in practice, but also robust probing secure. However, the evaluation of PROLEAD can be seen as incomplete, since the simulations cannot cover all possible inputs of large circuits. Therefore, it is possible that PROLEAD fails to detect existing leakage if a leaking test vector is not considered sufficiently by the simulator. Hence, increasing the number of simulations decreases the probability that existing leakage is not detected. To create confidence in the decision of PROLEAD, we provide a mechanism to determine an adequate number of simulations to consider. More specifically, we compute the minimum number of simulations needed to guarantee an acceptable small probability that existing leakage is not detected.

**Compare with Leakage Simulators.** The leakage simulators and evaluation techniques covered in Section 3 can give a preliminary overview of the vulnerability of the given design, often unprotected, i.e., not masked. However, they are not in conformity with the probing security model. As a result, they probably fail to detect leakage, particularly for masked implementations. In other words, they cannot guarantee the detection of leakages in flawed masked implementations. However, PROLEAD has two important differences compared to state-of-the-art leakage simulators.

1. PROLEAD operates directly on the simulated intermediates instead of modeling power traces. Consequently, the security evaluation does not depend on a hypothetical power model.
2. The evaluation procedure of PROLEAD is fully in line with the concept of robust probing security [FGMDP<sup>+</sup>18]. We investigate all necessary robust-probing adversaries that give PROLEAD the ability to detect any leakage according to the security model.

In short, PROLEAD fills the gap, i.e., uses the benefits of simulation-based tools (fast evaluations) and examines the security of large circuits based on robust probing models, which neither state-of-the-art simulation-based nor formal verification tools can handle. The tool is publicly available at [GitHub](https://github.com/ChairImpSec/PROLEAD)<sup>1</sup>.

## 1.2 Outline

Section 2 starts by introducing the basic concept of probing security and continues with more advanced security notions. Afterwards, Section 2 gives the relevant background on masking and the statistical hypothesis test. Next, we present the existing tools from the

<sup>1</sup><https://github.com/ChairImpSec/PROLEAD>

formal verification and leakage simulation domain in Section 3. In Section 4, we explain the methodology of PROLEAD with a focus on generating the probing sets and evaluation. Finally, we apply PROLEAD to a wide range of protected hardware implementations and discuss its findings in Section 5. Section 6 concludes this paper.

## 2 Background

### 2.1 Robust Probing Model

The robust  $d$ -probing model [FGMDP<sup>+</sup>18] extends the  $d$ -probing model [ISW03] by considering physical defaults violating security in non-ideal circuits, i.e., physical logic circuits and hardware implementations [BDF<sup>+</sup>17]. More precisely, under robust  $d$ -probing model information leakage sourced by combinational recombinations (glitches) [MPG05, MPO05], memory recombinations (transitions) [CGP<sup>+</sup>12, BGG<sup>+</sup>14], and routing recombinations (couplings) [CBG<sup>+</sup>17] are taken into account. We apply the notation from [FGMDP<sup>+</sup>18], annotating the robust probing model with a triple  $(g, t, c)$ . Each entry in the triple specifies whether glitches ( $g = 1$ ), transitions ( $t = 1$ ), or couplings ( $c = 1$ ) are considered. For example, the  $(0, 0, 0)$ -robust probing model, i.e., without considering any physical defaults, is equivalent to the  $d$ -probing model introduced in [ISW03] building the theoretical foundation for verifying the security of an ideal circuit. It defines a  $d$ -probing adversary who can place up to  $d$  (standard) probes on freely chosen spots to record noise-free and stable signals. Hence, the adversary gets access to a set of  $d$  intermediates. To cover a desired physical default in the  $(g, t, c)$ -robust  $d$ -probing model, a probe extension procedure is applied on the standard probes which further increases the set of intermediates. We remark that taking all possible occurrences of a specified type of physical default into account is very conservative but leads to a model that fits arbitrary circuits.

**Glitch-Extension.** Glitches are unexpected signal transitions occurring in combinational circuits. Due to imbalanced delay paths and switching delays, signals may arrive asynchronously at a gate resulting in a different but temporary output before the output signal reaches its intended state. To cover glitches within the robust probing model, glitch-extended probes replace all probes allowing an adversary to access all stable intermediates (either register outputs or primary inputs) that contribute to the probed wire.

**Transition-Extension.** Transitions potentially recombine the contents of the same memory element in two consecutive invocations. Hence, by overwriting a memory element, i.e., a register or a flip-flop, an adversary gains information about the old and the new values. To model transitions in the robust probing model, all probes are replaced by transition-extended probes recording the signal during two consecutive invocations.

**Coupling-Extension.** Couplings lead to unintentional and undesired recombinations of values on adjacent wires. To model couplings in the robust probing model, we need to replace all probes with coupling-extended probes that observe multiple neighboring wires.

### 2.2 Security Notions

According to the probing model, a circuit achieves  $(g, t, c)$ -robust  $d$ -probing security if a  $d$ -probing adversary cannot learn anything about the processed secret. More formally, the joint distribution of any observation set is statistically independent of the distribution of any secret. Unfortunately, for the verification of  $d$ -probing security, the statistical independence of each observation set must be checked. As the number of probe combinations grows with  $d$  and the complexity of the circuit, verification of  $d$ -probing security becomes

infeasible for large circuits and particularly at higher orders. To bypass the verification of  $d$ -probing security, small and secure building blocks, so-called gadgets, are composed to construct more complex circuits. Hence, the  $d$ -probing security of a large circuit is satisfied if all involved gadgets are composable without violating  $d$ -probing security. To formally verify the composability of a gadget, Barthe et al. introduced  $d$ -Strong Non-Interference (SNI) [BBD<sup>+</sup>16] as an extension of  $d$ -Non-Interference (NI) [BBD<sup>+</sup>15]. While gadgets satisfying  $d$ -SNI are indeed composable, it turned out that  $d$ -SNI is an over-conservative security notion in practice and leads to significant area and randomness overheads [CS20]. To relax the requirements in terms of area and randomness, Cassiers et al. introduced  $d$ -Probe-Isolating Non-Interference (PINI) as a less conservative and, therefore, more efficient security notion [CS20].

### 2.3 Boolean Masking

Boolean masking is a common and well-studied approach to protect hardware implementations against SCA attacks [CJRR99]. According to the concept of secret sharing [Sha79], Boolean masking splits a sensitive variable  $X \in \mathbb{F}_2^n$  into  $s > 1$  independently and uniformly distributed shares  $\{X^0, \dots, X^{s-1}\} \in (\mathbb{F}_2^n)^s$  that satisfy  $X = \bigoplus_{i=0}^{s-1} X^i$ . To generate a sharing of  $X$ , we sample  $\{X^0, \dots, X^{s-2}\}$  uniformly and randomly from  $\mathbb{F}_2^n$ . For the remaining  $X^{s-1}$ , it holds that  $X^{s-1} = \left(\bigoplus_{i=0}^{s-2} X^i\right) \oplus X$ . To avoid SCA leakages about  $X$ , all operations of the cipher need to be performed on  $\{X^0, \dots, X^{s-1}\}$ , i.e. the shared representation of  $X$ . To achieve security under the  $d$ -probing model (cf. Section 2.1) it must hold that  $s \geq d + 1$ .

### 2.4 Statistical Hypothesis Tests

Hypothesis tests apply statistical procedures to data to assess the strength of evidence against the underlying *null hypothesis*  $H_0$ . Usually,  $H_0$  denotes a general statement that there is no relation between two groups of samples. To accept or reject  $H_0$ , hypothesis tests provide a quantitative value in the form of a significance level.

For our purposes, we evaluate whether there is a significant dependency between two categorical variables  $R$  with  $r$  different categories and  $C$  with  $c$  different categories, both from a single population. Hence, the corresponding  $H_0$  states that  $R$  and  $C$  are statistically independent. The frequency of observed samples of  $R$  and  $C$  are stored in a two-way contingency table with  $r$  and  $c$  categories. For better understanding, we depict a generic two-way contingency table in Table 1.

**Table 1:** Contingency table for two variables  $R$  and  $C$ , with  $F_{i,j}$  denoting the frequency of observed samples when  $R = i$  and  $C = j$ .

|              |             |     |               |              |
|--------------|-------------|-----|---------------|--------------|
| $F_{i,j}$    | $j = 0$     | ... | $j = c - 1$   | <b>Total</b> |
| $i = 0$      | $F_{0,0}$   | ... | $F_{0,c-1}$   | $F_{0,*}$    |
| ...          | ...         | ... | ...           | ...          |
| $i = r - 1$  | $F_{r-1,0}$ | ... | $F_{r-1,c-1}$ | $F_{r-1,*}$  |
| <b>Total</b> | $F_{*,0}$   | ... | $F_{*,c-1}$   | $F_{*,*}$    |

**Pearson’s  $\chi^2$ -Test of Independence.** The  $\chi^2$ -test of independence [Pea92] rejects or accepts  $H_0$  based on a test statistic that follows a  $\chi^2$ -distribution. The test itself measures the divergence of the observation from the expectation. Such expectations are done by estimating the degree of freedom  $\nu$  and the corresponding expected frequency  $E_{i,j}$  for

each observed frequency  $F_{i,j}$ , given that  $H_0$  holds. Afterwards, the  $\chi^2$ -test statistic  $x$  is computed as follows.

$$v = (r-1)(c-1), \quad E_{i,j} = \frac{F_{i,*} \cdot F_{*,j}}{F_{*,*}}, \quad x = \sum_{i=0}^{r-1} \sum_{j=0}^{c-1} \frac{(F_{i,j} - E_{i,j})^2}{E_{i,j}} \quad (1)$$

The associated  $p$ -value, i.e., the probability to accept  $H_0$ , is finally computed based on the probability density function  $f$  and the gamma function  $\Gamma$ .

$$p = \int_x^\infty f(x, v) dx, \quad f(x, v) = \begin{cases} \frac{x^{\frac{v}{2}-1} e^{-\frac{x}{2}}}{2^{\frac{v}{2}} \Gamma(\frac{v}{2})} & x > 0 \\ 0 & \text{else} \end{cases} \quad (2)$$

**$G$ -Test of Independence.** The  $G$ -test of independence [Sok95, MoD09] is an alternative to the  $\chi^2$ -test of independence based on a likelihood ratio test. The method uses the multinomial distribution. It tests the goodness of fit of the observed to the expected frequencies, under the assumption that  $H_0$  holds, by estimating the  $G$ -statistic  $x$  as follows.

$$x = 2 \cdot \sum_{i=0}^{r-1} \sum_{j=0}^{c-1} F_{i,j} \cdot \ln \left( \frac{F_{i,j}}{E_{i,j}} \right), \quad (3)$$

where the expected frequencies  $E_{i,j}$  are computed as for the  $\chi^2$ -test, i.e., Equation (1). Since the distribution of  $x$  is approximately  $\chi^2$ -distributed with the same  $v$ , the  $p$ -value is computed in the same way as for the  $\chi^2$ -test, i.e., Equation (2).

#### 2.4.1 Hypothesis Testing for Leakage Detection

Due to its simplicity and efficiency, the  $\chi^2$ -test is more common than the  $G$ -test. If the contingency table is small, i.e.  $2 \times 2$ , the  $\chi^2$ -test can be easily calculated by hand. Moreover, the  $\chi^2$ -test computes a squaring while the  $G$ -test conducts the more complex logarithm function. In the field of leakage detection, the  $\chi^2$ -test is a complement to Welch's  $t$ -test as it detects some flaws which the  $t$ -test cannot detect. For a detailed comparison of the  $\chi^2$ -test and  $t$ -test, we refer to [MRSS18]. However, as the  $\chi^2$ -test is an approximation of the  $G$ -test based on Taylor expansion, it is not as accurate as of the  $G$ -test. If  $F_{i,j}$  and  $E_{i,j}$  are different, the  $\chi^2$ -test approximation overestimates the outlier what may lead to erroneous results while the  $G$ -test computes correctly [Sok95, Hoe12]. We remark that these inaccuracies occur if we deal with a sparse contingency table, i.e., the amount of data is low while the contingency table is large. As in the domain of SCA, contingency tables might often be sparse, we choose the  $G$ -test as the underlying hypothesis test of our tool PROLEAD.

For the sake of completeness, we would like to note that – in contrast to the  $G$ -test – exact hypothesis tests such as Fischer's exact test [Fis22, Agr92] do not approximate the  $p$ -value. Hence, they provide an accurate significance level. Exact tests are of great relevance when dealing with small data sets as the approximation error when applying the  $\chi^2$ -test increases the more expected frequencies are smaller than 5 [Yat34].

On the other hand, evaluating larger contingency tables with an exact hypothesis test is not feasible. Usually, Fischer's exact test is applied on up to  $2 \times 4$  contingency tables or combined with a probabilistic heuristic such as Monte Carlo sampling [M<sup>+</sup>87].

#### 2.4.2 Statistical Power Analysis

Whenever we apply statistical hypothesis tests, we ensure that we can trust their results. In particular, our experiment must satisfy that the underlying hypothesis test detects even a



small dependency. The question of whether a hypothesis test reliably detects a dependency is closely related to the metric of *statistical power*. To understand the statistical power, we first define the *estimation errors* that can occur while performing hypothesis tests in the context of SCA.

**Definition 1** (False Positive). *False positives* occur if we reject  $H_0$  despite it being true. It means that PROLEAD wrongly classifies a secure design as insecure.

**Definition 2** (False Negative). *False negatives* occur if we accept  $H_0$  despite it being false. It means PROLEAD cannot detect existing leakage and wrongly classifies an insecure design as secure.

While the  $p$ -value gives the probability of a false positive, the statistical power is related to the false-negative probability  $\beta$ . As the power of a test defines the probability that it rejects  $H_0$  correctly, it is computed as:

$$1 - \beta = F(x_{\text{crit}}, v, \lambda) = e^{-\frac{\lambda}{2}} \sum_{i=0}^{\infty} \frac{(\frac{\lambda}{2})^i}{i!} Q(x_{\text{crit}}, v + 2i) \quad Q(x, v) = \frac{\gamma(\frac{v}{2}, \frac{x}{2})}{\Gamma(\frac{v}{2})}$$

While  $F$  denotes the cumulative distribution function for the noncentral  $\chi^2$ -distribution,  $Q$  denotes the cumulative distribution function of the central  $\chi^2$ -distribution, and  $\gamma$  denotes the lower incomplete gamma function. Moreover,  $x_{\text{crit}}$  defines the critical value of the underlying distribution for a given confidence level  $a$ . Hence,  $x_{\text{crit}}$  estimates the  $a$ -quantile of the  $\chi^2$ -distribution. Last,  $\lambda$  is the noncentrality parameter depending on  $\varphi$  and  $F_{*,*}$  (given in Table 1) and is computed as:

$$\lambda = \varphi^2 \cdot F_{*,*}$$

As we want to detect even small effects, we choose  $\varphi = 0.1$  following the proposal of Cohen [Coh88] which estimates  $\varphi = 0.1$  as a small effect,  $\varphi = 0.3$  as medium effect, and  $\varphi = 0.5$  as huge effect. Consequently, we numerically estimate  $F_{*,*}$  for fixed  $\varphi = 0.1$  and  $\beta \leq 10^{-5}$ .

### 3 Related Works

The evaluation of masked implementations has a pivotal role in protecting devices against SCA attacks. Therefore, many researchers contribute to this topic by presenting automated tools revealing flaws in protected implementations. So far, the research concentrates on two different approaches, namely formal verification, and leakage simulation.

**Formal Verification.** Automated tools for formal verification can prove the security of masked implementations, assisted by predefined adversary models. As formal verification is complete, no false negatives occur and the result can be seen as security proof under the chosen model but, sometimes, the verification is too conservative leading to false positives [PMK<sup>+</sup>11, KSM20]. The reduction of masking’s security properties to simple and abstract adversary models allows the evaluation of complex circuits, i.e., masked implementations operating on many shares. Nevertheless, the verification under more sophisticated adversary models, i.e., models that cover physical defaults, is too computational complex for larger circuits. That is why formal verification tools are mostly applied only to gadgets. Today, a wide range of formal verification tools, evaluating robust probing security, is available [BBD<sup>+</sup>15, BBD<sup>+</sup>16, BBC<sup>+</sup>19, Cor18, BGI<sup>+</sup>18, KSM20, MKSM22, BK21, BMRT22] and we refer to [CGF21] for an exhaustive survey on formal verification tools. Below, we summarize the most relevant candidates related to this work including their capabilities. At EUROCRYPT 2015, Barthe et al. [BBD<sup>+</sup>15] presented `maskVerif`, a language-based

formal verification framework based on probabilistic information flow to prove the  $(0, 0, 0)$ - and  $(0, 1, 0)$ -robust  $d$ -probing security of masked implementations. Moreover, they introduced the  $d$ -NI security notion and integrated an automated verification procedure into `maskVerif`. Later, the authors of [BBD<sup>+</sup>16] extended  $d$ -NI to the  $d$ -SNI security notion to evaluate circuits with respect to their composability, and integrated the formal verification of  $d$ -SNI into `maskVerif`. The introduction of the SNI notion improves formal verification since small parts of the circuit (so-called gadgets) that fulfill  $d$ -SNI can be arbitrarily composed to construct larger and still secure designs. Hence, `maskVerif` can efficiently verify small gadgets to avoid the formal verification of larger gadget-based circuits. As these versions of `maskVerif` left glitches unconsidered, Bloem et al. developed `Rebecca` [BGI<sup>+</sup>18], an automated tool based on the approximated estimation of Fourier coefficients. As the Fourier coefficients are not fully computed for all underlying functions of the circuit but only approximated, the precision of `Rebecca` might be slightly low, resulting in a small number of false positives. In contrast to `maskVerif`, `Rebecca` can verify  $(0, 0, 0)$ -,  $(1, 0, 0)$ -, and  $(0, 1, 0)$ -robust  $d$ -probing security but no composability notions as  $d$ -NI and  $d$ -SNI are supported. Nevertheless, the computational cost for estimating Fourier coefficients is quite high which restricts `Rebecca` to the verification of small circuits and low security orders. Moreover, Barthe et al. extended `maskVerif` to evaluate higher-order leakages in the presence of glitches [BBC<sup>+</sup>19]. The extension is based on a simple but expressive intermediate language that annotates each instruction with a leakage expression. Since the to-date version of `maskVerif` can verify large sets of intermediates very efficiently, the performance is improved compared to former versions [BGI<sup>+</sup>18]. However, `maskVerif` follows a language-based approach resulting in verification purely based on the syntax instead of statistical evaluation. Therefore, a false positive can occur if a statistical independent output of a masked function violates non-completeness and its sharing is not refreshed by fresh randomness. Two examples for such a false classification are the Threshold Implementation (TI) of PRESENT S-Box given in [PMK<sup>+</sup>11] which `maskVerif` wrongly classifies as first-order insecure, and a small toy example presented in [KSM20]. In short, neither `Rebecca` nor `maskVerif` can fully avoid false positives. To counteract this problem, Knichel et al. presented `SILVER` [KSM20], which completely avoids false positives by analyzing the statistical independence of joint distributions. Moreover, `SILVER` supports all previously discussed security notions ( $d$ -probing security,  $d$ -NI, and  $d$ -SNI) as well as the  $d$ -PINI notion [CS20], which allows composability as well as a more efficient implementation compared to  $d$ -SNI. All security notions can be verified under the  $(0, 0, 0)$ - and  $(1, 0, 0)$ -robust  $d$ -probing model, i.e., covering glitches. Concerning efficiency, `SILVER` achieves fast verification of gadgets at higher orders as well as small S-Boxes on low orders. On the negative side, `SILVER` is not able to analyze even a middle-size masked circuit. By a recent extension [MKSM22], `SILVER` is also able to evaluate a certain form of circuits under the  $(1, 1, 0)$ -robust  $d$ -probing model. Hence, `SILVER` was the first formal verification tool that covers glitches and transitions simultaneously. While the current versions of `maskVerif` and `SILVER` can verify a design under various conditions, the overhead in terms of verification time, especially for verification with `SILVER` is impractical for large circuits.

**Leakage Simulation.** Since the formal verification of larger circuits becomes infeasible, another research branch focuses on evaluating these circuits by simulating the power consumption of a particular prototype running the implementation. For efficiency reasons, only a fixed set of input vectors is simulated resulting in an incomplete evaluation. Hence, the high performance of leakage simulation comes at the cost of accuracy and false negatives. We remark that [BBYS22] presents a wide range of leakage simulators. Therefore, we focus on how simulators abstract the leakage. The accuracy of a simulator mainly depends on the abstraction level of the simulation. Simulations at the Register-Transfer Level (RTL) are helpful to verify the security of a hardware implementation during the earliest design



stage. Since a high-level description is the only available source, the simulation cannot take any hardware-specific internals into account. Usually, the simulation of a trace on RTL is done by simulating the internal logic given in the high-level description and applying a leakage function like Hamming weight (HW) [AMM<sup>+</sup>06, Rep16, FGBR20] or Hamming distance (HD) [SBY<sup>+</sup>18] on the logically simulated intermediates. Then, conventional leakage assessment techniques as Test Vector Leakage Assessment (TVLA) [BCD<sup>+</sup>13] operate on the simulated traces. To increase accuracy, Debande et al. proposed a profiled logic simulator based on linear regression [DBBL12]. During a profiling phase, the simulator receives real traces from a profiling device and two successive states of all registers to fit a function of state bits and transitions [SLP05]. According to the resulting model, the simulator can generate new traces that are more accurate compared to non-profiled high-level simulators. Furthermore, some tools are built to identify possible hypothesis functions for an exploitation with DPA and CPA [HSZ13, HPN<sup>+</sup>19, NPH<sup>+</sup>20, ZPTF21]. If a gate-level netlist is available, simulations become more detailed and leakage can be detected for each cell separately. Nevertheless, taking every single gate into account increases the simulation time significantly. Most of the gate level-simulators consider simulated toggles which are stored in a Value Change Dump (VCD) file [KP07, SBY<sup>+</sup>18, SVRK19, YKES20]. Based on the toggles, they simulate the traces which are again processed by TVLA. Sometimes the user can annotate the netlist. For example, the simulator of Kirschbaum et al. [KP07] can simulate glitches if the user adds propagation delays to the netlist. Furthermore, *Karna* splits the placed netlist into  $N \times N$  regions and identifies the most leaking ones with localized TVLA if the user adds placement information [SVRK19]. To allow a fair comparison of gate-level leakage detection on a prototype, Kiaei et al. developed the *Saidoyoki* evaluation board [KLE<sup>+</sup>21]. The board includes two self-designed ASICs. The simulator receives a gate-level netlist, layout parasitics, and a test scenario (DPA, CPA, or TVLA) together with a fitting set of test vectors. Then, the procedure applies commercial tools for synthesis, hardware simulation, and gate-level leakage estimation. Simulators operating at a low abstraction level, e.g. transistor level, such as Simulation Program with Integrated Circuit Emphasis (SPICE) and its variants, can simulate the power consumption very accurately as they take many hardware-specific properties into account. Therefore, transistor-level simulations achieve the highest accuracy and are widely used to model side-channel leakage [TV04, AMM<sup>+</sup>06, RBE<sup>+</sup>07, RCS<sup>+</sup>09, SBY<sup>+</sup>18]. Nevertheless, covering all design internals comes at the cost of performance [TV03]. Hence, transistor-level simulations become infeasible for large circuits [BDG<sup>+</sup>14].

## 4 Technique

The procedure PROLEAD follows to verify  $d$ -probing security consists of two steps. Initially, PROLEAD generates all probing sets that must be considered for the desired leakage verification. Later, a verification step goes through all relevant probing sets and tests their information leakage. During verification, a simulator generates the inputs for the circuit (based on the given settings) and simulates the circuit to obtain intermediate values. Afterwards, a statistical hypothesis test evaluates the independence of the intermediate's underlying distributions. In this section, we present both steps in detail. We start by reviewing the leakage models PROLEAD supports and give some information about the specification of the settings.

### 4.1 Notation

We denote functions by sans-serif lower-case characters, e.g.  $f(\cdot)$ . We use an upper-case and bold character, like  $\mathbf{X}$ , to denote a list of elements. Further, we use subscripts to refer

to a specific element of a list. For example,  $x_i \in \mathbf{X}$  denotes the element with index  $i$  in the list.

## 4.2 Formal Models

### 4.2.1 Circuit Model

It holds that every sequential circuit that contains no combinational loop (i.e., not covering circuits with an asynchronous design architecture) can be uniquely modeled as a Mealy machine [Mea55] following the schematic of Figure 1. The circuit model combines combinational logic with a single register stage containing all synchronization elements. Hence, even if a circuit encompasses multiple register stages, it is modeled as it is depicted in Figure 1. The combinational logic processes the primary inputs  $\mathbf{I}$  and the register state  $\mathbf{S}$ , and returns all register inputs and primary outputs  $\mathbf{O}$ .

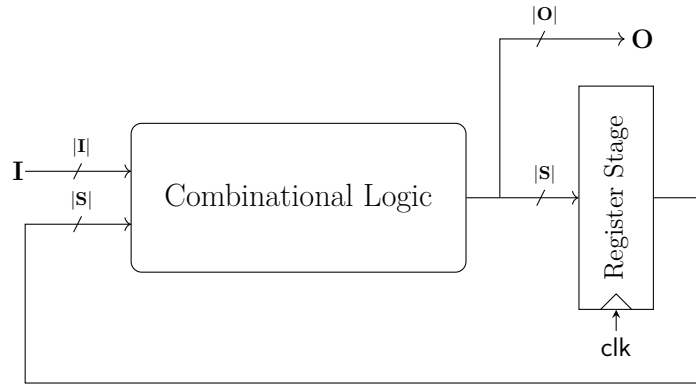


Figure 1: General model of a sequential circuit.

**Definition 3** (Mealy Machine). We represent a *Mealy Machine* as a directed graph  $(\mathbf{V}, \mathbf{E})$ . Each vertex  $v \in \mathbf{V}$  models a combinational gate or a register cell while we model each connection between the gates, i.e., wires, as an edge  $e \in \mathbf{E}$ . Each  $v \in \mathbf{V}$  is defined as a quadruple  $v = (\mathbf{E}_v^{\text{in}}, \mathbf{E}_v^{\text{out}}, f_v, g_v)$  with the following elements:

- The input wires  $\mathbf{E}_v^{\text{in}} \subseteq \mathbf{E}$  (and output wires  $\mathbf{E}_v^{\text{out}} \subseteq \mathbf{E}$ ) of  $v$ .
- The underlying Boolean function  $f_v$  computed by  $v$ .
- $g_v \in \mathbb{F}_2$  with  $g_v = \text{true}$  if  $v$  is combinational, and  $g_v = \text{false}$  if  $v$  is a sequential gate.

We remark that  $e \in \mathbf{E}$  only specifies the signal itself (i.e. the output value of a gate) but not the connection between multiple gates. Hence, a connection from  $v_0$  to  $v_1$  with  $v_0, v_1 \in \mathbf{V}$  exists if  $e \in \mathbf{E}_{v_0}^{\text{out}}$  and  $e \in \mathbf{E}_{v_1}^{\text{in}}$ . Further,  $e \in \mathbf{E}$  can be the input of multiple gates as the same signal can be the input of multiple gates. Additionally, we create a list  $\mathbf{V}_e \subseteq \mathbf{V}$  for each  $e \in \mathbf{E}$ .  $\mathbf{V}_e$  stores all  $v \in \mathbf{V}$  with  $e \in \mathbf{E}_v^{\text{in}}$ .

### 4.2.2 Leakage Models

To verify  $d$ -probing security, PROLEAD can be dictated to consider a specification of the  $(g, t, c)$ -robust  $d$ -probing model [FGMDP<sup>+</sup>18].

**Definition 4** (Probe). Let  $(\mathbf{V}, \mathbf{E})$  be the representation of a sequential circuit and  $\mathbf{T}$  be a list of considered time instances, i.e. clock cycles. A *probe*  $p = (e, t)$  with  $e \in \mathbf{E}$  and  $t \in \mathbf{T}$  records a signal on wire  $e$  during clock cycle  $t$ .

**Definition 5** (Probing Set). A *Probing Set*  $\mathbf{P} = \{p_0, \dots, p_{|\mathbf{P}|-1}\}$  defines a list of probes. We denote the extensions applied to a probing set  $\mathbf{P}$  with superscripts. For example, if  $\mathbf{P}$  contains standard probes, we refer to the same list when extended by glitches by  $\mathbf{P}^g$ .

By default, we perform the verification step under the  $(1, 0, 0)$ -robust  $d$ -probing model. This so-called glitch-extended  $d$ -probing model is a widely-used adversary model for hardware implementations. It considers physical defaults in terms of glitches. To formalize the behavior of glitches, i.e., their propagation through combinational logic, we utilize the conservative glitch-extension procedure from [FGMDP<sup>+</sup>18] given in Definition 6.

**Definition 6** ( $(1, 0, 0)$ -Robust  $d$ -Probing Model). Consider a list of  $d$ -probing sets  $\mathbf{P}$ . We transform all  $d$ -probing sets  $\mathbf{P}_i = \{p_0, \dots, p_{d-1}\} \in \mathbf{P}$  into glitch-extended probing sets  $\mathbf{P}_i^g \in \mathbf{P}^g$  by substituting each probe  $p \in \mathbf{P}_i$  individually. We substitute  $p = (e, t)$  by all probes placed on the input wires of combinational gates that contribute to  $e$  and record during clock cycle  $t$ .

Then, the verification step goes through all  $\mathbf{P}_i^g \in \mathbf{P}^g$  and analyzes each glitch-extended probing set. We give the details of the probe extension procedure in Section 4.4. Further, we can expand the verification by considering the joint occurrence of glitches and transitions under the  $(1, 1, 0)$ -robust  $d$ -probing model. Usually, hardware implementations are analyzed concerning glitches only. However, transitions can lead to security flaws, especially in iterative circuits, e.g., round-based cipher designs [HSS12, CS21]. Formalizing the influence of transitions, i.e., value changes, whose leakage depends on the previous as well as the new value, is done by extending all  $p \in \mathbf{P}^g$  by transitions according to Definition 7.

**Definition 7** ( $(1, 1, 0)$ -Robust  $d$ -Probing Model). Consider a list of glitch-extended  $d$ -probing sets  $\mathbf{P}^g$ . We transform all  $\mathbf{P}_i^g = \{p_0, \dots, p_{|\mathbf{P}_i^g|-1}\} \in \mathbf{P}^g$  into  $\mathbf{P}_i^{g,t} \in \mathbf{P}^{g,t}$  by substituting all  $p = (e, t) \in \mathbf{P}_i^g$  with a tuple  $\{p, p'\}$  recording the previous clock cycle  $p' = (e, t - 1)$ .

We should highlight that PROLEAD always considers the glitches and does not support non-robust  $(0, 0, 0)$  case. As PROLEAD is supposed to evaluate hardware circuits, we cannot ignore glitches. For a correct evaluation, the user should also consider transitions. However, we made the cover of transitions an optional feature to allow the designer to identify the source of leakage. More precisely, if PROLEAD reports the detection of leakage for glitch + transitions, the designer can turn off the transitional effect and re-evaluate the circuit to find whether the detected leakage is due to the glitches or not. Further, PROLEAD does not cover coupling as information about the layout becomes required, i.e., placement of cells and routing of signals. Since PROLEAD works with the gate-level netlist as the result of a synthesis process, such information is not available. Moreover, the user can restrict the verification of higher-order probing security to univariate leakages or cover multivariate leakages as well.

**Univariate Leakage.** Univariate attacks exploit leakage during a single point in time. Hence, we formalize the verification of univariate  $d$ -order probing security by an attacker who places up to  $d$  probes at the same clock cycle. Therefore, an attacker can spot up to  $d$  intermediates during a single but arbitrary clock cycle. We model the attacker's capabilities by verifying all relevant probing sets whose probes record during the same clock cycle. More formally, it holds that all probes  $p \in \mathbf{P}_i$  record during the same clock cycle.

**Multivariate Leakage.** In contrast to the univariate setting, a  $v$ -variate attack combines information of  $v$  different points in time [MM12]. Consequently, a multivariate attacker can place each probe at an arbitrary clock cycle. To formalize the adversary's behavior,

we make no restrictions on the probing sets concerning the clock cycles. Hence, a probing set can contain probes at any relevant gate and record at any clock cycle. Note that by ‘any relevant gate’ we refer to the list  $\mathbf{E}^*$  defined below, and by ‘any clock cycle’ we still stay with the targeted clock cycles (given as a list  $\mathbf{T}$ ) defined in the configuration file by the user.

### 4.3 Configuration

PROLEAD receives three input files.

1. A gate-level netlist written in Verilog as used in digital circuit design to abstract the circuit. Such a netlist is produced by synthesizing the circuit’s behavioral description (e.g., VHDL or Verilog) using a hardware synthesizer, e.g., Design Compiler [Inc] or Yosys [Wol].
2. Any ASIC standard cell library can be used for the synthesis, but the functional behavior of each cell of the library should be defined in a custom file, which should be given to PROLEAD as well. Such a file is required for the simulator of PROLEAD to understand how to simulate the cells used in the given netlist. We integrated the functional behavior of most of the cells in NanGate 45 nm open-cell library.
3. A custom configuration file, allowing the users to specify their requirements. All settings regarding simulation and verification take place in this file. Primarily, the user adjusts the simulator by defining the total number of simulations and the simulation time frame in terms of clock cycles. To start a simulation, the user should ensure a correct initialization of the primary inputs to the circuit by defining an input sequence. The input sequence formalizes the state of all primary inputs during an arbitrary number of initial cycles. For example, in case of a cryptographic core, how and when plaintext and key should be given to the circuit and how handshaking signals (like `reset`) are controlled.

For verification, PROLEAD supports a customized list of wires, which is also defined by the user. Formally, a wire is either considered in the verification or ignored. The user specifies this by including the considered wire into the list  $\mathbf{E}^*$ . To ease the definition  $\mathbf{E}^*$  by the user, we include all wires to the list by default. Hence, the default case is to perform a complete evaluation in terms of wires. Furthermore, the user specifies the leakage verification by setting an appropriate security order  $d$  and choosing a proper leakage model. The configuration file contains more fine-grained settings which we ignore to express here for the sake of brevity.

### 4.4 Generation of Probing Sets

After reading the given design and configuration files, and making the graph  $(\mathbf{V}, \mathbf{E})$ , as the first step PROLEAD generates all  $d$ -probing sets that fit the desired leakage evaluation specified in the configuration file. Depending on the leakage model, the probing set generation specifies either  $\mathbf{P}^g$  or  $\mathbf{P}^{g,t}$ . Both lists encompass all probing sets (either  $\mathbf{P}_i^g$  or  $\mathbf{P}_i^{g,t}$ ) that are considered for verification. In the following, we algorithmically describe all steps required to generate the probing sets.

#### 4.4.1 Extraction of Relevant Wires

We start by determining which  $e \in \mathbf{E}$  are relevant for the verification procedure. For efficiency reasons, we generate a small list of relevant wires  $\mathbf{H}$  as long as they fit the defined configuration. The generation of  $\mathbf{H}$  is presented in Algorithm 1. As we, at least, consider glitches, every probe  $p$  gets extended by probes on the whole combinational circuit

that contributes to the intermediate signal probed by  $p$ . According to the circuit model (cf. Figure 1), a probe  $p$  on an intermediate signal of the combinational logic leads to additional probes on a subset of primary inputs and register outputs. We consider the  $n$ -bit output of the combinational circuit, with  $n = |\mathbf{S}| + |\mathbf{O}|$ ,<sup>2</sup> as a set of  $n$  coordinate functions  $\{f_0(\mathbf{I}_0, \mathbf{S}_0), \dots, f_{n-1}(\mathbf{I}_{n-1}, \mathbf{S}_{n-1})\}$  while each  $f_j$  operates on an individual set of primary inputs  $\mathbf{I}_j \subseteq \mathbf{I}$  and register outputs  $\mathbf{S}_j \subseteq \mathbf{S}$ . For illustration, we consider the following example based on a single coordinate function  $f_j(\mathbf{I}_j, \mathbf{S}_j)$ . Since the subcircuit computing  $f_j(\mathbf{I}_j, \mathbf{S}_j)$  is fully combinational, a single probe on the output of  $f_j(\mathbf{I}_j, \mathbf{S}_j)$  expands to probes on all signals in  $\mathbf{I}_j$  and  $\mathbf{S}_j$ . Hence, placing additional probes on intermediate signals of a coordinate function is not necessary as all inputs of  $f_j$  are already covered. Therefore, we don't have to consider a probe on every intermediate signal of the circuit. It is enough to place probes on all output wires of the combinational circuit  $\{\mathbf{S} \cup \mathbf{O}\}$ . Consequently, we add the output wires of all coordinate functions to  $\mathbf{H}$ . To this end, we need to be in conformity with the user-defined configuration regarding allowed and ignored signals. More precisely, it should hold that  $\mathbf{H} \subseteq \mathbf{E}^*$ . We satisfy these properties for each  $e \in \mathbf{H}$  by analyzing only elements in  $\mathbf{E}^*$  (cf. Line 2 of Algorithm 1). To examine if a signal is the output of a coordinate function, we determine whether the circuit propagates the underlying signal to another combinational gate. This is done in Lines 4-8. Considering the circuit model, given in Section 4.2.1, each  $e \in \mathbf{H}$  can only be either an input of the registers or a primary output which is additionally not an input of any combinational gate.

---

**Algorithm 1** Extraction of relevant wires
 

---

**Input:**  $\mathbf{E}^*$  ▷ Lists of desired (included) wires  
**Output:**  $\mathbf{H}$  ▷ List of relevant wires

- 1:  $\mathbf{H} \leftarrow \emptyset$
- 2: **for**  $\forall e \in \mathbf{E}^*$  **do** ▷ Analyze all considered signals of the circuit
- 3:      $\alpha \leftarrow \text{true}$
- 4:     **for**  $\forall v \in \mathbf{V}_e$  **do** ▷ Analyze all cells that receive the signal  $e$  as input
- 5:         **if**  $g_v = \text{true}$  **then** ▷ Check if the signal is given to a combinational gate
- 6:              $\alpha \leftarrow \text{false}$
- 7:         **end if**
- 8:     **end for**
- 9:     **if**  $\alpha = \text{true}$  **then**
- 10:          $\mathbf{H} = \mathbf{H} \cup \{e\}$  ▷ Add the considered wire to the list
- 11:     **end if**
- 12: **end for**

---

#### 4.4.2 Combination of Probes

After inserting all relevant wires of the circuit into  $\mathbf{H}$ , based on the user-defined configuration we place and combine the probes resulting in multiple (yet non-extended)  $d$ -probing sets  $\mathbf{P}_i \in \mathbf{P}$ . For efficiency reasons, we avoid duplicates in all  $\mathbf{P}_i \in \mathbf{P}$ . Hence, it holds that  $p_j \neq p_k$  for all  $p_j, p_k \in \mathbf{P}_i$  with  $j \neq k$ . Furthermore, the order of probes inside the probing set does not affect the verification. We abstract the generation of all  $\mathbf{P}_i \in \mathbf{P}$  as finding all possible  $d$ -combinations of elements  $\mathbf{L}'$  with  $|\mathbf{L}'| = d$  and  $\mathbf{L}' \in \mathbf{L}'$  in a predefined list of elements  $\mathbf{L} = \{l_0, \dots, l_{|\mathbf{L}|-1}\}$  which is a common problem in combinatorics. We show the generation of all possible  $d$ -combinations in  $\mathbf{L}'$  in Algorithm 2. For simplicity, we operate on a bit-vector of  $|\mathbf{L}|$  indices  $\mathbf{M} = \langle m_0, \dots, m_{|\mathbf{L}|-1} \rangle$  with  $m_i \in \mathbb{F}_2$ . Initially, it holds that  $m_i = 1$  if  $i < d$  and  $m_i = 0$  otherwise to start with the first combination, i.e., Line 3 of

---

<sup>2</sup>There might be some overlap between the signals of  $\mathbf{S}$  and  $\mathbf{O}$ , but this does not harm the given definitions.

**Algorithm 2.** In Lines 15-32,  $\mathbf{M}$  is modified to represent the next possible  $d$ -combination of element indices. Based on the current combination of indices shown by  $\mathbf{M}$ , we store the corresponding  $d$ -combination  $\mathbf{L}'_i = \{l_{m_0}, \dots, l_{m_{d-1}}\}$  before the indices are updated (cf. Line 13). Our algorithm terminates if the last  $d$ -combination is reached. This is the case if it holds that  $m_i = 0$  for all  $i < |\mathbf{L}| - d$ . We capture the last combination in Line 6.

---

**Algorithm 2** Make  $d$ -combinations of elements

---

**Input:**  $\mathbf{L} = \{l_0, \dots, l_{|\mathbf{L}|-1}\}$  ▷ List of elements  
**Input:**  $d$  ▷ Size of combinations  
**Output:**  $\mathbf{L}' = \{\mathbf{L}'_0, \dots, \mathbf{L}'_{\binom{|\mathbf{L}|}{d}-1}\}$  ▷ Lists of  $d$ -combinations

```

1:  $\alpha \leftarrow \text{true}$ 
2:  $\mathbf{M} \leftarrow \langle m_0, \dots, m_{|\mathbf{L}|-1} \rangle, \forall m_i = 0$  ▷ Initialize a vector that stores the element indices
3: for  $i \in \{0, \dots, d-1\}$  do ▷ Get the first combination
4:    $m_i \leftarrow 1$ 
5: end for
6: while  $\alpha = \text{true}$  do
7:    $\mathbf{F} \leftarrow \emptyset$  ▷ Get an empty temporary probing set
8:   for  $i \in \{0, \dots, |\mathbf{L}|-1\}$  do
9:     if  $m_i = 1$  then
10:       $\mathbf{F} \leftarrow \mathbf{F} \cup \{l_{m_i}\}$  ▷ Store the element in the combination
11:    end if
12:  end for
13:   $\mathbf{L}' \leftarrow \mathbf{L}' \cup \mathbf{F}$  ▷ Add the  $d$ -combination to the list
14:   $\alpha \leftarrow \text{false}$ 
15:  for  $i \in \{0, \dots, |\mathbf{L}|-1\}$  do ▷ Get next combination
16:    if  $m_i = 1 \wedge m_{i+1} = 0$  then
17:       $\alpha \leftarrow \text{true}$ 
18:       $m_i \leftarrow 0$ 
19:       $m_{i+1} \leftarrow 1$ 
20:       $s \leftarrow 0$ 
21:      for  $j \in \{0, \dots, i-1\}$  do
22:        if  $m_j = 1$  then
23:           $m_j \leftarrow 0$ 
24:           $s \leftarrow s + 1$ 
25:        end if
26:      end for
27:      for  $j \in \{0, \dots, s-1\}$  do
28:         $m_j \leftarrow 1$ 
29:      end for
30:      break
31:    end if
32:  end for
33: end while

```

---

However, to compute the probing sets, we have to specify  $\mathbf{L}$  and add the relevant timing information. As the relevant timing combinations differ for the univariate and multivariate cases, we present each procedure separately.

**Univariate Probe Generation.** Covering only univariate leakages leads to a straightforward probe generation approach shown in [Algorithm 3](#). As all probes must be annotated with the same clock cycle, we generate  $d$ -combinations according to [Algorithm 2](#) applied on



**H.** Afterwards, we consider the  $d$ -combinations of wires  $\mathbf{H}' = \{\mathbf{H}'_0, \dots, \mathbf{H}'_{\binom{|\mathbf{H}|}{d}-1}\}$  and the relevant clock cycles  $\mathbf{T} = \{t_0, \dots, t_{|\mathbf{T}|-1}\}$ . For each  $\mathbf{H}'_i \in \mathbf{H}'$  with  $\mathbf{H}'_i = \{e'_0, \dots, e'_{d-1}\}$  and each  $t \in \mathbf{T}$ , we compute a probing set  $\mathbf{P} = \{p_0, \dots, p_{d-1}\}$  with  $p_j = (e'_j, t)$ . This procedure results in  $|\mathbf{H}'| \cdot |\mathbf{T}|$  probing sets. For example, suppose that  $d = 2$  and  $\mathbf{H}'_0 = \{e_0, e_1\}$  and three targeted clock cycles  $\mathbf{T} = \{t_0, t_1, t_2\}$ . To consider  $\{t_0, t_1, t_2\}$ , we generate  $|\mathbf{T}|$  probing sets

$$\mathbf{P}_0 = \{(e_0, t_0), (e_1, t_0)\}, \quad \mathbf{P}_1 = \{(e_0, t_1), (e_1, t_1)\}, \quad \mathbf{P}_2 = \{(e_0, t_2), (e_1, t_2)\}.$$

In other words,  $\mathbf{P}_i$  covers all probes from  $\mathbf{H}'_0$  recording at clock cycle  $t_i$ .

---

**Algorithm 3** Univariate probing set generation

---

**Input:**  $\mathbf{H} = \{e_0, \dots, e_{|\mathbf{H}|-1}\}$  ▷ List of relevant wires  
**Input:**  $\mathbf{T} = \{t_0, \dots, t_{|\mathbf{T}|-1}\}$  ▷ List of relevant clock cycles  
**Output:**  $\mathbf{P} = \{\mathbf{P}_0, \dots, \mathbf{P}_{\binom{|\mathbf{H}|}{d} \cdot |\mathbf{T}|-1}\}$  ▷ A list of probing sets

- 1:  $\mathbf{H}' \leftarrow$  result of [Algorithm 2](#) on  $\mathbf{H}$
- 2: **for**  $i \in \{0, \dots, \binom{|\mathbf{H}|}{d} - 1\}$  **do**
- 3:      $\{e'_0, \dots, e'_{d-1}\} \leftarrow \mathbf{H}'_i$
- 4:     **for**  $j \in \{0, \dots, |\mathbf{T}| - 1\}$  **do**
- 5:          $\mathbf{R} \leftarrow \emptyset$  ▷ Generate a temporary empty probing set
- 6:         **for**  $l \in \{0, \dots, d - 1\}$  **do**
- 7:              $p \leftarrow (e'_l, t_j)$  ▷ Generate and place a single probe
- 8:              $\mathbf{R} \leftarrow \mathbf{R} \cup \{p\}$  ▷ Add the probe to the probing set
- 9:         **end for**
- 10:         $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{R}$
- 11:     **end for**
- 12: **end for**

---

**Multivariate Probe Generation.** In this case, each probe  $p \in \mathbf{P}$  can be annotated with every targeted clock cycle  $t \in \mathbf{T}$ . Hence, each possible  $d$ -combination of  $\mathbf{T}$  is a possible annotation for  $\mathbf{P}$ . Before we apply [Algorithm 2](#), we generate a list of considered probes  $\mathbf{P}'$  by storing probes on all relevant wires  $e \in \mathbf{H}$  and at every time instance  $t \in \mathbf{T}$ . Hence, the resulting set  $\mathbf{P}'$  encompasses  $|\mathbf{H}| \cdot |\mathbf{T}|$  probes. Afterwards, we apply [Algorithm 2](#) on  $\mathbf{P}'$  in order to generate the  $d$ -probing sets. We again explain this by an example. Suppose  $d = 2$  and two relevant wires  $\mathbf{H} = \{e_0, e_1\}$  and the targeted clock cycles  $\{t_0, t_1, t_2\}$ . This results in six relevant probes  $\mathbf{P}' = \{p_0, p_1, p_2, p_3, p_4, p_5\}$ . It holds that:

$$p_0 = (e_0, t_0), p_1 = (e_0, t_1), p_2 = (e_0, t_2), \\ p_3 = (e_1, t_0), p_4 = (e_1, t_1), p_5 = (e_1, t_2)$$

Applying [Algorithm 2](#) on  $\mathbf{P}'$  returns  $\binom{|\mathbf{H}| \cdot |\mathbf{T}|}{d}$   $d$ -probing sets:

$$\mathbf{P}_0 = \{p_0, p_1\}, \mathbf{P}_1 = \{p_0, p_2\}, \mathbf{P}_2 = \{p_1, p_2\}, \mathbf{P}_3 = \{p_0, p_3\}, \mathbf{P}_4 = \{p_1, p_3\}, \\ \mathbf{P}_5 = \{p_2, p_3\}, \mathbf{P}_6 = \{p_0, p_4\}, \mathbf{P}_7 = \{p_1, p_4\}, \mathbf{P}_8 = \{p_2, p_4\}, \mathbf{P}_9 = \{p_3, p_4\}, \\ \mathbf{P}_{10} = \{p_0, p_5\}, \mathbf{P}_{11} = \{p_1, p_5\}, \mathbf{P}_{12} = \{p_2, p_5\}, \mathbf{P}_{13} = \{p_3, p_5\}, \mathbf{P}_{14} = \{p_4, p_5\}$$

We show the corresponding procedure, i.e., multivariate generation of  $d$ -probing sets, in [Algorithm 4](#). Storing all possible probes  $p \in \mathbf{P}'$  is given in [Line 2-7](#).

**Algorithm 4** Multivariate probing set generation

---

**Input:**  $\mathbf{H} = \{e_0, \dots, e_{|\mathbf{H}|-1}\}$  ▷ List of relevant wires  
**Input:**  $\mathbf{T} = \{t_0, \dots, t_{|\mathbf{T}|-1}\}$  ▷ List of relevant clock cycles  
**Output:**  $\mathbf{P} = \{\mathbf{P}_0, \dots, \mathbf{P}_{(|\mathbf{H}||\mathbf{C}|-1)}\}$  ▷ A list of probing sets

- 1:  $\mathbf{P}' \leftarrow \emptyset$  ▷ Get an empty list of considered probes
- 2: **for**  $i \in \{0, \dots, |\mathbf{H}| - 1\}$  **do**
- 3:     **for**  $j \in \{0, \dots, |\mathbf{C}| - 1\}$  **do**
- 4:          $p \leftarrow (e_i, c_j)$
- 5:          $\mathbf{P}' \leftarrow \mathbf{P}' \cup \{p\}$  ▷ Store all considered probes
- 6:     **end for**
- 7: **end for**
- 8:  $\mathbf{P} \leftarrow$  result of Algorithm 2 on  $\mathbf{P}'$

---

**4.4.3 Probe Extension**

Up to now,  $\mathbf{P}$  contains all probing sets relevant for verification under the  $(0, 0, 0)$ -robust  $d$ -probing model, i.e., still non-extended. However, as we take glitches and, if specified, transitions into account, we should extend  $\mathbf{P}$ . Both supported models cover glitches; hence, we start by transforming  $\mathbf{P}$  into  $\mathbf{P}^g$  containing all glitch-extended probing sets. For the sake of efficiency, we process every  $e \in \mathbf{H}$  and precompute its set of wires after glitch-extension  $e^g \in \mathbf{H}_e^g$ . To this end, we use a recursive backpropagation procedure given in Algorithm 5. In short, for the given  $e \in \mathbf{H}$ , the procedure checks if the probe extension stops, i.e., whether  $e$  is a register output or a primary input (see Line 6). If so,  $e$  is added to  $\mathbf{H}_e^g$ . Otherwise, the same procedure is repeated for all inputs of the gate whose output is  $e$  (Lines 10-13). Having the lists  $\mathbf{H}_e^g$  for all  $e \in \mathbf{H}$ , it is enough to substitute every probe  $p$  in all  $d$ -probing sets  $\mathbf{P}_i \in \mathbf{P}$  with the corresponding glitch-extended probing set. This is done by replacing  $p = (e, t)$  with  $p^g = (e^g, t)$  for all  $e^g \in \mathbf{H}_e^g$ . This way,  $\mathbf{P}^g$  which is equivalent to  $\mathbf{P}$  under the  $(1, 0, 0)$ -robust  $d$ -probing model is achieved.

**Algorithm 5** Glitch extension

---

**Input:**  $(\mathbf{V}, \mathbf{E})$  ▷ Circuit graph  
**Input:**  $e \in \mathbf{H}$  ▷ A single signal  
**Output:**  $\mathbf{H}_e^g$  ▷ The list of probed signals after glitch-extension

- 1: **for**  $i \in \{0, \dots, |\mathbf{V}| - 1\}$  **do**
- 2:     **if**  $e \in \mathbf{E}_{v_i}^{\text{out}}$  **then**
- 3:          $v \leftarrow v_i$  ▷ The cell whose output is  $e$
- 4:     **end if**
- 5: **end for**
- 6: **if**  $(\mathbf{E}_v^{\text{in}} = \emptyset) \vee (g_v = \text{false})$  **then** ▷ If  $e$  is a primary input or a register output
- 7:      $\mathbf{H}_e^g \leftarrow \{e\}$
- 8: **else**
- 9:      $\mathbf{H}_e^g \leftarrow \emptyset$
- 10:     **for**  $l \in \mathbf{E}_v^{\text{in}}$  **do** ▷ All inputs of the gate  $v$
- 11:          $\mathbf{G}_e^g \leftarrow$  result of Algorithm 5 on  $(\mathbf{V}, \mathbf{E}), l$
- 12:          $\mathbf{H}_e^g \leftarrow \mathbf{H}_e^g \cup \mathbf{G}_e^g$
- 13:     **end for**
- 14: **end if**

---

**Extension based on Transitions.** After the generation of  $\mathbf{P}^g$  and possible optimizations (explained below), the extension for transitions is done straightforwardly. Namely, every

probe  $p = (e, c) \in \mathbf{P}_i^g \in \mathbf{P}^g$  is substituted by a tuple of two probes recording the same signal but at two consecutive clock cycles, i.e.,  $\{p, p'\}$  with  $p = (e, c)$  and  $p' = (e, c - 1)$ . Hence,  $\mathbf{P}^{g,t}$  is constructed.

#### 4.4.4 Optimizations

The verification step can evaluate the list of probing sets  $\mathbf{P}^g$  or  $\mathbf{P}^{g,t}$  to verify  $d$ -probing security. However, avoiding unnecessary probes and probing sets accelerates the verification procedure. In particular, we remove probes and probing sets if they fulfill one of the properties defined below.

**Definition 8** (Duplicate). Consider a probing set  $\mathbf{P}$  and two probes  $p_i, p_j \in \mathbf{P}$ . We refer to the tuple  $(p_i, p_j)$  as a *duplicate* if  $p_i = p_j$  and  $i \neq j$ . Hence,  $\mathbf{P}$  contains the same probe twice.

**Definition 9** (Subsequence). Consider two probing sets  $\mathbf{P}, \mathbf{R}$ . We refer to  $\mathbf{P}$  as a *subsequence* of  $\mathbf{R}$  if  $\mathbf{P} \subseteq \mathbf{R}$ . Hence,  $\mathbf{P}$  is fully covered by  $\mathbf{R}$ .

Due to the construction of  $\mathbf{P}$  (cf. Algorithm 3 and Algorithm 4) neither duplicates nor subsequences can occur in the set of standard probes. However, duplicates, as well as subsequences, are introduced during the glitch extension. In particular, if multiple but different probes lead to overlapping probing sets after glitch extension. Therefore, we remove duplicates and subsequences after the glitch extension. As the transition extension is a bijection (two different probes never share the same transition-extended probe), extending  $\mathbf{P}^g$  with transitions does not introduce new duplicates or subsequences.

**Removing Duplicated Probes.** We remark that detecting duplicates in a sorted probing set  $\mathbf{P}_i^g$  has a complexity of  $\mathcal{O}(|\mathbf{P}_i^g|)$ . Therefore, we sort each duplicate-prone probing set  $\mathbf{P}_i^g \in \mathbf{P}^g$  with IntroSort [Mus97]. As IntroSort has a complexity of  $\mathcal{O}(|\mathbf{P}_i^g| \log |\mathbf{P}_i^g|)$  the sorting is very fast.

**Removing Subsequences.** First, we remove duplicated probing sets  $\mathbf{P}_i^g \in \mathbf{P}^g$  with  $\mathbf{P}_i^g = \mathbf{P}_{j \neq i}^g$  which are easy to identify. To this end, we apply IntroSort to sort the probing sets of  $\mathbf{P}^g$ . Afterwards, we go through all probing sets in  $\mathbf{P}^g$  and remove every  $\mathbf{P}_i^g \in \mathbf{P}^g$  with  $\mathbf{P}_i^g = \mathbf{P}_{i-1}^g$ . It means that we remove all duplicates of  $\mathbf{P}_i^g$  except its first occurrence. Second, we search for all tuples  $(\mathbf{P}_i^g, \mathbf{P}_{j \neq i}^g)$  with  $\mathbf{P}_i^g \subset \mathbf{P}_j^g$ . If a tuple is found, we mark  $\mathbf{P}_i^g$  as a probing set to be removed and ignore it in further searches. Finally, all marked probing sets are removed from  $\mathbf{P}^g$ . As the search for tuples has the complexity of  $\mathcal{O}(n^2)$ , we can increase the efficiency through parallelization. Each thread can compare a dedicated set of probing sets  $\mathbf{P}_i^g \in \mathbf{P}^g$  with all other probing sets in  $\mathbf{P}^g$  and mark  $\mathbf{P}_i^g$  in a shared memory if  $\mathbf{P}_i^g$  should be removed. After the termination of all threads, the entire marked probing sets can be removed. Since this process may take a very long time if the circuit is very large and a high security order  $d$  is defined, this can be deactivated through the user-defined configuration.

## 4.5 Verification

Given a list of probing sets  $\mathbf{P}^g$ , the verification step analyzes every  $\mathbf{P}_i^g \in \mathbf{P}^g$  by simulating intermediates recorded by the probes in  $\mathbf{P}_i^g$ . The same holds for  $\mathbf{P}^{g,t}$ . We give a high-level overview of the verification approach in Algorithm 6. By accomplishing the verification, the procedure returns a list encompassing the  $p$ -values  $p_i \in \mathcal{G}$  for each  $\mathbf{P}_i^g \in \mathbf{P}^g$ . The verification approach can be divided into three steps, presented as follows. Note that  $p$ -values refer to the result of statistical hypothesis tests explained in Section 2.4.

**Algorithm 6** Verification of probing sets

---

|  |  |
|--|--|
| <b>Input:</b> $(\mathbf{V}, \mathbf{E})$   | ▷ Circuit graph  |
| <b>Input:</b> $\mathbf{P}^g = \{\mathbf{P}_0^g, \dots, \mathbf{P}_{ \mathbf{P}^g -1}^g\}$  | ▷ List of probing sets (can be $\mathbf{P}^{g,t}$ as well) |
| <b>Input:</b> $n_{\text{total}}$   | ▷ Total number of simulations                              |
| <b>Input:</b> $n_{\text{step}}$  | ▷ Number of simulations per step                           |
| <b>Input:</b> $n_g$  | ▷ Number of groups   |
| <b>Output:</b> $\mathcal{G} = \{p_0, \dots, p_{ \mathbf{T} -1}\}$                          | ▷ List of $p$ -values                                      |
| 1: $\mathbf{D} \leftarrow \emptyset$   | ▷ Initialize distribution tables                           |
| 2: <b>for</b> $i \in \{0, \dots, \frac{n_{\text{total}}}{n_{\text{step}}} - 1\}$ <b>do</b> |  |
| 3: $\mathbf{S} \leftarrow \emptyset$   | ▷ Initialize simulation results                            |
| 4: <b>for</b> $j \in \{0, \dots, \frac{n_{\text{step}}}{64} - 1\}$ <b>do</b>               |  |
| 5:         Simulation( $n_g, \mathbf{V}, \mathbf{E}, \mathbf{S}_j$ )                       |  |
| 6: <b>end for</b>  |  |
| 7:     UpdateDistributions( $n_g, \mathbf{S}, \mathbf{P}^g, \mathbf{D}$ )                  |  |
| 8: $\mathcal{G} \leftarrow \text{Evaluation}(n_g, \mathbf{D})$                             |  |
| 9: <b>end for</b>  |  |

---

**4.5.1 Simulation**

The **Simulation** procedure in Line 5 of **Algorithm 6** emulates the circuit for an arbitrary input sequence to compute all probed intermediates. Through the configuration (see **Section 4.3**), the user specifies the number of groups  $n_g$  for which the statistical hypothesis test should be evaluated. Traditional tests in the context of SCA are either fixed versus random or fixed<sub>1</sub> versus fixed<sub>2</sub>, i.e., two groups. For each group, the user should naturally define the fixed value(s) or the random value (i.e., how many random bits are required). This should not be necessarily two groups, and **PROLEAD** supports any arbitrary number of groups, e.g., multiple fixed values.

During the definition of the initial primary input sequences (also stated in **Section 4.3**), the user defines which primary inputs at which clock cycles take a value assigned to one of the above-defined groups. This includes their masking as well. For example, two groups are defined as `64'h0000000000000000` and `64'h$$$$$$$$$$$$$$$$` referring to a fixed 64-bit vector fully filled by 0, and a 64-bit random vector. Exemplary, suppose that the circuit is an encryption function of a cipher masked with 3 shares, i.e., the 64-bit plaintext should be given by means of 3 shares  $P^0$ ,  $P^1$ , and  $P^2$  assigned to **SelectedGroup**<sup>0</sup>, **SelectedGroup**<sup>1</sup>, and **SelectedGroup**<sup>2</sup> respectively. For every simulation, the simulator selects one of the aforementioned groups randomly, and generates a 64-bit random vector if the random group is selected. Then, the masking (with 3 shares) of the selected 64-bit vector, so-called **SelectedGroup**, is constructed by 2 other 64-bit random vectors, i.e., selecting **SelectedGroup**<sup>0</sup> and **SelectedGroup**<sup>1</sup> at random and making **SelectedGroup**<sub>2</sub> as **SelectedGroup**  $\oplus$  **SelectedGroup**<sup>0</sup>  $\oplus$  **SelectedGroup**<sup>1</sup>.

The user further defines the maximum simulation length in number of clock cycles while the simulator emulates the circuit's state after each clock cycle iteratively. The user can also define an end condition to be checked during each simulated cycle. If the circuit state fulfills the end condition, the simulation terminates. A possible end condition is to stop if one or multiple primary outputs, e.g., a **done** signal, reached a specified value.

As given in **Section 4.2.1**, the given circuit is modeled as a Mealy machine consisting of a register stage storing the output of a combinational circuit, whose input is provided by a combination of the primary inputs and the registers' output. Therefore, once the masked primary inputs are prepared, the simulator starts iterating through the clock cycles. To this end, the following operations are performed per clock cycle until the end condition is met or the maximum number of clock cycles is reached.

1. The primary inputs are updated. More precisely, it is checked if for the current clock cycle a new value for the primary inputs is defined through the initial input sequence (see Section 4.3).
2. Register outputs are updated, i.e., the signals connected to registers' outputs reflect the corresponding values stored in the registers.
3. Now, the input of the combinational circuit is fully defined; hence, it can be evaluated. This is done following the concept of event-driven simulation. Since we consider no delay for the gates, this can be simplified by processing the combinational circuit in the order of the logical depth<sup>3</sup>. At the end of this step, the outputs of the combinational circuit (which are the circuit's primary outputs and/or the registers' input) are provided.
4. As the last step of every clock cycle, the registers store the values appearing at their inputs based on the status of their corresponding control signals, e.g., `clock`, `enable`, `reset`, etc.

Note that the above procedure is valid only for circuits, where all registers are synchronized, e.g., all of them see the positive-edge/negative-edge of the clock signal. PROLEAD optionally can handle other circuits, e.g., with latches and clock gating, but this requires to evaluate the combinational circuit two times per clock cycle, once when the clock signal is high and one more time when it is low (i.e., lower efficiency).

To decrease the runtime, the simulator of PROLEAD works on 64-bit variables, i.e., handling 64 independent simulations in parallel, which indeed follows the concept behind bit-slicing. We should highlight that only the above-given operations per clock cycle are performed on 64-bit variables. Almost all other operations, e.g., the group selection and preparation of masked primary inputs, are done ordinarily (not bit-sliced). Further, since simulations are fully independent of each other, several simulations are performed in parallel by means of multi-treading, which is also adjusted by the user through the configuration file. As shown in Algorithm 6, the total number of simulations is denoted by  $n_{\text{total}}$  which is divided by  $n_{\text{step}}$  (also defined by the user based on the available memory) denoting the size of each simulation set which should be performed before updating the evaluation results. This allows the user to observe the evaluation results after each  $n_{\text{step}}$  simulations.

#### 4.5.2 Update Distributions

Before the evaluation takes place, we convert the simulation results into one individual and independent distribution table per probing set. This is essential since the statistical hypothesis test requires such distributions to estimate  $p$ -values. For this, we go through all simulations and concatenate the recorded bits of all  $p \in \mathbf{P}_i$  into a value with at least  $n_{\text{probe}}$  bits while  $n_{\text{probe}} = |\mathbf{P}_i|$ . In the following, we refer to these concatenated  $n_{\text{probe}}$ -bit values as *keys*. Each entry of a distribution table stores an individual key and how often the key occurs per group, i.e.,  $n_g$  individual numbers.

For each set of  $n_{\text{step}}$  simulations, we compute the keys of each probing set and update the corresponding distribution table. To efficiently search for a key in the distribution table, we keep the distribution tables sorted by their keys. Hence, searching for keys has logarithmic complexity  $\mathcal{O}(\log n)$  in the number of observed keys. If a key is found in the table, the occurrence of the corresponding group is increased by one. Otherwise, an empty table entry with the new key (for all  $n_g$  groups) is added into the sorted distribution table

<sup>3</sup>Note that it should not be misunderstood that we do not cover glitches in our security evaluation. Indeed, we do not simulate the glitches, but the underlying glitch-extended probing model covers any form of glitches which may happen in the realization of the given circuit and affect its security.

before incrementing the corresponding occurrence. As we searched for the key before, the complexity for insertion into the sorted distribution table becomes independent of whether the key exists in the table or not. This process is also parallelized through multi-threading as the distribution table of different probing sets can be updated independently of each other. Hence, multiple distribution tables are updated in parallel and each table is modified only by a single thread.

### 4.5.3 Leakage Evaluation

As given in Section 2.4, we make use of the  $G$ -test as the statistical hypothesis test. More precisely, we apply the  $G$ -test on the  $n_g$  distribution tables of each probing set individually to achieve a measure for the independence of the distributions. Hence, we obtain the corresponding  $p$ -value  $p_i$  for each probing set, based on which we report the detectability of a leakage. This is the case if  $-\log_{10}(p)$  exceeds the predefined threshold, i.e., the null hypothesis  $H_0$  is rejected. Line 8 of Algorithm 6 performs this operations and stores  $p$ -values in  $\mathcal{G}$  which can be shown, printed, or stored in a file after each  $n_{\text{step}}$  simulations.

## 4.6 Statistical Confidence

We refer to our results as statistically confident as soon as the error probabilities become acceptable. For the false positive probability, we predefine a threshold probability and reject  $H_0$  if the computed  $p$ -value becomes smaller than the threshold. As  $p < 10^{-5}$  is a common threshold for leakage assessment, e.g. t-test [SM15] and  $\chi^2$ -test [MRSS18], we decide to set the threshold probability to  $10^{-5}$ . Hence, the chance to falsely reject  $H_0$ , i.e. to report a secure design as insecure is smaller than  $10^{-5}$  for each probing set. However, false positives can become very likely if the number of probing sets exceeds  $10^5$ . If this happens, e.g. if the experiment considers millions of probing sets, we recommend decreasing the threshold to an acceptable level. However, the decisive factor is not that the threshold is exceeded but that the  $p$ -value decreases continuously with an increasing number of simulations. To bring the false negative probability to an acceptable level, we apply the power analysis techniques introduced in Section 2.4. During the verification procedure, we continually monitor the sample size needed to satisfy statistical confidence for a predefined tuple  $(\beta, \varphi)$ . In particular, we numerically estimate the number of simulations required to satisfy an error probability of  $\beta$  for an effect size  $\varphi$ . To estimate the number of required simulations, we define a range in which we try to approximate the necessary number of simulations. In practice, we define a very large range from one to one billion to be sure that the number of required simulations lies in the range. Then, we apply a trial-and-improve strategy which is one-simulation accurate and achieves logarithmic complexity in the upper bound of the range. For the case studies in Section 5,  $\beta$  equals the threshold of  $10^{-5}$  for experiments with less than  $10^5$  probing sets while we set the effect size to  $\varphi = 0.1$ . Hence, by applying these parameters, we detect all small effects with an error rate of  $10^{-5}$ . While we suggest using these parameters as the default settings, we remark that the user can choose arbitrary parameters according to the desired security guarantees and the evaluated design. The estimation of required simulations takes place after each leakage evaluation step. As the computation of the statistical power depends on the degree of freedom, we estimate the number of required simulations for the probing set resulting in the highest degree of freedom. Hence, we can be sure that the estimated number of simulations is enough for all considered probing sets. Naturally, the degree of freedom grows with every new spotted intermediate that we store in the contingency table. Hence, the number of required simulations can only be estimated during the evaluation and grows if the degree of freedom grows. Nevertheless, the growth slows down or stops if all possible probed values of a set are considered in the contingency table. Hence, we stop the evaluation as soon as the number of performed simulations reaches the number of required simulations. In



Section 5.3, we visualize the progression of the different parameters based on two practical case studies in Figure 2.

## 5 Case Studies

In order to examine the ability as well as performance of PROLEAD, we evaluated several designs, which are mainly available through public repositories, like GitHub. In other cases, we either received the design from the corresponding authors or constructed the designs by ourselves. In short, PROLEAD can rapidly identify leakage in unmasked designs and those which we were aware of their security flaw. Further, we found out several mistakes and shortcomings of publicly-available masked designs which are supposed to be probing secure. In the following we elaborate each case study, but we keep the description of each one short and mainly refer to the original article.

**Setup.** We made use of Synopsis Design Compiler and NanGate45 ASIC standard cell library to synthesis and generate the netlist of each case study. We made sure to avoid optimization across different modules (i.e., keeping the design hierarchy<sup>4</sup>) to not violate any assumptions of the original designer, e.g., not violating non-completeness [NRS11]. We further provided the functional description of most of the cells in the NanGate45 library thereby PROLEAD can understand and simulate the given netlist (see Section 4.3). We ran the evaluations on a machine with an AMD EPYC 7352 (48 hyper-threading cores) and 128 GB of memory.

For the entire evaluations we considered two groups ( $n_g = 2$ ), one group fully random and the other one fully zero. In other words, we performed fixed versus random  $G$ -test when fixed inputs is  $\{0\}^t$  and random input  $\xleftarrow{\$} \mathbb{F}^t$ , where  $t$  stands for the size of the input vector. For all case studies, we kept the key of the design (if any) to the zero vector. Note that this does not have any effect on the result of our evaluations. However, if the design receives the key in a masked form, we gave the masked representation of the zero vector updated at the start of each circuit simulation.

For all case studies, we first conducted the evaluations by only covering glitches, i.e., no transitions. If we found no leakage, we then extended the evaluation by additionally covering the transitions. Therefore, if not stated, when we report vulnerability of a circuit, we mean when only glitches are taken into account. Likewise, when we report security of a circuit, we refer to the case where both glitches and transitions are covered. For all case studies, we set the effect size to  $\varphi = 0.1$ . Therefore, if we report a design as secure, we mean that no effect with a effect size of  $\varphi \geq 0.1$  was detected by PROLEAD. Only when we discover strong leakage, we increased  $\varphi$  to improve the readability of our results. The summary of all conducted evaluations are shown in Table 2.

### 5.1 Unmasked Designs

The vulnerability of unmasked designs is expected. Hence, just for sanity check, we evaluated a round-based implementation of SKINNY-64 [BJK<sup>+</sup>16], which is available [online](#)<sup>5</sup>. PROLEAD reports first-order leakage at all clock cycles in less than a second using less than 100 simulations.

**Hiding Countermeasures.** When evaluating probing security of unmasked implementation, adding either amplitude or temporal noise does not have any effect on their vulnerability. Examples include adding an independent noise module, adding jitter to the

<sup>4</sup>In Design Compiler this is achieved by setting `-no_autoungroup` and `-no_boundary_optimization` switches for the `compile_ultra` command.

<sup>5</sup><https://sites.google.com/site/skinnycipher/downloads>

clock, or randomizing the clock source [MOP07]. The same holds if the circuit is realized by a dual-rail pre-charge logic as a power-equalization technique. Placing a single probe on one of the rails would lead to leakage about the actual signal. It is even the same in case of Masked Dual-rail Pre-charge Logic (MDPL) [PM05], as one probe on a gate output rail propagates to e.g.,  $a_m = a \oplus m$ ,  $b_m = b \oplus m$ , and  $m$ , which clearly leaks information about  $a$  and  $b$ .

Here, we would like to stress that such dual-rail pre-charge logics are free of glitches, but the glitch-extended probes should still be propagated to the input of combinational circuit. The name “glitch-extended” is not bounded by glitches. The propagation delay of CMOS gates depends on the given input; for example there is a small delay difference when the input of an AND gate changes from 11 to 01 or to 10.<sup>6</sup> This makes the power consumption of the gate to depend on the given data, although no glitch happens on the circuit. A similar concept with larger granularity is known as “data-dependent time of evaluation” [MOP07]

Nevertheless, via several case studies elaborated below, we show that many implementations are not probing secure, while the authors have not seen leakage in practice using often 100 million measurements. This obviously depends on the quality of the measurement setup and many other factors involved in the experimental analyses. The leakage might be detected using another setup running in another environment, or if the gates of the underlying circuits are realized differently. This highlights the relevance of (robust) probing security model. If a circuit is probing secure when both glitches and transitions are taken into account, its security in practice is independent of how the gates are realized and how precise the measurement setup is. Of course, this statement does not include the coupling effects yet.

## 5.2 Small Masked Circuits

SILVER [KSM20] is able to evaluate the (robust)-probing security of small masked circuits, e.g., S-Boxes with low input size (including the masked inputs and fresh masks). For large circuits, either the tool cannot build the Binary Decision Diagram (BDD) of the given circuit or the evaluation is not accomplished in a reasonable time.

As the first masked circuit, we have taken the TI design of the S-Box of the PRESENT cipher [BKL<sup>+</sup>07] presented in [PMK<sup>+</sup>11], where the S-Box is decomposed to two quadratic bijections, and each of which is masked by three shares following classical TI scheme [NRS11]. Similar to SILVER, PROLEAD detects no leakage with glitches and transitions. As stated in Section 3, this is one of the cases in which `maskVerif` is too conservative and reports leakage.

As a flawed design, we took the PRESENT TI S-Box of [EGMP17], without correction terms (i.e., with non-uniform output sharing) which is supposed to be insecure. Both SILVER and PROLEAD find probes with first-order leakage.

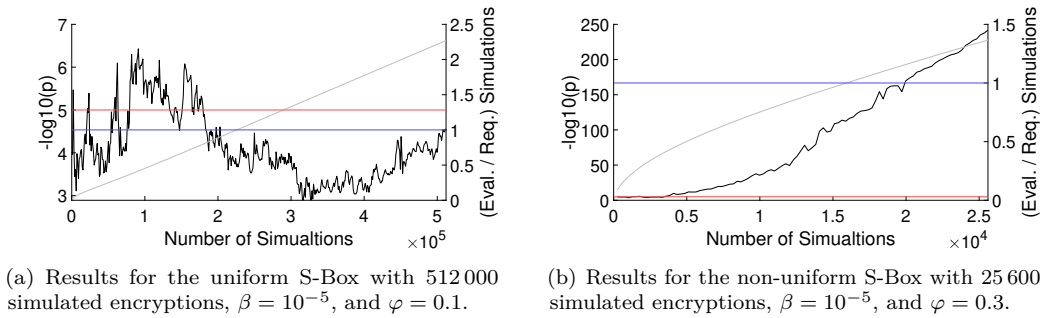
The first AES TI S-Box with 3 shares has been proposed in [MPL<sup>+</sup>11]. Due to the size of the circuit and its number of inputs (including 52 fresh masks), it is out of the capacity of SILVER. However, PROLEAD confirms its first-order security (glitches and transitions).

Other  $d + 1$  masked AES S-Boxes at arbitrary order have been introduced in [GMK16, GMK17, CRB<sup>+</sup>16]. The first-order designs have been successfully evaluated by SILVER in [KSM20], what PROLEAD also does. The higher-order designs, however, are too large for SILVER, while we are able to confirm their security by PROLEAD.

## 5.3 Masked Full Ciphers

**Serialized PRESENT.** The first complete cipher design, which we have evaluated, is the nibble-serialized PRESENT encryption function of [PMK<sup>+</sup>11], where the above-discussed

<sup>6</sup>For example, see [here](#) for NanGate 45 and [here](#) for TSMC 250.



**Figure 2:** PROLEAD results for the nibble-serialized PRESENT encryption functions.

TI S-Box is instantiated. PROLEAD has confirmed the first-order security of the design in less than two minutes while SILVER cannot handle such designs. By exchanging the S-Box with the flawed one, and evaluating the encryption function in whole with PROLEAD, we also detected first-order leakage rapidly.

To visualize the results and the statistical confidence, we show the progression of  $p$  and the number of required simulations for both serialized PRESENT designs in Figure 2. Each plot encompasses the  $p$ -value as  $-\log_{10}(p)$  which is drawn in black, while the horizontal line (red) indicates the  $10^{-5}$  threshold of the  $p$ -values. Moreover, the grey line shows the relationship between processed simulations and required simulations to achieve  $\beta < 10^{-5}$ . Hence, our results become statistically confident if as many simulations as needed are processed. This is visualized by another threshold at 1.0 (drawn in blue) which has to be passed by the grey line. For the uniform design, shown in Figure 2(a), we plot the results for 512 000 simulated encryptions. 512 000 turns out to be twice as much as needed since the result is confident after 225 567 simulations. We can conclude that the design is secure since  $p$  stays under the threshold for more than 225 567 simulations. For the non-uniform S-Box, shown in Figure 2(b), we detect leakage after thousands of simulations. Hence, we conclude that the corresponding effect size is higher than 0.1 and set  $\varphi = 0.3$  to detect only moderate effect sizes. We simulate 25 600 encryptions while  $\beta$  falls under  $10^{-5}$  after around 16 000 simulations. Although we can only detect medium effect sizes reliably, the leakage is visible as the  $p$ -value is far above the threshold.

**Serialized AES.** We further evaluated the masked full cipher designs of [MPL<sup>+</sup>11, GMK16, CRB<sup>+</sup>16], whose underlying masked S-Boxes we have already evaluated. In summary, we have not detected any first-order leakage when evaluating the first-order designs. For each design, PROLEAD required between 13 and 50 minutes to accomplish the full evaluation process, i.e., glitches and transition with an effect size of  $\varphi = 0.1$ .

**Registers with enable.** It is a common practice to use registers with enable in designs, where some registers should not store their inputs at every clock cycle. We noticed that NanGate 45 cell library does not contain any of such register cells, which are then realized by a normal register and a multiplexer to re-store the register’s output when the enable signal is low. Although this does not pose any problem to either the functionality or the security of the design, placing a probe on the input of such a register would propagate to its output as well (through the aforementioned multiplexer). Therefore, even without considering transitions, the evaluation would observe the input and output of those registers, hence inherently covering transitions.

**Null Fresh.** The authors of [SM21a] have introduced a technique to realize first-order secure implementation of (up to) cubic functions with two shares and without any fresh

masks. They applied the technique on the S-Box of several ciphers and provided the encryption function of the full ciphers in [GitHub](#)<sup>7</sup>. We evaluated all designs explained below.

**Midori-64 [BBI<sup>+</sup>15]**. It is a round-based implementation supporting both encryption and decryption. Considering glitches, we did not find any first-order leakage. The S-Box has an internal register stage; hence, the full cipher implementation forms a pipeline with two stages, i.e., two clock cycles per cipher round. Hence, two consecutive plaintexts (for encryption) can be given to the circuit. If the input (plaintext) is the same for both cycles, i.e., the `reset` signal is high for two clock cycles, PROLEAD detects first-order transitional leakage.

It is actually a general problem and not dedicated to this design. Suppose that the first pipeline stage realizes the function  $f(\cdot)$  and the second pipeline stage the function  $g(\cdot)$ . Let us denote the input of the cipher by  $A$  which is given to the circuit two consecutive clock cycles. After two clock cycles, the first pipeline stage has computed  $f(A)$  and the second one  $g(f(A))$ , which is equal to the application of one cipher round on  $A$ . In the next clock cycle, the input of the first pipeline stage changes from  $A$  to  $g(f(A))$ . Therefore, placing a transition-extended probe on the first pipeline stage would observe some bits of  $A$  and the corresponding bits of  $g(f(A))$ . It means that by one probe, the input and output of a cipher round are recorded, which most of the time leads to detectable leakage. As a general rule, in pipeline designs, consecutive inputs should not be the same, i.e., should not have the same masking (initial sharing). By filling one pipeline stage with another independently-masked input or a zero vector right after giving the desired input, i.e., inserting a bubble into the pipeline as suggested in [CS21], the observed first-order transitional leakage has vanished.

**PRESENT-80**. It is the encryption function based a nibble-serial design architecture. In short, we found out that the plugged masked S-Box is first-order secure, but not the encryption function. Considering only glitches, we have not observed any first-order leakage, but once transitions are taken into account, we rapidly (i.e., using less than 10 000 simulations) detect first-order leakage. The reason for such a leakage lies on its serialized architecture. In such a design, at every clock cycle some multiplexers decide whether the register stores the S-Box output or the P-Layer output, or loads in parallel (or serial). Hence, placing a probe at a register input propagates to many other register outputs including some of those belonging to the masked S-Box.

There is no problem during the serialized computation of the S-Box, after which the P-Layer is applied. Since the S-Box has an internal register stage, during the application of the P-Layer, one state nibble so-called  $X$  (on which the S-Box is already applied) appears at the S-Box input module, and the internal register is filled with lets say  $f(X)$ . The P-Layer stores one bit of  $X$  at the first nibble of the state register which is again given to the S-Box module at the next clock cycle. Hence, a probe placed at the output of the S-Box internal register observes some information about shares of  $X$  in two consecutive clock cycles. Together with other glitch-extended probes explained above, information about  $X$  is revealed. This can be avoided by disabling the S-Box internal register when performing P-Layer.

**PRINCE [BCG<sup>+</sup>12]**. Similar to that of Midori-64, it is a round-based implementation of both encryption and decryption, and forms a pipeline of two stages. The authors have themselves stayed that their masked S-Box does not provide a uniform output sharing. However, they claimed that the diffusion layer makes the masked input of every S-Box in the next round uniform. By PROLEAD we observed first-order leakages at clock cycles 7, 9, and 13, i.e., in rounds 4, 5 and 7 of the encryption. It is indeed confirmed that the diffusion layer helps since no leakage during the second encryption round are observed,

<sup>7</sup><https://github.com/Chair-for-Security-Engineering/NullFresh>

but after some rounds the sharing of the S-Box inputs becomes non-uniform. However, we should note that this leakage might be not exploitable since an attack would require to guess many key bits to be conducted in the cipher's middle rounds.

**AES-128.** The authors have provided two masked designs for inversion in  $GF(2^4)^2$ , one with one fresh mask and another one without any. As they stated, these masked S-Boxes do not have uniform output sharing. Hence, similar to that of PRINCE, they used the diffusion layer (MixColumns) to make the input sharing of the S-Boxes in the next round uniform. PROLEAD rapidly detected first-order leakage in these implementations. Through inspecting the reason, we found a design flaw in their scheme, which we explain as follows.

Let us denote the output of the  $GF(2^4)^2$  inversion of four S-Boxes by  $\mathbf{I} : \langle A', B', C', D' \rangle$  and the composition of the output isomorphism and the affine transformation of the AES S-Box by  $A_O(\cdot)$ . The authors wrote MixColumns  $\mathbf{O} = \mathbf{M} \cdot A_O(\mathbf{I})$ , where  $\mathbf{M}$  stands for the MixColumns matrix, as  $\mathbf{O} = \mathbf{M}' \cdot \beta \cdot A_O(\mathbf{I})$ , where  $\beta$  is a  $4 \times 4$  matrix with elements in  $\{0, 1\}$ . Therefore, they could write  $\mathbf{O} = \mathbf{M}' \cdot A_O(\beta \cdot \mathbf{I})$ . Then, they stored the result of  $\mathbf{I}' = \beta \cdot \mathbf{I}$  in dedicated registers, and then applied  $\mathbf{M}' \cdot A_O(\mathbf{I}')$  together. The first issue, which we found with PROLEAD, is exactly at this point. The authors claimed that each byte of  $\mathbf{I}'$  has now a uniform sharing, which is true, but not  $\mathbf{I}'$  in whole. Matrix  $\mathbf{M}'$  consists of two constants  $\{2, 3\}$  in each row, i.e., applied on two bytes of  $A_O(\mathbf{I}')$ . Hence, placing a probe on multiplication by  $\mathbf{M}'$  propagates to several bits of  $\mathbf{I}'$ , which are jointly not uniform, hence first-order leakage.

Placing an extra register at the output of  $A_O(\mathbf{I}')$  would solve this issue, but the output of the MixColumns is directly given to the input affine of the S-Box module (for the next round) without any register (see Figure 6 of [SM21a]). Therefore, a probe placed on the input affine combined with the input isomorphism of the S-Box would again propagate to the output of several registers storing  $A_O(\mathbf{I}')$  which are again not jointly uniform. The only solution which we found to mitigate this leakage is to add one more register at the output of MixColumns, i.e., one register to store  $\mathbf{I}'$ , one to store  $A_O(\mathbf{I}')$ , and one more to store the MixColumns output  $\mathbf{O}$ . However, the problem that we have reported above on the non-uniformity of the S-Box inputs in the middle rounds of PRINCE, holds true here as well, since this technique would avoid the leakage at the second cipher round, but not necessarily at all rounds.

**Null Fresh 2.** In a follow-up work [SM21b], the authors have extended their technique to the second-order with three shares, and presented second-order glitch-extended probing secure implementation of quadratic functions without any fresh masks. Composing such functions still necessitates the insertion of fresh masks, which can be at minimum. The authors applied their proposed scheme on the S-Box of several ciphers and provided the HDL code of the masked full cipher implementations in [GitHub](#)<sup>8</sup>, which we have taken and evaluated as given below.

**Keccak [BDPA].** Both first- and second-order designs realizing a round-based implementation of Keccak- $f[200]$  with three pipeline stages and without any fresh masks are provided by the authors. Following the principle given for Null Fresh Midori-64 design with respect to bubbles in the pipeline (in [page 334](#)), we detected no leakage.

**Midori-64, PRINCE, SKINNY-64.** These round-based implementations have often four pipeline stages (seven stages for PRINCE), and require 128-bit fresh mask bits per clock cycle. Our analyses did not find any first- or second-order leakage.

**PRESENT-80.** Similar to the first-order design [SM21a], it is a nibble-serial implementation of the encryption function. We have detected first- and second-order univariate leakage at the second cipher using around 5000 simulations. The reason is that the output

<sup>8</sup><https://github.com/Chair-for-Security-Engineering/NullFresh2>

sharing of the second stage of the decomposed S-Box is not uniform, although the authors claimed its uniformity. Hence, after the application of the masked S-Box on all state nibbles and the P-Layer, the probes which are placed on the S-Box in the second cipher round exhibit leakage. If the key is also masked, the leakage is restricted to only second order. Otherwise, first-order leakage is detected. This can be solved by either exchanging the S-Box design with another one with uniform output sharing or introducing 8-bit fresh mask at the end of each S-Box to refresh its output sharing.

**Low-Latency Keccak.** In [ZSS<sup>+</sup>21], the authors have introduced a technique to combine masked  $\chi$  and  $\theta$  functions of Keccak without placing any register in between. This allowed them to construct a generic round-based design with one clock cycle per round, supporting all variants of Keccak and at any arbitrary order. This comes at the cost of a relatively high demand for fresh masks.

Since the design is generic, we have evaluated the smallest variant, i.e., Keccak- $f$ [25] which is available in [GitHub](#)<sup>9</sup>. Focusing on the first-order design, by means of only glitches we have not found any first-order leakage confirming their security arguments. However, we have observed strong first-order leakage when transitions are also considered in the evaluation, i.e., using 2-3 million simulations. The reason for such a leakage is that the design is made to perform one masked Keccak round in every clock cycle, i.e., one register stage in the round-based implementation. A probe at the input of a single state register would propagate to output of several state registers, due to the composed combinational circuit  $\chi$  and  $\theta$ . Taking transitions into account, these probes record those state registers at two consecutive Keccak rounds. This is not a general statement, but when the output of a masked circuit is written on its input, transitions usually lead to leakage. For example, this holds true for most of the cases studied in [MKSM22]. This also holds true for this Keccak implementation. To be more precise, the leakage is detected only during the first and the last clock cycles, i.e., the first and last Keccak rounds, which potentially can lead to exploitable leakage. This is due to some multiplexers placed to load the input at the first clock cycle and another multiplexers to avoid the  $\theta$  function at the last Keccak round.

The same holds for higher-order designs. However, many more simulations are required to detect such higher-order transitional leakages. This is due to the size of the combinational circuit and the dependency of each register input to several other register outputs. For example, two probes placed on register inputs of a second-order design propagates to 328 other probes. This hardens the detection of leakages since the distribution tables need many samples to be filled, i.e., better estimated. For the second-order design we required around 500 million simulations to detect the aforementioned transitional second-order leakage. This becomes harder on higher-order designs, hence very unlikely exploitable.

**Low-Random Masking.** In order to reduce the fresh masks in designs that achieve second-order security, a technique has been introduced in [BDMS22] which allows to reuse the fresh masks at every cipher round. More precisely, the fresh masks should be updated only together with the given input (plaintext and key). The authors have applied the underlying technique on several ciphers and provided full cipher designs in [GitHub](#)<sup>10</sup>, which we fully evaluated. All designs have a round-based architecture, while for each cipher two designs are provided. One has a higher number of pipeline stages requiring the lowest number of fresh masks, and the another one has a lower latency necessitating a higher number of fresh masks. In none of the provided designs, we have found any first-order leakage. Hence, the evaluations given below focus only on the second-order leakages.

**LED-128 [GPPR11].** We have not found any leakage in the design with 5 pipeline stages. However, we detected multi-variate second-order leakage in the 3-stage design using few

<sup>9</sup>[https://github.com/Chair-for-Security-Engineering/Low-Latency\\_Keccak](https://github.com/Chair-for-Security-Engineering/Low-Latency_Keccak)

<sup>10</sup>[https://github.com/Chair-for-Security-Engineering/Low\\_Random\\_Masking](https://github.com/Chair-for-Security-Engineering/Low_Random_Masking)



hundred thousand simulations. In such designs, the authors coupled two masked S-Boxes and used their shares to blind each other computations to fulfill the non-completeness. Each S-Box is decomposed to two quadratic functions F and G. The second-order leakage is detected when a probe is placed on an output of F of one of the coupled S-Boxes and the second probe on an output of G of the other S-Box.

**Midori-64.** Evaluating the 4-stage encryption/decryption design, we detected multi-variate second-order leakage by around 1 million simulations. Since the same fresh masks are used in all clock cycles of an encryption, by placing a probe on a part of the S-Box e.g., at the 4-th clock cycle, information about some fresh masks are obtained. When the second probe is placed on another part of the S-Box in the next clock cycle where the same fresh masks are used, we observe detectable leakage. This leakage has been detected at the 4th and 5th clock cycles, i.e., at the border of the first and second cipher rounds; hence, it is expected to be exploitable.

Note that it is a pipelined design, and 4 consecutive plaintexts can be given to the encryption function. After giving the target plaintext, the 3 other pipeline stages can be filled by zero or random inputs, i.e., bubble strategy. We examined both scenarios, and have seen the same leakage. Potential solutions might be (1) to set fresh masks to zero in clock cycles when pipeline does not contain meaningful data, or (2) to swap between 4 different fresh masks corresponding to 4 given consecutive plaintexts. None of such hints are given in the original paper [BDMS22], nor the implementations in GitHub consider/suggest such scenarios.

The 3-stage design has a univariate second-order leakage which is not originating from the aforementioned source. We detected such a leakage using around 1 million simulations when two probes are placed at different outputs of the G function starting at the third clock cycle. The S-Box is decomposed to quadratic functions F and G.

**PRINCE.** When evaluating the 6-stage design, we observed univariate second-order leakage using around 37 million simulations when two probes are placed at the input affine function of the decomposed S-Box in the second cipher round, i.e., at the 7-th clock cycle. The origin of this leakage is not the reuse of fresh masks. We observed the same leakage when the fresh masks change at every clock cycle. The leakage is indeed due to the way the fresh mask bits are reused by different S-Boxes in a round. Suppose that an S-Box module receives fresh masks  $\langle r_0, r_1, \dots, r_{37} \rangle$ . The same fresh masks in the same order are given to all other S-Boxes in a cipher round. Changing this order may avoid the observed leakage. The 4-stage design also exhibits second-order (but multi-variate) leakage using 30 million simulations. The reason is similar to what we have explained for the 4-stage Midori-64 design, i.e., two probes in consecutive clock cycles.

**SKINNY-64.** The 4-stage design shows univariate second-order leakage similar to the 6-stage PRINCE design. We detected the leakage by 1 million simulations when two probes are placed on input affine function of the decomposed S-Box in the second cipher round, i.e., 5-th clock cycle. We further detected multi-variate second-order leakage (similar to several other cases) when two probes are placed on consecutive clock cycles by obtaining information about the fresh mask in one clock cycle and revealing some leakage about the secrets by the second probe in the next clock cycle.

The 3-stage design has also univariate second-order leakage (detected using 1 million simulations), similar to the 4-stage design, but at the 4-th clock cycle, as the design has one less stage.

## 5.4 GHPC

There are a few recent developments in the areas of composable security, i.e., constructing hardware gadget (for e.g., a 2-input AND gate) whose security is guaranteed when composed.

This is highly beneficial to construct a masked circuit at arbitrary order. To the best of our knowledge, the most efficient composable 2-input AND gadget extendable to any arbitrary order is known as Hardware Private Circuits (HPC2) [CGLS21] following the security notion PINI [CS20]. Generic Hardware Private Circuits (GHPC) [KSM22] extends the HPC2 to construct arbitrary large gadgets (of any input and output size), but is limited to first-order.

Let us consider an exemplary circuit made by two GHPC gadgets realizing two functions cascaded, i.e.,  $d = f(g(a, b), c)$ . More precisely,

$$\langle t_0, t_1 \rangle = \text{GHPC-g}\left(\langle a_0, a_1 \rangle, \langle b_0, b_1 \rangle, r_0\right), \quad \langle d_0, d_1 \rangle = \text{GHPC-f}\left(\langle t_0, t_1 \rangle, \langle c_0, c_1 \rangle, r_1\right),$$

where  $a = a_0 \oplus a_1$  (resp. for  $b, c, d$ , and  $t$ ) and  $r_0, r_1$  denote the fresh masks.

Each GHPC gadget has two register stages, i.e., the latency of two clock cycles. When the input  $\langle a, b, c \rangle$  is given, it takes a couple of clock cycles till the output is ready, i.e.,  $\langle d_0, d_1 \rangle$ . When the input  $\langle c_0, c_1 \rangle$  is not synchronized with the intermediate value  $\langle t_0, t_1 \rangle$  by means of extra registers, the circuit does not form a pipeline, and the inputs  $a, b, c$  should stay stable till the output is ready. In this non-pipeline scenario, the authors suggested to remove some optional internal registers of the GHPC gadgets for the sake of area efficiency. This is based on an assumption that not only the given input but also the fresh masks stay stable until the entire circuit is evaluated. In other words, all inputs including and fresh masks are given to the circuit and stay stable and unchanged for a couple of clock cycles till the output is ready. The same concept has been considered in the design of non-pipeline DOM multipliers [GMK16]. This theoretically, does not pose any issue. The authors have confirmed the security of their designs by SILVER. Since SILVER evaluates the circuit in the steady state, removing such optional internal registers would not affect the evaluation result of SILVER, i.e., with and without such internal registers SILVER reports the security of the designs made by GHPC gadgets.

However, PROLEAD reports first-order leakage when transitions are taken into account. The second GHPC gadget initially calculates the given input  $c$  based on  $r_1$  and the output of the first gadget  $\langle t_0, t_1 \rangle$ , which is not yet ready. After two clock cycles, when the output of the first gadget is valid, the second gadget calculates its output with the same fresh mask  $r_1$ . The transitional leakage related to the consecutive values stored in the registers of the second gadget would cancel the effect of fresh masks, leading to first-order leakage. This unfortunately cannot be solved by updating the fresh masks at every clock cycle, since the circuit without optional internal registers would not generate the correct output at every clock cycle. More precisely, if the optional internal registers are removed, the fresh masks must stay stable until the circuit is fully evaluated. The only way to operate GHPC circuits securely and functionally correct is to keep the optional internal registers and update the fresh masks at every clock cycle. Note that such optional internal registers are independent of extra registers which may be added to the circuit (outside of gadgets) to synchronize the input of every gadget and construct a fully-pipeline design.

## 5.5 Summary

We have provided several case studies, in which the security of only small components (e.g., S-Boxes) have been analyzed, e.g., by SILVER. However, when such modules are plugged into larger designs, the security of the final constructions could only be evaluated by experimental analysis, which can naturally be erroneous. The authors of those designs have not seen the leakages which we observed by PROLEAD, but it does not mean that the same implementations evaluated by means of a different setup in another environment (with lower noise) show the same level of robustness. This indeed re-highlights the application of composable secure gadgets and the necessity of having proofs for the implementations instead of being dependent on manually-crafted optimized designs and experimental

**Table 2:** Evaluation results, using enough traces to detect effects of size  $\varphi \geq 0.1$ . For the non-secure designs, the required time reflects the duration needed to detect the largest effect.

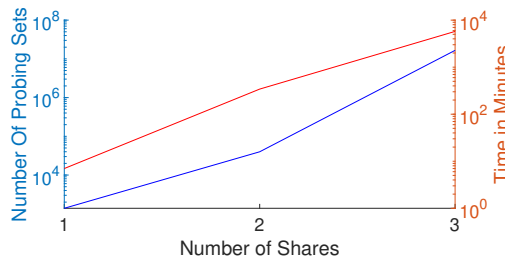
| Design                              | Ref.                  | Security   |   | Performance |          |
|-------------------------------------|-----------------------|------------|---|-------------|----------|
|                                     |                       | [expected] |   | [probes]    | [time]   |
| unmasked, SKINNY-64                 | [BJK <sup>+</sup> 16] | 0          | ✓ | 200         | 0.1 sec  |
| TI, PRESENT S-Box                   | [PMK <sup>+</sup> 11] | 1          | ✓ | 84          | 0.1 sec  |
| TI, PRESENT S-Box                   | [EGMP17]              | 1          | ✗ | 84          | 0.1 sec  |
| TI, AES S-Box                       | [MPL <sup>+</sup> 11] | 1          | ✓ | 1 359       | 3.6 min  |
| DOM, AES S-Box                      | [GMK16]               | 1          | ✓ | 1 386       | 1.5 min  |
| DOM, AES S-Box                      | [GMK16]               | 2          | ✓ | 39 873      | 2.4 hour |
| DOM, AES S-Box                      | [GMK16]               | 3          | ✓ | 16 583 602  | 3.7 week |
| CMS, AES S-Box                      | [CRB <sup>+</sup> 16] | 1          | ✓ | 1 530       | 3.5 min  |
| CMS, AES S-Box                      | [CRB <sup>+</sup> 16] | 2          | ✓ | 39 597      | 3.3 hour |
| TI, nibble-serial PRESENT-80        | [PMK <sup>+</sup> 11] | 1          | ✓ | 14 942      | 1.6 min  |
| TI, nibble-serial PRESENT-80        | [EGMP17]              | 1          | ✗ | 14 942      | 5.3 sec  |
| TI, byte-serial AES-128             | [MPL <sup>+</sup> 11] | 1          | ✓ | 43 855      | 50 min   |
| DOM, byte-serial AES-128            | [GMK16]               | 1          | ✓ | 28 420      | 13 min   |
| CMS, byte-serial AES-128            | [CRB <sup>+</sup> 16] | 1          | ✓ | 28 763      | 25 min   |
| Null Fresh, Midori-64               | [SM21a]               | 1          | ✓ | 3 855       | 8.5 min  |
| Null Fresh, PRESENT-80              | [SM21a]               | 1          | ✗ | 8 990       | 4.0 sec  |
| Null Fresh, PRINCE                  | [SM21a]               | 1          | ✗ | 4 736       | 18 sec   |
| Null Fresh, AES-128                 | [SM21a]               | 1          | ✗ | 12 958      | 13 sec   |
| Null Fresh 2, Keccak- $f$ [200]     | [SM21b]               | 1          | ✓ | 1 692       | 1.7 sec  |
| Null Fresh 2, Keccak- $f$ [200]     | [SM21b]               | 2          | ✓ | 39 185 460  | 5.8 day  |
| Null Fresh 2, Midori-64             | [SM21b]               | 2          | ✓ | 39 195 148  | 3.6 week |
| Null Fresh 2, PRINCE                | [SM21b]               | 2          | ✓ | 57 658 755  | 2.3 week |
| Null Fresh 2, SKINNY-65             | [SM21b]               | 2          | ✓ | 11 083 068  | 14 hour  |
| Null Fresh 2, PRESENT-80            | [SM21b]               | 1          | ✗ | 48 200      | 1.0 sec  |
| Null Fresh 2, PRESENT-80            | [SM21b]               | 2          | ✗ | 3 036 288   | 53 sec   |
| Low-Latency Keccak- $f$ [25]        | [ZSS <sup>+</sup> 21] | 1          | ✗ | 1 248       | 11 sec   |
| Low-Latency Keccak- $f$ [25]        | [ZSS <sup>+</sup> 21] | 2          | ✓ | 1 136 450   | 1.5 hour |
| Low-Rand, LED-128, 5-stage          | [BDMS22]              | 2          | ✓ | 13 597 965  | 8.1 day  |
| Low-Rand, LED-128, 3-stage          | [BDMS22]              | 2          | ✗ | 3 070 704   | 5.6 hour |
| Low-Rand, Midori-64, 4-stage        | [BDMS22]              | 2          | ✗ | 2 541 844   | 2.3 day  |
| Low-Rand, Midori-64, 3-stage        | [BDMS22]              | 2          | ✗ | 788 140     | 2.7 hour |
| Low-Rand, PRINCE, 6-stage           | [BDMS22]              | 2          | ✗ | 10 392 858  | 4.2 week |
| Low-Rand, PRINCE, 4-stage           | [BDMS22]              | 2          | ✗ | 18 110 620  | 3.4 week |
| Low-Rand, SKINNY-64, 4-stage        | [BDMS22]              | 2          | ✗ | 2 726 080   | 1.6 hour |
| Low-Rand, SKINNY-64, 3-stage        | [BDMS22]              | 2          | ✗ | 1 424 442   | 1.3 day  |
| GHPC gadgets, without optional regs | [KSM22]               | 1          | ✗ | 110         | 0.1 sec  |
| GHPC gadgets, with optional regs    | [KSM22]               | 1          | ✓ | 162         | 0.1 sec  |

evaluations. This might be highly relevant for the new activities and public call by NIST on “Masked Circuits for Block-ciphers”<sup>11</sup>.

We further would like to stress that the leakages which we found by PROLEAD in the aforementioned case studies are not necessarily exploitable. However, such designs cannot be considered probing secure, which is often in contradiction with the authors’ claims. A natural question is whether (robust) probing security is important in practice. It is true that (robust) probing model is relatively conservative, but it captures any leakage independent of how the actual circuit is realized in hardware and which timing information the cells of the underlying library have. In short, we believe that if a circuit is (robust) probing secure (guaranteed for example through composable gadgets), its physical realization is very likely secure in practice as long as the specifications of the circuit are not changed, e.g., the netlist stays unchanged. This statement stays valid even if the underlying ASIC library changes which just alters the power and timing characteristics of the instantiated gates.

## 5.6 Limitations

While PROLEAD can evaluate the probing security of larger designs whose probing security cannot be evaluated by formal verification tools, it turns out that some designs that satisfy higher-order probing security are even too large for a complete evaluation with PROLEAD. For example, consider the DOM, AES S-Box [GMK16] with different security orders evaluated during Section 5.



**Figure 3:** Evaluation time and the number of probing sets for a DOM protected AES S-Box with increasing security order.

In Figure 3, we see that the number of probing sets grows exponentially with the increasing number of shares leading to an exponentially increased runtime of PROLEAD. We can conclude from Table 2 that we can analyze a DOM AES S-Box up to the third order, while a fourth-order evaluation possibly takes months. Besides runtime, the memory requirements grow exponentially and may become the limiting factor on some devices. On the one hand, the number of contingency tables grows. On the other hand, each contingency table may grow exponentially as the number of probes in a probing set increases with the desired security order. Moreover, we remark that an increased number of probes per set and the resulting larger contingency tables only lead to statistically confident results if more simulations are considered. Hence, the number of required simulations to reliably detect an effect also grows with the desired security order. Nevertheless, we remark that a partial evaluation of larger designs is still possible as the user can limit the evaluation in terms of considered probes and clock cycles. Thus, although a complete evaluation is not feasible, PROLEAD can help the user to find potential vulnerabilities in a design.

<sup>11</sup><https://csrc.nist.gov/Projects/masked-circuits>

## 6 Conclusions

In this work, we introduced PROLEAD, a new simulation-based approach to evaluate the probing security of masked implementations. Although being dependent on simulations, in contrast to the state of the art, PROLEAD is free of any leakage/power model and directly examines the robust probing security of the given implementations. Thanks to gate-level simulations, bit slicing, and parallelisms, PROLEAD enjoys a high-performance feature being able to evaluate masked full cipher implementations in a reasonable time, e.g., first-order security of a masked AES-128 encryption function in a couple of hours. This is certainly out of the capacity of formal verification tools (like SILVER), which can at most evaluate subcircuits, e.g., gadgets or small S-Boxes. We should also mention that since PROLEAD is based on simulations and statistical hypothesis tests, its evaluation results cannot be considered as a proof when it reports the robustness of the given design. However, if PROLEAD detects a leakage, the found probing set can confidently be considered as a counterexample violating the desired security. Furthermore, PROLEAD can estimate the reliability of the results by reporting the false-negative probability. Users can adapt the required statistical confidence level to their needs while PROLEAD computes the minimum number of simulations that are needed to satisfy the required security level. Due to the estimation of confidence, it is always clear what a user can expect from a result given by PROLEAD and how reliable the results are. Naturally, the results of PROLEAD are more reliable when a higher number of simulations are considered in the evaluations. Hence, similar to any statistical hypothesis test, there is a trade-off between the confidence level and the number of samples involved in the evaluation. Nevertheless, we believe that PROLEAD is a highly helpful tool to rapidly examine the probing security of masked implementations prior to experimental analyses and/or fabrications. The tool can also be used by practitioners and engineers without having access to any SCA measurement setup. Through several cases studies, we have shown the ability of PROLEAD to find design flaws in implementations which are claimed robust probing secure. At the moment, PROLEAD supports glitch- and transition-extended probing security. A natural follow-up work would be in the direction of extending its features to cover security evaluations based on random probing model.

## Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and through the project 435264177 (SAUBER).

## References

- [Agr92] Alan Agresti. A Survey of Exact Inference for Contingency Tables. *Stat. Sci.*, 7(1):131 – 153, 1992.
- [AMM<sup>+</sup>06] Manfred Josef Aigner, Stefan Mangard, Francesco Menichelli, Renato Menicocci, Mauro Olivieri, Thomas Popp, Giuseppe Scotti, and Alessandro Trifiletti. Side channel analysis resistant design flow. In *ISCAS 2006*. IEEE, 2006.
- [BBC<sup>+</sup>19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS 2019*, volume 11735 of *LNCS*, pages 300–318. Springer, 2019.

- [BBD<sup>+</sup>15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 457–485. Springer, 2015.
- [BBD<sup>+</sup>16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *ACM CCS 2016*, pages 116–129. ACM, 2016.
- [BBI<sup>+</sup>15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015.
- [BBYS22] Ileana Buhan, Lejla Batina, Yuval Yarom, and Patrick Schaumont. SoK: Design Tools for Side-Channel-Aware Implementations. In *ASIA CCS 2022*, pages 756–770. ACM, 2022.
- [BCD<sup>+</sup>13] Georg T. Becker, Jim Cooper, Elizabeth K. DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, T. Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. Test Vector Leakage Assessment (TVLA) methodology in practice. 2013.
- [BCG<sup>+</sup>12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012.
- [BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
- [BDF<sup>+</sup>17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 535–566, 2017.
- [BDG<sup>+</sup>14] Shivam Bhasin, Jean-Luc Danger, Tarik Graba, Yves Mathieu, Daisuke Fujimoto, and Makoto Nagata. Physical Security Evaluation at an Early Design-Phase: A Side-Channel Aware Simulation Methodology. In *ES4CPS 2014*, page 13. ACM, 2014.
- [BDMS22] Tim Beyne, Siemen Dhooghe, Amir Moradi, and Aein Rezaei Shahmirzadi. Cryptanalysis of Efficient Masked Ciphers: Applications to Low Latency. *TCHES*, 2022(1):679–721, 2022.
- [BDPA] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak Reference.
- [BGG<sup>+</sup>14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the Cost of Lazy Engineering for Masked Software Implementations. In *CARDIS 2014*, volume 8968 of *LNCS*, pages 64–81. Springer, 2014.



- [BGI<sup>+</sup>18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In *EUROCRYPT 2018*, volume 10821 of *LNCS*, pages 321–353. Springer, 2018.
- [BJK<sup>+</sup>16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.
- [BK21] Nicolas Bordes and Pierre Karpman. Fast Verification of Masking Schemes in Characteristic Two. In *EUROCRYPT 2021*, volume 12697 of *LNCS*, pages 283–312. Springer, 2021.
- [BKL<sup>+</sup>07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.
- [BMRT22] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. IronMask: Versatile Verification of Masking Security. In *IEEE SP 2022*. IEEE, 2022.
- [CBG<sup>+</sup>17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does Coupling Affect the Security of Masked Implementations? In *COSADE 2017*, volume 10348 of *LNCS*, pages 1–18. Springer, 2017.
- [CGF21] Ana Covic, Fatemeh Ganji, and Domenic Forte. Circuit Masking: From Theory to Standardization, A Comprehensive Survey for Hardware Security Researchers and Practitioners. *CoRR 2021*, abs/2106.12714, 2021.
- [CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.
- [CGP<sup>+</sup>12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of Security Proofs from One Leakage Model to Another: A New Issue. In *COSADE 2012*, volume 7275 of *LNCS*, pages 69–81. Springer, 2012.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
- [Coh88] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1988.
- [Cor18] Jean-Sébastien Coron. Formal Verification of Side-Channel Countermeasures via Elementary Circuit Transformations. In *ACNS 2018*, volume 10892 of *LNCS*, pages 65–82. Springer, 2018.
- [CRB<sup>+</sup>16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with  $d+1$  Shares in Hardware. In *CHES 2016*, volume 9813 of *LNCS*, pages 194–212. Springer, 2016.

- [CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.
- [CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *TCHES*, 2021(2):136–158, 2021.
- [DBBL12] Nicolas Debande, Maël Berthier, Yves Bocktaels, and Thanh-Ha Le. Profiled Model Based Power Simulator for Side Channel Evaluation. *IACR Cryptol. ePrint Arch.*, page 703, 2012.
- [DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, 2014.
- [EGMP17] Maik Ender, Samaneh Ghandali, Amir Moradi, and Christof Paar. The First Thorough Side-Channel Hardware Trojan. In *ASIACRYPT 2017*, volume 10624 of *LNCS*, pages 755–780. Springer, 2017.
- [FGBR20] Muhammad Arsath K. F, Vinod Ganesan, Rahul Bodduna, and Chester Rebeiro. PARAM: A Microprocessor Hardened for Power Side-Channel Attack Resistance. In *HOST 2020*, pages 23–34. IEEE, 2020.
- [FGMDP<sup>+</sup>18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults and the Robust Probing Model. *TCHES*, 2018(3):89–120, 2018.
- [Fis22] R. A. Fisher. On the Interpretation of  $\chi^2$  from Contingency Tables, and the Calculation of P. *J. R. Stat. Soc.*, 85(1):87–94, 1922.
- [GBTP08] Benedikt Gierlich, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In *CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.
- [GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic Low-Latency Masking in Hardware. *TCHES*, 2018(2):1–21, 2018.
- [GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for sidechannel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.
- [GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptogr. Eng.*, 8(2):109–124, 2018.
- [GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS@CCS 2016*, page 3. ACM, 2016.
- [GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017*, volume 10159 of *LNCS*, pages 95–112. Springer, 2017.
- [GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001.

- [GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011.
- [GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In *CRYPTO 2014*, volume 8616 of *LNCS*, pages 444–461. Springer, 2014.
- [Hoe12] Jesse Hoey. The Two-Way Likelihood Ratio (G) Test and Comparison to Two-Way Chi Squared Test. *arXiv preprint arXiv:1206.4881*, 2012.
- [HPN<sup>+</sup>19] Miao Tony He, Jungmin Park, Adib Nahiyan, Apostol Vassilev, Yier Jin, and Mark Mohammad Tehranipoor. RTL-PSC: Automated Power Side-Channel Leakage Assessment at Register-Transfer Level. In *VTS 2019*, pages 1–6. IEEE, 2019.
- [HS13] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS 2013*, volume 8419 of *LNCS*, pages 219–235. Springer, 2013.
- [HSS12] Annelie Heuser, Werner Schindler, and Marc Stöttinger. Revealing side-channel issues of complex circuits by enhanced leakage models. In *DATE 2012*, pages 1179–1184. IEEE, 2012.
- [HSZ13] Sorin A. Huss, Marc Stöttinger, and Michael Zohner. AMASIVE: an adaptable and modular autonomous side-channel vulnerability evaluation framework. In *NTC 2013*, volume 8260 of *LNCS*, pages 151–165. Springer, 2013.
- [Inc] Synopsys Inc. Design Compiler Graphical.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
- [KLE<sup>+</sup>21] Pantea Kiaei, Zhenyuan Liu, Ramazan Kaan Eren, Yuan Yao, and Patrick Schaumont. Saidoyoki: Evaluating side-channel leakage in pre- and post-silicon setting. *IACR Cryptol. ePrint Arch.*, page 1235, 2021.
- [KMMS22] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated Generation of Masked Hardware. *TCHES*, 2022(1):589–629, 2022.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
- [KP07] Mario Kirschbaum and Thomas Popp. Evaluation of Power Estimation Methods Based on Logic Simulations. In *Austrochip 2007*, pages 45–51. TU Graz, 2007.
- [KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020*, volume 12491 of *LNCS*, pages 787–816. Springer, 2020.

- [KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits Towards Automated Generation of Composable Secure Gadgets. *TCHES*, 2022(1):323–344, 2022.
- [M<sup>+</sup>87] Nicholas Metropolis et al. The beginning of the Monte Carlo method. *LA Science*, 15(584):125–130, 1987.
- [Mea55] George H Mealy. A method for synthesizing sequential circuits. *Bell Syst. tech. j.* 1955, 34(5):1045–1079, 1955.
- [MKSM22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional Leakage in Theory and Practice – Unveiling Security Flaws in Masked Circuits. *TCHES*, 2022(2), 2022.
- [MM12] Amir Moradi and Oliver Mischke. How Far Should Theory Be from Practice? - Evaluation of a Countermeasure. In *CHES 2012*, volume 7428 of *LNCS*, pages 92–106. Springer, 2012.
- [MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *TCHES*, 2019:256–292, 2019.
- [MoD09] J.H. McDonald and University of Delaware. Sparky House Publishing, 2009.
- [MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
- [MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA 2005*, volume 3376 of *LNCS*, pages 351–365. Springer, 2005.
- [MPL<sup>+</sup>11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011.
- [MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In *CHES 2005*, volume 3659 of *LNCS*, pages 157–171. Springer, 2005.
- [MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage Detection with the x2-Test. *CHES 2018*, 2018(1):209–237, 2018.
- [Mus97] David R. Musser. Introspective Sorting and Selection Algorithms. *Softw. Pract. Exp.*, 27(8):983–993, 1997.
- [NPH<sup>+</sup>20] Adib Nahiyani, Jungmin Park, Miao Tony He, Yousef Iskander, Farimah Farahmandi, Domenic Forte, and Mark Mohammad Tehranipoor. SCRIPT: A CAD Framework for Power Side-channel Vulnerability Assessment Using Information Flow Tracking and Pattern Generation. *ACM TODAES 2020*, 25(3):26:1–26:27, 2020.
- [NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schl affer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptol.*, 24(2):292–321, 2011.

- [Pea92] Karl Pearson. *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to have Arisen from Random Sampling*. Springer, 1992.
- [PM05] Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In *CHES 2005*, volume 3659 of *LNCS*, pages 172–186. Springer, 2005.
- [PMK<sup>+</sup>11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptol.*, 24(2):322–345, 2011.
- [RBE<sup>+</sup>07] Francesco Regazzoni, Stéphane Badel, Thomas Eisenbarth, Johann Großschädl, Axel Poschmann, Zeynep Toprak Deniz, Marco Macchetti, Laura Pozzi, Christof Paar, Yusuf Leblebici, and Paolo Ienne. A Simulation-Based Methodology for Evaluating the DPA-Resistance of Cryptographic Functional Units with Application to CMOS and MCML Technologies. In *IC-SAMOS 2007*, pages 209–214. IEEE, 2007.
- [RBN<sup>+</sup>15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *LNCS*, pages 764–783. Springer, 2015.
- [RCS<sup>+</sup>09] Francesco Regazzoni, Alessandro Cevrero, François-Xavier Standaert, Stéphane Badel, Theo Kluter, Philip Brisk, Yusuf Leblebici, and Paolo Ienne. A Design Flow and Evaluation Framework for DPA-Resistant Instruction Set Extensions. In *CHES 2009*, volume 5747 of *LNCS*, pages 205–219. Springer, 2009.
- [Rep16] Oscar Reparaz. Detecting Flawed Masking Schemes with Leakage Detection Tests. In *FSE 2016*, volume 9783 of *LNCS*, pages 204–222. Springer, 2016.
- [SBY<sup>+</sup>18] Danilo Sijacic, Josep Balasch, Bohan Yang, Santosh Ghosh, and Ingrid Verbauwhede. Towards Efficient and Automated Side Channel Evaluations at Design Time. In *PROOFS 2018*, volume 7 of *Kalpa Publications in Computing*, pages 16–31. EasyChair, 2018.
- [Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.
- [SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In *CHES 2005*, volume 3659 of *LNCS*, pages 30–46. Springer, 2005.
- [SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer, 2015.
- [SM21a] Aein Rezaei Shahmirzadi and Amir Moradi. Re-Consolidating First-Order Masking Schemes Nullifying Fresh Randomness. *TCHES*, 2021(1):305–342, 2021.
- [SM21b] Aein Rezaei Shahmirzadi and Amir Moradi. Second-Order SCA Security with almost no Fresh Randomness. *TCHES*, 2021(3):708–755, 2021.
- [Sok95] Robert Sokal. *Biometry : the principles and practice of statistics in biological research*. W.H. Freeman, 1995.

- [SVRK19] Patanjali SLPSK, Prasanna Karthik Vairam, Chester Rebeiro, and V. Kamakoti. Karna: A Gate-Sizing based Security Aware EDA Flow for Improved Power Side-Channel Attack Protection. In *ICCAD 2019*, pages 1–8. ACM, 2019.
- [Tri03] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.*, page 236, 2003.
- [TV03] Kris Tiri and Ingrid Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In *CHES 2003*, volume 2779 of *LNCS*, pages 125–136. Springer, 2003.
- [TV04] Kris Tiri and Ingrid Verbauwhede. Place and Route for Secure Standard Cell Design. In *CARDIS 2004*, volume 153 of *IFIP*, pages 143–158. Kluwer/Springer, 2004.
- [Wol] Clifford Wolf. Yosys Open SYnthesis Suite.
- [Yat34] F. Yates. Contingency Tables Involving Small Numbers and the x2 Test. *J. R. Stat. Soc.*, 1(2):217–235, 1934.
- [YKES20] Yuan Yao, Tarun Kathuria, Baris Ege, and Patrick Schaumont. Architecture Correlation Analysis (ACA): Identifying the Source of Side-channel Leakage at Gate-level. In *HOST 2020*, pages 188–196. IEEE, 2020.
- [ZPTF21] Tao Zhang, Jungmin Park, Mark Mohammad Tehranipoor, and Farimah Farahmandi. PSC-TG: RTL Power Side-Channel Leakage Assessment with Test Pattern Generation. In *DAC 2021*, pages 709–714. IEEE, 2021.
- [ZSS+21] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziye Salarifard, and Amir Moradi. Low-Latency Keccak at any Arbitrary Order. *TCHES*, 2021(4):388–411, 2021.