Transitional Leakage in Theory and Practice

Unveiling Security Flaws in Masked Circuits

Nicolai Müller[1], David Knichel[1], Pascal Sasdrich[1] and Amir Moradi[2]

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

[2] University of Cologne, Institute for Computer Science, Cologne, Germany
firstname.lastname@uni-koeln.de

Abstract. Accelerated by the increased interconnection of highly accessible devices, the demand for effective and efficient protection of hardware designs against Side-Channel Analysis (SCA) is ever rising, causing its topical relevance to remain immense in both, academia and industry. Among a wide range of proposed countermeasures against SCA, *masking* is a highly promising candidate due to its sound foundations and well-understood security requirements. In addition, formal adversary models have been introduced, aiming to accurately capture real-world attack scenarios while remaining sufficiently simple to efficiently reason about the SCA resilience of designs. Here, the d-probing model is the most prominent and well-studied adversary model. Its extension, introduced as the *robust* d-probing model, covers physical defaults occurring in hardware implementations, particularly focusing on *combinational recombinations* (glitches), *memory recombinations* (transitions), and *routing recombinations* (coupling).

With increasing complexity of modern cryptographic designs and logic circuits, formal security verification becomes ever more cumbersome. This started to spark innovative research on automated verification frameworks. Unfortunately, these verification frameworks mostly focus on security verification of hardware circuits in the presence of glitches, but remain limited in identification and verification of transitional leakage. To this end, we extend SILVER, a recently proposed tool for formal security verification of masked logic circuits, to also detect and verify information leakage resulting from combinations of glitches *and* transitions. Based on extensive case studies, we further confirm the accuracy and practical relevance of our methodology when assessing and verifying information leakage in hardware implementations.

Keywords: Side-Channel Analysis · Transitional Leakage · Masking · Hardware

1 Introduction

Physical Attacks and Countermeasures. Even though *design* of secure cryptographic algorithms is a well-researched topic, secure *implementation* remains a late-breaking challenge. Principally, cryptographic algorithms are designed to withstand adversaries in the *black-box* model, limiting adversarial observations to inputs and outputs. However, practical implementations on physical devices usually entail side-channel information. Within the context of such a *gray-box* model, adversaries may observe any physical behavior and characteristics of an electronic device during execution of security-critical applications, such as temporal behavior [Koc96], instantaneous power consumption [KJJ99], Electromagnetic (EM) radiations [GMO01], or temperature and heat dissipation [HS13], in order to reveal secret and sensitive information.

Consequently, many protection mechanisms have been proposed, among which *masking* (based on the concepts of *secret-sharing*) prevails as most promising candidate, due to a formal and sound theoretical foundation [CJRR99]. Specifically in the context of hardware implementations, different masking schemes and flavors have been proposed over the years [ISW03, NRR06, RBN+15, GMK17, GM18], continuously improving efficiency and security. At the same time, not a few schemes have been shown to be insecure due to design flaws [MMSS19]. Nowadays, there is a growing awareness that design, implementation, and protection of cryptographic algorithms is still a (mostly) manual, delicate, and error-prone process requiring long-standing expertise and experience in hardware design and security. Moreover, this recognition propels an entirely new branch of research intending to consolidate security models and formal methods to assess and verify the security of hardware implementations.

Formal Verification of Hardware Security. Strong theoretical foundations, such as formal models for adversaries and physical execution environments, along with formal verification methods can support and accelerate design, implementation, and verification of secure cryptographic devices and applications. In connection with *hardware masking*, the simple and abstract Ishai, Sahai, and Wagner (ISW) d-probing adversary and security model [ISW03] usually provides a baseline verification model to reason about security in a *gray-box* context. In its basic appearance, the d-probing model allows an adversary to arbitrarily observe up to d intermediate values of an ideal circuit[1] during the processing of sensitive information. Along with some basic assumptions on noise, independence of inputs, and circuit encoding, the security of masked cryptographic implementations can be verified and proved under the d-probing model.

With respect to physical hardware and digital logic circuits, however, unconsidered and unintentional physical effects, such as *glitches* [MPG05, MS06], *transitions* [CGP+12, BGG+14], or *couplings* [CBG+17], and implementation defects due to architectural conditions, such as *parallelism* [BDF+17] or *pipelining* [CGD18], have been shown to be responsible for security degradation of theoretically secure designs and implementations. Most of all, *glitches* are long-known causes for leakage of sensitive information even in the presence of appropriate masking countermeasures [MPG05]. Adjusting the deficiencies in the original ISW d-probing model, a more robust extension of the standard model has been proposed recently [FGP+18], particularly allowing to incorporate unintentional physical defaults as part of the adversarial model.

Automated Tools for Formal Verification. Through the ever increasing integration of modern circuits and systems, complexity of hardware designs and implementations grows further. In a sense, this continuous progress and increasing complexity of hardware structures limits manual verification of masked circuits mostly to atomic parts and minor components only, often considered as masked *gadgets* in recent literature. To this end, the development of automated tools assisting designers in formal verification of masked circuits is a natural and long overdue step.

Set within the context of automated formal verification, `maskVerif` [BBD+15, BBD+16, BBC+19] is the first proposal addressing this direction of research. Originally, `maskVerif` was designed to verify d-probing security for ideal circuits in the presence of transitional leakage, but later versions of the tool have been extended to cover d-probing security and security notions for *composability*, i.e., Non-Interference (NI) [BBD+15] and Strong Non-Interference (SNI) [BBD+16], in the presence of glitches, however, still considering transitions for ideal circuits only. While `maskVerif` pursuits a language-based formal verification approach, Rebecca [BGI+18] and Coco [GHP+21] rely on Fourier coefficient

[1] In an ideal circuit, it is supposed that the gates have no delay to propagate the input changes to the output, hence no glitches.

estimation to automatically verify security of masked circuits. Even though Rebecca was the first tool to explicitly consider glitches as physical defaults, it is still limited to verification of d-probing security only, without supporting verification of additional composability notions (the same holds for Coco, extending Rebecca to verification of masked software implementations on CPUs). Only recently, SILVER [KSM20] has been presented, allowing to verify pure d-probing security and all recent security notions for composability (NI, SNI, and PINI [CS20]) in the presence of glitches while using an exact verification approach based on Reduced Ordered Binary Decision Diagrams (ROBDDs), avoiding false negatives (in contrast to `maskVerif` which relies on an optimistic approach).

(In-)complete Modeling of Physical Defaults. Comparing these existing approaches and tools for formal verification of masked hardware circuits, what becomes immediately apparent is the fact that all tools only consider glitches as undesired physical defaults while neglecting information leakage due to *transitions*. In a sense, even though transitional leakage is well-known to cause security degradation [CGP+12, BGG+14, CS21], only `maskVerif` implements a rudimentary verification under such conditions, however, focusing on ideal logic circuits only, hence, neglecting glitches as further source of unintentional security degradation. As a consequence, none of the existing tools supports a complete modeling of unintentional physical defaults occurring in hardware, i.e., glitches *and* transitions, for the security verification of masked circuits. Even worse, under certain circumstances, all tools may lead to false positives, i.e., reporting a hardware design secure in the presence of glitches, while the presence of transition leakage actually breaks the security of the masking scheme.

Our Contributions. In this work, we translate the transition-related findings of [FGP+18] into an algorithmic evaluation approach well suited for its integration into leakage verification tools. The result is a novel probe-extension procedure allowing us to model information leakage due to physical effects originating in combinational and memory recombinations, i.e., glitches and transitions. In a next step, we extend the state-of-the-art leakage verification framework SILVER to assess the security of masked hardware circuits considering glitches and transitions simultaneously. For this, we further extend the capabilities of SILVER to process and analyze iterative digital logic circuits, i.e., digital logic that processes data in a sequential fashion. Eventually, demonstrating the power of our extended model and verification framework as well as the practical relevance of accurately modeling glitches and transitions during security verification, we analyze different iterative 8-bit S-box constructions, proposed by Boss et al. [BGG+16], and report information leakage due to transitions, both in formal security verification as well as in experimental Side-Channel Analysis (SCA) evaluations. In a second set of case studies, we deal with iterative circuits, while Hardware Private Circuit (HPC) gadgets are integrated. In contrast to the original version of SILVER, we are able to evaluate such circuits made by Output-Probe-Isolating Non-Interference (O-PINI) secure gadgets as presented in [CS21]. Thanks to our extended version of SILVER, while confirming the issues reported and claims made in [CS21], we provide other insights through theoretical and experimental analyses when HPC gadgets are processed iteratively.

2 Background

2.1 Notations

We denote Boolean variables $x \in \mathbb{F}_2$ by lower case letters and vectors of multiple Boolean variables $X \in \mathbb{F}_2^n$ by upper case letters. To denote single elements of a vector $X \in \mathbb{F}_2^n$ we use subscripts. Hence, $x_i \in F_2$ denotes the element at position i of X. Moreover, we use

subscripts and curly brackets to denote a primary input at a specific point in time. For example, $x_{\{i\}}$ is forwarded to a circuit in clock cycle i. Further, we denote shares of a variable with superscripts. Hence, $X^i \in \mathbb{F}_2^n$ denotes share i of the unshared variable X. To formalize the probing model, we use upper case bold letters to denote a set of probes, e.g., $\mathbf{P}$ while we apply lower case bold letters to denote a single probe $\mathbf{p} \in \mathbf{P}$. Again, we denote the i^{th} probe $\mathbf{p_i} \in \mathbf{P}$ of a set with subscripts. For Boolean functions f and vectorial Boolean functions F we use sans-serif fonts.

2.2 d-Probing Model

In the standard d-probing model, originally introduced in [ISW03], an adversary is granted the ability to probe up to d wires in an ideal circuit. This means, every probe is instantaneous and independent of all other probes while providing access to the exact, noise-free, and stable signal of a wire upon invocation of the circuit, i.e., for specific input assignments and states of memory elements. Following this model, a circuit provides *d^{th}-order probing security*, if and only if an adversary is not able to learn anything about the secret, i.e., the joint distribution over observations made by up to d probes is independent of the distribution of any secret processed by the circuit.

By introducing the *robust* probing model in [FGP+18], Faust et al. extended the standard probing model in order to sufficiently capture information leakage originating from physical defaults occurring in physical logic circuits and hardware implementations. More precisely, the work covers *combinational recombinations* (glitches), *memory recombinations* (transitions), and *routing recombinations* (couplings), aiming to accurately cover all phenomenons relevant to side-channel leakage in hardware. For each considered physical default, a probe extension was introduced to increase the number of observed values within a circuit according to the considered physical default.

Glitch-Extended Probes. *Glitches* are unintentional and undesired signal transitions due to different delay paths and switching delays within a combinational circuit. In order to model such a behavior within the framework of the robust d-probing adversarial model, any glitch-extended probe allows an adversary to learn all stable inputs (and hence all possible combinations) flowing into the computation of the probed wire. As such, assuming a worst-case scenario, a single glitch-extended probe on a combinational circuit is then substituted by all relevant (standard) probes on the outputs of the last register stage or on primary inputs.

Transition-Extended Probes. *Transitions* are unintentional and undesired recombinations of memory contents due to consecutive invocations (e.g., clock cycles) of a circuit. In order to model this behavior within the context of the robust d-probing adversarial model, each *transition-extended* probe, placed on a memory element, is substituted by two (standard) probes, one on the memory input and one on the memory output, effectively doubling the number of adversarial probes.

Coupling-Extended Probes. *Couplings* are unintentional and undesired recombinations of values carried on neighboring wires. In order to model such a behavior in the robust d-probing adversarial model, each *coupling-extended probe* is replaced by the set of (standard) probes observing the set of neighboring wires.

(g, t, c)-Extended Probes. We apply the notation proposed in [FGP+18], denoting a (g, t, c)-extended probe as a glitch-extended (if $g = 1$), transition-extended (if $t = 1$), and coupling-extended (if $c = 1$) probe within the robust d-probing adversarial model. More specifically, throughout the given work, we will focus on $(g, t, 0)$-extended probes, as we intend to analyze circuits on a gate-level for which layout and routing information usually is not available and coupling cannot be modeled with adequate precision.

2.3 Boolean Masking

Following the approach of *secret sharing*, Boolean masking splits a secret $X \in \mathbb{F}_2^n$ into s independently and uniformly distributed shares X^i, $0 \leq i < s$, such that the sum over all these shares equals X, i.e., $X = \bigoplus_{i=0}^{s-1} X^i$. Known as uniform sharing, this is commonly achieved by drawing X^i uniformly and at random from $\mathbb{F}_2^n$ for all $0 \leq i < s-1$ and computing the remaining share as $X^{s-1} = X \oplus \bigoplus_{i=0}^{s-2} X^i$, transforming the unshared secret $X \in \mathbb{F}_2^n$ into its shared representation $X' = \left(X^i\right)_{0 \leq i < s} \in \mathbb{F}_2^{n \times s}$. Naturally, this approach requires splitting any secret in at least $d+1$ shares to achieve security against an adversary that is able to probe up to d wires in a circuit, i.e., to achieve d-probing security.

2.4 Threshold Implementation

Threshold Implementations (TIs) have been introduced and developed as a way to realize masked circuits and to avoid leakage in hardware implementations caused due to glitches [Bil15, CBR+15, BGN+14, NRR06, NRS08]. The number of input (s_{in}) and output sharings (s_{out}) necessary to realize a d^{th}-order TI of a given Boolean function $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ with algebraic degree t is restricted by $s_{in} \geq t \times d + 1$ and $s_{out} \geq \binom{s_{in}}{t}$ where $\mathsf{F}_{\mathsf{TI}} : \mathbb{F}_2^{n \times s_{in}} \mapsto \mathbb{F}_2^{m \times s_{out}}$ denotes the resulting TI which can be denoted as a set of s_{out} component functions $\mathsf{F}_{\mathsf{TI}}^{i \leq s_{out}} : \mathbb{F}_2^{n \times s_{in}} \mapsto \mathbb{F}_2^m$. A secure d^{th}-order TI further should fulfill the following properties:

Correctness. Let $X' \in \mathbb{F}_2^{n \times s_{in}}$ be a valid Boolean masking of $X \in \mathbb{F}_2^n$. Then $Y' = \mathsf{F}_{\mathsf{TI}}(X') \in \mathbb{F}_2^{m \times s_{out}}$ is a valid Boolean masking of $Y = \mathsf{F}(X) \in \mathbb{F}_2^m$.

Non-Completeness. Any combination of up to d component functions $\mathsf{F}_{\mathsf{TI}}^{\mathsf{i}}$ of F_{TI} must be independent of at least one input share for every secret. Intuitively, when solitary considering a circuit fulfilling non-completeness, an adversary in the d-probing model will not learn anything about a secret X, as non-completeness guarantees that any observation made by up to d probes will be independent of at least one circuit share and due to the uniformity property of Boolean masking, these circuit shares will hence be independent of the secret.

Uniformity. When grouped corresponding to their unshared input value X, each valid output sharing Y' of $Y = \mathsf{F}(X)$ within this group has to occur p times, where p is constant over all X and Y', while each invalid sharing does not occur at all. This guarantees that Y' always is a uniform sharing of Y when given as input to a subsequent circuit.

2.5 Trivially Composable Gadgets

Finding secure, masked circuits for high security orders and large functions has proven to be a hard task. For this, a wide variety of composability notions have been established, aiming to define properties of masked circuits that are sufficient to guarantee security in the d-probing model when combined to a larger design. This enables the construction of small sub-circuits (typically realizing simple two-input gates for AND and XOR) that lead to d-probing secure circuits when combined and interconnected. In recent literature, these composable, atomic sub-circuits are commonly referred to as *gadgets*.

In the course of this direction of research, Non-Interference (NI) and Strong Non-Interference (SNI) [BBD+15, BBD+16] were introduced first, where SNI remedied composability flaws discovered for NI by further restricting *propagation* [CS20] of probes placed on output wires of a gadget. As the scope of SNI was initially limited to single-output gadgets, Cassiers et al. [CS20] extended this notion to capture multiple-output gadgets as well, but

in the same work, they introduced the notion of Probe-Isolating Non-Interference (PINI) as a way to further reduce overhead requirements of composable gadgets and allowing trivial implementation of linear functions.

Probe-Isolating Non-Interference. Following the principles of Domain Oriented Masking (DOM) [GMK17], share domains were introduced for PINI [CS20] and output probes are restricted to propagate within their own share domain while internal probes are limited to propagate into a single, but arbitrary, domain. This allows trivial implementation of linear functions, i.e., their share-wise application, while enabling trivial composition of gadgets fulfilling the notion of PINI by simply interconnecting different input- and output-shares with respect to their share domain, i.e., not mixing share domains through interconnections.

Output-Probe-Isolating Non-Interferences. Recently, in order to model and achieve trivial composition under transitions, the notion of O-PINI was introduced in [CS21], as an extension to the original definition of PINI, aiming to solve issues that may occur when there exist a feedback loop in the design, e.g., one input of a gadget depends on an output of itself. In order to model this, at first, it is determined in which input share domains the original probes propagate, before output probes are added (if they do not already exist) to all these domains. The gadgets must then be PINI with respect to the new set of probes in its original definition. This extension is motivated by the fact that adding an internal probe placed on a gadget allows an additional – but arbitrary – domain to be in the set of input shares resulting from propagation. Now, if there is a feedback loop in the design and an output of the gadget is input to itself, these input shares will propagate into the previous iteration, which can be effectively modeled by putting another probe on the corresponding output domain, possibly reducing the security order. For example, as presented in [CS21], Figure 1 visualizes why HPC2 is not O-PINI when initialized for the second security order. For three input shares, we consider $\mathbf{p_0}$ and $\mathbf{p_1}$ placed on the gadget internals $\overline{x^1}\, r_{01}$ propagating into share domain 1 (due to x^1) and $\overline{x^2}\, r_{21}$ propagating into share domain 2 (due to x^2). Since there exists a feedback loop from its outputs to the gadget's inputs, $\mathbf{p_0}$ and $\mathbf{p_1}$ would further propagate into the outputs belonging to share domain 1 and 2, which is similar to placing additional output probes $\mathbf{p_6}$ and $\mathbf{p_7}$. Now, given $\{\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_6}, \mathbf{p_7}\}$, i.e., two output probes placed on domain 1 and domain 2 and two internal probes, PINI only guarantees that in this case the propagation is limited to input share domain 1 and 2 (due to $\mathbf{p_6}$ and $\mathbf{p_7}$) and at most two other, domains (due to the internal probes) which may potentially include share domain 0. In summary, $\{\mathbf{p_0}, \mathbf{p_1}\}$ may now propagate into three share domains 0, 1, and 2, rendering the design insecure.

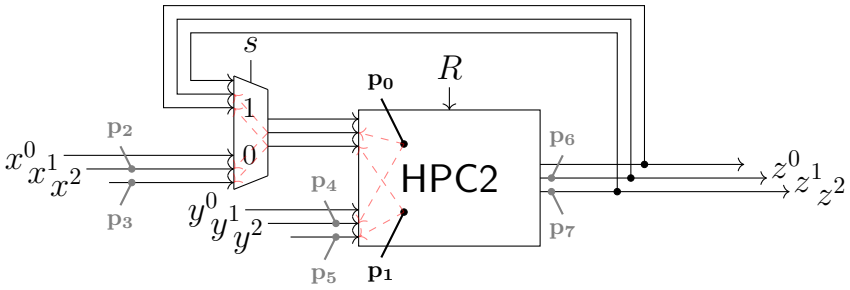


Figure 1: Iterative HPC2 gadget.

By adding output probes corresponding to the input share domains resulting from propagation and still guaranteeing that the original number of input domains, in which the new set of probes propagates into, does not increase, O-PINI enables trivial composition of iterative designs in the presence of transitions.

HPC Gadgets. In [CGLS21], Cassiers et al. introduced novel gadget constructions called Hardware Private Circuits (HPCs). This included HPC2, a gadget realizing a 2-input AND gate and offering trivial composability under the notion of PINI in the glitch-extended probing model for arbitrary security orders. This allows trivial construction of secure circuits by simply substituting AND gates with its shared representation, i.e., HPC gadgets. The security and composability guarantees provided by these gadgets are in conformity with the notion of PINI in the presence of only glitches, excluding transitions. As a refined notion which is able to handle transitions, O-PINI was introduced in [CS21], and at the same time HPC2 was further adapted to be in conformity with it an hence be trivially composable under transitions combined with glitches.

2.6 Formal Verification

In order to verify the security and composability of a design in the d-probing model, a wide variety of tools have been proposed which differ in the their abstraction and accuracy levels [BGI+18, BBC+19, KSM20, BDM+20, BBD+16, BGR18, BBD+15, CGLS21]. However, all these tools offer a valuable assistance in finding provable secure masking schemes and enable a designer to catch flaws in an early stage of the design process.

SILVER. In [KSM20], Knichel et al. introduced SILVER, a verification tool that utilizes a methodology based on Binary Decision Diagrams (BDDs) for checking statistical independence between observations made by (glitch-extended) probes and secret values or set of shares, effectively enabling a verification of all common security and composability notions (including PINI) in the standard and glitch-extended robust d-probing model. As SILVER currently is the only publicly-available[2] tool supporting the notion of PINI and is free of false negatives, we opted to integrate our results into its verification framework. Fortunately, following the concepts outlined Section 2.2, extending the probes with respect to physical defaults is a simple pre-processing step, possibly increasing the number of (standard) probes which have to be considered. This means, the core of SILVER, i.e., the check for statistical independence utilizing BDDs, remains unchanged regardless of any probe extension (which is already demonstrated for glitch-extended probes in the current version of the tool).

Limitations. As SILVER constructs BDDs for representing Boolean functions, the verification run time mainly depends on the complexity of the underlying Boolean functions, in particular, the number of product terms in the disjunctive normal form of the circuit [KSM20]. Therefore, SILVER is applicable to small circuits like gadgets or single S-Boxes, but not to a full cipher design. Moreover, the complexity of constructing all relevant probe combinations grows exponentially with the security order, and each combination itself encompasses a larger set of random variables. Consequently, higher-order leakage verification with SILVER is only possible for small circuits. Further, due to its nature, the construction of BDDs for a circuit containing a loop is not possible. Hence, SILVER is not able to handle such circuits.

3 Transitional Leakage

As discussed in Section 2.2, the (g, t, c)-robust d-probing security model formalizes models for capturing three different type of physical defaults (glitches, transitions, and couplings). In particular, modeling of any considered default [FGP+18] or even a combination of multiple defaults relies on extending a d-set of probes on intermediate values according to the corresponding extension scheme.

[2] https://github.com/Chair-for-Security-Engineering/SILVER

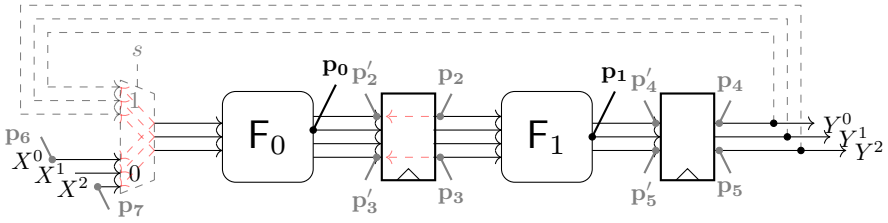


Figure 2: Iterative design of F with static primary inputs.

Currently, verifications are mostly performed for the widely used specification of $(1, 0, 0)$-robust d-probing, i.e., a d-set, generated by glitch-extended probes, is used to take combinational recombinations into account. However, to further allow the verification given the combined occurrence of glitches and transitions, we present an automated procedure for the combination of glitch-extended and transition-extended probes. For this, the final set of probes can then be used for verification within the $(1, 1, 0)$-robust d-probing model as part of security verification tools such as SILVER [KSM20].

For the sake of clarity, we start this section by presenting two examples and explain how the expansion of probes is performed when (1) the primary inputs are stable and (2) they change during the evaluation of the circuit. As discussed in [CS21], leakages due to transitions often lead to problems if the same physical instance of a function F is evaluated multiple times by different but not necessarily independent inputs. This holds for the evaluation of an *iterative circuit* which is the underlying architecture of our experiments. For this, we first define the structure of an iterative circuit.

Definition 1 (Iterative Circuit)**.** An *iterative circuit* implements an *iteration function* $\mathsf{F} : \mathbb{F}_2^n \to \mathbb{F}_2^m, U \mapsto Y$ driven by a multiplexer $\mathsf{MUX} : \mathbb{F}_2^n \times \mathbb{F}_2^n \times \mathbb{F}_2 \to \mathbb{F}_2^n, (X, V, s) \mapsto U$ as

$$U = \begin{cases} X & \text{if } s = 0, \\ V & \text{otherwise} \end{cases},$$

while X denotes the primary input and Y the primary output. Further, $V \subseteq Y$ stands for the feedback meaning that it is taken from the output of F. One *execution* of the circuit means iterating F k times, for a constant $k \geq 2$. Depending on s, the MUX selects either the primary input X or the feedback V to be given to F. In a typical case, X is selected for the first iteration while a part of the output of F is given to its input for the following iterations.

Given this definition, we consider an iterative circuit for the examples given below, which realize an exemplary function $\mathsf{F} = \mathsf{F}_1 \circ \mathsf{F}_0$ using two combinational circuits F_0 and F_1 whose outputs are directly stored in registers. Hence, each iteration of F takes two clock cycles. Further, the circuit receives three shares of an n-bit primary input (X^0, X^1, X^2), $X^i \in \mathbb{F}_2^n$, while the shared output of the function is also fed back as the new input, selected by a multiplexer. After k iterations, the circuit output is taken as the (shared) result of such an execution (Y^0, Y^1, Y^2), $Y^i \in \mathbb{F}_2^n$. Further, in order to ensure the correctness of the operation, the select signal of the multiplexer $s = 0$ if $k = 0$ (first iteration) while $s = 1$ for any subsequent iteration $k \neq 0$. Figure 2 and Figure 3 depict such an iterative circuit.

Example 1 (Static Primary Inputs)**.** For the first example, we suppose that all primary inputs remain stable throughout the entire execution. In this sense, the primary inputs do not change during each of the k iterations. Now, let us consider a set of two probes $\mathbf{P} = \{\mathbf{p_0}, \mathbf{p_1}\}$ placed on an output of F_0 and F_1 respectively, as shown in Figure 2. In the following, we intend to extend the set $\mathbf{P}$ to cover both glitches *and* transitions.

Step 1: Glitch-Extension. We start to model glitches for all $\mathbf{p} \in \mathbf{P}$ by adding the relevant probes, according to Section 2.2, into $\mathbf{P}_g$, i.e., the set of probes after glitch-extension. As $\mathbf{p_0}$ (resp. $\mathbf{p_1}$) is placed on one output share of F_0 (resp. F_1), we need to add probes on all inputs of the combinational logic that contribute to the computation of $\mathbf{p_0}$ (resp. $\mathbf{p_1}$). As an exemplary case, we assume the following extensions.

$$\mathbf{p_1} \xrightarrow{(1,0,0)} \{\mathbf{p_2}, \mathbf{p_3}\}, \qquad \mathbf{p_0} \xrightarrow{(1,0,0)} \{\mathbf{p_4}, \mathbf{p_5}, \mathbf{p_6}, \mathbf{p_7}\}, \qquad \mathbf{P}_g = \{\mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4}, \mathbf{p_5}, \mathbf{p_6}, \mathbf{p_7}\}$$

Supposing that $\mathbf{p_0}$ is extended to two input shares of F_0, we need to examine all corresponding inputs of the multiplexer. More precisely, if s switches from 0 to 1, the multiplexer output replaces the primary inputs with the feedback signals. Hence, any glitches, propagated to the multiplexer outputs, depend on the primary inputs as well as the feedback signals, i.e., the outputs of the previously accomplished iteration. We model this effect by adding probes on the corresponding feedback signals to $\mathbf{P}_g$, i.e., adding $\mathbf{p_4}$ and $\mathbf{p_5}$, and on the corresponding primary inputs, i.e., $\mathbf{p_6}$ and $\mathbf{p_7}$.

Step 2: Transition-Extension. Second, we now extend the probes in $\mathbf{P}_g$ to take transitions into account. Note, that it is also possible to first extend the probes based on transitions and then continue with the extension scheme for glitches. This eventually will result in the same d-set of probes that can be evaluated for statistical independence.

In a sense, according to [FGP+18], transitions combine two values that are consecutively seen on the same register. Modeling the overwriting effect for registers allows the extension of a probe $\mathbf{p} \in \mathbf{P}_g$ placed on a register output by another probe $\mathbf{p}'$ on the corresponding register input. Consequently, our approach analyzes $\mathbf{P}_g$ and extends all contained probes after the glitch-extension that are already at the register outputs (or primary inputs). For instance, as $\mathbf{p_2}$ (resp. $\mathbf{p_3}$) is placed on an output of the first register stage, it holds that:

$$\mathbf{p_2} \xrightarrow{(0,1,0)} \{\mathbf{p_2}, \mathbf{p_2'}\}, \qquad \mathbf{p_3} \xrightarrow{(0,1,0)} \{\mathbf{p_3}, \mathbf{p_3'}\}$$

The same extensions must be applied on $\mathbf{p_4}$ and $\mathbf{p_5}$ placed on outputs of the second register stage. Hence, it holds that:

$$\mathbf{p_4} \xrightarrow{(0,1,0)} \{\mathbf{p_4}, \mathbf{p_4'}\}, \qquad \mathbf{p_5} \xrightarrow{(0,1,0)} \{\mathbf{p_5}, \mathbf{p_5'}\},$$
$$\mathbf{P}_{g,t} = \{\mathbf{p_2}, \mathbf{p_2'}, \mathbf{p_3}, \mathbf{p_3'}, \mathbf{p_4}, \mathbf{p_4'}, \mathbf{p_5}, \mathbf{p_5'}, \mathbf{p_6}, \mathbf{p_7}\}.$$

As a consequence, $\{\mathbf{p_2'}, \mathbf{p_3'}, \mathbf{p_4'}, \mathbf{p_5'}\}$ denote the transition-extension probes on the register inputs. Note that all register inputs are stable when stored. Hence, no further glitch-extension on $\{\mathbf{p_2'}, \mathbf{p_3'}, \mathbf{p_4'}, \mathbf{p_5'}\}$ is necessary and this step of our algorithm terminates here. Ultimately, $\mathbf{P}_{g,t}$ now covers glitches and transitions, and its statistical independence to the secrets (i.e., the unmasked primary input $X = (X^0 \oplus X^1 \oplus X^2)$ can be examined implying the evaluation of two probes $\{\mathbf{p_0}, \mathbf{p_1}\}$ under a $(1, 1, 0)$-robust 2-probing model.

3.1 Changes on Primary Inputs

In the previous example, we mainly followed the concepts and assumptions provided in [CS21], particularly assuming that all primary inputs remain stable and static for the entire execution of the circuit through multiple iterations. In one of the case studies, which we present in Section 5, we demonstrate that such an assumption in fact can be the source of security flaws. For this, in the following example, we allow transitions on primary inputs such that each primary input can change at every clock cycle.

Example 2 (Dynamic Primary Inputs)**.** In order to deal with inputs being changed at every clock cycle, we introduce the notion of *input sequences*, denoting input values

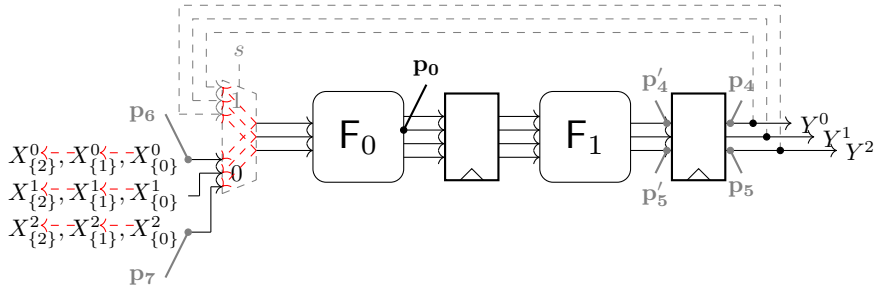


Figure 3: Iterative design of F with transitional primary inputs.

given per clock cycle. Hence, with such an implicit notion of time, we define the primary input X at cycle l as $X_{\{l\}}$. We further consider the same iterative circuit as shown in Example 1, but, for the sake of simplicity, we limit the set of probes to a single probe $\mathbf{P} = \{\mathbf{p_0}\}$ as visualized in Figure 3. Again, we start with extending the probes based on glitches. Following the same procedure as in Example 1, $\mathbf{p_0}$ is extended such that $\mathbf{p_0} \xrightarrow{(1,0,0)} \{\mathbf{p_4}, \mathbf{p_5}, \mathbf{p_6}, \mathbf{p_7}\}$. Now, the fundamental difference to Example 1 is that we have different input values consecutively given in every clock cycle. As a consequence, transitions not only occur between input and output of registers, but may also occur due to the consecutive primary input values. Hence, modeling the transitions by extending the probes to cover registers, i.e., $\{\mathbf{p'_4}, \mathbf{p'_5}\}$, is not sufficient. For this, let us examine the probes on the primary inputs $\{\mathbf{p_6}, \mathbf{p_7}\}$. Generally, if we are in clock cycle l, probe $\mathbf{p_6}$ observes a bit of $X^0_{\{l\}}$ and $\mathbf{p_7}$ a bit of $X^2_{\{l\}}$. Obviously, such an extension does not consider the fact that input values in cycle l have been replaced by the associated values of cycle $l+1$, i.e., the corresponding bits of $X^0_{\{l+1\}}$ and $X^2_{\{l+1\}}$. Therefore, we need to include two additional standard probes. Since the feedback signals and primary inputs have two clock cycles distance (see Figure 3), if we are in the second clock cycle ($l = 1$), the first iteration of the circuit $k = 1$ is accomplished, the feedback signals traverse the output of F_1 being stored in the register, and the select signal of the multiplexer switches. This means that $\mathbf{p_4}$ and $\mathbf{p_5}$ observe the feedback value, and $\mathbf{p_6}$ and $\mathbf{p_7}$ the corresponding bits of $X^0_{\{1\}}$ and $X^2_{\{1\}}$. Due to the primary input transitions, $\mathbf{p_6}$ and $\mathbf{p_7}$ are also extended to $\mathbf{p'_6}$ and $\mathbf{p'_7}$ observing the corresponding bits of $X^0_{\{2\}}$ and $X^2_{\{2\}}$. As a consequence, the resulting extensions can be modeled as:

$$\mathbf{p_4} \xrightarrow{(0,1,0)} \{\mathbf{p_4}, \mathbf{p'_4}\}, \quad \mathbf{p_5} \xrightarrow{(0,1,0)} \{\mathbf{p_5}, \mathbf{p'_5}\}, \quad \mathbf{p_6} \xrightarrow{(0,1,0)} \{\mathbf{p_6}, \mathbf{p'_6}\}, \quad \mathbf{p_7} \xrightarrow{(0,1,0)} \{\mathbf{p_7}, \mathbf{p'_7}\}$$
$$\mathbf{P}_{g,t\{2\}} = \{\mathbf{p_4}, \mathbf{p'_4}, \mathbf{p_5}, \mathbf{p'_5}, \mathbf{p_6}, \mathbf{p'_6}, \mathbf{p_7}, \mathbf{p'_7}\}.$$

Note that in the former cycles $l < 1$, the feedback signal does not yet carry the output of F_1 depending on the given primary inputs. Hence, the extension of the probes is slightly different. For example, in the first clock cycle ($l = 0$), $\mathbf{p_0}$ extends to

$$\mathbf{p_0} \xrightarrow{(1,1,0)} \mathbf{P}_{g,t\{1\}} = \{\mathbf{p_6}, \mathbf{p'_6}, \mathbf{p_7}, \mathbf{p'_7}\},$$

and the following primary inputs are probed:

$$\mathbf{p_6} \xleftarrow{\text{probe}} X^0_{\{0\}}, \quad \mathbf{p'_6} \xleftarrow{\text{probe}} X^0_{\{1\}}, \quad \mathbf{p_7} \xleftarrow{\text{probe}} X^2_{\{0\}}, \quad \mathbf{p'_7} \xleftarrow{\text{probe}} X^2_{\{1\}}$$

3.2 Modeling Glitch- and Transition-Extended Probes

After giving a first intuition on the modeling and generation of $(1, 1, 0)$-extended probes in the d-probing model using the two above examples, we now formally define the algorithmic

solution for the identified problems and challenges. In addition, we show how to integrate these algorithmic concepts into state-of-the-art formal verification tools, such as SILVER[3].

Circuit Model. We model a circuit as *Directed Acyclic Graph (DAG)* that represents all gates as nodes and all wires as edges. Hence, as the first step, such a graph should be made for the circuit under evaluation. For circuits without loops, this can be trivially done, knowing the netlist of the circuit.[4] However, as shown above, we deal with iterative circuits, which contain loops. To handle this, and extract a loop-free graph[5], we remove all loops and store a list of transitions $\mathcal{F}$ indicating the feedback signals. Each list entry $f \in \mathcal{F}$ formalizes a single feedback loop together with its corresponding input multiplexer. We denote each $f \in \mathcal{F}$ as a triple $f = (i, j, l)$, where $i \in \mathcal{I}$ denotes a primary input overwritten by a feedback signal j (i.e. both inputs of the multiplexer) during clock cycle l. With $\mathcal{F}$, we keep all information regarding loops while structurally removing them from the circuit. $\mathcal{F}$ itself is created by analyzing all initial multiplexers $\mathsf{MUX}(i, j, s)$ that select (depending on the selection signal s) if a primary input i or a feedback signal j is given to the circuit. The corresponding clock cycle can then be found by analyzing the latency of the feedback signal. As j is computed during one iteration, l corresponds to the number of clock cycles per iteration. For example, if we consider Figure 2, we analyze and remove three multiplexers $\mathsf{MUX}_0(X^0, Y^0, s)$, $\mathsf{MUX}_1(X^1, Y^1, s)$, and $\mathsf{MUX}_2(X^2, Y^2, s)$. The list of transitions is $\mathcal{F} = \{(X^0, Y^0, 2), (X^1, Y^1, 2), (X^2, Y^2, 2)\}$ and $\mathcal{I} = \{X^0, X^1, X^2\}$ which are given to F_1. Based on the given circuit model, we formalize the glitch and transition extension scheme as shown in Algorithm 1. For better understanding, we define the following functions:

`gate`(w):	*returns the source gate of wire* w.
`type`(g):	*returns* `reg` *if gate* g *is a register.*
`inputs`(g):	*returns a set containing all input wires of gate* g.
`dfs`(w):	*performs a depth-first search starting at wire* w *but stops going deeper if a found wire* w' *carries a synchronized element (either register output or primary input). It returns all found* w'.

Algorithm 1 receives a d-set of probes together with a manually created list of all primary inputs $\mathcal{I}$ and a list of all transitions based on feedback signals $\mathcal{F}$ for all evaluated points in time. The output of Algorithm 1 is the set of probes after glitch and transition extension.

Modeling Glitches. For any $\mathbf{p} \in \mathbf{P}$, we model glitches by placing probes on all synchronized elements (either register outputs or primary inputs) that contribute to the computation of the value that $\mathbf{p}$ observes. We realize the extension by a depth-first search starting at $\mathbf{p}$ as the root node. The search itself is performed backwards and stops evaluating a branch if a synchronized element is found. Only for the synchronized element, we add a probe on its output wire. This can be seen in Lines 6-8 of Algorithm 1. Since the circuit became loop-free, for any probe placed on an primary input, we check if the primary input is input of an initial multiplexer during the evaluated clock cycle. This is done by searching the tuple with the primary input and the evaluation cycle in $\mathcal{F}$. If the tuple exists, we place probes on both multiplexer inputs. Note that the glitches related to both primary input and feedback occur during an individual clock cycle and on a specific input given during this clock cycle. Hence, we create one set of probes per clock cycle containing its individual probes. This can be seen in Lines 10-18 of Algorithm 1.

[3]We opted to integrate our solution into SILVER as it allows to avoid false negatives and provides extensive support for additional composability notions.

[4]In digital circuit design, a netlist is a description of the connectivity of a circuit. In its simplest form, a netlist consists of a list of the cells in a circuit and a list of the nodes they are connected to.

[5]It is essential for some verification tools, e.g., SILVER, as evaluating, e.g., constructing BDDs of a circuit with loop, becomes impossible.

Algorithm 1 Glitch and transition extension of a probing set

Input: $\mathbf{P}$ ▷ Probing set $\mathbf{P}$
Input: $\mathcal{I}, \mathcal{F}$ ▷ List of primary inputs $\mathcal{I}$, list of feedback signals $\mathcal{F}$
Input: n ▷ Total latency of one interation
Output: $\mathbf{P}_{g,t_{\{0\}}}, \mathbf{P}_{g,t_{\{1\}}}, \ldots, \mathbf{P}_{g,t_{\{n-1\}}}$ ▷ Final probing sets for all clock cycles

$\mathbf{P}_g \leftarrow \emptyset$ ▷ Initialize set of probes
for $\forall l \in \{0, \ldots, n-1\}$ **do**
    $\mathbf{R}_{\{l\}} \leftarrow \emptyset, \mathbf{S}_{\{l\}} \leftarrow \emptyset, \mathbf{P}_{g,t_{\{l\}}} \leftarrow \emptyset$ ▷ Initialize set of probes for cycle l
end for

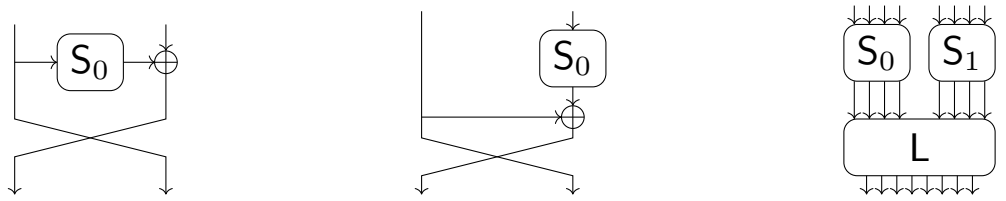
for all $\mathbf{p} \in \mathbf{P}$ **do** ▷ Perform glitch-extension
    $\mathbf{P}_g \leftarrow \mathbf{P}_g \cup \texttt{dfs}(\mathbf{p})$
end for

for $\forall l \in \{0, \ldots, n-1\}$ **do**
    for all $\mathbf{p} \in \mathbf{P}_g$ **do** ▷ Perform glitch-extension on feedbacks
        $x \leftarrow \mathbf{p}$
        if $\exists (x, y, l) \in \mathcal{F}$ **then** ▷ Check if x exist in $\mathcal{F}$
            $\mathbf{p}_1 \xleftarrow{\text{probe}} x, \mathbf{p}_2 \xleftarrow{\text{probe}} y$
            $\mathbf{R}_{\{l\}} \leftarrow \mathbf{R}_{\{l\}} \cup \{\texttt{dfs}(\mathbf{p}_1), \texttt{dfs}(\mathbf{p}_2)\}$
        else
            $\mathbf{R}_{\{l\}} \leftarrow \mathbf{R}_{\{l\}} \cup \{\mathbf{p}\}$
        end if
    end for

    for all $\mathbf{r} \in \mathbf{R}_{\{l\}}$ **do** ▷ Perform transition-extension on registers
        if $\texttt{type}(\mathbf{r}) = \texttt{reg}$ **then** ▷ Search all probes on register outputs
            $\{x\} \leftarrow \texttt{inputs}(\mathbf{r})$
            if $\exists (x, y, l) \in \mathcal{F}$ **then**
                $\mathbf{r}' \xleftarrow{\text{probe}} \{y\}$
            else
                $\mathbf{r}' \xleftarrow{\text{probe}} \{x\}$
            end if
            $\mathbf{S}_{\{l\}} \leftarrow \mathbf{S}_{\{l\}} \cup \{\mathbf{r}, \mathbf{r}'\}$
        else
            $\mathbf{S}_{\{l\}} \leftarrow \mathbf{S}_{\{l\}} \cup \{\mathbf{r}\}$
        end if
    end for

    for all $\mathbf{s} \in \mathbf{S}_{\{l\}}$ **do** ▷ Perform transition-extension on primary inputs
        if $l > 0 \quad \wedge \quad \mathbf{s} = x_{\{l\}} \in \mathcal{I}$ **then**
            $\mathbf{s}' \xleftarrow{\text{probe}} x_{\{l-1\}}$ ▷ Add the primary input of the previous clock cycle
            $\mathbf{P}_{g,t_{\{l\}}} \leftarrow \mathbf{P}_{g,t_{\{l\}}} \cup \{\mathbf{s}, \mathbf{s}'\}$
        else
            $\mathbf{P}_{g,t_{\{l\}}} \leftarrow \mathbf{P}_{g,t_{\{l\}}} \cup \{\mathbf{s}\}$
        end if
    end for
end for

Modeling Transitions. For any probe $p \in \mathbf{P}_g$ placed on a register output, we model transitions on the corresponding register by placing a standard probe on the register input.



(a) Feistel (b) MISTY (c) SPN

Figure 4: The underlying constructions of the 8-bit S-boxes introduced in [BGG+17].

Technically, we place the probe on the wire that drives the register. This can be seen in Lines 19-31 of Algorithm 1. For any probe placed on a primary input $x_{\{l\}}$ we place an additional standard probe on $x_{\{l-1\}}$. This step must be repeated for all evaluated clock cycles. This can be seen in Lines 32-39 of Algorithm 1.

3.3 Integration into SILVER

Since manual verification is time-consuming and hardly feasible for larger constructions, we opted to automate our approach and integrated our probe-extension algorithm into the already existing leakage verification tool SILVER [KSM20]. Given that SILVER operates on an annotated gate-level netlist, we extended the initial parsing and pre-processing procedures to support the conversion of an iterative circuit to a loop-free graph as well as the generation of the list of transitions $\mathcal{F}$. Further, transitions due to sequences of primary inputs are indicated through an extended annotation scheme in the original gate-level netlist.

In terms of analysis and verification, since SILVER already implements the security verification within the $(1, 0, 0)$-robust probing model, we leave the already provided glitch-extension scheme unchanged but only integrate our novel transition-extension scheme accordingly, as shown above. Eventually, the final statistical independence check is now performed on the newly created set of probes $\mathbf{P}_{g,t_{\{l\}}}$ instead of $\mathbf{P}$ or $\mathbf{P}_g$, as before. As a result, if SILVER detects a statistical dependency for any $\mathbf{P}_{g,t_{\{l\}}}$, i.e., the set of probes during cycle l, it will report information leakage accordingly.

4 Case Study 1: Strong 8-bit S-boxes

In order to apply the above-presented leakage evaluation procedure based on iterative circuits existing in literature, we consider the strong 8-bit S-box designs originally proposed at CHES 2016 [BGG+16] and later extended in [BGG+17]. During the following case study, we evaluate all S-box designs with our extended version of SILVER and compare the outcome with practical results based on physical measurements.

S-Box Architecture. The core idea of [BGG+17] is to construct a set of 8-bit S-boxes using smaller 4-bit S-boxes and some linear operations in known constructions, namely a Feistel network, a MISTY construction, or a Substitution Permutation Network (SPN). Figure 4 presents all such constructions, where the 4-bit S-boxes are denoted by S_i and a linear layer by L. These constructions should be iterated k times with $k \geq 2$ to build a cryptographically strong 8-bit S-box. This allowed the authors to build a relatively small masked circuit (TI) trivially realizing one round of such constructions (denoted by SB) and iterate it to achieve an area-efficient masked implementation of the 8-bit S-box. Figure 5 shows such an iterative design, which is similar to what the authors proposed in [BGG+17, Fig. 3(c)]. The authors have mainly constructed first-order secure TIs of their designs using

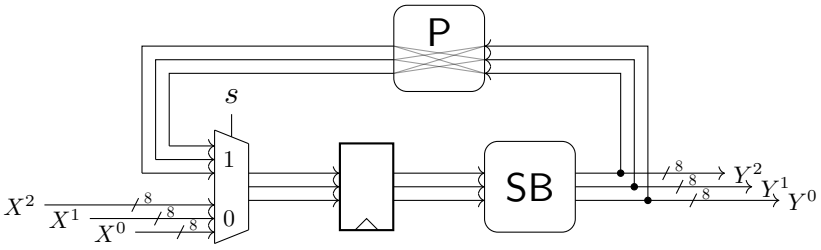


Figure 5: Iterative S-box architecture.

$td + 1 = 3$ shares, which satisfy $(1, 0, 0)$-robust 1-probing security. As shown in Figure 5, all input shares of SB are synchronized by a single register stage. Consequently, each iteration of SB is performed within a single clock cycle. In total, the authors introduced a set of eight different S-boxes SB_1 to SB_8. We give the basic properties of each S-box in Table 1, where Perm. refers to Figure 4(c) with linear layer L being a bit permutation, and Matrix a matrix multiplication.

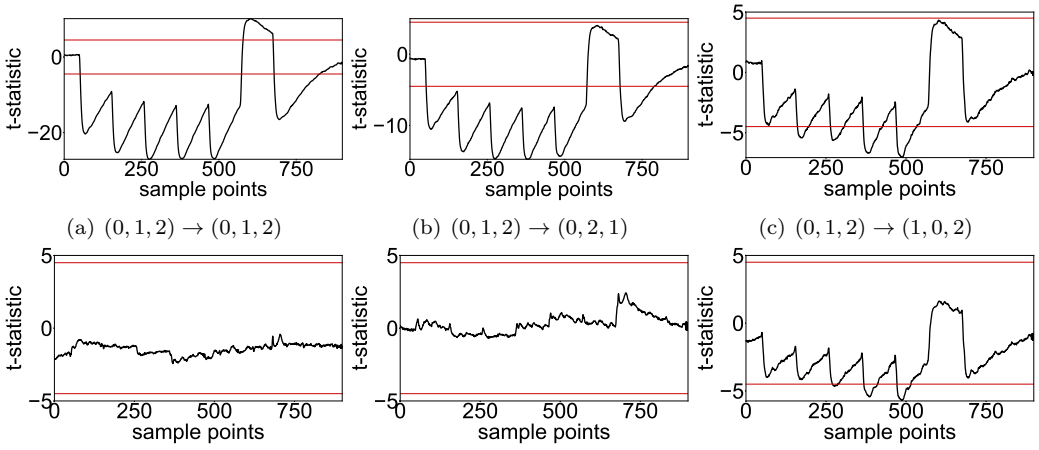
We implemented the same circuits receiving three input shares $(X^0, X^1, X^2) \in \mathbb{F}_2^{8\times 3}$ and returning three output shares $(Y^0, Y^1, Y^2) \in \mathbb{F}_2^{8\times 3}$. The output of the combinational logic builds the feedback signal and is given as the new input to the circuit if $k \neq 0$. This is handled by a multiplexer driven by the select signal $s \in \mathbb{F}_2$. Additionally, we have instantiated a share permutation module, denoted by P in the feedback path (see Figure 5). The purpose of such a module is to permute the shares. Although it has not been discussed or pointed out in [BGG+17], the output shares of SB should not necessarily be identically given to its input shares in the successive iteration. We denote this permutation as a mapping of share indices. For example $(0, 1, 2) \to (0, 1, 2)$ denotes the identity as all shares are mapped to their original position. Note, however, that each of the six possible permutations has no impact on the claimed first-order security of this construction considering the $(1, 0, 0)$-robust probing model.

Example 3 (Strong 8-bit Iterative S-Boxes)**.** Example 1 and Example 2 indicated that iterative circuits are prone to transitional leakage. Therefore, using our transition-extended version of SILVER, we tried to examine all eight different S-box constructions. We first examined the combinational function of all such designs by $(1, 0, 0)$-robust probing model and verified with SILVER that considering only glitches, the combinational function is first-order secure and provides a uniform output sharing, i.e., its composition does not violate security properties. As stated, permuting the shares of the feedback signals should not affect the security if the probes are only extended due to glitches. This is also trivially confirmed by SILVER. However, such a permutation may have an effect when transitions are taken into account. Therefore, we evaluated all eight iterative designs for all six possible share permutations considering the $(1, 1, 0)$-robust probing model. The results of such evaluations are summarized in Table 1. Beside the result of first-order evaluations, we give the corresponding number of evaluated probe combinations and the execution time next to each evaluation result.

It can be seen that considering the identity $(0, 1, 2) \to (0, 1, 2)$, i.e., the original construction in [BGG+17], none of the constructed iterative S-boxes is secure when the probes are extended based on glitches and transitions. Interestingly, our extended version of SILVER reports no leakage for three designs SB_1, SB_4, and SB_5 when feedback shares are permuted as $(0, 1, 2) \to (1, 2, 0)$. The same holds for two designs SB_1 and SB_4 for permutation $(0, 1, 2) \to (2, 0, 1)$. Since these designs belong to different categories, we believe that it is a coincidence with no dependency on the underlying construction of these S-boxes. As a consequence, among the iterative designs introduced in [BGG+17] there is no certain design category combined with a specific permutation of feedback shares which guarantees the security in presence of glitches and transitions, independent of the employed

Table 1: First-order evaluation results of all 8-bit S-boxes reported by SILVER under the $(1, 1, 0)$-robust probing model, including the number of probe combinations and the execution time on a Windows 10 server with 96 cores and 256GB RAM.

S-Box	Type	Iter.	Area	Share Permutation P: $(0,1,2) \rightarrow$					
		#	GE	$(0,1,2)$	$(0,2,1)$	$(1,0,2)$	$(1,2,0)$	$(2,0,1)$	$(2,1,0)$
SB_1	Perm.	8	241	✗$^{3}_{2.2\,s}$	✗$^{19}_{9.9\,s}$	✗$^{3}_{2.1\,s}$	✓$^{108}_{2.2\,min}$	✓$^{116}_{2.2\,min}$	✗$^{11}_{10.3\,s}$
SB_2	Matrix	2	446	✗$^{1}_{1.5\,min}$	✗$^{1}_{1.5\,min}$	✗$^{1}_{1.5\,min}$	✗$^{2}_{10.7\,min}$	✗$^{1}_{1.3\,min}$	✗$^{2}_{10.2\,min}$
SB_3	Matrix	4	458	✗$^{1}_{3.6\,min}$	✗$^{1}_{1.5\,min}$	✗$^{1}_{1.4\,min}$	✗$^{1}_{19.8\,min}$	✗$^{1}_{2.0\,h}$	✗$^{1}_{20.5\,min}$
SB_4	Feistel	5	363	✗$^{5}_{1.2\,s}$	✗$^{21}_{22.8\,min}$	✗$^{5}_{1.5\,s}$	✓$^{52}_{1.2\,h}$	✓$^{52}_{1.5\,h}$	✗$^{13}_{21.8\,min}$
SB_5	Perm.	9	263	✗$^{3}_{3.7\,s}$	✗$^{3}_{3.6\,s}$	✗$^{3}_{3.8\,s}$	✓$^{56}_{6.9\,min}$	✗$^{3}_{3.4\,s}$	✗$^{11}_{13.3\,s}$
SB_6	Matrix	4	420	✗$^{1}_{37.6\,s}$	✗$^{1}_{34.5\,s}$	✗$^{1}_{37.0\,s}$	✗$^{1}_{7.4\,min}$	✗$^{1}_{4.6\,min}$	✗$^{1}_{7.7\,min}$
SB_7	Feistel	4	431	✗$^{7}_{6.9\,s}$	✗$^{7}_{36.2\,s}$	✗$^{7}_{1.9\,s}$	✗$^{7}_{33.1\,s}$	✗$^{7}_{32.3\,s}$	✗$^{7}_{33.7\,s}$
SB_8	Feistel	8	332	✗$^{8}_{2.7\,s}$	✗$^{8}_{1.1\,min}$	✗$^{8}_{2.7\,s}$	✗$^{8}_{1.1\,min}$	✗$^{8}_{2.3\,min}$	✗$^{8}_{1.1\,min}$



(a) $(0,1,2) \rightarrow (0,1,2)$ (b) $(0,1,2) \rightarrow (0,2,1)$ (c) $(0,1,2) \rightarrow (1,0,2)$

(d) $(0,1,2) \rightarrow (1,2,0)$ (e) $(0,1,2) \rightarrow (2,0,1)$ (f) $(0,1,2) \rightarrow (2,1,0)$

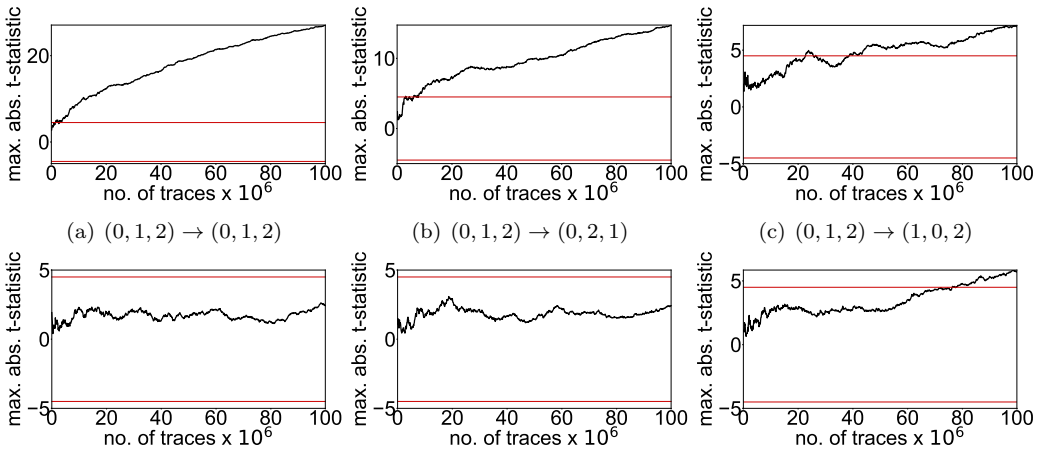
Figure 6: Iterative SB_1, first-order fixed vs. random t-test results over time, using 100 million traces.

components (i.e., 4-bit S-boxes, linear layers, round function of the Feistel network).

4.1 Experimental Analysis

As a sanity check, we examined the leakage of one of the iterative designs using real measurements. To this end, we have taken the iterative design of SB_1 (which is a first-order TI with 3 shares) and implemented the corresponding circuit on the Spartan-6 target Field-Programmable Gate Array (FPGA) of a SAKURA-G evaluation board [SAK] and recorded the dynamic power consumption by means of a digital oscilloscope at a sampling rate of 625 MS/s. During all measurements, the target architecture was being operated by a 6 MHz stable clock source. For the analysis, we applied the common Test Vector Leakage Assessment (TVLA) [GJJR11] approach on a set containing 100 million traces, measured while the circuit receives either a fixed or a random 3-share 8-bit input (to the 8-bit S-box). We have examined the same circuit for all six share permutations and limited our analyses to only first-order t-tests, as the underlying constructions are only supposed to provide first-order security.

The resulting t-statistics, which are shown in Figure 6 and Figure 7, precisely confirm our theoretical findings reported by our extended version of SILVER. More specifically, for



(a) $(0,1,2) \to (0,1,2)$ (b) $(0,1,2) \to (0,2,1)$ (c) $(0,1,2) \to (1,0,2)$

(d) $(0,1,2) \to (1,2,0)$ (e) $(0,1,2) \to (2,0,1)$ (f) $(0,1,2) \to (2,1,0)$

Figure 7: Iterative SB_1, first-order fixed vs. random t-test results over number of traces.

all designs that exhibit information leakage according to SILVER, we also observe leakage in the real measurements. Interestingly, for the original settings $(0,1,2) \to (0,1,2)$ and $(0,1,2) \to (0,2,1)$, the t-statistics surpasses the 4.5 threshold by less than five million traces, while for other insecure cases, many more traces are required to observe a significant leakage. This experiment indicates and confirms that transitional leakage, which we have detected by our transition-extended version of SILVER, is not only a purely theoretical issue but can have a severe practical impact on the security of a design.

5 Case Study 2: HPC Gadgets

As given in Section 2.5, HPC2 gadgets offer trivial composability under the notion of PINI only in the $(1,0,0)$-robust probing model [CGLS21]. In order to be in conformity with the refined notion of O-PINI under transitions and glitches, a novel O-PINI2 multiplication gadget has been proposed in [CS21]. In contrast to the original HPC2 gadget, such an extended O-PINI2 gadget requires d additional fresh random bits and an extra register stage, with d referring to the security order of the gadget, i.e., operating on $d+1$ shares of each input.

In the course of this case study, we first verify the results from [CS21] with our extension of SILVER and confirm the reported issues related to transitional leakage when iterating PINI-secure gadgets as described. Moreover, we verify the $(1,1,0)$-robust probing security of an iterative design if the considered gadget is in conformity with the O-PINI notion. Second, we show that the proposed O-PINI security notion is only necessary for a scenario where the circuit's loop is made by a single register stage. Eventually, we show how to avoid transitional leakages even without the application of an O-PINI secure gadget but solely relying on the length of the circuit's loop.

Example 4 (Original Design)**.** We first focus on the issues reported in [CS21]. More precisely, we consider the iterative circuit shown in Figure 8, where input shares x and y are given to an HPC 2-input multiplication gadget, which is iterated several times while using the gadget output shares z as new input shares instead of x. Following [CS21], we instantiated both, an HPC2 and an O-PINI2 gadget as the underlying 2-input multiplication gadget in Figure 8, and verified both constructions up to the third security order, i.e., $d \leq 3$, using our extended version of SILVER. As a side note, there is an inherent latency *imbalance* in the design of any HPC2 and O-PINI2 gadget, i.e., the required refreshing of

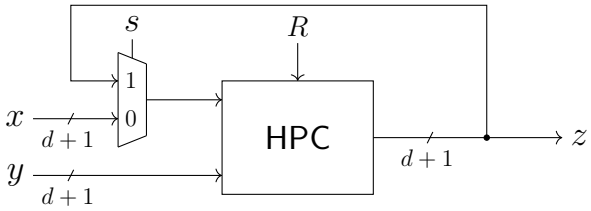


Figure 8: Iterative circuit for Examples 4 and 5.

Table 2: Leakage verification results of iterated HPC2 and O-PINI2 gadgets under (1,1,0)-robust probing model, including the number of probe combinations and the execution time on a Windows 10 server with 96 cores and 256GB RAM.

Gadget	Area	Random.	Security Order d		Complexity	
	[GE]	[bits]	[expected]	[achieved]	[# of probes]	[run time]
HPC2	104	1	1	0 ✗	8	0.01 s
HPC2	250	3	2	1 ✗	51	0.2 s
HPC2	459	6	3	2 ✗	1 862	53.8 min
O-PINI2	142	2	1	1 ✓	22	0.03 s
O-PINI2	309	5	2	2 ✓	1 035	2.7 min
O-PINI2	539	9	3	3 ✓	73 226	1.5 d

a single input introduces an additional cycle of latency for this input only. Hence, with respect to the generation of the output z, the input x has a lower latency compared to the other input y. Table 2 summarizes our evaluation results.

Our results indeed confirm the issues reported in [CS21], i.e., transitional leakage decreases the security order by one if the gadget is PINI secure but not O-PINI secure. Hence, in these cases, d-order security under the $(1,1,0)$-robust probing model is only satisfied if the gadget is at least $(d+1)$-order $(1,0,0)$-robust probing secure. When evaluating the HPC2 gadgets, we noticed that SILVER detects the leakage in the second clock cycle, when the feedback z is given as the new input instead of x. We show the details of this issue for $d = 1$ in Figure 9(a) which depicts half of the circuit, i.e., the part of the circuit which generates the first output share z^0.

For this example, SILVER identified a single probe $\mathbf{P} = \{\mathbf{p_0}\}$ indicating first-order leakage. More precisely, $\mathbf{p_0}$ is placed on an AND gate computing $r\,\overline{x^0}$ during the second clock cycle when r is stored in a register. Note that we provided the tool with a sequence of primary inputs r, i.e., distinct $r_{\{i\}} \in \mathbb{F}_2$ in cycle i indicating that the fresh mask is updated at every clock cycle.

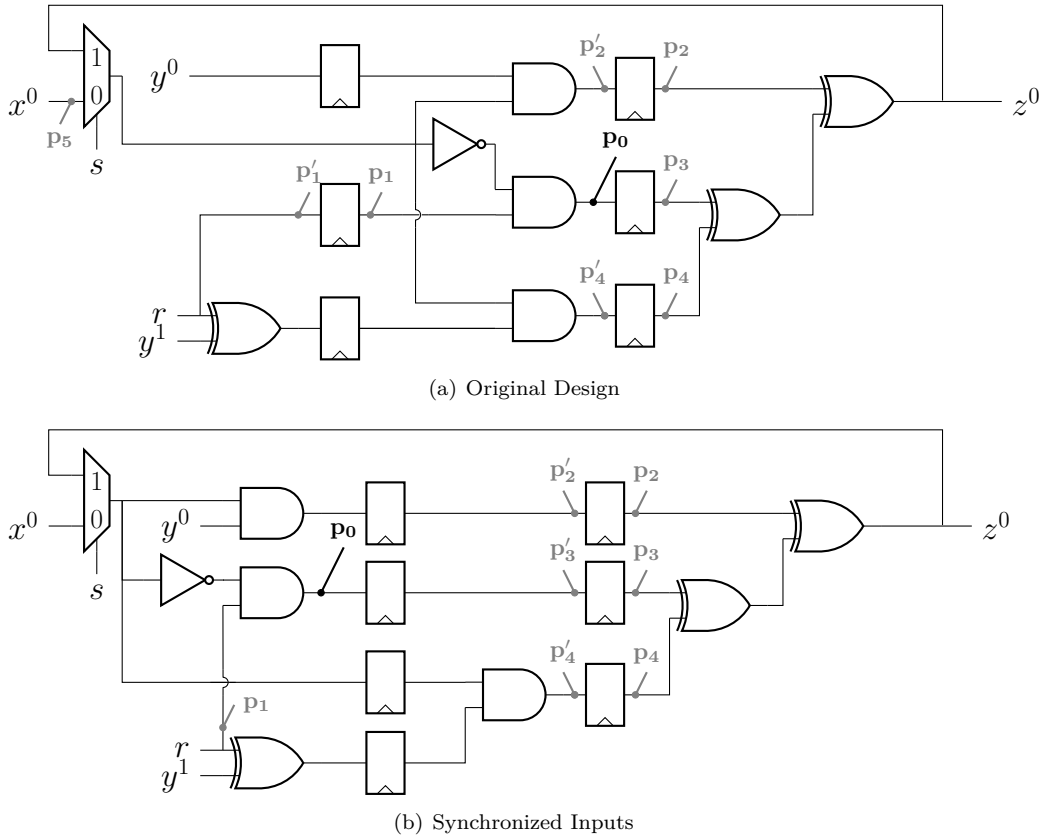
Starting with the glitch-extension scheme, $\mathbf{p_0}$ is extended as follows:

$$\mathbf{p_0} \xrightarrow{(1,0,0)} \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4}, \mathbf{p_5}\}, \qquad \mathbf{P}_g = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4}, \mathbf{p_5}\},$$

where $\mathbf{p_1}$ observes the register which stores r and $\mathbf{p_5}$ the primary input x^0. During the next clock cycle, x^0 is replaced by the feedback signal z^0, hence a transition between x^0 and z^0. This leads to the following glitch-extended and transition-extended set of probes.

$$\mathbf{p_1} \xrightarrow{(0,1,0)} \{\mathbf{p_1}, \mathbf{p_1'}\}, \quad \mathbf{p_2} \xrightarrow{(0,1,0)} \{\mathbf{p_2}, \mathbf{p_2'}\}, \quad \mathbf{p_3} \xrightarrow{(0,1,0)} \{\mathbf{p_3}, \mathbf{p_0}\}, \quad \mathbf{p_4} \xrightarrow{(0,1,0)} \{\mathbf{p_4}, \mathbf{p_4'}\},$$
$$\mathbf{P}_{g,t} = \{\mathbf{p_1}, \mathbf{p_1'}, \mathbf{p_2}, \mathbf{p_2'}, \mathbf{p_3}, \mathbf{p_0}, \mathbf{p_4}, \mathbf{p_4'}, \mathbf{p_5}\}$$

Now, we consider the output after the first iteration, i.e., $z^0 = x^0y^0 \oplus x^0y^1 \oplus r_{\{0\}}$, while the extended probes $\{\mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4}\}$ observe $\big(x^0y^0, r_{\{0\}}\overline{x^0}, x^0(y^1 \oplus r_{\{0\}})\big)$. Simultaneously, $\mathbf{p_1'}$ observes the fresh mask at the second clock cycle, i.e., $r_{\{1\}}$. Therefore, probe $\mathbf{p_0}$



(a) Original Design

(b) Synchronized Inputs

Figure 9: Single share computation of a first-order HPC2 2-input multiplication gadget.

leads to an observation including $\big(r_{\{0\}}, r_{\{1\}}, x^0, x^0y^0, r_{\{0\}}\overline{x^0}, x^0(y^1 \oplus r_{\{0\}})\big)$. This trivially leaks information about y. For an O-PINI2 gadget, z^0 would be refreshed with another mask independent of r and stored in a register, which avoids such a first-order leakage by extending $\mathbf{p_0}$ under the (1,1,0)-robust probing model.

Example 5 (Delayed Feedback)**.** As stated, the latency of x to z is one clock cycle. Therefore, the feedback is propagated at a point in time when the applied fresh mask is updated simultaneously. Hence, the transition-extended probes would capture the applied fresh mask and the feedback signal which is blinded by the same fresh mask, hence an undesired information leakage. A potential solution is to delay the feedback for (at least) one additional clock cycle in such a way that, when the transition between the old input x and the feedback signal z occurs, the fresh masks are updated (at least) one clock cycle before. We realize this by delaying input x one clock cycle to be synchronized with the other input y. This is shown in Figure 9(b).

Hence, a single probe $\mathbf{p_0}$ would still be extended by glitches to $\mathbf{P}_g = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4}, \mathbf{p_5}\}$. At the third clock cycle, $\underline{\mathbf{p_5}}$ observes x_0 while $\{\mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4}\}$ observe the result of the first iteration, i.e., $\big(x^0y^0, r_{\{0\}}\overline{x^0}, x^0(y^1 \oplus r_{\{0\}})\big)$. At the same clock cycle, $\mathbf{p_1}$ observes two consecutive fresh masks $r_{\{1\}}$ and $r_{\{2\}}$. This obviously does not lead to any leakage as the result of the first iteration is blinded by $r_{\{0\}}$ which is not observed by $\mathbf{p_1}$. Placing any other single probe on this circuit does not lead to any first-order leakage under $(1, 1, 0)$-probing model. We have confirmed this by evaluating such a circuit (for $d \leq 3$) with our extended version of SILVER.

As a side note, such a synchronization of inputs of the HPC2 gadget would solve the

issue and at the same time keep its total latency of two clock cycles, in contrast to O-PINI2 multiplication gadget which has a latency of three clock cycles. As a result, if HPC gadgets are used in an iterative circuit as shown in Figure 8, transitional leakage does not degrade their security if the sequential loop (involving the feedback signal) consists of at least two register stages and essentially fresh masks are updated at every clock cycle.

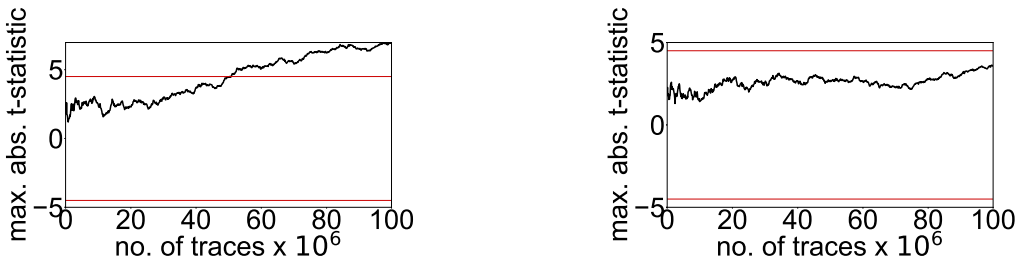
5.1 Experimental Analysis

Similar to the first case study (Section 4), we confirm our theoretical findings by means of an experimental analysis. We have used the same measurement setup, and implemented both designs of Example 4 and Example 5 for $d = 1$, collected 100 million traces recording the instantaneous power consumption during 5 iterations, and conducted the same evaluation, i.e., fixed-versus-random t-test.

The results, shown in Figure 10 and Figure 11, are inline with those reported by SILVER. Most importantly, we do not observe any first-order leakage from the design used in Example 5, while this is not the case for the design of Example 4. Note that the leakage detected for Example 4 is slightly above the threshold, i.e., more than 50 million measurements are required to observe the leakage. By this, we would like to highlight that with the theory and tools at hand (SILVER), we can detect the security flaws in such designs, but we cannot acquire any overview on their detectability and/or exploitability in practice.

(a) Original Design (b) Delayed Feedback

Figure 10: Iterative HPC gadget, first-order fixed v. random t-test results over time, using 100 million traces.



(a) Original Design (b) Delayed Feedback

Figure 11: Iterative HPC gadget, first-order fixed v. random t-test results over number of traces.

6 Conclusions

In this work, we present a novel methodology for modeling transition- and glitch-extended probes, enabling us to integrate the verification of transition-based leakage into SILVER, an existing software framework for formal verification of masked circuits which before was limited to perform verification in the $(1, 0, 0)$-robust d-probing model only, i.e., under the occurrence of glitches. With the integration of our methodology into SILVER, we enable designers to now also formally evaluate the security of hardware designs in the presence of glitches combined with transitions (i.e., in the $(1, 1, 0)$-robust d-probing model), which is highly relevant for constructing SCA-resilient iterative hardware designs.

For this, we present fundamental concepts to model and verify transition-based information leakage originating from memory recombinations, feedback loops, and transitions in primary inputs. We further demonstrate the relevance of our model extension by means of two different case studies. More precisely, the first case study demonstrates the power of the extended version of SILVER, for the first time enabling the verification and detection of security flaws in the iterative S-box designs introduced in [BGG+17].

Additionally, our second study confirms the composition flaws of HPC2 multiplication gadgets, as initially discussed in [CS21], when operated iteratively. In particular, this also allows us to show that some constructions proposed in [CS21] might be seen over-conservative with respect to security. Ultimately, for both case studies, we validate and confirm our findings (i.e., information leakage reported by our extended version of SILVER) by means of experimental leakage assessments.

While this work covers formal verification of SCA-resilience under glitches combined with transitions and is publicly available at GitHub[6], we should stress that its ability to cover transitional leakage is limited to a certain form of circuits. More precisely, the transitions associated to the input sequences are covered as long as the probes are placed at the combinational circuit receiving such primary inputs. If the probes are places on the combinational circuits which are not directly fed by the primary inputs, the transitional leakage originating from the input sequences might not be detected. Further, this tool does not yet cover any coupling-related leakage. As the detection of this would require additional routing information, the overall extension scheme cannot be performed on netlist level anymore. Nevertheless, a complete leakage verification should also take coupling effects into account. Hence, the automated verification under the $(1, 1, 1)$-robust d-probing model is a promising topic for future works.

Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the projects 393207943 GreenSec, 435264177 SAUBER, and 406956718 SuCCESS.

References

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

[6] https://github.com/Chair-for-Security-Engineering/SILVER/tree/transitional-leakage

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

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

[BDF+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 *Lecture Notes in Computer Science*, pages 535–566, 2017.

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

[BGG+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 *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

[BGG+16] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit Sboxes with Efficient Masking in Hardware. In *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 171–193. Springer, 2016.

[BGG+17] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit sboxes with efficient masking in hardware extended version. *J. Cryptogr. Eng.*, 7(2):149–165, 2017.

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

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In *ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.

[Bil15] Begül Bilgin. *Threshold implementations : as countermeasure against higher-order differential power analysis*. PhD thesis, University of Twente, Enschede, Netherlands, 2015.

[CBG+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 *Lecture Notes in Computer Science*, pages 1–18. Springer, 2017.

[CBR+15] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-Order Threshold Implementation of the AES S-Box. In *CARDIS 2015*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.

[CGD18] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors. In *COSADE 2018*, volume 10815 of *Lecture Notes in Computer Science*, pages 82–98. Springer, 2018.

[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+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 *Lecture Notes in Computer Science*, 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 *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Transactions on Information Forensics and Security*, 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. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.

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

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

[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.

[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 *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.

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

[HS13] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS 2013*, volume 8419 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2013.

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

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

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

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

[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. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[MS06] Stefan Mangard and Kai Schramm. Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In *CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2006.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[NRS08] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In *ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[SAK] SAKURA. Side-channel Attack User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html.