IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 1, pp. 323–344. DOI:10.46586/tches.v2022.i1.323-344

Generic Hardware Private Circuits

Towards Automated Generation of Composable Secure Gadgets

David Knichel, Pascal Sasdrich and Amir Moradi

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

Abstract. With an increasing number of mobile devices and their high accessibility, protecting the implementation of cryptographic functions in the presence of physical adversaries has become more relevant than ever. Over the last decade, a lion's share of research in this area has been dedicated to developing countermeasures at an algorithmic level. Here, *masking* has proven to be a promising approach due to the possibility of formally proving the implementation's security solely based on its algorithmic description by elegantly modeling the circuit behavior. Theoretically verifying the security of masked circuits becomes more and more challenging with increasing circuit complexity. This motivated the introduction of security notions that enable masking of single gates while still guaranteeing the security when the masked gates are composed. Systematic approaches to generate these masked gates – commonly referred to as gadgets – were restricted to very simple gates like 2-input AND gates. Simply substituting such small gates by a secure gadget usually leads to a large overhead in terms of fresh randomness and additional latency (register stages) being introduced to the design.

In this work, we address these problems by presenting a generic framework to construct trivially composable and secure hardware gadgets for arbitrary vectorial Boolean functions, enabling the transformation of much larger sub-circuits into gadgets. In particular, we present a design methodology to generate first-order secure masked gadgets which is well-suited for integration into existing Electronic Design Automation (EDA) tools for automated hardware masking as only the Boolean function expression is required. Furthermore, we practically verify our findings by conducting several case studies and show that our methodology outperforms various other masking schemes in terms of introduced latency or fresh randomness – especially for large circuits.

Keywords: Masking, Generic and Composable Hardware Gadgets, Automated Masking, Side-Channel Analysis

1 Introduction

Even though Side-Channel Analysis (SCA) has been studied extensively by academic and industrial researchers, secure implementation of strong cryptographic implementations remains a challenging task. In the wake of the seminal description by Paul Kocher [Koc96], different approaches for countermeasures against SCA adversaries have been proposed. Among all candidates, *masking*, inspired by *secret sharing* concepts, is fascinating by its theoretical and sound security foundation [CJRR99] and has been applied manifold until today [ISW03, Tri03, NRS11, RBN$^+$15, GMK17, GM18]. Unfortunately, not many of the proposed schemes have survived due to design flaws, inaccurate models, or invalid assumptions [MMSS19]. As a consequence, this trend of schemes whose assumptions have been proven invalid only confirms that, to the present day, design and implementation

Received: 2021-07-15 Accepted: 2021-09-15 Published: 2021-11-19

of masking schemes is still a mostly manual, complex, and error-prone process, even for experienced security experts and hardware designers.

Facing such challenges, researchers recently started to focus on development of formal and accurate models of physical adversaries, hardware platforms, and execution environments as a mandatory foundation for formal verification and provably-secure schemes. In this light, formal verification of masked circuits is frequently conducted in the simple and abstract Ishai-Sahai-Wagner (ISW) d-probing security model [ISW03], given some basic assumptions on input and noise distribution, and its extension for more accuracy in the presence of physical defaults, e.g., glitches, transitions, and couplings [FGP+18].

Even though the introduction of a simple, yet practical, formal model accelerated verification, most security proofs are still limited to small circuits and masked gadgets only, mostly due to constraints in computational complexity. Naturally, modern approaches endeavor to extend formal verification to larger circuits through composition of formally verified gadgets, however, experience has shown that composition of secure gadgets is non-trivial and security proofs do not extend immediately.

Accordingly, several security notions for secure and trivial composition of masked gadgets have been proposed recently [BBD+15, BBD+16, CS20]. Although the security notions aim to assist in design and verification of larger circuits, creation of gadgets according to these rules in order to meet the requirements is still a challenge. More specifically, design of efficient gadgets under several optimization metrics, e.g., computational complexity, area demands, randomness requirements, performance in terms of latency and throughput, or higher-order protection still requires manual interaction and long-standing experience. To this end, the list of existing secure gadgets is limited, as most of them are hand-crafted, mainly focusing on protection of small gates, e.g., a 2-input AND [Tri03, BDF+17, FGP+18, CGLS20]. More importantly, these approaches usually are limited to atomic Boolean functions, e.g., AND and XOR, but do not provide a generic or automated approach to design secure, efficient, and trivially composable gadgets for different or arbitrary Boolean functions.

Contributions. In this work, we present a novel and generic framework that allows to easily construct trivially composable gadgets for arbitrary vectorial Boolean functions. In particular, relying on the glitch-extended probing adversary model and the secure composability notion of Probe-Isolating Non-Interference (PINI), our framework enables simple and generic construction of hardware private circuits and opens the possibility to transform any unprotected Boolean function into a first-order secure and composable gadget. In addition, backed by a thorough and sound theoretical security analysis and formal security arguments, our constructions enable efficient formal verification of entire cryptographic circuits and systems with respect to the PINI security notion. Eventually, we show practical relevance of our construction through experimental verification using different case studies and compare implementation results with respect to area, latency, and fresh randomness for various gadget constructions and related work.

Outline. Before we present our fundamental design principles based on Shannon's Decomposition and provide a dedicated security analysis for our first-order secure construction schemes in Section 3, we first briefly present underlying assumptions and concepts in Section 2, including circuit representation, adversary model, security notions, and Boolean masking. In Section 4, we discuss and compare our proposed constructions to related works from literature, focusing on the metrics of area, latency, randomness, and composability. We further present different case studies to emphasize practical application of our concepts in Section 5 and experimentally confirm our theoretical security analyses based on leakage assessment for different PRESENT and AES designs. Eventually, we give a conclusion of the research conducted in this work in Section 6.

2 Background

2.1 Notation

Let us denote functions using sans-serif fonts, e.g., F. Next, we denote random variables with uppercase letters, e.g., X, while sets of random variables are given in bold, such as $\mathbf{X}$. Further, we use subscripts to indicate elements within a set while superscripts are used to denote (randomized) shares of random variables. Moreover, lowercase letters are used for the value of a random variable and bold lowercase letters indicate values for sets of variables accordingly. As a special case, the set of all shares of each random variable in $\mathbf{X}$ is denoted as $Sh(\mathbf{X})$ and $P[\mathbf{X} = \mathbf{x}] = P[\mathbf{X}]$ denotes the joint probability that every $X_i \in \mathbf{X}$ takes the value $x_i \in \mathbf{x}$. Moreover, $\mathbf{X}^j$, $j \geq 0$, denotes the set containing all shares with index j. If $\mathbf{S}$ is a set over arbitrary shares X_i^j, i.e., $\mathbf{S} \subset Sh(\mathbf{X})$, then $|\mathbf{S}|_i$ denotes the number of shares in $\mathbf{S}$ that correspond to X_i.

2.2 Circuit Model

Throughout this work, any deterministic logic circuit C will be considered and modeled as a Directed Acyclic Graph (DAG) $\mathsf{G}_\mathsf{C} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ gives the list of vertices and $\mathcal{E}$ the list of edges of the DAG. Further, each vertex $v \in \mathcal{V}$ represents a single combinational or sequential gate in the netlist while each edge $e \in \mathcal{E}$ represents a single wire carrying an element from the finite field $\mathbb{F}_2$. In its entirety, a circuit realizes a vectorial Boolean function $\mathsf{F} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ given its *coordinate functions* $\mathsf{F}_0, \ldots, \mathsf{F}_{m-1}$, where F is defined over its input $\mathbf{X} \in \mathbb{F}_2^n$.

Encoded Circuit Model As formalized by Ananth et al. [AIS18], a circuit compiler is a set of algorithms $\{\mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode}\}$, such that $\mathsf{Compile}$ is a deterministic algorithm that takes as input a circuit C and generates a randomized and encoded circuit $\bar{\mathsf{C}}$. Further, Encode is a probabilistic algorithm that takes as input a set of (secret) random variables $\mathbf{X}$ and generates a shared representation $Sh(\mathbf{X})$ with respect to some masking scheme (e.g., *Boolean masking*). Lastly, Decode is a deterministic algorithm that takes as input a shared representation $Sh(\mathbf{Y})$ and reconstructs the according set of random variables $\mathbf{Y}$. Moreover, the circuit compiler has to satisfy *correctness* such that:

$$\mathsf{Decode}(\bar{\mathsf{C}}(\mathsf{Encode}(\mathbf{X}))) = \mathsf{C}(\mathbf{X}), \forall \mathbf{X}.$$

2.3 Adversary Model

Before discussing common security notions and Boolean masking as theoretically sound countermeasure against SCA, we introduce the foundational d-probing adversary model which is used in modern literature to model side-channel adversaries and verify security of hardware circuits in presence of such adversaries.

Traditional d-Probing Model. In the traditional ISW d-probing model [ISW03], the adversarial strength is solely defined and limited by the number of probes that are granted to an adversary. Each probe can be used to observe and extract information carried on a single circuit[1] wire at a time. Assuming an ideal circuit, all gates and wires are updated simultaneously and each wire only carries the result of the driving gate under the current assignment of primary inputs. Then, depending on the number of granted probes, an adversary can combine information of up to d wires in the circuit to infer sensitive information. Further, a circuit is assumed to be secure under the d-probing adversary

[1] In the remainder of this work, we assume that the adversary is only able to observe an encoded circuit $\bar{\mathsf{C}}$ while the Encode and Decode algorithms are unavailable for the adversary.

Algorithm 1 glitch-extend

```
Input: Probe $P \in \mathbb{F}_2$
Output: Glitch-extended probe $\mathbf{P}_{ext} \in \mathbb{F}_2^e, e > 0$
  if $P$ is placed on an output of a combinational gate then
      $\mathbf{P}_{ext} \leftarrow \bigcup_{0 \leq i < n}$ glitch-extend($P_i$)    ▷ where $P_i$, $0 \leq i < n$ are all inputs to the combinational gate
  else
      if $P$ is placed on an output of a register or on a primary input then
          $\mathbf{P}_{ext} \leftarrow \{P\}$
      end if
  end if
```

model if for any combination of up to d probes, the adversary is not able to learn about sensitive information (more details are given in Section 2.5).

Probing in the Presence of Glitches. Research has shown that physical circuits do not have an ideal behavior but physical defaults, e.g., glitches, transitions, and coupling [FGP$^+$18], may cause unintentional leakages. In particular glitches, causing unintentional transitions on circuit wires due to non-ideal gates and different path delays, have been shown to introduce design vulnerabilities even for circuits that are secure under the standard d-probing adversary model [MPG05]. As a consequence, a robust d-probing model has been proposed [FGP$^+$18] assuming a worst-case scenario under glitch-occurrence in physical circuits. More precisely, in contrast to the standard model, probes in the robust model are considered as *glitch-extended* and grant an adversary not only access to the signal on the probed wire but also any combination of stable driving signals (primary or registered inputs).

If $P \in \mathbb{F}_2$ is a probe in the standard probing model, the corresponding glitch-extended probe can be derived by performing glitch-extend(P), where glitch-extend($\cdot$) can be defined recursively by returning either all extended probes on the input to the combinational gate, if P is placed on the output of such gate, or is defined by the identity function, if P is placed on an output of a register or a primary input. This is formally defined in Algorithm 1. A set of probes can be extended by union of the glitch extension of each probe.

2.4 Probe Simulatability

The concept of probe simulation helps to formally argue about dependencies between the probability distribution over probe observations and inputs to a masked (i.e., encoded) circuit.

Definition 2.1 (Perfect Probe Simulation). *Given a set* $\mathbf{P} = \{P_0, P_1, \ldots, P_{l-1}\}$ *of* l *(glitch-extended) probes on a masked circuit* C *with input* $Sh(\mathbf{X})$, $\mathbf{P}$ *can be* perfectly simulated *with a set over arbitrary shares* $\mathbf{S}$ *iff there exists a simulator* Sim *such that for any shared input* $Sh(\mathbf{X})$ *to* C, *the probability distribution over* $\mathbf{P}$ *and* $Sim(\mathbf{S})$ *are equal, where* $Sim : \mathbb{F}_2^{|\mathsf{S}|} \mapsto \mathbb{F}_2^l$ *with input* $\mathbf{S} \subseteq Sh(\mathbf{X})$ *is a probalistic polynomial time (p.p.t.) simulator.*

Probe Propagation in Composed Circuits. Experience has shown that composition of secure circuits, in the presence of d adversarial probes, may not result in secure composed circuits, given the same adversarial strength. More specifically, even though each circuit separately can be proven to resist up to d adversarial probes, the effect of *probe propagation* may provide the adversary with more information than initially assumed. In particular, as downstream sub-circuits in a composed circuit usually process and combine results of upstream sub-circuits, placing up to d adversarial probes in those sub-circuits can provide information that, for isolated circuits, might only be obtainable by placing more than d probes, hence, virtually extending the adversarial strength beyond the limit of d probes. With the help of Definition 2.1, we can formally define probe propagation as:

Definition 2.2 (Probe Propagation)**.** *A probe $P \in \mathbb{F}_2^l$ is said to* propagate *into an input wire $I \in \mathbb{F}_2$ iff I is required to perfectly simulate P, i.e., I has to be in the simulation set S as defined in Definition 2.1.*

2.5 Security Notions

Since the seminal introduction of the ISW d-probing adversary model [ISW03], many different security notions to analyze and verify the security of physical circuits have been proposed, in particular to ensure composition of secure circuits from provably secure sub-circuits. Below, we introduce the most common security notions, based on the consolidated definitions in [DBR19] and their generalization, unification, and extension as recently presented in [KSM20].

Probing Security. Granted access to internal values of a circuit through adversarial probes, an adversary may learn (partial) information on the processed secrets. Hence, in order to achieve probing security in the presence of up to d adversarial probes, any combination of up to d probes on internal values carried on wires must be statistically independent of the processed secrets. More specifically, this will limit the partial information any d-probing adversary can learn on the secrets, such that correct guessing and recovering of the sensitive information is impossible. More formally, probing security can be defined through Definition 2.3.

Definition 2.3 (d-Probing Security)**.** *An encoded circuit $\bar{\mathsf{C}}$, with secret input $\mathsf{Encode}(\mathbf{X})$, $\mathbf{X} \in \mathbb{F}_2^n$, is d-probing secure, if and only if for any observation $\mathbf{Q}$ of $t \leq d$ wires, $\mathbf{X}$ is statistically independent of the observation, i.e., $P[\mathbf{Q}|\mathbf{X}] = P[\mathbf{Q}]$.*

Non-Interference. While d-probing security purely focuses on the security of circuits in the presence of adversarial probes, the security notion of *Non-Interference (NI)* additionally targets the composition of masked circuits, usually considered as *gadgets*, such that security spans across the composed circuit instead of isolated gadgets only.

Through the concept of NI, flow of sensitive information is limited, although a d-probing adversary is still allowed to gain partial information on internal values and wires through adversarial probes. However, the original circuit, and in particular the original distribution of probed values, must not be distinguishable from a simulated distribution generated only based on the available partial information. As a consequence, each adversarial probe must be perfectly simulatable on partial information comprising a subset of all primary input shares limited by the security order d. More formally, the security notion of NI can be expressed through Definition 2.4.

Definition 2.4 (d-Non-Interference)**.** *An encoded circuit $\bar{\mathsf{C}}$, with secret input $\mathsf{Encode}(\mathbf{X})$, $\mathbf{X} \in \mathbb{F}_2^n$, is d-non-interfering if and only if for any observation $\mathbf{Q}$ of $t \leq d$ wires, there exists a set $\mathbf{S}$ of input shares, with $|\mathbf{S}|_i \leq t, \forall i$, such that $P[\mathbf{Q}|\mathbf{S}] = P[\mathbf{Q}|Sh(\mathbf{X})]$.*

Strong Non-Interference. Unfortunately, the security notion of NI could not ensure *composability* of d-probing secure gadgets, due to the problem of *probe propagation* in composed circuits. More precisely, composing gadgets may result in combination of partial information such that the placing of adversarial probes on downstream gadgets may propagate into upstream gadgets and grant the adversary access to partial information that otherwise could only be observed by placing more than d adversarial probes.

Hence, to correct deficiencies in the NI notion, once it comes to composition of secure gadgets, the stronger notion of Strong Non-Interference (SNI) was introduced. In particular, the concept of SNI intercepts probe propagation at the primary output of gadgets which again limits the partial information accessible through adversarial probes. As a consequence, each primary output of an SNI-secure gadget must be perfectly simulatable even without any partial information gained through adversarial probes. More formally, the security notion of SNI can be expressed through Definition 2.5.

Definition 2.5 (d-Strong Non-Interference)**.** *An encoded circuit* $\bar{\mathsf{C}}$*, with secret input* $\mathsf{Encode}(\mathbf{X}), \mathbf{X} \in \mathbb{F}_2^n$*, is d-strong-non-interfering if and only if for any observation* $\mathbf{Q}$ *of* $t = t_1 + t_2 \leq d$ *wires, with* t_1 *being internal wires and* t_2 *being output wires, there exists a set* $\mathbf{S}$ *of input shares, with* $|\mathbf{S}|_i \leq t_1, \forall i$*, such that* $P[\mathbf{Q}|\mathbf{S}] = P[\mathbf{Q}|Sh(\mathbf{X})]$*.*

Probe-Isolating Non-Interference. Although the security notion of SNI resolves shortcomings in NI and allows secure composition of gadgets, this security notion, however, is rather conservative and inefficient in practice with respect to fresh entropy and circuit area. Moreover, Cassiers and Standaert [CS20] have shown that the concept of SNI is limited to single-output gadgets only, but does not scale for multi-output gadgets, again due to *probe propagation*. Although the concept of Multiple-Input-Multiple-Output SNI (MIMO-SNI) could fix the deficiencies, PINI was introduced as a more elegant and efficient solution.

In particular, the approach of PINI is inspired by trivial composition of linear functions (assuming Boolean masking) and the concept of *domain separation* as introduced in [GMK17]. More precisely, PINI-secure gadgets limit the propagation of adversarial probes with respect to share domains (also referred to as *circuit shares*), i.e., each share domain is separated and any adversarial probe will only propagate into its associated share domain. Given this, PINI-gadgets are trivially composable, similar to linear gadgets, regardless of the number of primary outputs. More formally, the security notion of PINI can be expressed through Definition 2.6.

Definition 2.6 (d-Probe-Isolating Non-Interference)**.** *An encoded circuit* $\bar{\mathsf{C}}$ *with secret input* $\mathsf{Encode}(\mathbf{X}), \mathbf{X} \in \mathbb{F}_2^n$*, is probe-isolating non-interfering if and only if for any observation* $\mathbf{Q}$ *of* $t = t_1 + t_2 \leq d$ *wires, with* t_1 *being internal wires and* t_2 *being output wires, there exists a set of* $\mathbf{I}_{\text{PI}}$ *primary input indices, with* $|\mathbf{I}_{\text{PI}}| \leq t_1$*, and* $\mathbf{I}_{\text{PO}}$ *primary output indices, with* $|\mathbf{I}_{\text{PO}}| \leq t_2$*, such that* $\mathbf{Q}$ *can be perfectly simulated by* $\mathbf{S} = Sh(\mathbf{X})^{\mathbf{I}_{\text{PI}} \cup \mathbf{I}_{\text{PO}}}$*.*

2.6 Boolean Masking

Due to its sound theoretical foundation, Boolean masking has been established as the most predominant approach to mitigate side-channel leakage in digital logic. In general, Encode for Boolean masking relies on concepts of secret sharing to split sensitive variables $\mathbf{X}$ into Boolean shares $\mathbf{X}^i$, such that $\mathbf{X} = \bigoplus_{i=0}^{d} \mathbf{X}^i$, which allows simple masking of linear functions, but requires special considerations for non-linear operations.

Assuming that each Boolean share $\mathbf{X}^i$ is independent of the secret $\mathbf{X}$ and all other shares, a circuit implementing Boolean masking with $d+1$ shares can be evaluated securely even in the presence of d adversarial probes. However, as already mentioned, transient computations, i.e., glitches in hardware circuits, may recombine independent shares resulting in secret-dependent evaluations that may leak sensitive information. Hence, careful construction and layout of the masking scheme is imperative and a variety of different schemes has been proposed to ensure resistance even in the presence of glitches.

As a consequence, different hardware masking schemes have been proposed over the last years [ISW03, Tri03, NRS11, RBN+15, GMK17, GM18], most of them being extendable to higher-order protection and providing different trade-offs for computational and area complexity, memory requirement, latency, and randomness demand.

3 Generic Hardware Private Circuits (GHPC)

3.1 Shannon Decomposition

In general, our construction for the design of generic and composable hardware private circuits for arbitrary Boolean functions utilizes the so-called *Shannon Decomposition* which was initially presented by Boole in [Boo48].

Theorem 1 (Shannon Decomposition)**.** *Any Boolean function* $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ *can be written as*

$$\begin{aligned}\mathsf{F}(X_0, X_1, \dots, X_i, \dots, X_{n-1}) =& \overline{X_i} \cdot \mathsf{F}(X_0, X_1, \dots, 0, \dots, X_{n-1}) \oplus \\ & X_i \cdot \mathsf{F}(X_0, X_1, \dots, 1, \dots, X_{n-1}),\end{aligned}$$

or in short: $\mathsf{F} = \overline{X_i} \cdot \mathsf{F}|_{X_i=0} \oplus X_i \cdot \mathsf{F}|_{X_i=1}$*, where* $\mathsf{F}|_{X_i=0}$ *and* $\mathsf{F}|_{X_i=1}$ *are called the Shannon cofactors.*

Note that in the original definition, the Shannon cofactors were connected by a simple OR operation instead of an XOR. Correctness of both versions is nonetheless obvious, as by assigning a value to X_i, the corresponding Shannon cofactor is selected as output function, such that:

$$\mathsf{F} = \begin{cases} \mathsf{F}|_{X_i=0}, & \text{if } X_i = 0 \\ \mathsf{F}|_{X_i=1}, & \text{if } X_i = 1 \end{cases} \tag{1}$$

Since $\mathsf{F}|_{X_i=0}$ and $\mathsf{F}|_{X_i=1}$ are again Boolean functions, this decomposition can be applied recursively, depending on arbitrary input variables. For example, F can be decomposed choosing X_i and X_j, $i \neq j$, then leading to:

$$\begin{aligned}\mathsf{F} =\ & \overline{X_i}\,\overline{X_j} \cdot \mathsf{F}|_{X_i=0,X_j=0} \oplus \\ & \overline{X_i}\,X_j \cdot \mathsf{F}|_{X_i=0,X_j=1} \oplus \\ & X_i\,\overline{X_j} \cdot \mathsf{F}|_{X_i=1,X_j=0} \oplus \\ & X_i\,X_j \cdot \mathsf{F}|_{X_i=1,X_j=1}.\end{aligned} \tag{2}$$

In essence, translating a Shannon Decomposition of F into a logic circuit can be represented as a multiplexer (cascade) selecting the cofactors depending on the decomposition variables. For this, Equation 1 results in a 2-input multiplexer selecting depending on X_i, while Equation 2 results in a 4-input multiplexer selecting depending on X_i and X_j.

3.2 Design

A high-level overview of our methodology for generating composable private circuits from unprotected circuits is depicted in Figure 1. Given an unprotected circuit C realizing a Boolean function $\mathbf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and knowing the function expression of $\mathbf{F}$, our masking approach enables the construction of a first-order protected and composable hardware private circuit GHPC with two input and output shares under the PINI security notion (even in the presence of glitches). Further, the number of refreshing random bits of our approach is limited to only a single fresh random bit per coordinate function (i.e., $\mathbf{R} \in_R \mathbb{F}_2^m$). In fact, the result of the GHPC is a textbook sharing of each original coordinate

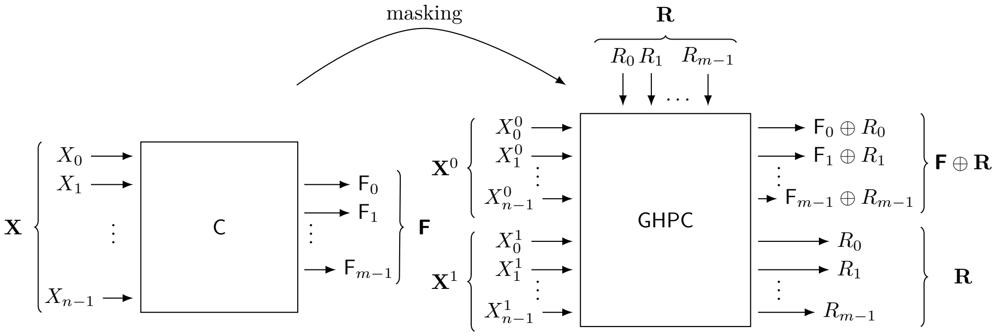


Figure 1: An overview of first-order masking

function F_i, i.e., each coordinate function is blinded by a different fresh mask $R_i \in \mathbf{R}$, while the second share is simply assigned the chosen random value R_i drawn from $\mathbf{R}$, hence, immediately ensuring uniformity and correctness of the sharing.

Construction Principle. Given the circuit C, our construction principle for translation into a GHPC allows to process and transform each coordinate function F_i with $0 \leq i < m$ independently. For this, we will restrict the discussion of the construction principle to arbitrary single-output Boolean functions $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2$, as extension to vectorial Boolean functions is given trivially through application on each coordinate function separately.

In general, given a shared function expression obtained through *direct sharing* of the original function F, i.e., $\mathsf{F}' = \mathsf{F}(X_0^0 \oplus X_0^1, X_1^0 \oplus X_1^1, \ldots, X_{n-1}^0 \oplus X_{n-1}^1)$, our construction can be seen as Shannon Decomposition of F', where each Shannon cofactor is blinded by R and F' is evaluated and decomposed based on shares from a single share domain. However, it is important to note that F' itself is never constructed explicitly, as a Shannon Decomposition based on one share allows to construct F' implicitly, as this, given simple Boolean masking, results in substituting any variable in the original function F with the corresponding (possibly negated) other share. For instance, it holds that if $X_0^1 = 0$, $X_1^1 = 1$, $\ldots$, and $X_{n-1}^1 = 1$, then $\mathsf{F}'|_{\mathbf{X}^1=(0,1,\ldots,1)} = \mathsf{F}(X_0^0, \overline{X_1^0}, \ldots, \overline{X_{n-1}^1})$. Interestingly, in this case, the Shannon cofactors only depend on a single share domain, while selection of the correct computation, i.e., the selection of the correct cofactor, only depends on the other share domain.

Hence, the foundation of our construction is a multiplexer design that selects function evaluations restricted to one share, each evaluation blinded by the same random value $R \in_R \mathbb{F}_2$, as shown in Figure 2 and algorithmically described in Algorithm 2. Then, the selection of the correct evaluation only depends on the second share, e.g., shares from domain 1 for the given design. Note, however, that naming of share domains is not fixed but may be chosen arbitrarily, as long as the share domain naming is applied consistently throughout the entire design and the naming of the output domains is adopted accordingly (to ensure security under the PINI notion). Then, each blinded Shannon cofactor is stored in a register, and selected according to the other share subsequently. For this, the registers depicted in dashed lines (and denoted as $\mathsf{Reg}_{\mathsf{pipe}}[]$ in Algorithm 2) ensure synchronization and enable a pipelined architecture, but do not have any effect on the security in the glitch-extended robust probing model in general. Eventually, as only the correct Shannon cofactor is enabled through an AND gate but all other factors are gated, summing up the values in the final register stage results in the correct but blinded output, hence, assigning R to the second share of the GHPC ensures correctness.

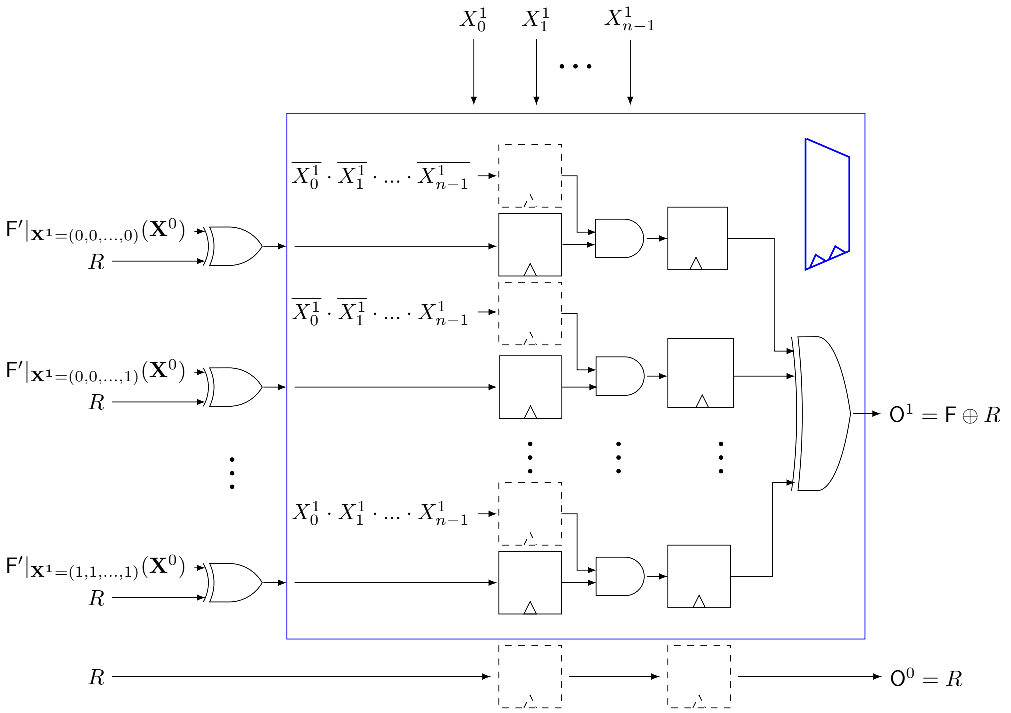

X_0^1
X_1^1
X_{n-1}^1
$\overline{X_0^1} \cdot \overline{X_1^1} \cdot \ldots \cdot \overline{X_{n-1}^1}$
$\mathsf{F}'|_{\mathbf{X}^1=(0,0,\ldots,0)}(\mathbf{X}^0)$
R
$\overline{X_0^1} \cdot \overline{X_1^1} \cdot \ldots \cdot X_{n-1}^1$
$\mathsf{F}'|_{\mathbf{X}^1=(0,0,\ldots,1)}(\mathbf{X}^0)$
R
$X_0^1 \cdot X_1^1 \cdot \ldots \cdot X_{n-1}^1$
$\mathsf{F}'|_{\mathbf{X}^1=(1,1,\ldots,1)}(\mathbf{X}^0)$
R
$\mathsf{O}^1 = \mathsf{F} \oplus R$
R
$\mathsf{O}^0 = R$

Figure 2: General GHPC design.

Security Analysis. In this section, we briefly prove the correctness and security of our construction under the notion of PINI, as stated in Theorem 2, assuming the d-probing model with glitch-extended probes.

Theorem 2. *For an arbitrary circuit* C*, realizing a Boolean function* $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2$*, the transformation into a* GHPC *results in a correct and first-order PINI-secure circuit under the glitch-extended d-probing model.*

Proof.

Correctness: The correctness of the Shannon Decomposition directly implies that $\mathsf{O}^1 = \mathsf{F}' \oplus R$. As $[\mathbf{X}^0, \mathbf{X}^1]$ is a valid sharing of $\mathbf{X}$, it follows that $\mathsf{O}^1 = \mathsf{F} \oplus R$, hence $\mathsf{O} = \mathsf{O}^0 \oplus \mathsf{O}^1 = \mathsf{F}$.

PINI: Considering Figure 2 and Algorithm 2, any extended probe on an input to the non-optional elements of the first register stage reveals all variables contributing to $T_i \leftarrow \mathsf{Reg}[\mathsf{F}'|_{\mathbf{X}^1=\text{BIN}(i)} \oplus R]$ for a fixed $0 \leq i < 2^n - 1$, i.e., the joint distribution over $[R, \mathbf{X}^0]$, which can be perfectly simulated with shares restricted to share domain 0 and drawing $R \in_R \mathbb{F}_2$. Further, any extended probe on the input of the second register stage reveals every stable variable contributing to $M_i \leftarrow \mathsf{Reg}[S_i \cdot T_i]$, which translates to a leakage of the joint distribution over $[T_i, \mathbf{X}^1]$. Due to the blinding with R, T_i can be simulated by drawing $T_i \in_R \mathbb{F}_2$. Hence, every observation can be simulated with shares restricted to domain 1. Eventually, placing an extended probe on the output O^1 reveals the joint distribution over $[M_0, M_1, \ldots, M_{2^n-1}]$. Here, depending on which input T_i is selected as the output, each observation will be of the form $[0, \ldots, 0, M_i, 0, \ldots, 0]$, i.e., a vector where all coordinates are zero except the one that shows a function perfectly blinded by R. This is due to the fact that by construction of the multiplexer design, only one second-stage register contains the selected input — all others contain zero. The resulting vector can be

Algorithm 2 GHPC

```
Input: input shares $\mathbf{X}^0$,$\mathbf{X}^1 \in \mathbb{F}_2^n$, fresh randomness $R \in \mathbb{F}_2$
Output: $\mathbf{F} \in \mathbb{F}_2^2, \mathbf{F} = [\mathsf{F}^0, \mathsf{F}^1] = [R, \mathsf{F} \oplus R]$
  $\mathsf{O}^1 \in \mathbb{F}_2, \mathsf{O}^1 \leftarrow 0$
  $\mathsf{O}^0 \leftarrow \mathsf{Reg}_{\mathsf{pipe}}[\mathsf{Reg}_{\mathsf{pipe}}[R]]$          ▷ Computation of $\mathsf{O}^0$

  for $\forall i \in \{0, \ldots, 2^n - 1\}$ do                 ▷ Computation of $\mathsf{O}^1$
      $S_i \leftarrow \mathsf{Reg}_{\mathsf{pipe}}[\textsc{product}(i, \mathbf{X}^1)]$
      $T_i \leftarrow \mathsf{Reg}[\mathsf{F}'|_{\mathbf{X}^1=\textsc{bin}(i)} \oplus R]$     ▷ $\textsc{bin}(i)$ is the binary representation of $i$

      $M_i \leftarrow \mathsf{Reg}[S_i \cdot T_i]$
      $\mathsf{O}^1 \leftarrow \mathsf{O}^1 \oplus M_i$
  end for

  function $\textsc{product}(V \in \mathbb{F}_2^n, \mathbf{X} \in \mathbb{F}_2^n)$
      $P \in \mathbb{F}_2, P \leftarrow 1$
      for $\forall i \in \{0, \ldots, n - 1\}$ do
          if $[\textsc{bin}(V)]_i = 1$ then     ▷ $[\textsc{bin}(V)]_i$ is the $i$-th bit of the binary representation of $V$
              $P \leftarrow P \cdot X_i$
          else
              $P \leftarrow P \cdot \overline{X_i}$
          end if
      end for
  return $P$
  end function
```

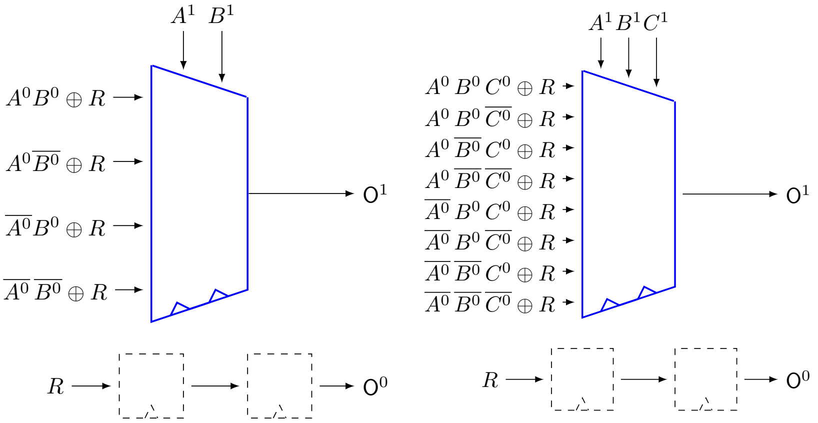
perfectly simulated by drawing a fresh random bit and placing it at the right position i of the vector. However, note that in this example, the position i depends on the shares from share domain 1, hence, in order to provide PINI-security, exchanging the indices of the circuit output shares is not allowed. Eventually, every extended probe on the output O^0 will only reveal a fresh random bit R, i.e., no information about the original input and/or output. □

Examples. In Figure 3, the designs resulting from masking a 2-input AND (Figure 3a) and a 3-input AND (Figure 3b) are given as examples. For this, we would like to highlight the clocked multiplexer symbol used in these figures, which refer to the same module identified by a blue border in Figure 2. As previously explained, the inputs to the multiplexer can be simply derived by inserting every combination of negated/non-negated shares from domain 0 into $\mathsf{F} = A\,B$. This results in $2^{n=2}$ input functions for the multiplexer design realizing a 2-input AND and in $2^{n=3} = 8$ input functions for the 3-input AND. As an extra verification, we checked these designs with SILVER [KSM20] – a software tool for formal verification of masked circuits – which confirmed our theory by reporting first-order security under the PINI notion in the robust probing model.

3.3 Reducing the Latency

If desired, in order to reduce the overall latency, the number of register stages in the design can be reduced to a single stage.

Construction Principle. For this, every input to the multiplexer must be blinded by a different freshly drawn random mask, as only this way, the second register stage in the first


A^1 B^1
$A^0B^0 \oplus R$
$A^0\overline{B^0} \oplus R$
$\overline{A^0}B^0 \oplus R$
$\overline{A^0}\,\overline{B^0} \oplus R$
O^1
R
O^0
$A^1 B^1 C^1$
$A^0\,B^0\,C^0 \oplus R$
$A^0\,B^0\,\overline{C^0} \oplus R$
$A^0\,\overline{B^0}\,C^0 \oplus R$
$A^0\,\overline{B^0}\,\overline{C^0} \oplus R$
$\overline{A^0}\,B^0\,C^0 \oplus R$
$\overline{A^0}\,B^0\,\overline{C^0} \oplus R$
$\overline{A^0}\,\overline{B^0}\,C^0 \oplus R$
$\overline{A^0}\,\overline{B^0}\,\overline{C^0} \oplus R$
O^1
R
O^0

(a) 2-input AND

(b) 3-input AND

Figure 3: Examples for first-order PINI-secure GHPC constructions.

Algorithm 3 $\mathsf{GHPC_{LL}}$

Input: input shares $\mathbf{X}^0$,$\mathbf{X}^1 \in \mathbb{F}_2^n$, fresh randomness $\mathbf{R} \in \mathbb{F}_2^{2^n}$
Output: $\mathbf{F} \in \mathbb{F}_2^2, \mathbf{F} = [\mathsf{F}^0, \mathsf{F}^1] = [R, \mathsf{F} \oplus R]$, with $R \in \mathbf{R}$

```
$R \in \mathbb{F}_2, R \leftarrow 0$
$\mathsf{O}^1 \in \mathbb{F}_2, \mathsf{O}^1 \leftarrow 0$
for $\forall i \in \{0, \ldots, 2^n - 1\}$ do
    $S_i \leftarrow \mathsf{Reg_{pipe}}[\textsc{product}(i, \mathbf{X}^1)]$        ▷ PRODUCT(.) as defined in Algorithm 2
    $T_i \leftarrow \mathsf{Reg}[\mathsf{F}'|_{\mathbf{X}^1=\textsc{bin}(i)} \oplus R_i]$
    $M_i \leftarrow S_i \cdot T_i$
    $R \leftarrow R \oplus S_i \cdot R_i$
    $\mathsf{O}^1 \leftarrow \mathsf{O}^1 \oplus M_i$
end for
$\mathsf{O}^0 \leftarrow \mathsf{Reg}[R]$
```

circuit share can be entirely omitted. For the output of circuit share 0, i.e., for O^0, the corresponding randomness has to be selected by a standard multiplexer before it is stored in a register. The resulting design with one register stage – referred to as $\mathsf{GHPC_{LL}}$ in the following – can be seen in Figure 4 and in Algorithm 3. The architecture of circuit share 1 uses the same multiplexer architecture as presented in Figure 2, but with the second register stage omitted. Hence, this design has a reduced latency of 1 clock cycle, while expanding the demand for fresh randomness to 2^n bits.

Security Analysis. Again, we briefly prove the correctness and security under the notion of PINI, as stated in Theorem 3, assuming the glitch-extended d-probing model.

Theorem 3. *For an arbitrary circuit* C*, realizing a Boolean function* $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2$*, the transformation into a* $\mathsf{GHPC_{LL}}$ *results in a correct and first-order PINI-secure circuit under the glitch-extended probing model.*

Proof.

Correctness: Following the same argumentation as for Theorem 2, O^1 outputs $\mathsf{F} \oplus R_i$ with $i \in \{0, \ldots, 2^n - 1\}$ and for any valid input sharing. As by construction, O^0 equals R_i, correctness of this masking is fulfilled.

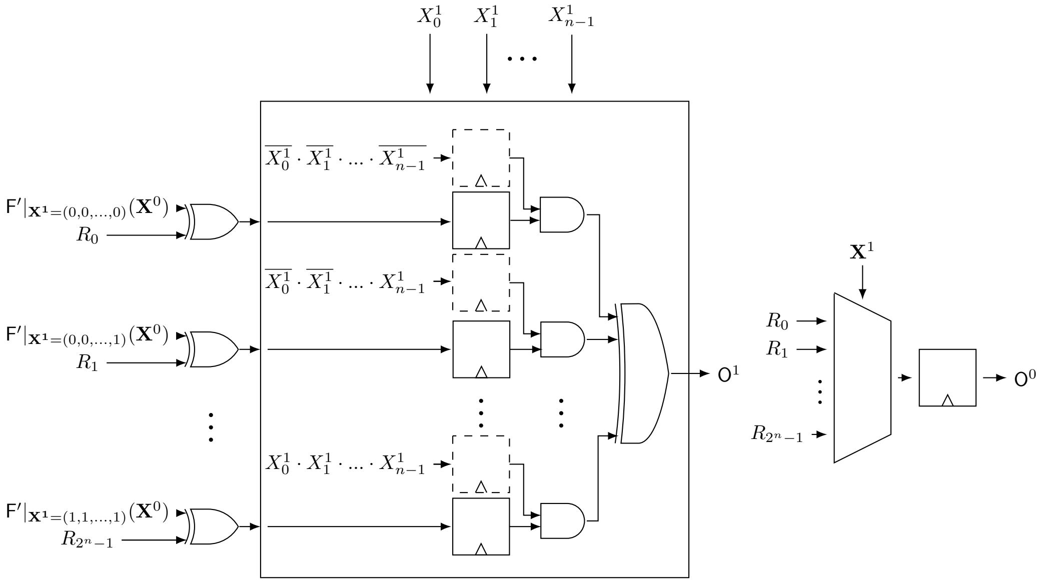

X_0^1
X_1^1
X_{n-1}^1
$\overline{X_0^1} \cdot \overline{X_1^1} \cdot \ldots \cdot \overline{X_{n-1}^1}$
$\overline{X_0^1} \cdot \overline{X_1^1} \cdot \ldots \cdot X_{n-1}^1$
$X_0^1 \cdot X_1^1 \cdot \ldots \cdot X_{n-1}^1$
$\mathsf{F}'|_{\mathbf{X}^1=(0,0,\ldots,0)}(\mathbf{X}^0)$
R_0
$\mathsf{F}'|_{\mathbf{X}^1=(0,0,\ldots,1)}(\mathbf{X}^0)$
R_1
$\mathsf{F}'|_{\mathbf{X}^1=(1,1,\ldots,1)}(\mathbf{X}^0)$
R_{2^n-1}
O^1
$\mathbf{X}^1$
R_0
R_1
R_{2^n-1}
O^0

Figure 4: General $\mathsf{GHPC}_{\mathsf{LL}}$ design with reduced latency.

PINI: Considering Figure 4 and Algorithm 3, the joint distribution $[\mathbf{X}^0, R_i]$ using a probe on the input of $T_i \leftarrow \mathsf{Reg}[\mathsf{F}'|_{\mathbf{X}^1=\mathrm{BIN}(i)} \oplus R_i]$ can be simulated using only shares from domain 0 and drawing $R_i \in_R \mathbb{F}_2$. An extended output probe on O^1 will observe a joint distribution of the form $[\mathbf{X}^1, T_0, T_1, \ldots, T_{2^n-1}]$, which can be simulated with $\mathbf{X}^1$ and drawing 2^n fresh random bits as $T_i \in_R \mathbb{F}_2$, $\forall 0 \leq i < 2^n - 1$. Due to the output register, a probe on O^0 can be perfectly simulated using one fresh random bit, while each internal probe (on R) is perfectly simulatable by fresh randomness and shares drawn only from share domain 1, as the whole circuit share does not involve computation on shares from share domain 0. □

4 Comparisons

In this section, we briefly discuss and compare our proposed constructions to state-of-the-art masking schemes with respect to common metrics, such as latency, demand for fresh randomness, area consumption, and composability of gadgets. To this end, Table 1 lists recent approaches from literature and their application to hardware circuits. In particular, we align our discussion and comparison by focusing on basic non-linear gates, i.e., 2-input AND gates, which are commonly used to create secure and composable gadgets required for construction of larger circuits. Further, we extend our discussion by comparing different techniques with respect to larger circuits, particularly using the PRESENT, PRINCE, Skinny, Prøst, Rectangle, Class-13, and AES S-boxes as illustrating examples. For the 4-bit S-boxes, we realized the corresponding descriptions given in [CGLS20] which are optimized with respect to the number of cascaded 2-input AND gates, i.e., favoring HPC1 and HPC2 as instantiated gadget. For the AES S-box, we considered the design given in [BP12], where – based on a tower field representation – a low-depth circuit has been constructed. It contains two isomorphisms at the start and end of the GF(2^8) inversion, and excluding the XOR gates, has at most 4 cascaded 2-input AND gates, which also is in favor of HPC1 and HPC2. In addition, we also include results reported in [GIB18], proposing a generic approach for low-latency masking which has been applied to the AES S-box considering different low-latency constructions.

Further, note that the descriptions given in [CGLS20] are without considering pipeline

Table 1: Comparison of different first-order masking schemes. (using Synopsis Design Compiler and UMC 180 standard cell library)

Target	Scheme	Func.		Latency	Rand.	Area	Compos.	Ref.
		n	m	[cycle]	[bit]	[GE]	notion	
AND2	DOM	2	1	2	**1**	56	SNI	[FGP+18]
	HPC1			2	2	94	PINI	[CGLS20]
	HPC2			2	**1**	66	PINI	[CGLS20]
	GHPC			2	**1**	82	PINI	**new**
	GHPC$_{LL}$			**1**	4	**59**	PINI	**new**
PRESENT S-box	HPC1	4	4	3	8	403	PINI	[CGLS20]
	HPC2			3	**4**	320	PINI	[CGLS20]
	GHPC$_{LL}$-AND			2	16	**310**	PINI	**new**
	GHPC			2	**4**	1308	PINI	**new**
	GHPC$_{LL}$			**1**	64	959	PINI	**new**
PRINCE S-box	HPC1	4	4	4	12	645	PINI	[CGLS20]
	HPC2			4	6	467	PINI	[CGLS20]
	GHPC$_{LL}$-AND			2	24	**445**	PINI	**new**
	GHPC			2	**4**	1384	PINI	**new**
	GHPC$_{LL}$			**1**	64	987	PINI	**new**
Skinny S-box	HPC1	4	4	4	8	467	PINI	[CGLS20]
	HPC2			4	**4**	301	PINI	[CGLS20]
	GHPC$_{LL}$-AND			2	16	**288**	PINI	**new**
	GHPC			2	**4**	1232	PINI	**new**
	GHPC$_{LL}$			**1**	64	951	PINI	**new**
Prøst S-box	HPC1	4	4	3	8	432	PINI	[CGLS20]
	HPC2			3	**4**	309	PINI	[CGLS20]
	GHPC$_{LL}$-AND			2	16	**302**	PINI	**new**
	GHPC			2	**4**	1225	PINI	**new**
	GHPC$_{LL}$			**1**	64	952	PINI	**new**
Rectangle S-box	HPC1	4	4	3	8	439	PINI	[CGLS20]
	HPC2			3	**4**	319	PINI	[CGLS20]
	GHPC$_{LL}$-AND			2	16	**311**	PINI	**new**
	GHPC			2	**4**	1229	PINI	**new**
	GHPC$_{LL}$			**1**	64	962	PINI	**new**
Class-13 S-box	HPC1	4	4	3	8	432	PINI	[CGLS20]
	HPC2			3	**4**	304	PINI	[CGLS20]
	GHPC$_{LL}$-AND			2	16	**303**	PINI	**new**
	GHPC			2	**4**	951	PINI	**new**
	GHPC$_{LL}$			**1**	64	933	PINI	**new**
AES S-box	CMS	8	8	5	54	2530	-	[CRB+16]
	DOM			8	18	2851	-	[GMK17]
	GLLM			1	2048	60730[2]	-	[GIB18]
	GLLM			2	416	6740[2]	-	[GIB18]
	HPC1			5	68	3504	PINI	[BP12]
	HPC2			5	34	2452	PINI	[BP12]
	GHPC$_{LL}$-AND			4	136	**2376**	PINI	**new**
	GHPC			2	**8**	77145	PINI	**new**
	GHPC$_{LL}$			**1**	2048	64111	PINI	**new**

[2] *These designs were synthesized using a different UMC 90 nm process technology.*

registers to synchronize the inputs of each gate. Therefore, in order to provide a fair comparison, all performance figures reported in Table 1 are for non-pipeline designs. Besides, similar to the state of the art, we did not include the area required for generation of fresh masks in the reported area footprints. All area results have been obtained by synthesizing the Hardware Description Language (HDL) code of the design using Synopsis

Design Compiler and UMC 180 nm standard cell library, unless indicated otherwise.

Latency. Since the final latency of our constructions does not depend on the underlying Boolean function and its algebraic degree, its application on larger functions leads to a higher efficiency with respect to latency compared to other approaches. Certainly, for small circuits and simple Boolean functions, e.g., a 2-input AND gate, hand-crafted and optimized gadgets might be more efficient in terms of latency. However, as our approach easily scales for larger functions, even with high algebraic degree, e.g., for an entire AES S-box, construction of masked circuits with low latency becomes feasible. More specifically, focusing on our low-latency approach $\mathsf{GHPC}_{\mathsf{LL}}$ (at cost of additional randomness), our secure constructions outperform all other schemes listed in Table 1 in terms of latency. More precisely, to the best of our knowledge, the $\mathsf{GHPC}_{\mathsf{LL}}$ is the only first-order composable construction with one clock cycle latency.

Although GroÃ§ et al. proposed a generic low-latency masking approach in [GIB18], the resulting AES S-box constructions certainly have a comparable latency (along with demand for fresh randomness and area), but do not result in a composable design but only focus on proving a probing-secure construction.

Randomness. In contrast to latency, the demand for fresh randomness of our constructions is mainly governed by the underlying Boolean function. The number of required fresh random bits r per circuit evaluation is given as

$$r_{\mathsf{GHPC}} = m, \qquad r_{\mathsf{GHPC}_{\mathsf{LL}}} = m \cdot 2^n.$$

More precisely, for GHPC this number is independent of the number of inputs n but only depends on the number of outputs m of the underlying Boolean function $\mathbf{F} : \mathbb{F}_2^n \to \mathbb{F}_2^m$. Hence, for large functions, such as 4-bit or 8-bit S-boxes, this results in randomness-efficient designs. This view for sure changes when lower latency is favorable, i.e., $\mathsf{GHPC}_{\mathsf{LL}}$ whose required fresh randomness depends on both m and n.

Then again, efficiency of our approach, in terms of required fresh randomness, does not change with the optimizations done on the implementation of the Boolean function. Taking the HPC1 and HPC2 of the S-boxes covered by Table 1, the foundational S-box implementations have been optimized through application of SAT solvers in order to reduce the latency and number of 2-input AND gates [CGLS20, BP12]. However, for our approach, the number of random bits is independent of how the Boolean function is realized. Instead, it only depends on its number of input and output bits. For this, our approach is particularly suitable for integration into EDA tools and automated integration of masking countermeasures into logic circuits.

Area. Besides latency and demand for fresh randomness during execution, the footprint in terms of area of the resulting design is an often considered metric in evaluation of efficiency and expense of a final design. In this regard, reduction in area usually can be traded for increasing latency and demand for fresh randomness.

In turn, this implies that our proposals optimized for low-latency ($\mathsf{GHPC}_{\mathsf{LL}}$) and low-randomness (GHPC) naturally are outperformed by hand-crafted and sophisticatedly optimized gadgets and constructions. However, observing that our $\mathsf{GHPC}_{\mathsf{LL}}$ construction for a 2-input AND not only provides best results in latency but also is smaller than all related (PINI-secure) hand-crafted gadgets, we opted to instantiated all S-box constructions provided in Table 1 with our $\mathsf{GHPC}_{\mathsf{LL}}$ gadget instead of HPC1 or HPC2. Given this, we can observe that all our $\mathsf{GHPC}_{\mathsf{LL}}$-AND S-box constructions outperform related designs in terms of area (and even latency to some extent), but at cost of additional randomness that is required for secure execution. Construction of larger $\mathsf{GHPC}_{\mathsf{LL}}$-AND gadgets (3- or 4-bit input) to be used in the implementation of the S-boxes is also possible, but since the

S-box descriptions we have in hand are not optimized to efficiently use such large-input gates, we have not included such cases in the presented results.

Composability. Eventually, our construction allows to build secure and composable hardware gadgets from arbitrary Boolean functions. As a consequence, even entire S-boxes, as shown in Table 1 can be transformed into securely-composable gadgets under the PINI notion. However, in contrast to the existing designs focusing on construction of secure circuits through composition of secure AND and XOR gates, our approach efficiently scales for arbitrary Boolean functions. For instance, AES S-box designs presented in [GMK17, CRB+16] rely on a careful instantiation of secure 2-input (and larger) AND gates. However, as these gadgets are not trivially composable, the resulting S-box circuit is indeed probing secure, but does not necessarily provide composability.

In contrast to this, our approach always results in PINI-secure gadgets, independent of the underlying function, allowing to construct gadgets even for larger circuits such as the AES S-box. As a result, we can conclude that due to its flexibility, our approach provides a clear road map for automatization of masking arbitrary circuits through generation of composable secure gadgets. It is true that a secure variant of any circuit can be constructed by HPC1 and HPC2 2-input AND (and XOR) gadgets, but there is a lower bound for the latency of the resulting circuit – which is of crucial importance in hardware designs – defined by the algebraic degree of the components of the underlying cryptographic function. However, our scheme uncouples this dependency while maintaining the same generality.

5 Case Studies

Below, we present the experimental results obtained when applying our construction principle to different block cipher implementations.

5.1 Target Device and Measurement Setup

The analyses have been conducted on a SAKURA-G board [SAK], where a Spartan-6 Field-Programmable Gate Array (FPGA) is embedded to host cryptographic cores. For all case studies given in the remainder of this section, the power consumption traces of the target FPGA have been collected by monitoring the voltage drop over a 1 Ω resistor placed in the Vdd path amplified by an on-board AC amplifier. During the measurements performed by a digital oscilloscope at the sampling rate of 500 MS/s, the implemented cryptographic core was supplied by a stable and jitter-free clock source at the frequency of 6 MHz.

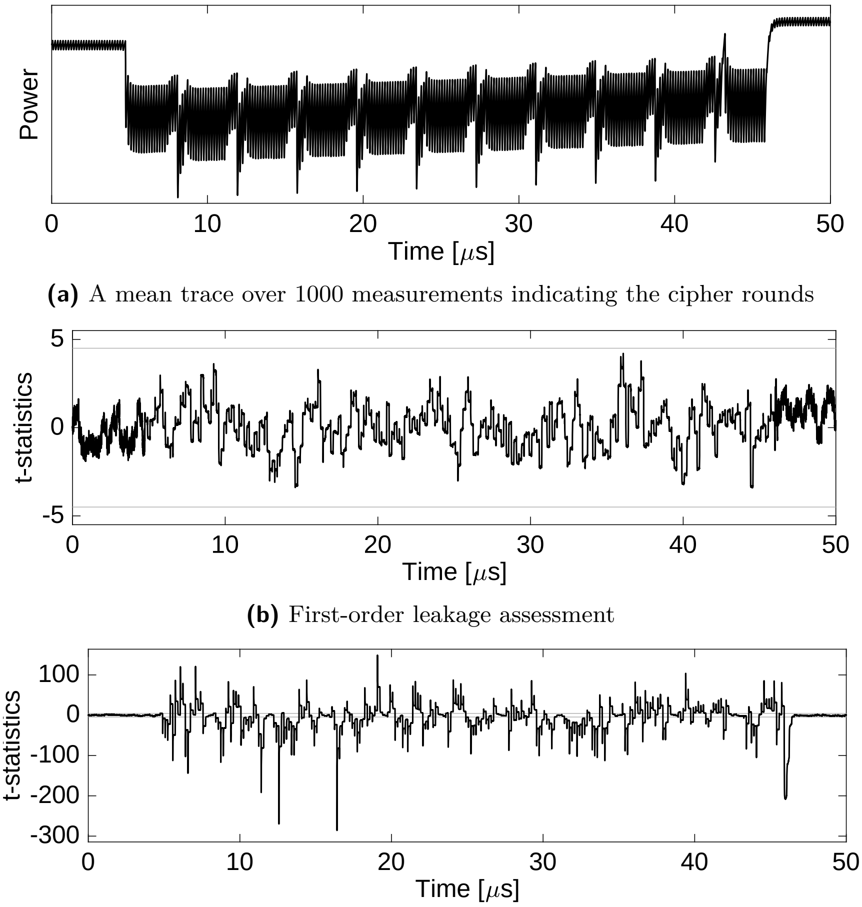
PRNG. For the generation of each fresh random bit, we constructed a 31-bit Linear-Feedback Shift Register (LFSR) with the feedback polynomial $x^{31} + x^{28} + 1$, which has a maximum cycle of $2^{31} - 1$ with only two taps [WM12]. Each LFSR is initialized by an arbitrary value right after the FPGA power-up, making sure that no LFSR is entirely filled by zero, and there is no common initialization value for two LFSRs.

5.2 Byte-Serial AES

For our first case study, we opted to implement a first-order secure AES encryption based on the byte-serial architecture of Moradi et al. [MPL+11] with only minor modifications in the control logic due to the increased S-box latency. It is worth to highlight that all our HDL designs of the case studies are provided in the GitHub: https://github.com/Chair-for-Security-Engineering/GHPC.

Design. For this, a single GHPC AES S-box is instantiated and shared between data path and key expansion circuits, requiring both, data and key to be shared using $d+1 = 2$ shares. Further, due to the two-cycle latency of the GHPC S-box design, and this component being the bottleneck of the architecture, we opted to include all pipelining registers to enable processing of all byte substitutions within 22 cycles, i.e., 2 cycles initial latency, 16 cycles for the round function S-box computations, and 4 cycles for the key expansion. Further, shifting of rows and mixing of columns is done in one respectively four cycles, while the key is updated simultaneously. In total, a single first-order secure AES round function (including key expansion) requires 23 cycles, resulting in a total of 230 cycles for an AES-128 encryption. Note that, although mixing of columns is omitted in the last round, expansion of the final post-whitening key is stalling the final round computation. We also provide a generic HDL description of our architecture in the GitHub: https://github.com/Chair-for-Security-Engineering/GHPC which allows to select GHPC or $\mathsf{GHPC}_{\mathsf{LL}}$ as the underlying design while adjusting the S-box, control logic, and randomness automatically (see more details in Appendix A). However, we considered only the GHPC design in our experimental analyses due to the similarity of the results.

Then, as given in Table 2, our entire first-order AES encryption architecture has a size of 86.3 kGE, using an 180 nm cell library while it requires only 8-bit fresh randomness per



(a) A mean trace over 1000 measurements indicating the cipher rounds

(b) First-order leakage assessment

(c) Second-order leakage assessment

Figure 5: Experimental analysis of our first-order AES byte-serial encryption design (covering the entire encryption); fixed vs. random t-test results using 100 million traces.

Table 2: Performance figures of our case studies.
(using Synopsis Design Compiler and UMC 180 standard cell library)

Design	Scheme	Order	Random.	Area	Delay	Latency
		d	[bit]	[GE]	[ns]	[cycle]
AES Serial	unprotected	0	0	3646	8.13	195
	GHPC	1	8	86 326	21.34	215
	GHPC$_{\text{LL}}$	1	2048	76 339	23.48	205
PRESENT Serial	unprotected	0	0	2139	5.12	545
	GHPC	1	4	5604	5.76	607
	GHPC$_{\text{LL}}$	1	64	5253	5.92	576
PRESENT Round-based	unprotected	0	0	2798	4.39	31
	GHPC	1	72	31 559	5.22	62
	GHPC$_{\text{LL}}$	1	1152	25 264	5.28	31

clock cycle to maintain the first-order security.

Leakage Assessment. Using a fix-versus-random Test Vector Leakage Assessment (TVLA) methodology according to [SM15], Figure 5b and Figure 5c show evaluation results for first-order and second-order statistical moments using 100 million power traces. As expected, our design does not exhibit any observable leakage for the first-order statistical moment while expectedly we could observe significant differences in the second-order statistical moment. These results indeed confirm our theoretical security evaluations for the GHPC construction, showing its applicability to arbitrary Boolean functions in order to construct generic and composable PINI-secure gadgets.

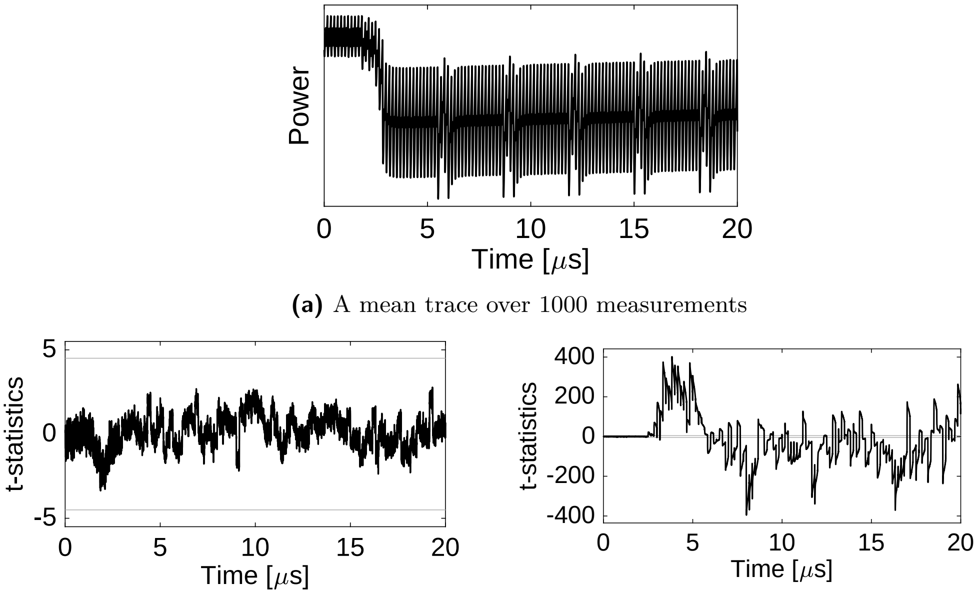
5.3 Nibble-Serial PRESENT

For our second case study, we implemented the nibble-serial design of Poschmann et al. [PMK+11], realizing the PRESENT encryption where a single S-box instance is shared for the entire data and key processing. Per clock cycle, both state and key registers are shifted nibble-wise to conduct key addition and S-box look-up at the same time, while the permutation layer is done in parallel (in a single clock cycle). Again, we constructed a general design, in which the user can set the desired GHPC or GHPC$_{\text{LL}}$ scheme. The number of required fresh masks as well as the latency of the S-box, required for the control logic, is automatically adjusted accordingly. For more detail on the design architecture, we refer to the HDL code given in the GitHub: https://github.com/Chair-for-Security-Engineering/GHPC. Table 2 also lists the performance figures of our designs including the area overhead, required fresh randomness, latency, and delay.

Focusing on our GHPC design, we collected 100 million traces and performed fix-versus-random TVLA at different orders. The results shown in Figure 6 confirm our claims and expectations on the security level of our construction.

5.4 Round-Based PRESENT

We also implemented the PRESENT encryption function in a round-based fashion. The unprotected design performs each cipher round in a single clock cycle, resulting in 31 cycles for the entire encryption. The first-order GHPC design needs 2 clock cycles per round while forming a pipeline design, i.e., encrypting two plaintexts in consecutive clock cycles, resulting in 62 clock cycles for two encryptions. Note that we made use of the internal registers of the S-box as the state register. This allowed us to keep 31 clock cycle

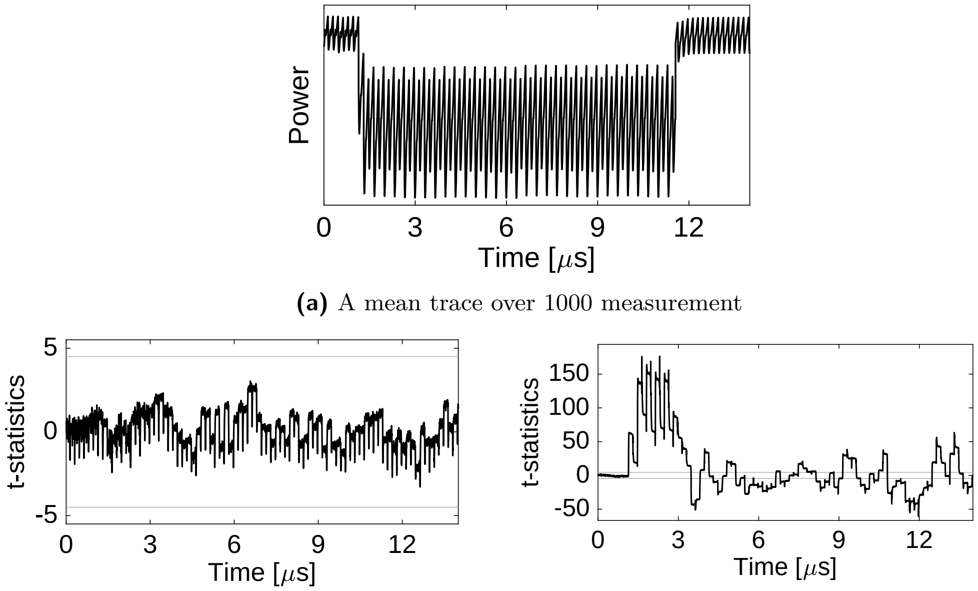


(a) A mean trace over 1000 measurements

(b) First-order

(c) Second-order

Figure 6: Experimental analysis of our first-order PRESENT nibble-serial encryption design (covering the first five rounds); fixed vs. random t-test results using 100 million traces.



(a) A mean trace over 1000 measurement

(b) First-order

(c) Second-order

Figure 7: Experimental analysis of our first-order PRESENT round-based encryption design (covering the entire encryption); fixed vs. random t-test results using 100 million traces.

latency in $\mathsf{GHPC}_{\mathsf{LL}}$ design (see Table 2). Similar to all other case studies, we practically examined this construction by performing the same leakage assessment at different orders. The results, which are along the same line as the formerly presented ones, are shown in

Figure 7. Nevertheless, Table 2 covers the performance figures of this design as well.

As a remark, the evaluation results of the $\mathsf{GHPC}_{\mathsf{LL}}$ circuits of all case studies are very similar to the figures presented above. Therefore, we omit showing the identical results.

6 Discussions and Conclusions

In this work, we developed and presented a generic framework to construct trivially composable hardware private circuits with a compact latency from arbitrary vectorial Boolean functions. Following the concept of Shannon's decomposition, we derived generic circuit constructions which offer both, first-order probing security in the presence of glitches and trivial composability, by fulfilling the notion of PINI in the robust probing model. More specifically, we presented the fundamental design principles, security analyses, and simple examples to illustrate our contribution.

After establishing the concept, we compared our constructions to state-of-the-art masking schemes. Based on this comparison, we conclude that our approach can be used to achieve optimized designs for different metrics, in particular focusing on latency, randomness, and area.

In terms of **latency**, our proposed $\mathsf{GHPC}_{\mathsf{LL}}$ gadgets outperform all handcrafted and carefully optimized constructions, independent of the complexity and appearance of the underlying Boolean function. In fact, all our presented results constructed according to the $\mathsf{GHPC}_{\mathsf{LL}}$ approach have the smallest latency of a single cycle, while still being PINI-secure, however, introducing higher demands on area and fresh randomness.

Then, considering **low-randomness** optimizations, our GHPC provides the best results in comparison to related works due to its independence on the targeted Boolean function and its appearance, while the number of random bits only depends on the number of outputs the underlying function has. Focusing on the designs with low fresh randomness, our GHPC approach scales well particularly for larger functions, e.g., an AES S-box, with only modest increase of latency in comparison to the $\mathsf{GHPC}_{\mathsf{LL}}$ approach while requiring only 8 bit fresh randomness as it is a function with 8-bit output.

Eventually, **low-area** constructions for all S-boxes can be achieved by replacing the HPC1 or HPC2 2-input AND gadget with our proposed $\mathsf{GHPC}_{\mathsf{LL}}$-AND construction. Interestingly, such constructions provide the smallest area footprint along with competitive latency. This however comes at cost of increased demand for fresh randomness.

Furthermore, our methodology was verified by (i) testing small examples with the state-of-the-art formal verification tool SILVER [KSM20], and by (ii) experimentally evaluating several case studies. As a first case study for our first-order approach, we decided to analyze a byte-serialized version of AES where the entire S-box was translated into a single first-order secure and composable gadget based on our presented design principle. For a second case study, we constructed a secure nibble-serialized and round-based PRESENT encryption, again translating the entire S-box into a single gadget following the corresponding first-order design concepts. All case studies evidently support our theoretical findings by showing no leakage in the first-order statistical moment when performing a non-specific leakage assessment.

To conclude, our generic methodology for constructing masked gadgets for arbitrary vectorial Boolean functions pioneers automatic generation of masked circuits in hardware solely based on the function expression and with a constant latency of 2 clock cycles for GHPC and 1 clock cycle for $\mathsf{GHPC}_{\mathsf{LL}}$ (at cost of higher fresh randomness). A fundamental question, which is not yet answered and still needs proper attention, is how expensive it is to generate a certain number of fresh masks per clock cycle. For this, choosing area, energy, power, or delay as the metric and cost function are certainly possible choices. However, without any detailed insight on the cost factors, we cannot easily prefer one design over another even though they have, for example, the same latency.

Since the presented solution is restricted to the first order with 2 shares, extension of the technique to cover higher-order security is naturally among our future works. As our approach pioneers the automated construction of masked hardware circuits, development of a proper tool is an interesting exercise to pursue. In this context, exploration of trade-offs between randomness and latency for functions larger than 4 bits through clever construction of atomic gadgets is an interesting question also left open for future work.

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 project 393207943 GreenSec.

References

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private Circuits: A Modular Approach. In *CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455. Springer, 2018.

[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.

[Boo48] George Boole. The calculus of logic. 1848.

[BP12] Joan Boyar and René Peralta. A Small Depth-16 Circuit for the AES S-Box. In *Information Security and Privacy Conference, SEC 2012*, volume 376 of *IFIP*, pages 287–298. Springer, 2012.

[CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Transactions on Computers*, 2020.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CRB+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 *Lecture Notes in Computer Science*, 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. Information Forensics and Security*, 15:2542–2555, 2020.

[DBR19] Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. Consolidating Security Notions in Hardware Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):119–147, 2019.

[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.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic Low-Latency Masking in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[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 *LNCS*, pages 95–112. Springer, 2017.

[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.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96*, 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.

[MPL$^+$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 *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptol.*, 24(2):292–321, 2011.

[PMK$^+$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. Cryptology*, 24(2):322–345, 2011.

[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.

[SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.

[Tri03] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.*, 2003:236, 2003.

[WM12] Roy Ward and Timothy C.A. Molteno. Table of Linear Feedback Shift Registers. Technical Report 2012-1, University of Otago, 2012. http://www.physics.otago.ac.nz/reports/electronics/ETR2012-1.pdf.

A Appendix

We have provided the HDL code of our case studies of Section 5 in the GitHub: https://github.com/Chair-for-Security-Engineering/GHPC. A “PINI_pkg.vhd” file is given for each design, where the settings of the desired implementation can be adjusted. This includes parameters like “low_latency” with which the gadget type (GHPC/ $\mathsf{GHPC}_{\mathsf{LL}}$) can be selected, and “pipeline” which sets if pipeline registers should be instantiated into the designs.

We further constructed the designs in such a way that it can easily realize different Boolean functions (i.e., different S-boxes). In the “PINI_pkg.vhd” file, the number of input bits and the output bits (via parameters “in_size” and “out_size”) can be adjusted, and the target Boolean function can be set as a case statement of the “PINI_Step1.vhd” file as a look-up table. This eases the process of automatic generation of GHPC and $\mathsf{GHPC}_{\mathsf{LL}}$ gadgets of arbitrary Boolean functions.