Randomness Optimization for Gadget Compositions in Higher-Order Masking

Jakob Feldtkeller[1], David Knichel[1], Pascal Sasdrich[1], Amir Moradi[2] and Tim Güneysu[1,3]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
firstname.lastname@rub.de
[2] University of Cologne, Institute for Computer Science, Germany
firstname.lastname@uni-koeln.de
[3] DFKI, Bremen, Germany

Abstract. Physical characteristics of electronic devices, leaking secret and sensitive information to an adversary with physical access, pose a long-known threat to cryptographic hardware implementations. Among a variety of proposed countermeasures against such Side-Channel Analysis attacks, *masking* has emerged as a promising, but often costly, candidate. Furthermore, the manual realization of masked implementations has proven error-prone and often introduces flaws, possibly resulting in insecure circuits. In the context of automatic masking, a new line of research emerged, aiming to replace each physical gate with a secure gadget that fulfills well-defined properties, guaranteeing security when interconnected to a large circuit. Unfortunately, those gadgets introduce a significant amount of additional overhead into the design, in terms of area, latency, and randomness requirements.
In this work, we present a novel approach to reduce the demands for randomness in such gadget-composed circuits by reusing randomness across gadgets while maintaining security in the probing adversary model. To this end, we embedded the corresponding optimization passes into an Electronic Design Automation toolchain, able to construct, optimize, and implement masked circuits, starting from an unprotected design. As such, our security-aware optimization offers an additional building block for existing or new Electronic Design Automation frameworks, where security is considered a first-class design constraint.

Keywords: Masking · Probing Security · Strong Non-Interference · Probe Isolating Non-Interference · Security-Aware Optimization · Security-Aware EDA

1 Introduction

In 1999, Paul Kocher introduced Differential Power Analysis (DPA) as a powerful threat to the security of cryptographic hardware and devices [KJJ99]. In the aftermath, this remarkable work sparked interest and innovation in entirely new branches of research dealing with Side-Channel Analysis (SCA), showing that the observation of physical characteristics of an electronic device, such as timing behavior [Koc96], instantaneous power consumption [KJJ99], electromagnetic (EM) radiations [GMO01], or temperature and heat dissipation [HS13], can reveal secret and sensitive information to any observer and adversary with physical access. Over the last two decades, researchers, among other things, pursued research covering the development of novel *side-channel attacks*, the enhancement of *analysis techniques*, and the design and implementation of *effective countermeasures*.

In this context, *masking* (based on the concepts of *secret sharing*) has been established as the most promising candidate for protection against SCA due to its theoretical and

sound security foundations [CJRR99]. Consequently, many different schemes and variants have been proposed and implemented in hardware over the last years [ISW03, Tri03, NRS11, RBN+15, GMK17, GM18], of which not a few have been shown to be insecure due to design flaws or inaccurate assumptions, e.g., [MMSS19]. Even today, the design and implementation of secure hardware is still a mostly manual and error-prone process, requiring long-standing expertise in hardware design and security. Consequently, recent research focuses on novel approaches and concepts facilitating the implementation of security in hardware through accurate models and consolidated security notions.

Security and Composability Notions. In the context of *hardware masking*, the simple and abstract Ishai-Sahai-Wagner (ISW) t-probing adversary and security model [ISW03] provides strong theoretical foundations and clear formal models for adversaries and physical execution environments of modern hardware devices. In its basic form, the t-probing model grants an adversary access to up to t intermediate values of an ideal circuit during the processing of sensitive information. Extended with unconsidered and unintentional physical effects, including *glitches* [MPG05, MS06], *transitions* [CGP+12, BGG+14], and *couplings* [CBG+17], the *robust t-probing model* [FGP+18] provides an accurate verification model to reason about the security of masked digital logic circuits. Unfortunately, the design and implementation of masked circuits is a non-trivial problem, as complexity increases with the design size, and the combination and composition of masked circuits may result in insecure constructions.

Accordingly, modern approaches endeavor to extend security to larger circuits through the composition of atomic masked circuits, often denoted as *gadgets*, guaranteeing security in the t-probing model when interconnected. To this end, novel security notions, such as Non-Interference (NI) [BBD+15], Strong Non-Interference (SNI) [BBD+16], and Probe Isolating Non-Interference (PINI) [CS20], have been proposed recently, defining gadget-contained properties that enable secure composition. Although the security and composability notions assist in the construction and verification of secure circuits, each gadget instantiation comes with additional overhead in terms of *latency increase*, *area occupation*, and *randomness demand*. More specifically, the local overhead for each gadget, introduced through the composability notions, accumulates for the entire circuit and often results in non-optimal solutions.

Security-Aware Construction and Optimization. Even beyond performance and area overhead, the composed constructions are often more secure than strictly required, as composability properties are often maintained even for the combined circuit, although eventually, only security against the t-probing adversary is required. For this, modern security-aware Electronic Design Automation (EDA) tools should not only assist designers in the construction of secure circuits through combining and composing masked gadgets but also provide ways and means to optimize the final construction in terms of area or performance while maintaining the desired level of security. This becomes even more important and pronounced for higher-order masking, as the performance and area penalty usually grows exponentially in the security order.

While there are already some initial concepts and tools to design and verify composed circuits [BGR18, CGLS21, KMMS22], all of the given tools maintain gadget barriers and boundaries without further optimization and improvement of the final masked circuits. Consequently, this leads to the question, whether dissolving gadget barriers and boundaries can help to share resources and, hence, decrease the overhead introduced through conservative security notions.

Our Contributions. Given these observations and research questions, we present a novel approach to optimize and reduce the final randomness consumption and requirements

of masked circuits created through the composition of securely masked gadgets. Set within this context, we attempt to use a holistic view of the composed circuit to identify clusters of gadgets that can share and re-use randomness while maintaining an appropriate level of security under the t-probing adversary model. More precisely, our approach is designed and well-suited for higher-order masking (i.e., $t \geq 2$) using state-of-the-art gadget constructions created under the SNI and PINI composability notions. Embedded into an efficient EDA flow, using the Multi-Level Intermediate Representation (MLIR) framework as a fundamental basis, we present novel optimization strategies and passes that allow constructing, optimizing, and implementing masked circuits, starting from an unprotected design. Depending on the context and application scenario, our optimization strategies allow reduction of the randomness demand of a masked AES-128 by up to 13% when utilizing state-of-the-art PINI gadgets and up to 94% when using SNI gadgets.

Outline. We first provide some core concepts and definitions used throughout the paper in Section 2. Then, in the main part of this paper, we introduce our optimization techniques for both SNI and PINI compositions in Section 3 and Section 4, respectively. In Section 5 we describe the implementation and integration of our optimization passes into an EDA toolchain for construction and optimization of masked circuits while we evaluate the performance and assess the practical security of our randomness reduction techniques in Section 6. Section 7 discusses existing and related work as well as possible extensions for the future, before we conclude in Section 8.

2 Preliminaries

In the following, we introduce necessary notations and recall the most important definitions from the literature required throughout the paper.

2.1 Notations

In Table 1, we provide a summary of important notation and variables used throughout this work. Further, calligraphic fonts are used to denote sets while we use superscripts to indicate the index of a gadget within a set, while subscripts are used for indices of shares and random bits.

2.2 Modeling and Masking Circuits

Any logic circuit realizing a Boolean function $\mathsf{F} : \mathbb{F}_2^u \mapsto \mathbb{F}_2^w$, can be abstracted as a graph [ISW03, FGP+18]. More formally, any stateful and deterministic circuit C can be modeled as a Directed Acyclic Graph (DAG) $\mathsf{D}_\mathsf{C} = \{\mathcal{V}, \mathcal{E}\}$, with vertices $v \in \mathcal{V}$ describing either combinatorial gates or synchronization elements, i.e., registers, and edges $e \in \mathcal{E}$ describing wires carrying elements drawn from $\mathbb{F}_2$.

Boolean Masking. Boolean masking splits a secret $x \in \mathbb{F}_2^u$ into $d > 1$ independent and uniformly distributed shares $x_i \in \mathbb{F}_2^u, 0 \leq i < d$, such that $x = \bigoplus_{i=0}^{d-1} x_i$. For this, each x_i for $0 \leq i < d-1$ is usually sampled from a uniform random distribution over $\mathbb{F}_2^u$, while the remaining share is derived as $x_{d-1} = \bigoplus_{i=0}^{d-2} x_i \oplus x$.

Encoded Circuit Model. The secure computation of $y = \mathsf{C}(x)$ can be formally defined through a *circuit compiler* [AIS18] consisting of three algorithms, i.e., COMPILE, ENCODE, and DECODE.

Table 1: Summary of notation and variables.

n	Number of gadgets
t	Security order
d	Number of shares
m	Number of random elements required for one gadget
$\hat{m}$	Number of random elements overall
$\mathcal{R}^k$	Ordered set of random elements used in the k'th gadget
r_i	i'th randomness element of overall randomness
r_i^k	i'th randomness element usd in the k'th gadget
$r_{i,j}^k$	Randomness used to mask products between share index i and j in the k'th gadget
T	(Simplified) computation tree
$\mathcal{V}$	Set of vertices in T
$\mathcal{E}$	Set of edges in T, i.e., shares of intermediate values
$\bar{\mathcal{E}}$	Set of unshared edges in T, i.e., intermediate values
$\bar{E}^{i,j}$	Unshared edge in T between $i \in \mathcal{V}$ and $j \in \mathcal{V}$
$\bar{P}^{i,j}$	Unshared path in T between $i \in \mathcal{V}$ and $j \in \mathcal{V}$
a^k	Left-hand-side input to the k'th gadget
b^k	Right-hand-side input to the k'th gadget
c^k	Output the k'th gadget
a_i^k	i'th share of the left-hand-side input to the k'th gadget
b_i^k	i'th share of the right-hand-side input to the k'th gadget
c_i^k	i'th share of the output of the k'th gadget

While COMPILE is a deterministic algorithm which takes a circuit C as input and outputs a masked circuit[1] $\tilde{\mathsf{C}}$, the ENCODE algorithm is probabilistic and transforms a secret x into its masked representation $\tilde{x}$. DECODE is again a deterministic algorithm converting a masked representation $\tilde{y}$ into the plain value y. Eventually, the secure computation of y from x is given as the concatenated execution of ENCODE, followed by $\tilde{\mathsf{C}}$, terminated with DECODE.

2.3 ISW t-Probing Model

In the literature, the ISW t-probing model [ISW03] has proven to be an accurate adversary model enabling the formal verification of masked circuits on the one hand, and construction of masking schemes on the other.

Probing on Ideal Circuits. In the standard t-probing model, an adversary is granted the ability to probe up to t arbitrary wires of a masked circuit $\tilde{\mathsf{C}}$, without having access to any information related to ENCODE and DECODE. However, assuming an ideal circuit, probes are restricted to only observe the current value of the wire. Then, t-probing security is achieved if an adversary is not able to retrieve any information on the secret given up to t probes.

Probing in the Presence of Glitches. Since ideal circuits do not capture physical defects in hardware implementations, i.e., *glitches*, *transitions*, and *couplings*, the t-probing model was extended to the *robust probing model* by Faust et al. [FGP+18]. In particular, glitches are a well-known source of information leakage in logic circuits due to unintentional activities caused by different signal and switching delays. To accurately capture these effects, in the glitch-extended robust t-probing model, each adversarial probe is extended to capture all values up to the last synchronization point (register or primary input),

[1] Throughout this work, we always assume a Boolean masking scheme.

contributing to the observed value. For this, each standard probe is substituted by a glitch-extend variant, and security is defined as the adversary not learning any information on the secret given up to t glitch-extended probes.

Probe Simulation and Propagation. According to Cassiers and Standaert [CS20], a probe propagates into another wire if the value on this wire is needed to perfectly simulate the distribution of the probed value, where *perfect simulation* is defined by Definition 1. This intuitively means that probes spread backwards through a circuit until they are stopped by some randomness refreshing. Hence, probes can capture information leakage of other wires contributing to the computation of the observed value.

Definition 1 (Perfect Probe Simulation)**.** Let $\mathcal{P}$ be a set of probes placed on a masked circuit. Then $\mathcal{P}$ is *perfectly simulatable* by a set $\mathcal{X}$ of input shares iff there exist a probabilistic polynomial time (ppt) simulator sim, such that for any values of the inputs to the masked circuit, the probability distribution over $\mathcal{P}$ and $sim(\mathcal{X})$ are equal.

2.4 Composability Notions

As formal verification and construction of masking schemes are increasingly challenging for higher security orders and larger circuits, different composability notions have been established. Each of them aimed at enabling the construction of small, masked sub-circuits, commonly denoted as *gadgets*, which provable lead to security in the (robust) t-probing model when composed to a larger circuit.

For this, all common composability notions restrict probe propagation for single gadgets, allowing to argue about probe propagation and security in composed circuits. More precisely, a single (extended) probe placed inside a gadget is restricted to propagate only to a limited set of input shares of the respective gadget, where the concrete limitation is specific to the composability notion.

***t*-Non-Interference.** The notion of NI [BBD+15], as defined in Definition 2, restricts probe propagation without differentiating between internal and output probes. It implies t-probing security but has proven to be insufficient to guarantee composability.

Definition 2 (Non-Interference)**.** A masked circuit $\tilde{\mathsf{C}}$ is t-NI iff for every probe set $\mathcal{P}$ containing $t' \leq t$ probes, there exists a set $\mathcal{X}$ containing at most t' shares per input, such that $\mathcal{P}$ can be perfectly simulated by $\mathcal{X}$.

Strong Non-Interference. For the SNI composability notion [BBD+16], an adversary is allowed to learn partial information on shared inputs when placing probes on wires inside the gadget. However, for every probe positioned on an output of the gadget, the adversary is not allowed to learn any additional input shares. More formally, t-SNI is defined according to Definition 3.

Definition 3 (t-Strong Non-Interference)**.** A single-output, masked circuit $\tilde{\mathsf{C}}$ is t-SNI iff for every probe set $\mathcal{P}$ containing t_1 internal probes and t_2 output probes, such that $t_1 + t_2 \leq t$, there exists a set $\mathcal{X}$ containing at most t_1 shares per input, such that $\mathcal{P}$ can be perfectly simulated by $\mathcal{X}$.

As this definition is limited to single-output gadgets, SNI was extended to Multiple-Output Strong Non-Interference (MO-SNI) by Cassiers and Standaert [CS20] to cover multiple-output gadgets as well.

Definition 4 (Multiple-Output Strong Non-Interference)**.** A masked circuit $\tilde{\mathsf{C}}$ provides t-MO-SNI iff for every probe set $\mathcal{P}$ containing t_1 internal probes and up to t_2 probes on each output, such that $t_1 + t_2 \leq t$, there exists a set $\mathcal{X}$ containing at most t_1 shares per input, such that $\mathcal{P}$ can be perfectly simulated by $\mathcal{X}$.

Probe Isolating Non-Interference. Since trivial (i.e., share-wise) implementations of linear operations are not SNI, Cassiers and Standaert introduce the notion of t-PINI [CS20], enabling trivial implementations by restricting probes to only propagate within a single-share domain. More formally, t-PINI is defined according to Definition 5.

Definition 5 (Probe Isolating Non-Interference)**.** Let $\mathcal{P}$ be a set of t_1 internal probes and some probes on outputs. Further, let $\mathcal{I}_\mathcal{O}$ be the set of share indices assigned to these output probes in $\mathcal{P}$, with $|\mathcal{I}_\mathcal{O}| = t_2$ and $t_1 + t_2 \leq t$. A masked circuit $\tilde{\mathsf{C}}$ is t-PINI iff for every $\mathcal{P}$, there exists a set of indices $\mathcal{I}_\mathcal{I}$ with $|\mathcal{I}_\mathcal{I}| \leq t_1$, such that the distribution observed by $\mathcal{P}$ can be perfectly simulated by input shares corresponding to share domains drawn from $\mathcal{I} = \mathcal{I}_\mathcal{I} \cup \mathcal{I}_\mathcal{O}$.

2.5 Restricted Composability Notions

In the context of this work, we introduce a variant of MO-SNI and PINI, which restricts the number of adversarial probes placed on gadget outputs more strictly. More precisely, our variants only allow t_2 adversarial probes placed on all outputs instead of t_2 probes placed at each output.

Definition 6 (t-Restricted Multiple-Output Strong Non-Interference)**.** A masked circuit $\tilde{\mathsf{C}}$ provides t-Restricted Multiple-Output Strong Non-Interference (RMO-SNI) iff for every probe set $\mathcal{P}$ containing t_1 internal probes and t_2 output probes, such that $t_1 + t_2 \leq t$, there exists a set $\mathcal{X}$ containing at most t_1 shares per input, such that $\mathcal{P}$ can be perfectly simulated by $\mathcal{X}$.

Definition 7 (t-Restricted Probe Isolating Non-Interference)**.** Let $\mathcal{P}$ be a set of t_1 internal probes and t_2 output probes with $t_1 + t_2 \leq t$. Further, let $\mathcal{I}_\mathcal{O}$ be the set of share indices assigned to the output probes. A masked circuit $\tilde{\mathsf{C}}$ provides t-Restricted Probe Isolating Non-Interference (R-PINI) iff for every $\mathcal{P}$, there exists a set of indices $\mathcal{I}_\mathcal{I}$ with $|\mathcal{I}_\mathcal{I}| \leq t_1$, such that the distribution observed by $\mathcal{P}$ can be perfectly simulated by input shares corresponding to share domains drawn from $\mathcal{I} = \mathcal{I}_\mathcal{I} \cup \mathcal{I}_\mathcal{O}$.

Obviously, these notions do not enable arbitrary compositions but require special care and construction, ensuring that a set of probes placed onto the composed circuit does not propagate into more than t output probes of a gadget. These definitions can be used to make statements about probing security and secure composition even when the full composability notions do not hold after optimization. We show how those notions can be ensured and leveraged to optimize the randomness used across gadget boundaries.

Further, we highlight that both RMO-SNI and R-PINI imply NI respectively. The reason is that both RMO-SNI and R-PINI restrict the adversary to at most $t' < t$ probes in total, which are simulated with at most t' shares of each input. This is exactly the definition of NI. In other words, if an R-PINI or RMO-SNI simulator exists for $\mathcal{P}$, then the NI simulator simply runs the same simulation on the same simulation set.

2.6 Computation Tree

In order to perform our cross-gadget optimizations, it is necessary to argue about connections between gadgets. For this, we define further abstractions of our circuit model, such that vertices in the graph represents gadgets, and each edge represent one share of an intermediate value. Please note, there always exist $t + 1$ parallel edges in a masked computation tree.

Definition 8 (Computation Tree)**.** We define the *computation tree* of a masked circuit $\tilde{\mathsf{C}}$ as a Directed Acyclic Multigraph (DAMG) $\mathsf{T} = (\mathcal{V}, \mathcal{E})$, with vertices $\mathcal{V} = \{G^0, G^1, \ldots, G^{n-1}\}$ representing all gadgets from $\tilde{\mathsf{C}}$ in addition to primary inputs and outputs, while edges $\mathcal{E} = \{E_\ell^{i,j} \mid i \in \mathcal{V}, j \in \mathcal{V}, \ell \leq t\}$ represents all wires between those gadgets in $\tilde{\mathsf{C}}$.

Further, following Cassiers et al. [CS20], a *simplified computation tree* is constructed according to Definition 9. In essence, the transformation removes the connection between inputs and outputs of SNI gadgets.

Definition 9 (Simplified Computation Tree)**.** A *simplified computation tree* is defined as a computation tree where all SNI gadgets G are divided into two separate parts G_{in} and G_{out}, such that the inputs of G are connected to G_{in}, and G_{out} is connected to the outputs of G. Further, G_{in} has no outputs and G_{out} has no inputs.

Since each edge in the (simplified) computation tree represents a wire in the circuit, each share belonging to one intermediate value is represented with its own edge. For situations where we only care about the existence of an edge connecting two vertices, we say an *unshared edge* is the set of edges belonging to the same unshared intermediate value. Using this, we can define an *unshared path* as a collection of connected unshared edges. Hence, for an unshared path, we only consider the existence of a connection between two gadgets and not the possible routes over different shares. However, we emphasize that this is just a formal concept with no impact on the underlying physical implementation.

Definition 10 (Unshared Path)**.** Let $\mathsf{T} = (\mathcal{V}, \mathcal{E})$ be a (simplified) computation tree. Further, let $\bar{\mathcal{E}}$ be a set of *unshared edges* such that $\bar{E}^{i,j} \in \bar{\mathcal{E}}$ iff $E_\ell^{i,j} \in \mathcal{E}$ for some ℓ. Then we define an *unshared path* $\bar{P}^{a,b}$ in T as a sequence of connected edges $\{\bar{E}^{a,i_1}, \bar{E}^{i_1,i_2}, \ldots, \bar{E}^{j_1,j_2}, \bar{E}^{j_2,b}\}$, with $\bar{E}^{i,j} \in \bar{\mathcal{E}}$.

3 Randomness Reduction for DOM-Gadget Composition

We first consider gadget compositions based on the notion of SNI. For this, we discuss how to use properties of SNI to cluster multiple gadgets before we show how to reuse randomness within such clusters. In the following, we assume a design composed of NI-addition gadgets and Domain-Oriented Masking (DOM)-multiplication and DOM-refresh gadgets [GMK17] where only the DOM gadgets require fresh randomness.

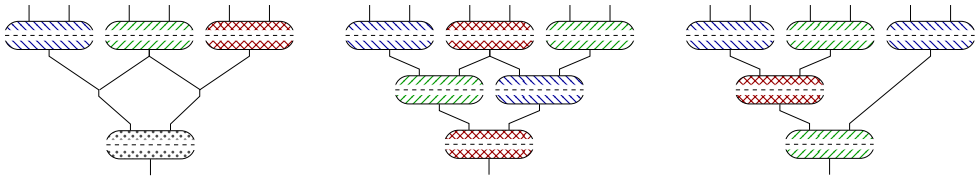
3.1 Clustering DOM Gadgets

Our first goal is to divide the set of DOM multiplication and DOM refresh gadgets into different clusters, such that we can reuse randomness within each cluster, i.e., two gadgets belonging to the same cluster may share some randomness.

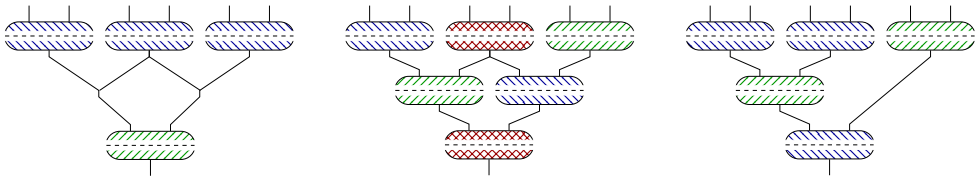
When reusing randomness among different gadgets, it has to be ensured that refreshing one value does not actually result in the removal of already introduced randomness. Hence, values dependent on the same randomness need always to be separate from each other.

In addition, sharing and reusing randomness among different gadgets potentially causes inter-gadget leakage, i.e., a probe placed within one gadget can capture leakage of additional input shares through the leakage of other gadgets. While the SNI property of a gadget generally restricts the amount of information that is leaked and captured by a single probe (i.e., the number of input shares), it does not constrain the source of information leakage (i.e., the share index). In consequence, to prevent critical cross-gadget leakage from revealing additional input shares, gadgets that can share and reuse the same randomness must have inputs independent of each other.

Further, we observe that the essential property of SNI gadgets is a share refreshing that removes any dependency between the output and the gadget inputs. Basically, this property is the foundation for SNI composition, introduced by Barthe et al. [BBD+16], and was generally shown by Belaïd et al. [BGR18]. However, this property allows not only to cluster parallel gadgets from different paths of the computation tree due to independent inputs, but also gadgets that lie in different paths within the *simplified* computation tree (which can be at the same path in the computation tree).



(a) Example assignments for Definition 11



(b) Example assignments for Definition 13

Figure 1: Example assignments in different clusters, where nodes represent DOM gadgets (according to a simplified computation tree) and NI gadgets are left out for brevity (at intersections or at edges). Matching colors and fill patterns resemble gadgets belonging to the same cluster and all edges represent unshared edges.

Definition 11 (Clustering of DOM Gadgets)**.** Let $\mathsf{T} = (\mathcal{V}, \mathcal{E})$ be a simplified computation tree of vertices $\mathcal{V} = \{G^0, G^1, \ldots, G^{n-1}\}$ and edges $\mathcal{E} = \{E_\ell^{i,j} \mid i \in \mathcal{V}, j \in \mathcal{V}, \ell \leq t\}$. A cluster of DOM gadgets is a set $\mathcal{C} \subseteq \mathcal{V}$ such that:

$$\forall \{G^i, G^j\} \in \mathcal{C}^2, i \neq j \mid \nexists G \in \mathcal{V} : \exists \bar{P}^{G,G^i} \wedge \exists \bar{P}^{G,G^j} \tag{1}$$

$$\forall \{G^i, G^j\} \in \mathcal{C}^2, i \neq j \mid \nexists \bar{P}^{G^i,G^j} \tag{2}$$

$$\forall \{G^i, G^j\} \in \mathcal{C}^2, i \neq j \mid \nexists G \in \mathcal{V} : \exists \bar{P}^{G^i,G} \wedge \exists \bar{P}^{G^j,G} \tag{3}$$

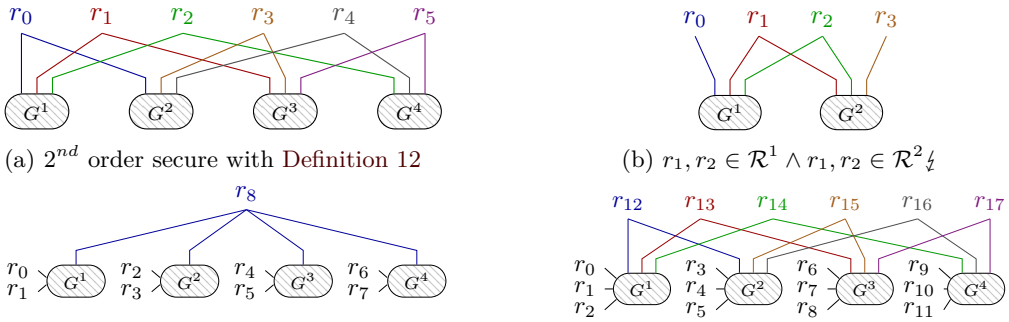
$$\forall G^i \in \mathcal{C}, \forall G^j \in \mathcal{V}, i \neq j \mid \nexists \bar{P}^{G^i,G^j} \vee \exists! \bar{P}^{G^i,G^j} \tag{4}$$

Here, Property (1) ensures that all inputs to the clusters are independent of each other, as there is no path from one *source* to multiple inputs of the cluster. Property (2) requires that there is no path from an output of the cluster to an input of the same cluster, ensuring the independence of the contained gadgets. Intuitively, those two properties restrict the cross-gadget leakage. Property (3) is symmetric to Property (1) in that it ensures that there is no path from multiple outputs of the cluster to the same *sink* gadget. This property is required to restrict the number of probes that can propagate to the outputs of the cluster. For the same reason, Property (4) restricts the number of paths from one output to all other gadgets to at most one. However, we included this property for completeness only, as it is always true for secure compositions of NI and SNI gadgets. The clustering principle is illustrated in Figure 1a for different scenarios.

We now prove that such a cluster can have at most t (propagated-)probes at outputs of the containing gadgets, meaning that the cluster requires RMO-SNI only, instead of MO-SNI, for probing security. We use Properties (3) and (4) from Definition 11 here, while the other properties are required later to prove Theorem 2.

Theorem 1. *Let $\mathsf{T} = (\mathcal{V}, \mathcal{E})$ be a simplified computation tree with gadgets in $\mathcal{V}$ that are t-NI or t-SNI and let $\mathcal{C}$ be a cluster according to Definition 11. Further, let $\mathcal{P}$ be a set of probes on $\mathcal{V}$ and $\mathcal{E}$, with $|\mathcal{P}| \leq t$. Then, there are at most t probes at output shares of $\mathcal{C}$ including probe propagation and glitch extension of probes.*

Proof. Assume a cluster $\mathcal{C}$ according to Definition 11 within a simplified computation tree $\mathsf{T} = (\mathcal{V}, \mathcal{E})$. We further assume a set of probes $\mathcal{P}$ on $\mathcal{V}$ and $\mathcal{E}$ with $|\mathcal{P}| \leq t$. In general,



(a) 2^{nd} order secure with Definition 12

(b) $r_1, r_2 \in \mathcal{R}^1 \wedge r_1, r_2 \in \mathcal{R}^2$ ↯

(c) 2^{nd} order secure with Definition 14

(d) 3^{nd} order secure with Definition 14

Figure 2: Example of randomness assignment for clusters of DOM gadgets, where (2a) represents a valid assignment. The assignment in (2b) is insecure, as the pair of randomness $\{r_1, r_2\}$ is used both in G^1 and G^2. In (2c) and (2d) a secure assignments with gadget unique randomness for 2^{nd} and 3^{rd} order security is shown.

there are two scenarios how the output of a gadget is probed: (1) by directly probing the corresponding signal or (2) by probe propagation after placing a probe on an internal signal (for this proof we consider glitch extension of probes as a form of probe propagation). Assume $|\mathcal{P}| = t_1 + t_2$ probes, among which t_1 probes are placed directly on outputs of $\mathcal{C}$, and t_2 probes elsewhere. It trivially holds that $t_1 \leq t$. Due to Properties (3) and (4), there is at most one unshared path from each actual probe to outputs of $\mathcal{C}$. Hence, a probe can not duplicate during probe propagation through different paths. In addition, the definition of NI and SNI guarantee that probes can not duplicate within gadgets. Therefore, there are at most t_2 probes capturing the outputs of $\mathcal{C}$ derived from probe propagation. This means, there are $t_1 + t_2 \leq t$ probes capturing the outputs of of $\mathcal{C}$, which proofs Theorem 1. □

Randomness Assignment for DOM Gadgets. After clustering gadgets according to Definition 11, the assignment of fresh randomness to gadgets follows a simple rule: Randomness can be reused between gadgets of the same cluster as long as there is no pair of random elements $\{r_i, r_j\}$ which is used in more than one gadget. In other words, there is no pair of gadgets that share more than one random element. We provide illustrative examples for the correct randomness reuse and an incorrect one in Figure 2a and Figure 2b respectively. Intuitively, this rule ensures that a probe placed in a single gadget can only leak one additional input share of other gadgets through cross-gadget leakage, as there is at most one random element that is reused between two gadgets. Definition 12 captures this rule of correctly reusing randomness for DOM gadgets within a cluster more formally.

Definition 12 (RMO-SNI-Randomness Assignment)**.** Given a cluster $\mathcal{C}$ as defined in Definition 11, for all DOM gadgets $G^k \in \mathcal{C}$ random elements can be assigned arbitrarily to G^k as long as the following condition holds:

$$\forall k, \forall \{r_i, r_j\} \in \mathcal{R}^k \mid \nexists k' \neq k : \{r_i, r_j\} \in \mathcal{R}^{k'}, \tag{5}$$

where r_i stands for the i-th random element and $\mathcal{R}^k$ is the set of randomness used for the gadget G^k.

Security Proof. We now prove that this rule ensures t-RMO-SNI for a cluster of DOM gadgets. More precisely, we take the gadget variant of Faust et al. [FGP+18], which was made t-SNI by adding an additional register layer at the output. The description of such

Algorithm 1: A single cluster, containing n DOM-multiplication gadgets, with reduced randomness (equal to definition in [FGP+18] for $n = 1$).

function clusterDOM$((a^0, b^0), ..., (a^{n-1}, b^{n-1}))$**:**
  Require: $t \geq 2$
  Require: All inputs are independent of each other
  Require: $a^k, b^k \in \mathbb{F}_2^{t+1}$ such that $a^k : (a_0^k, \ldots, a_t^k), b^k : (b_0^k, \ldots, b_t^k)$ with $\sum_j a_j^k = a^k$ and $\sum_j b_j^k = b^k$
  `// Initialize randomness`
2 **for** $k = 0$ *to* $n - 1$ **do**
3 $R^k = \langle r_0^k, \ldots, r_{m-1}^k \rangle \leftarrow$ Distribution according to Definition 12
4 $q \leftarrow 0$
5 **for** $i = 0$ *to* t **do**
6 **for** $j = i + 1$ *to* t **do**
7 $r_{i,j}^k \leftarrow r_q^k$; $r_{j,i}^k \leftarrow r_q^k$
8 $q \leftarrow q + 1$
  `// Compute multiplication gadgets`
9 **for** $k = 0$ *to* $n - 1$ **do**
10 **for** $i = 0$ *to* t **do**
11 $w_i^k \leftarrow a_i^k \cdot b_i^k$
12 **for** $j = i + 1$ *to* t **do**
13 $u_{i,j}^k \leftarrow a_i^k \cdot b_j^k + r_{i,j}^k$
14 $u_{j,i}^k \leftarrow a_j^k \cdot b_i^k + r_{j,i}^k$
15 **for** $i = 0$ *to* t **do**
16 $s_i^k \leftarrow \mathtt{Reg}[w_i^k] + \sum_{j=0, j\neq i}^{t} \mathtt{Reg}[u_{i,j}^k]$
17 $c_i^k \leftarrow \mathtt{Reg}[s_i^k]$
  Ensures: $c^k : (c_0^k, \ldots, c_t^k) \in \mathbb{F}_2^{t+1}$ such that $\sum_j c_j^k = a^k \cdot b^k, \forall k$
18 **return** $c^0, \ldots, c^{n-1}$

a cluster with n DOM gadgets is given in Algorithm 1. Please note that we can see the composition of the n DOM gadgets, constructing the cluster, as a new gadget for which we prove security. Further, we can restrict the argumentation to multiplication gadgets only, as SNI-refresh gadgets can be trivially constructed by SNI-multiplication gadgets where one input is a constant Boolean sharing of 1, e.g., $\langle 1, 0, \ldots, 0 \rangle$ (cf. [BGR18]).

Theorem 2. clusterDOM*, as defined in Algorithm 1, is t-RMO-SNI secure under the glitch-robust t-probing model.*

Proof. Let us denote a set of internal probes $\mathcal{P}_\mathcal{I}$ and a set of output probes $\mathcal{P}_\mathcal{O}$ with $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O} \leq t$. Without loss of generality, we restrict the probes to only capture w_i^k, $u_{i,j}^k$, s_i^k, and c_i^k as, due to glitch extension, all other probe positions capture a portion of these probes. In Algorithm 2, we give an algorithm that, given the set of probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$, returns a set of input shares $\mathcal{X}$ required to simulate the probes for the gadget defined in Algorithm 1. In particular, Algorithm 2 returns all inputs such that all required intermediate values can be computed exactly as in Algorithm 1 except for some u_{ij}^k. We first show that, for all possible probe placements, Algorithm 2 adds at most one share of each input to $\mathcal{X}$ for each internal probe. Afterwards, we show that the inputs in $\mathcal{X}$ enable a simulation of the set of probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$, as required for t-RMO-SNI.

For some probe placed on w_i^k it is obvious that Algorithm 2 only adds one share of each input to $\mathcal{X}$ (i.e., a_i^k, b_i^k). The same is true for a probe on s_i^k when $\nexists\{k', x, y, j\}$ such that $r_{x,y}^{k'} = r_{i,j}^k$ with a probe placed on $u_{x,y}^{k'}$ or $s_x^{k'}$, since then, Line 12 of Algorithm 2 is not

Algorithm 2: Input-share chooser for the simulator of ParDOM multiplication.

```
1  function ShareChooser^clusterDOM(P_I ∪ P_O):
2      X ← ∅
3      for i = 0 to t do
4          for k = 0 to n − 1 do
5              if w_i^k or s_i^k is probed then
6                  X ← X ∪ {a_i^k, b_i^k}
7              for j = 0 to t, j ≠ i do
8                  if u_ij^k is probed then
9                      X ← X ∪ {a_i^k, b_j^k}
10                 if s_i^k and at least one value u_x,y^k' or s_x^k' such that r_xy^k' = r_i,j^k is probed then
11                     X ← X ∪ {b_j^k}
12     return X
```

reached. However, if there is a probe placed on s_i^k and $\exists\{k', x, y, j\}$ such that $r_{x,y}^{k'} = r_{i,j}^k$ and a probe is placed on $u_{x,y}^{k'}$, then the set $\left\{a_i^k, b_i^k, b_j^k, a_x^{k'}, b_y^{k'}\right\}$ is added to $\mathcal{X}$. Naturally, this adds at most two shares of each input (using two probes) if $k' \neq k$. For $k' = k$, either $y = i$ or $y = j$, since $r_{i,j}^k$ is only used at most twice in each internal gadget, and $r_{i,j}^k = r_{j,i}^k$. Hence, again, only at most two shares of each input are added to $\mathcal{X}$ (using two probes). Otherwise, if a probe is placed on s_i^k as well as on some $s_x^{k'}$ such that $\exists\{j, y\} : r_{x,y}^{k'} = r_{i,j}^k$, then the set $\left\{a_i^k, b_i^k, a_x^{k'}, b_x^{k'}, b_j^k, b_y^{k'}\right\}$ is added to $\mathcal{X}$. With the same argument as before, this also adds only two shares per input to $\mathcal{X}$ (using two probes). In the last case, a probe is placed on some $u_{i,j}^k$ but not on some $s_x^{k'}$ containing the same random value. In this case, only $\left\{a_i^k, b_j^k\right\}$ is added to $\mathcal{X}$. The argumentation holds for any number of probes placed on values containing the same randomness, as all inputs are independent of each other (Property 1 from Definition 11) and two probes placed on values containing the same randomness already reveal all shares required for the actual computation. Please note, that a probe placed on some output share (i.e., c_i^k) does not add any item to $\mathcal{X}$, as required by RMO-SNI.

We now show that the input shares contained in $\mathcal{X}$ are sufficient to simulate the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$. For this, we compute all required intermediate values exactly as in the gadget (Algorithm 1) by replacing all values $u_{i,j}^k$, that can not be computed, by some fresh random value. For probes placed on output shares c_i^k where $a_i^k, b_i^k \notin \mathcal{X}$ we ignore w_i^k and compute only the sum of the corresponding $u_{i,j}^k, \forall j$. This ensures that all required intermediate values $w_i^k, u_{i,j}^k, s_i^k$ and output shares c_i^k are well defined in the simulator.

Finally, in order to prove Theorem 2, we only need to show that the above defined simulator has the same output distribution for the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$ as a computation of the probes with Algorithm 1. This is true for the following reason. All intermediate values are computed exactly the same for both the simulator and Algorithm 1, except for some $u_{i,j}^k$ which are replaced by fresh random values, and some output shares c_i^k.

Let us assume we replace some $u_{i,j}^k$ with randomness, then this value is not directly probed, as otherwise $a_i^k, b_j^k \in \mathcal{X}$ according to Algorithm 2 and $u_{i,j}^k$ can be computed, but a probe is placed on either s_i^k or c_i^k as those are the only values requiring $u_{i,j}^k$. Let us take a closer look into these cases in the following.

Case I. Assume a probe is placed on s_i^k. From Algorithm 2 it follows that $a_i^k \in \mathcal{X}$ (Line 7), hence, $b_j^k \notin \mathcal{X}$, as otherwise $u_{i,j}^k$ can be computed. From this and Algorithm 2 it also follows that $\nexists\{k', x, y\}$ with $r_{i,j}^k = r_{x,y}^{k'}$ such that a probe is placed on $u_{x,y}^{k'}$ or $s_x^{k'}$ (Line 12). If, on the one hand, there is no probe placed at any output $c_x^{k'}, k' \neq k$ that

contains $r^k_{i,j}$, then $r^k_{i,j}$ is only observable through $u^k_{i,j}$ (in the computation for s^k_i) and replacing $u^k_{i,j}$ with fresh randomness does not change the output distribution (a probe placed on c^k_i does not reveal any additional information to s^k_i). If, on the other hand, a probe is placed on some $c^{k'}_x, k' \neq k$ that also contains $r^k_{i,j}$, then $c^{k'}_x$ also contains $t-1$ other random values. From Definition 12 it follows that any two probes placed in different gadgets can have at most one random value in common. As there is no other internal probe containing $r^k_{i,j}$ and again as $b^k_j \notin \mathcal{X}$, there is no probe on $s^{k'}_x$, which is the only value sharing more than one random value with $c^{k'}_x$. Hence, $t-1$ more probes are required to remove all uniform random behavior from $c^{k'}_x$, but only $t-2$ probes are left. Therefore, $r^k_{i,j}$ is still only observable through $u^k_{i,j}$ and replacing $u^k_{i,j}$ with fresh randomness does not change the output distribution.

Case II. Now, assume a probe is placed on c^k_i. If there is some probe placed on some $s^{k'}_x$ which contains $r^k_{i,j}$, we can use the argumentation of Case I. Therefore, we assume only probes on output shares containing the randomness $r^k_{i,j}$. If c^k_i is the only output depending on $r^k_{i,j}$, this random value is only observable through $u^k_{i,j}$ and replacing $u^k_{i,j}$ with fresh randomness does not change the output distribution. If there are at least two output probes that contain $r^k_{i,j}$ then again we require $t-1$ probes to remove all uniform random behavior from c^k_i but only $t-2$ probes are left. Hence, $r^k_{i,j}$ is not observable at all and replacing $u^k_{i,j}$ with fresh randomness does not change the output distribution.

Therefore, replacing some $u^k_{i,j}$ with fresh randomness does not change the output distribution of the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$. If there is a probe placed on some c^k_i, and w^k_i cannot be computed as either $a^k_i \notin \mathcal{C}$ or $b^k_i \notin \mathcal{X}$, then there are no probes placed on w^k_i, or s^k_i. Hence, there is no probe placed on a value that shares more than one random value with c^k_i, again as the only such probe would be one placed at s^k_i. Hence, t more probes are required to remove all randomness from c^k_i but only $t-1$ probes are left. Therefore, w^k_i has no influence on the distribution of c^k_i and removing it from the computation does not change the output distribution.

We conclude that the simulator is able to fully simulate Algorithm 1 with input shares $\mathcal{X}$, which proves Theorem 2. □

We emphasize that the resulting cluster is RMO-SNI and hence, special care needs to be taken when composing gadgets belonging to different clusters. We envision the usage of this approach during optimization of an already-secure design, where the clusters can be selected such that the required composition properties hold (given in Definition 11). As a result, this transformation leads to an overall design that is at most RMO-SNI. Hence, probing secure and NI secure designs will maintain their properties, but SNI designs are potentially weakened. To maintain SNI security, it is necessary to ensure that on the path from each output of a cluster to an output of the circuit, there is at least one SNI gadget.

3.2 Relaxed Clustering of DOM Gadgets

In Definition 11, we used a strong notion of clustering that requires an information flow analysis for both the inputs and the outputs of each gadget. The resulting small clusters offer high potential for randomness optimization, i.e., all randomness bits can be potentially reused, but also have the potential of weakening the notion of composition, i.e., RMO-SNI instead of MO-SNI. A cluster that sustains the stronger notion of MO-SNI can be built by adding more restrictions to the randomness reuse within the cluster. In particular, some randomness is required to be unique, hiding randomness being reused at the outputs. More specifically, we need to ensure that the only combination of probes that can remove all randomness from an output share are probes placed on all shares of that output. Then, due to the stronger notion of composition, the clustering rules can be relaxed by omitting the information flow analysis of the gadget outputs and focusing solely on the gadget

inputs. More formally, only Property (1) and (2) from Definition 11 are required while Property (3) and (4) can be dropped.

Definition 13 (Relaxed Clustering of DOM Gadgets)**.** Let $\mathsf{T} = (\mathcal{V}, \mathcal{E})$ be a simplified computation graph of vertices $\mathcal{V} = \{G^0, G^1, \ldots, G^{n-1}\}$ and edges $\mathcal{E} = \{E_\ell^{i,j} \mid i \in \mathcal{V}, j \in \mathcal{V}, \ell \leq t\}$. A cluster of DOM gadgets is a set $\mathcal{C} \subseteq \mathcal{V}$ such that:

$$\forall \{G^i, G^j\} \in \mathcal{C}^2, i \neq j \mid \nexists G \in \mathcal{V} : \exists \bar{P}^{G,G^i} \wedge \exists \bar{P}^{G,G^j} \tag{1}$$

$$\forall \{G^i, G^j\} \in \mathcal{C}^2, i \neq j \mid \nexists \bar{P}^{G^i,G^j} \tag{2}$$

An illustration of the relaxed clustering is given in Figure 1b. In contrast to clustering, the assignment of randomness to gadgets within a cluster is more restricted and, hence, the formal definition requires additional properties compared to Definition 12.

Definition 14 (MO-SNI-Randomness Assignment)**.** Given a cluster $\mathcal{C}$ as defined in Definition 13, for all DOM gadgets $G^k \in \mathcal{C}$ random elements can be assigned to G^k as long as the following conditions hold:

$$\forall k, \forall \{r_i, r_j\} \in \mathcal{R}^k \mid \nexists k' \neq k : \{r_i, r_j\} \in \mathcal{R}^{k'} \tag{5}$$

$$\forall k, \forall r_i \in \hat{\mathcal{R}}^k \mid \nexists k' \neq k : r_i \in \hat{\mathcal{R}}^{k'} \vee r_i \in \mathcal{R}^{k'} \tag{6}$$

$$\exists i, \forall j \neq i \mid r_{i,j} \in \hat{\mathcal{R}}^k, \tag{7}$$

where $\hat{\mathcal{R}}^k$ is the set of unique randomness used by gadget G^k, $\mathcal{R}^k$ is the set of random values used for the gadget G^k, that are potentially shared with other gadgets, and $r_{i,j}$ is the random element blinding the cross domain (i, j), i.e., the product of shares from i and j (e.g., $r_{i,j}^k$ in Algorithm 1).

Intuitively, Property (6) resembles the unique usage of randomness in each gadget, Property (7) ensures that one output share is blinded completely by gadget-specific fresh randomness, ensuring that all output shares are required to remove the gadget-unique randomness of one output share, while Property (5) is already used in Definition 12 and determines how to reuse randomness between gadgets of the same cluster. Illustrative examples are given in Figure 2c and Figure 2d.

Using Definition 13 and Definition 14 together with the same DOM gadget implementation as before, we now prove MO-SNI-security of a corresponding cluster.

Theorem 3. clusterDOM *as defined in Algorithm 1, where Definition 14 is used as a replacement for Definition 12 in Line 3, is t-MO-SNI under the glitch-robust probing model.*

The only difference between Theorem 3 and Theorem 2 is the security property, i.e., MO-SNI instead of RMO-SNI, and the randomness assignment. In particular, an adversary is allowed to place $t_2 \leq t$ probes on each output instead of $t_2 \leq t$ for all outputs together. Now we show that the same simulator as used in the proof of Theorem 2 can be used to proof Theorem 3.

Proof. Assume the same simulator as used in the proof of Theorem 2. In particular, the required input shares are determined using Algorithm 2 and all values are computed exactly as in Algorithm 1, except for some $u_{i,j}^k$ and some c_i^k. Without loss of generality, we assume that the first $t-1$ random values assigned to each gadget are unique to this gadget, resulting in the first output share being only masked with unique randomness while all other output shares being masked with exactly one unique random value.

As before, we restrict our analysis to probes placed on w_i^k, $u_{i,j}^k$, s_i^k, and c_i^k, as other probes are strictly less powerful. Again, we need to show that the above defined simulator results in the same output distribution for the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$ as the computation of

the probes with Algorithm 1. The only values computed differently in the simulator and Algorithm 1 are some $u_{i,j}^k$ and c_i^k.

Recall, that if some value $u_{i,j}^k$ is replaced by randomness, then there is a probe placed on either s_i^k or c_i^k, as those are the only values requiring the value $u_{i,j}^k$.

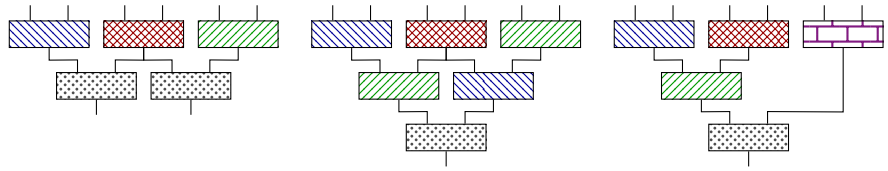
Case I. Assume s_i^k is probed. If this (and optionally a probe on c_i^k, which gives no additional information) is the only probe containing $r_{i,j}^k$, then this random value is observable only through $u_{i,j}^k$ and the output distribution does not change by replacing $u_{i,j}^k$ with fresh randomness. If $\exists\{k' \neq k, x, y\} : r_{i,j}^k = r_{x,y}^{k'}$, and a probe is placed on $c_x^{k'}$, then it follows from Property (7) that $c_x^{k'}$ also contains some random value unique to the k'-th gadget. Removing the masking with a random value which is unique to the gadget, requires either some internal probe (e.g., the randomness itself) or a probe on $c_0^{k'}$, which itself is masked with $t-2$ additional random values unique to the gadget. Removing them requires either one internal probe (e.g., $s_0^{k'}$) or all other output shares. Hence, removing some gadget's unique randomness from an output share requires either some internal probe or all $t+1$ shares of that particular output, where the later is prohibited by the number of available probes for each output ($\leq t$). In addition to the gadget's unique randomness, $c_x^{k'}$ also contains $t-2$ random values that are potentially shared with other gadgets. Property (5) ensures that two different gadgets do not share more than one random value. Hence, the only value sharing more than one random value with $c_x^{k'}$ is $s_x^{k'}$, where no probe is placed as otherwise $a_i^k \in \mathcal{X}$ (Line 6 in Algorithm 2) and $b_j^k \in \mathcal{X}$ (Line 11 in Algorithm 2) which means $u_{i,j}^k$ can be computed. Therefore, at least one additional probe is required to remove one reused random value in $c_x^{k'}$. This probe can either be an internal probe or an output probe, which itself is masked by some unique randomness different to the unique randomness in $c_x^{k'}$. Therefore, to remove all random behavior except $r_{i,j}^k$ from $c_x^{k'}$ at least $t-1$ probes are required, but only at most $t-2$ probes are left to the adversary, since there is already a probe on s_i^k and $c_x^{k'}$. Hence, the randomness $r_{i,j}^k$ is observable only through $u_{i,j}^k$ in s_i^k and replacing $u_{i,j}^k$ with fresh randomness does not change the output distribution.

Case II. Now, assume a probe placed on c_i^k and $r_{i,j}^k$ is only captured by probes on outputs. If there is a probe on some $s_x^{k'}$ such that $r_{i,j}^k = r_{x,y}^{k'}$, then the arguments from Case I hold. If c_i^k is the only output probe containing $r_{i,j}^k$, then this random value is observable only through $u_{i,j}^k$ and replacing that value by fresh randomness does not change the output distribution. If $\exists\{k' \neq k, x, y\} : r_{i,j}^k = r_{x,y}^{k'}$ and a probe is placed on $c_x^{k'}$, then c_i^k and $c_x^{k'}$ share at most one random value ($r_{i,j}^k$), due to Property (5), while both contain at least one gadget-unique random value and $t-2$ random values potentially shared with other gadgets. Removing the random behavior of one of the two outputs requires $t-1$ internal probes (with the same argument as above) but at most $t-2$ probes are left to the adversary, since c_i^k and $c_x^{k'}$ are already probed. Hence, the randomness $r_{i,j}^k$ is not observable at all and replacing $u_{i,j}^k$ by fresh randomness does not change the output distribution.

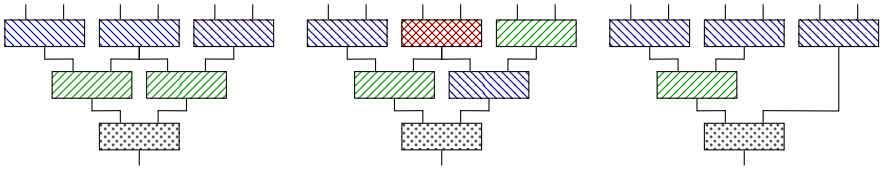
In conclusion, $u_{i,j}^k$ can be perfectly simulated by a random value. Now, assume a probe placed on some c_i^k, where w_i^k can not be computed, i.e., either $a_i^k \notin \mathcal{X}$ or $b_i^k \notin \mathcal{X}$. From Line 6 in Algorithm 2 follows that there is no probe placed on s_i^k, which is the only value sharing more than one random value with c_i^k. Hence, with the same argument as above, t internal probes are required to remove all random behavior from c_i^k but only $t-1$ probes are left to the adversary. Therefore, w_i^k is not observable and ignoring it during computation of c_i^k does not change the distribution.

We conclude, that the simulator fully simulates $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$ which proves Theorem 3. □

Note, however, that this optimization does not change the security properties of the original circuit, as the cluster remains MO-SNI.



(a) Example assignments for Definition 15



(b) Example assignments for Definition 17

Figure 3: Example assignments in different clusters, where nodes represents an HPC_2 gadget. Matching colors and fill patterns resemble gadgets belonging to the same cluster. The graphs represent the simplified computation tree (here equal to the computation tree) where all edges are represented unshared.

4 Randomness Reduction in HPC_2 Composition

As the second notion of composition, we consider PINI. Again, we first discuss how gadgets can be clustered in the context of PINI, i.e., within designs composed from HPC_2 gadgets [CGLS21], before we show how to reuse randomness within such clusters.

4.1 Clustering HPC_2 Gadgets.

In contrast to SNI, the essential property of PINI is the isolation of leakage within share domains, i.e., one probe can only cause the leakage of values within one share domain. As this must also hold for cross-gadget leakage caused by shared randomness, gadgets within one cluster are now allowed to have dependent inputs, effectively removing the requirement of Property (1). At the same time, a probe propagating through the design, using different paths but reaching the same gadget output, always results in only one probed output share, i.e., at the share index, the actual probe was placed. Hence, Property (4) is obsolete in the context of PINI. This results in the following formal definition:

Definition 15 (Clustering of HPC_2 Gadgets)**.** Let $\mathsf{T} = (\mathcal{V}, \mathcal{E})$ be a simplified computation tree of vertices $\mathcal{V} = \{G^0, G^1, ..., G^{n-1}\}$ and edges $\mathcal{E} = \{E_\ell^{i,j} \mid i \in \mathcal{V}, j \in \mathcal{V}, \ell \leq t\}$. A cluster of HPC_2 gadgets is a set $\mathcal{C} \subseteq \mathcal{V}$ such that:

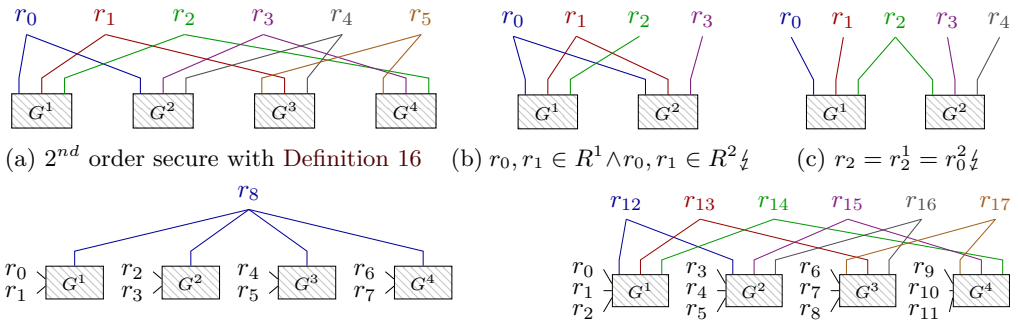
$$\forall \{G^i, G^j\} \in C^2, i \neq j \mid \nexists \bar{P}^{G^i, G^j} \tag{2}$$

$$\forall \{G^i, G^j\} \in C^2, i \neq j \mid \nexists G \in \mathcal{V} : \exists \bar{P}^{G^j, G} \wedge \exists \bar{P}^{G^i, G} \tag{3}$$

As in Definition 11, Property (2) ensures that gadgets within one cluster are independent of each other, while Property (3) guarantees that one probe cannot propagate to multiple probes at the outputs of the cluster.

We emphasize that PINI does not enforce a refreshing layer rendering the gadget output independent of the input and, hence, due to Property (3), PINI gadgets from the same path in the computation tree cannot be clustered together. The clustering can still be done based on a simplified computation tree, however, a PINI gadget does not split a path on its own.

Again, such a cluster can have at most t (propagated-)probes at outputs of the cluster, meaning that the cluster requires to be R-PINI only, instead of PINI, for probing security.



(a) 2^{nd} order secure with Definition 16 (b) $r_0, r_1 \in R^1 \wedge r_0, r_1 \in R^2$ ↯ (c) $r_2 = r_2^1 = r_0^2$ ↯

(d) 2^{nd} order secure with Definition 18 (e) 3^{rd} order secure with Definition 18

Figure 4: Example of randomness assignment for clusters of HPC_2 gadgets, where (4e) represents a valid assignment. The assignment in (4b) is insecure, as the pair of randomness $\{r_0, r_1\}$ is used both in G^1 and G^2, and the assignment in (4c) is insecure as r_2 is used both in position 2 and 0. In (4e) and (4e) a secure assignment with gadget unique randomness for 2^{nd} and 3^{rd} order security is shown.

Theorem 4. *Let $\mathsf{T} = (\mathcal{V}, \mathcal{E})$ be a simplified computation tree with gadgets in $\mathcal{V}$ that are t-PINI and let $\mathcal{C}$ be a cluster according to Definition 15. Further, let $\mathcal{P}$ be a set of probes on $\mathcal{V}$ and $\mathcal{E}$, with $|\mathcal{P}| \leq t$. Then there are at most t probes at output shares of $\mathcal{C}$ including probe propagation and glitch extension of probes.*

The corresponding proof follows the same line of arguments as the proof of Theorem 1 with only straight-forward changes related to PINI. Hence, we omit the proof for brevity.

Randomness Distribution for HPC_2. The distribution of randomness within clusters of HPC_2 gadgets is a bit more complex than in the case of DOM. First, the same rule as for DOM is required, namely that no pair of random elements $\{r_i, r_j\}$ is used in more than one gadget. In addition, PINI requires that random elements always mask the same share indices, i.e., are used at the same position within all gadgets that use this random element. This ensures the isolation of share domains even for cross-gadget leakage. We provide illustrative examples of secure and insecure randomness reuse for HPC_2 clusters in Figure 4 (first row) and define the property more formally as follows:

Definition 16 (R-PINI-Randomness Assignment)**.** Given a cluster $\mathcal{C}$ as defined in Definition 15, for all HPC_2 gadgets $G^k \in \mathcal{C}$, random elements can be assigned arbitrarily to G^k as long as the following conditions hold:

$$\forall k, \forall \{r_i, r_j\} \in \mathcal{R}^k \mid \nexists k' \neq k : \{r_i, r_j\} \in \mathcal{R}^{k'} \tag{5}$$

$$\forall \{k, i, x\} : r_i^k = r_x \mid \nexists \{k', j\}, k' \neq k, j \neq i : r_j^{k'} = r_x, \tag{8}$$

where r_x is the x'th randomness element overall, r_i^k is the i'th random element assigned to gadget G^k, and $\mathcal{R}^k$ is the set of random elements assigned to the gadget G^k.

Here, Property (5) is already used in Definition 12 and ensures that no two gadgets share more than one random element. In contrast, Property (8) is PINI specific, restricting the position of reused randomness within each gadget. More precisely, a random bit can only be reused to blind identical share domains. This ensures that cross-gadget leakage also adheres to the PINI definition.

Security Proof. We now prove that this rule ensures t-R-PINI for a cluster of HPC_2 gadgets, of which a description is given in Algorithm 3.

Algorithm 3: A single cluster, containing n HPC_2 gadgets, with reduced randomness (equal to Figure 5 in [CGLS21] for $n = 1$).

1 **function** clusterHPC$_2$($(a^0, b^0), ..., (a^{n-1}, b^{n-1})$):
  Require: $t \geq 2$
  Require: $a^k, b^k \in \mathbb{F}_2^{t+1}$ such that $a^k : (a_0^k, \ldots, a_t^k), b^k : (b_0^k, \ldots, b_t^k)$ with $\sum_j a_j^k = a^k$ and $\sum_j b_j^k = b^k$
  `// Initialize randomness`
2 **for** $k = 0$ *to* $n - 1$ **do**
3 $R^k = \langle r_0^k, \ldots, r_{m-1}^k \rangle \leftarrow$ Distribution according to Definition 16
4 $q \leftarrow 0$
5 **for** $i = 0$ *to* t **do**
6 **for** $j = i + 1$ *to* t **do**
7 $r_{i,j}^k \leftarrow r_q^k$; $r_{j,i}^k \leftarrow r_q^k$
8 $q \leftarrow q + 1$
  `// Compute` HPC_2 `gadgets`
9 **for** $k = 0$ *to* $n - 1$ **do**
10 **for** $i = 0$ *to* t **do**
11 $w_i^k \leftarrow a_i^k \cdot \mathtt{Reg}[b_i^k]$
12 **for** $j = 0$ *to* $t, j \neq i$ **do**
13 $u_{i,j}^k \leftarrow (a_i^k + 1) \cdot \mathtt{Reg}[r_{i,j}^k]$
14 $v_{i,j}^k \leftarrow b_j^k + r_{i,j}^k$
15 $z_{i,j}^k \leftarrow a_i^k \cdot \mathtt{Reg}[v_{i,j}^k]$
16 **for** $i = 0$ *to* t **do**
17 $c_i^k \leftarrow \mathtt{Reg}[w_i^k] + \sum_{j=0, j\neq i}^{t} (\mathtt{Reg}[u_{i,j}^k] + \mathtt{Reg}[z_{i,j}^k])$
  Ensures: $c^k : (c_0^k, \ldots, c_t^k) \in \mathbb{F}_2^{t+1}$ such that $\sum_j c_j^k = a^k \cdot b^k, \forall k$
18 **return** $c^0, \ldots, c^{n-1}$

Theorem 5. clusterHPC$_2$ *as defined in Algorithm 3 is t-R-PINI secure in the glitch-robust model.*

Our proof follows closely the argumentation of Cassiers et al. for a single HPC_2 gadget [CGLS21].

Proof. Let us denote a set of internal probes $\mathcal{P}_\mathcal{I}$ and a set of output probes $\mathcal{P}_\mathcal{O}$, such that $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O} \leq t$. Without loss of generality, we restrict the probes to only capture c_i^k, w_i^k, $u_{i,j}^k$, $v_{i,j}^k$, and $z_{i,j}^k$ as other extended probes are less powerful, due to glitch extension. In Algorithm 4, we give an algorithm that, given the set of probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$, returns a set of input shares $\mathcal{X}$ required to simulate the probes for the gadget defined in Algorithm 3. In particular, Algorithm 4 returns all inputs such that all required intermediate values can be computed exactly as in Algorithm 3, except for some $v_{i,j}^k$. We first show that, for all possible probe placements, Algorithm 4 adds at most one share index per probe to $\mathcal{X}$, where the share indices of probes capturing an output are always in $\mathcal{X}$. Afterwards, we show that the inputs in $\mathcal{X}$ enable a simulation of the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$, as required for R-PINI.

From the construction of Algorithm 4 it is obvious that all probes that do not capture an output c_i^k add at most one share index to $\mathcal{X}$. If all probes in $\mathcal{P}_\mathcal{O}$ capture different share indices then only Line 6 in Algorithm 4 is activated, which adds the respective share index to $\mathcal{X}$. Now, assume there are multiple probes capturing outputs of the same share index i. From Property (5) it follows that each pair $\{c_i^k, c_i^{k'}\}, k \neq k'$ of such probes can have at most one random value in common and Property (8) guaranties that this random value is

Algorithm 4: Input-share chooser for the simulator of $\mathsf{clusterHPC}_2$.
(equal to the description in the proof for Proposition 4 in [CGLS21] for $n = 1$.)

```
1  function ShareChooser^{clusterHPC_2}(P):
2      X ← ∅
3      for i = 0 to t do
4          for k = 0 to n − 1 do
5              if w_i^k or c_i^k is probed then
6                  X ← X ∪ {i}
7              for j = 0 to t, j ≠ i do
8                  if u_{i,j}^k, v_{i,j}^k, or z_{i,j}^k is probed and i ∈ X or j ∈ X then
9                      X ← X ∪ {i, j}
10                 else if u_{i,j}^k or z_{i,j}^k is probed then
11                     X ← X ∪ {i}
12                 else if v_{i,j}^k is probed then
13                     X ← X ∪ {j}
14                 if at least two probes on c_i^k, c_i^{k'} with k' ≠ k and r_{i,j}^k = r_{i,j}^{k'} then
15                     X ← X ∪ {j}
16     return X
```

used to blind the same two share indices i and j. Hence, all such probes will add the same share index i to $\mathcal{X}$ (Line 6) and all but the first add at most one additional share index j to $\mathcal{X}$ (Line 21). This also holds for combinations of probes capturing up to t output shares, as this can always be reduced to multiple pairs of probes which add redundant share indices to $\mathcal{X}$. Therefore, Algorithm 4 adds at most one share index per probe, where, for each share index i of an output captured by a probe, Line 6 guaranties $i \in \mathcal{X}$. We emphasize that this is not true for standard PINI, where an entire output share index is probed at once and, hence, an attacker gets combination of output probes for free.

We now show that the share indices in $\mathcal{X}$ are sufficient to simulate the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$. For this we compute all required intermediate values exactly as defined by the gadget (Algorithm 3) by replacing all values $v_{i,j}^k$, that cannot be computed by some fresh random value. Hence, all extended probes capturing w_i^k, $u_{i,j}^k$, and $v_{i,j}^k$ can be directly computed by construction of Algorithm 4, and probes capturing c_i^k, or $z_{i,j}^k$, where some share index $j \notin \mathcal{X}$ exists, are computed by replacing the corresponding intermediate value $v_{i,j}^k$ by fresh randomness.

This simulator results in the same output distribution for the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$ as a computation of the probes with Algorithm 3, for the following reason: All values are computed exactly the same for both, the simulator and Algorithm 3, except for some $v_{i,j}^k$ which are replaced by fresh random values. Let us assume we replace some $v_{i,j}^k$ with randomness, then it holds that either $i \notin \mathcal{X}$ or $j \notin \mathcal{X}$ and a value depending on $v_{i,j}^k$ is probed ($v_{i,j}^k$, $z_{i,j}^k$, or c_i^k).

If a probe is placed directly on $v_{i,j}^k$ then $j \in \mathcal{X}$, according to Algorithm 4, which means $i \notin \mathcal{X}$. Therefore, there cannot be any other probe placed on some $u_{i,j}^{k'}$, $u_{j,i}^{k'}$, $v_{i,j}^{k'}$, $v_{j,i}^{k'}$, $z_{i,j}^{k'}$, $z_{j,i}^{k'}$, $c_i^{k'}$, or $c_j^{k'}$ for all k'. However, from Property (8) and Algorithm 3 it follows that those are the only values that can dependent on the random value $r_{i,j}^k$, which means $r_{i,j}^k$ is only observable through $v_{i,j}^k$ and replacing $v_{i,j}^k$ with fresh randomness does not change the output distribution.

Otherwise, if a probe is placed on $z_{i,j}^k$ or c_i^k then $i \in \mathcal{X}$, according to Algorithm 4,

which means $j \notin \mathcal{X}$. Therefore, there cannot be any other probe placed on some $u^{k'}_{j,i}$, $v^{k'}_{i,j}$, $z^{k'}_{j,i}$, or $c^{k'}_{j}$ for all k'. Similar, there is at most one probe placed on $u^{k'}_{i,j}$, $v^{k'}_{j,i}$, $z^{k'}_{i,j}$ or $c^{k'}_{i}$, with $r^{k}_{i,j} = r^{k'}_{i,j}$, for all k'. Again, those are all possible values dependent on $r^{k}_{i,j}$ and, hence, there is exactly one probe placed on a value depending on $r^{k}_{i,j}$, which is either $z^{k}_{i,j}$ or c^{k}_{i}.

Case I: Assuming there is a probe placed on $z^{k}_{i,j}$, the corresponding extended probe is $\{a^{k}_{i}, v^{k}_{i,j}, z^{k}_{i,j}\}$, which contains the only values through which $r^{k}_{i,j}$ can be observed (with the argumentation from above). Therefore, replacing $v^{k}_{i,j}$ with fresh randomness does not change the output distribution.

Case II: Assuming there is a probe placed on c^{k}_{i}, the only observations depending on $r^{k}_{i,j}$ can be made via the extended probe of c^{k}_{i} (with the same argumentation as above), in particular, through the values $u^{k}_{i,j}$ or $z^{k}_{i,j}$. If $a^{k}_{i} = 0$, then it holds that $z^{k}_{i,j} = a^{k}_{i} \cdot v^{k}_{i,j} = 0$ independent of $v^{k}_{i,j}$, which means replacing $v^{k}_{i,j}$ with fresh randomness is not observable. Otherwise, if $a^{k}_{i} = 1$, it holds that $u^{k}_{i,j} = (a^{k}_{i} + 1) \cdot r^{k}_{i,j} = 0$ independent of $r^{k}_{i,j}$, which means $r^{k}_{i,j}$ is only observable through $v^{k}_{i,j}$ and, thus, replacing this value by fresh randomness does not change the output distribution.

Therefore, replacing some $v^{k}_{i,j}$ with a random value does not change the output distribution of the probes $\mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{O}}$ and, hence, the simulator is able to fully simulate Algorithm 3 with share indices in $\mathcal{X}$, which proves Theorem 5. □

We emphasize that the resulting cluster is R-PINI which requires special care when composing gadgets belonging to different clusters. As in the case of RMO-SNI, we envision the usage during optimization of an already-secure design, where the clusters can be selected such that the required composition properties hold (given in Definition 15). This transformation leads to an overall design that is at most R-PINI, and in contrast to RMO-SNI, there is no natural way within the context of PINI to recover the PINI properties.

4.2 Relaxed Clustering

The rules for clustering defined in Definition 15 are quite restrictive, especially as PINI gadgets do not split any path in the simplified computation tree. Hence, even when the defined clustering allows a high level of randomness reuse within a cluster, for most designs only small clusters can be created. This effect is even stronger than for SNI. In addition, we have a weakening of the resulting notion of composition. Unfortunately, as PINI does not stop probe propagation from outside a gadget to the inside, the clustering cannot be relaxed in the same manner for PINI-only gadgets as described in Section 3.2 for SNI. The reason is that some gadget-unique randomness cannot hide the reusable randomness completely at the outputs.

In the case of HPC_2 gadgets, this can be fixed by adding a register layer at the output, effectively making the gadget SNI as well. More precisely, we add some $d^{k}_{i} \leftarrow \mathtt{Reg}[c^{k}_{i}]$ to the algorithm (as Line 18), which is then considered the output share i of the k'th gadget. We call this gadget version HPC_2^{+}.

The resulting rules for clustering and randomness reuse are similar to Definition 13 and Definition 14 but integrate the considerations made so far for HPC_2. As the resulting gadget is SNI, the corresponding path is split in the simplified computation tree, and the resulting clusters are both MO-SNI and PINI, however, at the cost of additional area and latency. More formally, we can define the relaxed clustering of HPC_2^{+} gadgets as follows:

Definition 17 (Relaxed Clustering of HPC_2^+ Gadgets)**.** Let $\mathsf{T} = (\mathcal{V}, \mathcal{E})$ be a simplified computation tree of vertices $\mathcal{V} = \{G^0, G^1, ..., G^{n-1}\}$ and edges $\mathcal{E} = \{E_\ell^{i,j} \mid i \in \mathcal{V}, j \in \mathcal{V}, \ell \leq t\}$. A cluster of HPC_2^+ gadgets is a set $\mathcal{C} \subseteq \mathcal{V}$ such that:

$$\forall \{G^i, G^j\} \in \mathcal{C}^2 \mid \nexists \bar{P}^{G^i, G^j}. \tag{2}$$

Hence, only Property (2), which ensures independence of the gadgets within a cluster, is required, and all other clustering properties can be dropped. For such a clustering, we can utilize the same randomness assignment as given in Definition 14, where we additionally ensure that randomness is only reused at the same position within different gadgets, i.e., Property (8).

Definition 18 (PINI-SNI-Randomness Assignment)**.** Given a cluster $\mathcal{C}$ as defined in Definition 17, for HPC_2^+ gadgets $G^k \in \mathcal{C}$ random elements can be assigned arbitrarily to G^k as long as the following conditions hold:

$$\forall k, \forall \{r_i, r_j\} \in \mathcal{R}^k \mid \nexists k' \neq k : \{r_i, r_j\} \in \mathcal{R}^{k'} \tag{5}$$

$$\forall k, \forall r_i \in \hat{\mathcal{R}}^k \mid \nexists k' \neq k : r_i \in \hat{\mathcal{R}}^{k'} \vee r_i \in \mathcal{R}^{k'} \tag{6}$$

$$\exists i, \forall j \neq i \mid r_{i,j} \in \hat{\mathcal{R}}^k \tag{7}$$

$$\forall \{k, i, x\} : r_i^k = r_x \mid \nexists \{k', j\}, k' \neq k, j \neq i : r_j^{k'} = r_x, \tag{8}$$

where r_i is the i'th random element overall, $r_{i,j}$ is a random element hiding share indices i and j, $\hat{\mathcal{R}}^k$ is the set of unique randomness used for the gadget G^k, and $\mathcal{R}^k$ is the set of randomness used for the gadget G^k that is potentially reused by other gadgets.

Using Definition 17 and Definition 18 together with the HPC_2^+ gadget we now prove PINI for the corresponding cluster.

Theorem 6. $\mathsf{clusterHPC}_2$ *as defined in Algorithm 3, where Definition 18 is used as a replacement for Definition 16 in Line 3 and an output register* ($d_i^k \leftarrow \mathtt{Reg}[c_i^k]$) *is added, is* t*-PINI secure in the glitch-robust model.*

Proof. Assume the same simulator as used in the proof of Theorem 5, where we extend Algorithm 4 such that $\mathcal{X} \leftarrow \mathcal{X} \cup \{i\}$ when there is a probe placed on d_i^k. In particular, the simulator computes all values exactly as specified in Algorithm 3, except for some $v_{i,j}^k$ and the extension of d_i^k. Without loss of generality, we assume the first $t-1$ random values assigned to each gadget are unique to this gadget, resulting in the output share index 0 being only masked with unique randomness while all other output shares being masked with exactly one unique random value.

As before, we restrict our analysis to probes placed on w_i^k, $u_{i,j}^k$, $v_{i,j}^k$, $z_{i,j}^k$, c_i^k, as other probes only provide a subset of information. In addition, an attacker can probe an output share index i, which reveals the set $\{d_i^{k'} \mid \forall k'\}$. Again, we need to show that the above defined simulator results in the same output distribution for the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$ as the computation of the probes with Algorithm 3 extended by the computation of d_i^k. The only values computed differently in the simulator and Algorithm 3 are some $v_{i,j}^k$.

Recall, that if some $v_{i,j}^k$ is replaced by randomness, then either $i \notin \mathcal{X}$ or $j \notin \mathcal{X}$. For $j \in \mathcal{X}$, the only probes that can depend on $r_{i,j}^k$ are a probe placed on $v_{i,j}^k$ (see proof of Theorem 5) and/or a probe placed on the output share index j. Otherwise, if $i \in \mathcal{X}$, there is exactly one probe placed on either $z_{i,j}^k$, or c_i^k (see proof of Theorem 5) and/or a probe placed on the output share index i.

Case I. Assume exactly one probe is placed on either $v_{i,j}^k$, $z_{i,j}^k$, or c_i^k and no probe is placed on the output share indices i and j. Then the output distribution of the simulator and the gadget are equal following same argument as in the proof of Theorem 5.

Case II. Assume there is no probe on $v_{i,j}^k$, $z_{i,j}^k$, or c_i^k, but a probe placed on the outputs share index i. Each $d_i^{k'}$ is masked by at least one random value unique to the k'-th gadget. Removing the masking with a random value, which is unique to the gadget, requires either one internal probe (e.g., the randomness itself) or a probe on the output share index 0. Each $d_0^{k'}$ has t random values unique to the k'-th gadget and removing those masks requires either one internal probe (e.g., $c_0^{k'}$), or probes placed on all output share indices (which would require $t+1$ probes). Hence, removing the random behavior introduced by a random value unique to a gadget requires at least one internal probe. In addition, each $d_i^{k'}$ is masked by up to $t-1$ random values that are also used by other gadgets. Property (5) ensures that the only value having more than one random value in common with $d_i^{k'}$ is $c_i^{k'}$ (which is not probed). Therefore, removing one random value, shared with other gadgets, requires either some internal probe (e.g., the randomness itself) or some output share, which itself is masked by some gadget-unique randomness that requires at least one internal probe do be removed. The output distribution only changes if the randomness $r_{i,j}^k$ is observable through more than one output share (otherwise it is only observable through $v_{i,j}^k$). Removing all random behavior, except $r_{i,j}^k$, from one output share, requires at least $t-1$ internal probes, which means an adversary has no probes left to remove the gadget-unique randomness of some other output share. Therefore, the output distribution equals the one corresponding to the gadget.

Case III. Assume a probe is placed on $v_{i,j}^k$ and on the output share index j. Then d_i^k does not reveal new information and any output share $d_j^{k'}$ with $r_{i,j}^k = r_{j,i}^{k'}, k' \neq k$ is masked by $t-1$ other random values and removing them requires $t-1$ internal probes (with the same argument as above). As only $t-2$ probes are left to the adversary, $r_{i,j}^k$ is only observable through $v_{i,j}^k$, and replacing $v_{i,j}^k$ with randomness does not change the output distribution.

Case IV. There is exactly one probe placed on either $z_{i,j}^k$ or c_i^k and the output share index i is probed. This case is similar to the last case, in that the output d_i^k does not reveal any additional information and all other outputs shares are masked with $t-1$ other random values and removing those requires $t-1$ probes (with the same argument as above). Again the adversary has only $t-2$ probes left, which means $r_{i,j}^k$ is only observable through $v_{i,j}^k$ and, hence, the output distribution is not changed by the simulator.

We conclude, that there exists a simulator for the probes $\mathcal{P}_\mathcal{I} \cup \mathcal{P}_\mathcal{O}$ which proves Theorem 6. □

This optimization does not change the security property of the original circuit, as the cluster remains PINI. However, the latency and area impact is higher than using HPC_2 gadgets.

4.3 Double SNI Gadgets

Of course, our optimization techniques can also be used for DoubleSNI [CS20], i.e., PINI gadgets composed from an SNI refresh and an SNI multiplication gadget. As ultimately this is a composition of SNI gadgets, the proceeding follows the methods outlined in Section 3, where the resulting circuit is only PINI if the relaxed clustering with gadget-unique randomness is used (Section 3.2).

When using the DOM multiplication gadget and the corresponding refresh gadget, used in Section 3, PINI security follows directly from the proof of clusterDOM (given in Section 3) and the proof of DoubleSNI (given in [CS20]).

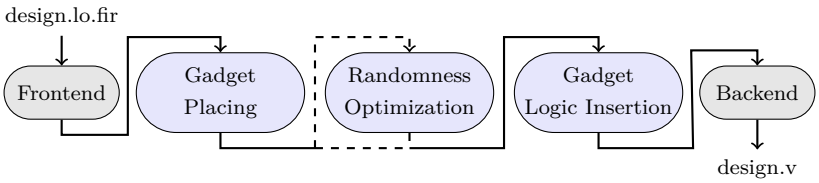


Figure 5: Structure of our MLIR-based tool for randomness optimization.

5 Implementation

To analyze the practical impact of the proposed randomness optimization, we implemented a software tool based on the compiler framework MLIR [LAB+21]. For this, we created an Intermediate Representation (IR) dialect, called SecFIR, which is closely related to the FIRRTL [IKL+17] dialect of MLIR. Eventually, our tool[2] allows transforming an insecure gate-level design into a side-channel-secure one with optimized randomness reuse.

Tool Flow. The general tool structure is shown in Figure 5, where all passes can be enabled and controlled by command-line arguments. Following a pass-based structure and using a dedicated IR results in a flexible tool that is well equipped for further extensions and integration into modern EDA tools. As a first step, and as part of the frontend, a parser reads a design in FIRRTL and translates it to our internal IR (SecFIR). A FIRRTL design can be generated from Chisel using a dedicated compiler [IKL+17] or from Verilog utilizing Yosis [yos]. Please note, we currently support designs consisting of a single module with registers and Boolean operators, i.e., a gate-level netlist. Afterward, as a second step within the frontend, multiple passes are applied sequentially to prepare the design for masking and optimization. This includes a transformation of combinational logic to an XOR-AND Graph (XAG), i.e., a logic only consisting of XOR, AND, and NOT gates. At its core, our tool consists of three passes performing the actual masking and randomness optimization before a backend pass outputs a protected design in Verilog.

5.1 Gadget Placing

The first pass identifies positions within the design where side-channel secure gadgets need to be placed. Precisely, nonlinear AND gates are replaced by masked gadgets and refreshing gadgets are inserted where necessary, while linear gates (i.e., XOR and NOT) remain unaffected. As discussed before, we either support DOM [CGLS21] (cf. Section 3) or HPC_2 gadgets [CGLS21] (cf. Section 4), targeting different notions of composition and impacting the performance and properties of the resulting design.

Probing Security. When composing for probing security as the final security notion, we replace all AND gates with DOM multiplication gadgets. Then additional DOM refresh gadgets are added where required according to the `tightPROVE` algorithm [BGR18].

Strong Non-Interference. For SNI, we again replace all AND gates with DOM multiplication gadgets. However, afterward, we insert DOM refresh gadgets for all gates with fan-out larger than one, namely one refresh gadget for all but one fan-out. This ensures that the design is NI, as shown by Barthe et al. [BBD+16] (share-wise implementations of XOR and NOT are NI). To ensure SNI, Barthe et al. add an SNI refresh layer on either the input or the output of the considered circuit. As we know the internal structure of the design, we can do this in a slightly optimized fashion by ensuring that there is no path

[2] Available at https://github.com/Chair-for-Security-Engineering/SAIREDA

from an input to an output in the simplified computation tree. This can be ensured by inserting one SNI refresh gadget when necessary, i.e., only on paths where no SNI gadget already exists. This is sufficient to remove the dependency between input and output and, hence, to ensure SNI.

Probe Isolating Non-Interference. Lastly, our tool supports two different methods to achieve PINI-secure designs. The first method replaces all AND gates with DoubleSNI gadgets, i.e., an DOM refresh gadget followed by an DOM multiplication gadget [CS20]. For the second method, the tool replaces all AND gates with HPC_2 gadgets. PINI is invariant under composition [CS20]. Hence, since DoubleSNI and HPC_2 gadgets offer composability under the notion of PINI and the trivial, i.e., share-wise, implementations of linear functions (XOR and NOT) offer the same, the resulting composed circuits are PINI-secure designs as well.

5.2 Randomness Optimization

The second pass optimizes the use of randomness across gadgets. For this, we first determine appropriate clusters of gadgets and then optimize the randomness usage within each such cluster, as outlined in Section 3 and Section 4. Note, that the gadgets we use all have the same initial requirement of randomness, namely $m = \frac{t(t+1)}{2}$ random elements per gadget.

5.2.1 Clustering Gadgets

Naturally, the algorithm assigning gadgets to different clusters impacts the overall result of the optimization pass. Intuitively, larger clusters allow more sharing of randomness and, hence, provide more efficient solutions (more details are given in Section 6). It is, therefore, desirable to include as many gadgets as possible in one cluster, which in itself is a non-trivial optimization problem, that is highly dependent on the design structure. For example, a clear hierarchical design, with many parallel paths in the simplified computation tree, allows for larger clusters than a design structure with strong interdependencies between gadgets. Moreover, when considering the combination of differently-sized clusters, it is not clear that optimizing each cluster yields a global optimum. Sometimes, creating multiple medium-sized clusters instead of a single large and plenty of small clusters may result in a better solution. Hence, given these constraints, the complexity of the original optimization problem increases.

Our implementation is rooted in the simple but heuristic approach of going through the gadgets one by one (in order of the design specification) and assigning them to the first suitable cluster. If no such cluster exists, the algorithm creates a new one. For this, we determine gadgets that obey the clustering rules outlined before, leveraging a simple information flow analysis. Here, SNI gadgets stop the backtracking through the computation tree, while for all other gates (including PINI gadgets), the search continues with all inputs or outputs, in accordance with the simplified computation tree.

5.2.2 Randomness Assignment

After achieving clustering, finding efficient randomness assignments within a single cluster of gadgets poses an additional optimization problem. For this, we present three different optimization algorithms: (i) a heuristic algorithm for DOM gadgets, (ii) a heuristic algorithm for HPC_2 gadgets, and (iii) a Satisfiability Modulo Theories (SMT) solver based algorithm for DOM gadgets. Further, in cases where gadgets require some unique randomness (i.e., in case of relaxed clustering), we first assign those unique random values and then run the assignment algorithms for the remaining fresh randomness only.

Algorithm 5: Heuristic randomness-assignment algorithm for DOM gadgets.

```
1  function HeuristicDOM(𝒞):
       Require: n = |𝒞| ≥ 2
2      m̂ ← m = t(t+1)/2
3      ℛ^0 ← ⟨r_0, ..., r_{m-1}⟩
4      for k = 1 to n − 1 do
5          for i = 0 to m̂ do
6              ℛ^k ← r_i
7              for j = i + 1 to m̂ do
8                  if {ℛ^0, ..., ℛ^{k-1}, ℛ^k ∪ r_j} fulfills Property (5) then
9                      Store state of algorithm
10                     ℛ^k ← ℛ^k ∪ r_j
11                 if |ℛ^k| = m then
12                     Continue with next gadget (Line 4)
13                 if j = m̂ − 1 then
14                     Return to last stored state in algorithm (Line 9)
15             if i = m̂ − 1 then
16                 m̂ ← m̂ + 1
17                 Start again with same gadget (Line 5)
18     return {ℛ^0, ..., ℛ^{n-1}}
```

Heuristic DOM Algorithm. The first algorithm is outlined in Algorithm 5 and finds a randomness assignment following the rules from Definition 14, i.e., a distribution where no two gadgets share more than one random element. Here, we first initialize the number of available random elements $\hat{m}$, the number of randomness each gadget requires, and assign the first m random elements to the first gadget. Afterward, we iteratively search for an assignment $\mathcal{R}^k$, that together with the already found assignments, does not violate Definition 14. If no such assignment can be found, the number of available random elements $\hat{m}$ is incremented.

Heuristic HPC$_2$ Algorithm. The second algorithm has the same working principle as the first. However, in order to follow the rules from Definition 18, it is necessary to keep track of the cross domains at which a random element is used. Hence, when searching for a new random element at position i, i.e., for a certain cross-domain of a gadget, the algorithm only needs to search in the set of random elements already used at position i. This reduces the search space, which in turn makes the search more efficient. If no valid assignment for position i exists, the algorithm assigns a random element to this position that is not already part of any previous assignments.

SMT-based DOM Algorithm. The last algorithm uses the SMT solver Z3 [dMB08] and returns either a valid assignment of $\hat{m}$ random elements for n gadgets, or *false* if no such assignment exits. We encode the rule of Definition 14 with two sets of Boolean variables, where $u_{k,i} = 1$ indicates that the random element r_i is used in the gadget G^k, and $s_{k,k',i} = 1$ indicates that the gadgets G^k and $G^{k'}$ share the random element r_i, for $0 \leq k, k' < n$ and $0 \leq i < \hat{m}$. Further, we use three constraints to ensure that each gadget has m random elements assigned (Equation (9)), and each pair of gadgets only shares at

most one random element (Equation (10) and Equation (11)).

$$0 \leq k < n : \sum_{i=0}^{\hat{m}-1} u_{k,i} = m \tag{9}$$

$$0 \leq k < k' < n : \sum_{i=0}^{\hat{m}-1} s_{k,k',i} \leq 1 \tag{10}$$

$$\bigwedge_{k=0}^{n-1} \bigwedge_{k'=k+1}^{n-1} \bigwedge_{i=0}^{\hat{m}-1} (u_{k,i} \wedge u_{k',i} \wedge s_{k,k',i}) \vee (\overline{u}_{k,i} \wedge \overline{u}_{k',i} \wedge \overline{s}_{k,k',i}) \tag{11}$$

To solve the optimization problem this approach starts by requesting a solution with m random elements and increases the number of random elements until the solver returns a valid assignment. In theory, this algorithm returns an optimal solution, however, it may take a long time to find it. In order to make the application practicable, we set a timeout for the SMT solver of 5 minutes, after which it continues with a higher number of random elements. Hence it is not guaranteed that the found solution is optimal but well enough to serve as a baseline for comparison. We determined the timeout experimentally to keep the timing practical while still achieving good results.

5.3 Gadget Logic Insertion

Finally, the last pass replaces the gadget instructions with the actual logic of the corresponding gadget for a given security order t. In addition, the transformation pass replaces all XOR gates with a share-wise XOR and all NOT gates with an inversion of the corresponding first share. Hence, the result is a masked design adhering to the specified order and security notion.

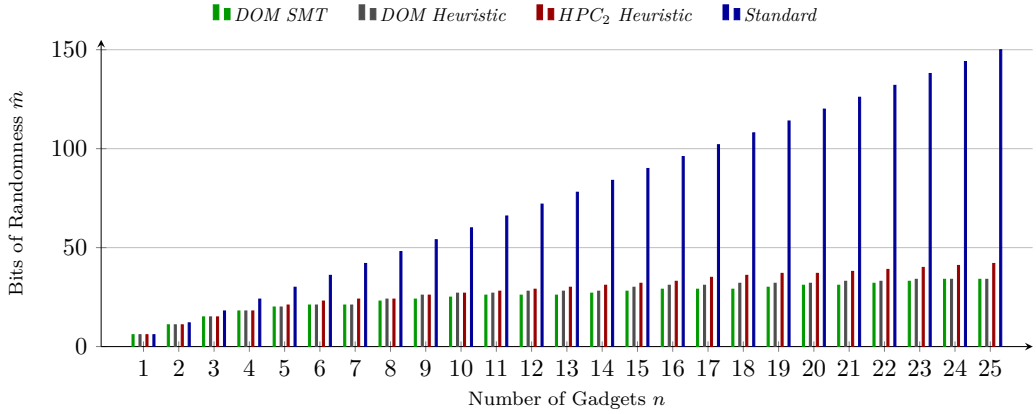
6 Evaluation

In the following, we first evaluate the general effectiveness of the outlined strategies for randomness reduction before considering concrete, real-world implementations of cryptographic designs.

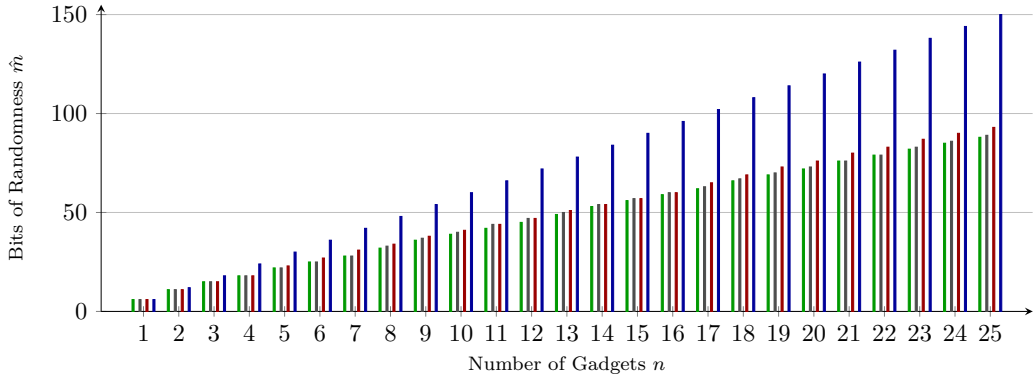
6.1 Randomness-Distribution Algorithms

Given the randomness assignment rules for RMO-SNI and R-PINI, as defined in Definition 12 and Definition 16 respectively, there is no closed formula for the optimal number of required randomness. Instead, we compare the randomness requirements identified by the three algorithms described in Section 5.2.2 and the standard state-of-the-art assignment, as shown in Figure 6a (exemplarily for clusters of 3^{rd} order gadgets). While the standard assignment has a linear slope, the outlined optimization strategies only increase logarithmically, resulting in a significant reduction for larger clusters. However, dealing with randomness assignment within the relaxed clustering, we further notice that, even though the same algorithms are used for the random elements shared between gadgets, the number of gadget-unique randomness increases linearly with the number of gadgets, dominating the requirement. This results in a nearly-linear increase for the overall required randomness, as shown in Figure 6b (again, exemplarily for clusters of 3^{rd}-order gadgets). Hence, much larger clusters are required to have a comparable reduction to the strict clustering.

When comparing the three assignment algorithms for different cluster sizes, we notice that both algorithms for DOM gadgets provide similar results for the amount of randomness.



(a) 3^{rd} Order (strict clustering)



(b) 3^{rd} Order (relaxed clustering)

Figure 6: Randomness requirements for clusters of different sizes. The numbers are determined by the three algorithms described in Section 5.2.2 and the standard assignment, where all gadgets get unique randomness only.

In contrast, the third algorithm, specifically targeting HPC_2, results in a slightly higher randomness requirement due to the additional constraint with regards to cross domains. However, according to the graphics shown in Figure 6, we can conclude that both heuristic algorithms also result in near-optimal assignments while simultaneously being orders of magnitude faster, as shown in Table 2.

6.2 Real-World S-boxes

The results shown in Figure 6 indicate a clear reduction in the amount of fresh randomness for large clusters; however, up to now, it remains unclear to which extend clusters can be created in practical designs. Hence, to evaluate the impact of our randomness optimization for practical designs, we selected multiple cryptographic S-boxes commonly considered for side-channel secure implementations. Namely, we analyze S-boxes of PRESENT [BKL+07], Keccak [BDPvA11], PRINCE [BCG+12], and two variants of AES [Can05, BP12]. More precisely, we use implementations that are optimized for masking in the sense that each construction requires a minimal amount of cascaded multiplication gadgets [CGLS21, BP12]. Further, we consider the security orders $t = 2, 3, 4$ for each case study while applying the four strategies for gadget placing discussed in Section 5.1 for both the strict and relaxed

Table 2: Execution time for the three algorithms described in Section 5.2.2 running on a system with a 2.6 GHz CPU and 32 GB RAM. Values are given in milliseconds, seconds, and minutes for easier comprehension.

	2^{nd} Order		3^{rd} Order		4^{th} Order	
	$n = 10$	$n = 15$	$n = 10$	$n = 15$	$n = 10$	$n = 15$
DOM SMT	10.515 m	30.541 m	106.759 m	140.971 m	343.838 m	446.273 m
DOM Heuristic	0.106 ms	0.153 ms	8.158 ms	15.465 ms	5.594 s	21.857 s
HPC_2 *Heuristic*	0.106 ms	0.139 ms	0.910 ms	1.136 ms	126.245 ms	156.870 ms

Table 3: Optimization opportunities for cryptographic S-boxes, using the strict and relaxed clustering methods with different gadget-placing strategies (Section 5.1). Here, t indicates the security order, and n the number of multiplication and refresh gadgets, while *Std* and *Opt* refer to without and with optimization, respectively, and *Red* gives the reduction of randomness in percent.

			Probing Security				SNI				PINI - *DoulbeSNI*				PINI - HPC_2			
		t	n	Std	Opt	Red	n	Std	Opt	Red	n	Std	Opt	Red	n	Std	Opt	Red
PRESENT	[CGLS21]	2	4	12	11	8.3%	6	93	51	45.2%	8	24	17	29.2%	4	12	11	8.3%
	strict*	3	4	24	23	4.2%	6	186	136	26.9%	8	48	41	14.6%	4	24	23	4.2%
		4	4	40	39	2.5%	6	310	260	16.1%	8	80	73	8.8%	4	40	39	2.5%
	[CGLS21]	2	4	12	10	16.7%	7	93	69	25.8%	8	24	18	25.0%	4	12	10	16.7%
	relaxed	3	4	24	22	8.3%	7	186	133	28.5%	8	48	36	25.0%	4	24	22	8.3%
		4	4	40	38	5.0%	7	310	237	23.5%	8	80	68	15.0%	4	40	38	5.0%
Keccak	[BDPvA11]	2	5	15	11	26.7%	5	45	21	53.3%	10	30	14	53.3%	5	15	8	46.7%
	strict*	3	5	30	26	13.3%	5	90	60	33.3%	10	60	40	33.3%	5	30	21	30.0%
		4	5	50	46	8.0%	5	150	120	20.0%	10	100	80	20.0%	5	50	40	20.0%
	[BDPvA11]	2	5	15	12	20.0%	5	45	33	26.7%	10	30	22	26.7%	5	15	11	26.7%
	relaxed	3	5	30	26	13.3%	5	90	66	26.7%	10	60	44	26.7%	5	30	23	23.3%
		4	5	50	46	8.0%	5	150	120	20.0%	10	100	80	20.0%	5	50	41	18.0%
PRINCE	[CGLS21]	2	6	18	17	5.6%	8	108	48	55.6%	12	36	24	33.3%	6	18	18	0.0%
	strict*	3	6	36	35	2.8%	8	216	132	38.9%	12	72	56	22.2%	6	36	36	0.0%
		4	6	60	59	1.7%	8	360	272	24.4%	12	120	104	13.3%	6	60	60	0.0%
	[CGLS21]	2	6	18	14	22.2%	10	108	78	27.8%	12	36	27	25.0%	6	18	14	22.2%
	relaxed	3	6	36	29	19.4%	10	216	151	30.1%	12	72	53	26.4%	6	36	29	19.4%
		4	6	60	53	11.7%	10	360	260	27.8%	12	120	95	20.8%	6	60	53	11.7%
AES	[Can05]	2	40	120	87	27.5%	29	492	114	76.8%	80	240	101	57.9%	40	120	116	3.3%
	strict*	3	40	240	207	13.8%	29	984	284	71.1%	80	480	257	46.5%	40	240	236	1.7%
		4	40	400	367	8.2%	29	1640	622	62.1%	80	800	516	35.5%	40	400	396	1.0%
	[BP12]	2	34	102	85	16.7%	28	486	123	74.7%	68	204	108	47.1%	34	102	95	6.9%
	strict*	3	34	204	187	8.3%	28	972	303	68.8%	68	408	275	32.6%	34	204	197	3.4%
		4	34	340	323	5.0%	28	1620	664	59.0%	68	680	531	21.9%	34	340	333	2.1%
	[Can05]	2	40	120	90	25.0%	46	492	337	31.5%	80	240	167	30.4%	40	120	83	30.8%
	relaxed	3	40	240	180	25.0%	46	984	597	39.3%	80	480	309	35.6%	40	240	155	35.4%
		4	40	400	322	19.5%	46	1640	913	44.3%	80	800	495	38.1%	40	400	246	38.5%
	[BP12]	2	34	102	79	22.5%	42	486	333	31.5%	68	204	146	28.4%	34	102	72	29.4%
	relaxed	3	34	204	165	19.1%	42	972	589	39.4%	68	408	282	30.9%	34	204	141	30.9%
		4	34	340	301	11.5%	42	1620	901	44.4%	68	680	481	29.3%	34	340	235	30.9%

*Design results in weaker composability notion (RMO-SNI or R-PINI).

clustering methods. The results are presented in Table 3.[3]

We first observe that for most of the designs, we achieve a significant reduction in the amount of required fresh randomness while implementations with high initial randomness requirements benefit the most. The only design where our method results in no optimization at all is the PRINCE S-box when selecting strict clustering and HPC_2 gadgets, due to a design structure without any parallelism and, hence, no clusters following Definition 15. Contrary, the Keccak S-box has a mostly parallel structure, allowing significant reduction even with HPC_2 gadgets. In terms of optimization opportunities, the SNI composition strategy performs best, with up to 76.8% for the 2^{nd}-order AES implementation. This large reduction can be achieved due to the high number of available gadgets (SNI composition requires by far the highest number of gadgets) and the SNI-specific clustering rules (i.e., the independence property of inputs and outputs). However, even this gain cannot compensate for the high initial randomness demand and, hence, even when optimized, the randomness requirements remain larger than that of the probing secure and HPC_2 standard variants. However, this also means that the composition of SNI gadgets initially contains the most (unnecessary) overhead in terms of randomness. The lowest randomness requirement is generally achieved by the relaxed clustering in combination with HPC_2 (with additional output registers). This also achieves the strongest notion of composition, i.e., both SNI and PINI. However, the implementation achieving probing security without the need for additional registers is nearly as effective, but the final result does not adhere to any notion of composition. Nevertheless, this is a viable option for a cipher implementation where every component is only used once and where no feedback loops exist.

Besides, when comparing the strict and relaxed clustering methods, none of them consistently results in the lowest number of random elements. While the strict clustering method always outperforms the relaxed clustering in an SNI-composition or DoubleSNI implementation, the situation is nearly vise versa for probing security and implementations composed from HPC_2. We also observe a decrease of optimization potential with increasing security order in the strict version, while the opportunities can also increase in the relaxed version. The reason for this is a fixed set of clusters across security orders, providing less randomness reuse opportunities without violating the requirement that each pair of gadgets is not allowed to share more than one random element. The relaxed clustering has the same requirement; however, with increasing security order, the number of reusable random elements increases proportionally to the set of gadget-unique elements.

6.3 Parallel S-box Structures

Hardware implementations often use parallel structures to trade area for performance. The same holds for cryptographic ciphers, where often multiple S-box instances are implemented in parallel. Such parallel structures are a natural fit for the optimization strategies presented in this work, and we, therefore, extend our analysis of the AES S-box [BP12] for various levels of parallelism[4]. In Figure 7, we show the proportion of the used randomness in the optimized case with respect to the unoptimized variant, i.e., where every gadget receives unique fresh randomness and where there is no cross-gadget randomness re-usage. For the strict clustering (Figures 7a to 7c), the optimization opportunities increase significantly with each additional parallel S-box instance. More precisely, the results indicate a logarithmic increase in the number of random elements that can be removed with each additional S-box instance. This means that adding only a few parallel structures can already result in huge gains. In contrast, the gain is much smaller for the relaxed clustering (Figures 7d to 7f)

[3]Excluding the area required for the generation of fresh randomness, our optimization has no impact on the required area or latency of the design, except for the relaxed clustering in combination with HPC_2, where we require $n \cdot t$ additional registers compared to the usual HPC_2 gadget composition.

[4]Parallel structures of smaller S-boxes result in a similar, but slightly less pronounced behavior as the analyzed AES S-box and we omit the corresponding data for brevity.

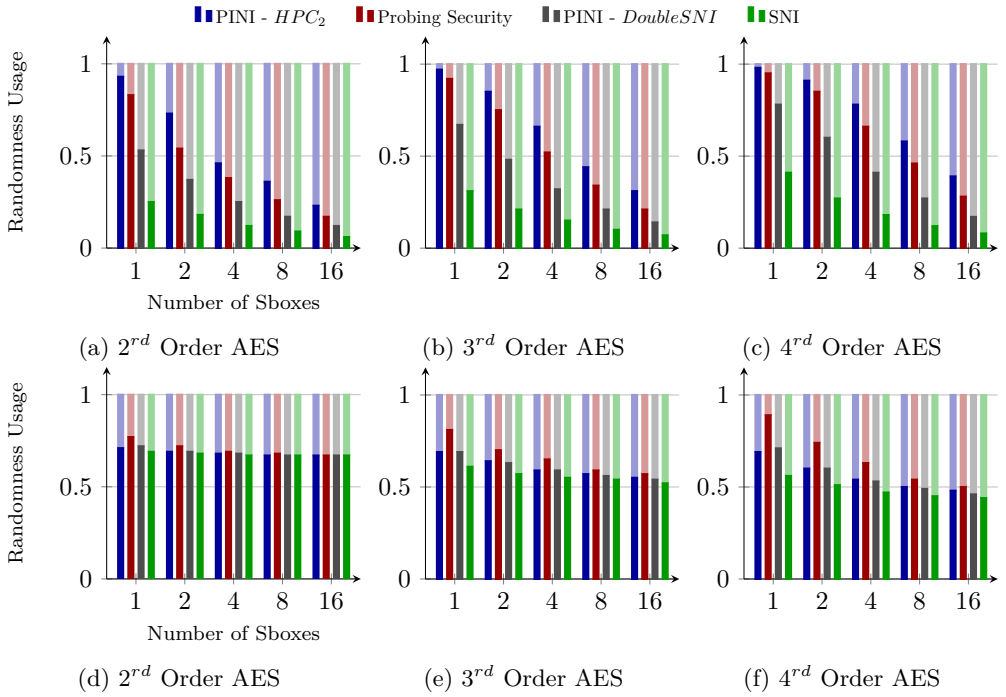


Figure 7: Proportion of used random elements when optimized compared to the standard variant in parallel implementations of the AES S-box. The first row shows the results corresponding to the strict clustering while the second row to the relaxed clustering.

as a large portion of random elements are used uniquely by one gadget. Interestingly, the gain is nearly constant in the 2^{nd}-order case but improves with increasing security order due to the rise in the amount of reusable random values. This indicates that the (unnecessary) overhead caused by gadget composition in terms of randomness accumulates and impairs with increasing security order.

The difference between the strict and relaxed clustering methods is even more obvious when considering absolute numbers. In Figure 8, we show the randomness requirement per S-box when implementing multiple instances in parallel. For strict clustering (Figures 8a to 8d) the significant reduction from Figure 7 translates to numbers that match or surpass the randomness requirements of a standard 1^{st}-order implementation in some cases. For the relaxed clustering (Figures 8e to 8h), we observe a nearly constant requirement for 2^{nd}-order, and a logarithmic behavior for 3^{rd}- and 4^{th}-order security, which indicates that higher-order security converges to a randomness requirement higher than the 1^{st} order.

6.4 Full AES Cipher

Finally, we analyze a complete AES-128 implementation to evaluate the optimization opportunities over multiple rounds, including permutation layers. In Table 4, we show the number of required randomness for an unrolled AES implementation, where the S-box is implemented either using the version of Canright [Can05], or Boyar and Peralta [BP12]. For the implementation using the Canright S-box, the `tightPROVE` algorithm [BGR18] did not finish within 48 hours, after which we canceled the computation.

For the strict clustering in combination with 4^{th} order SNI composition, the randomness-assignment algorithm (Algorithm 5) did not finish within 7 days (most of the other configurations finished within minutes, others within hours). Hence, we introduced a

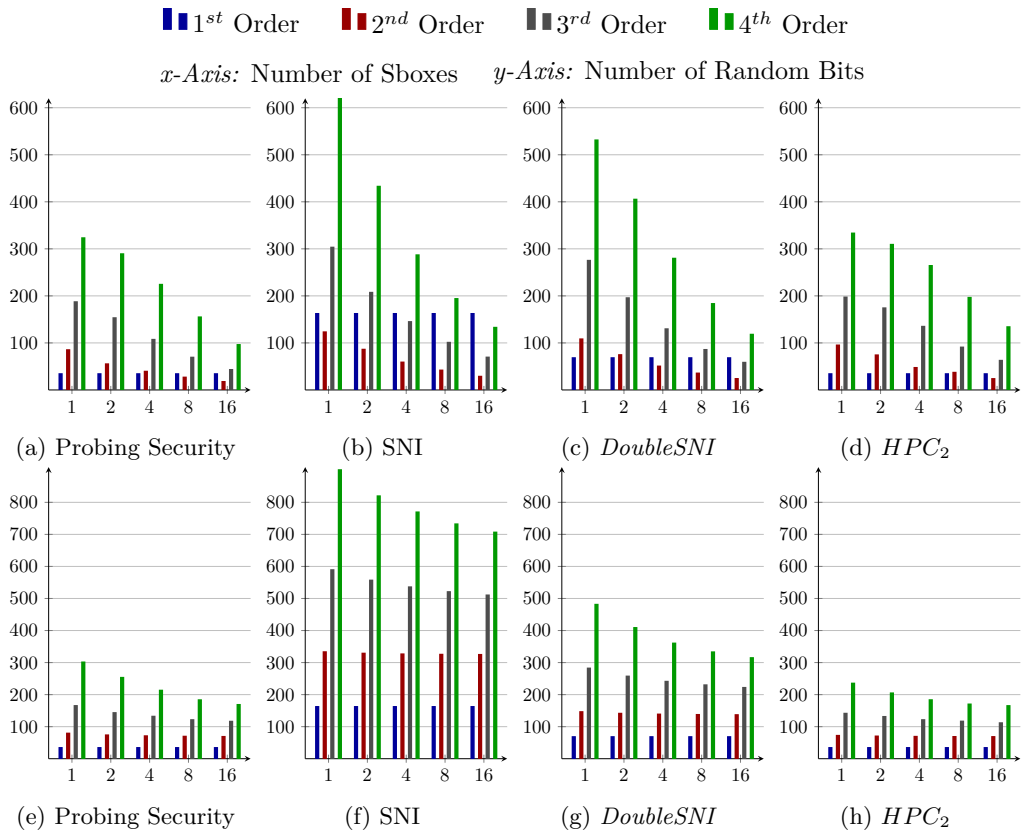


Figure 8: Randomness requirements per S-box (in average) when implementing multiple AES S-box instances in parallel. The first row shows the results corresponding to the strict clustering while the second row to the relaxed clustering. The data corresponding to 1^{st} order is drawn from the standard randomness assignment and shown for comparison.

maximum cluster size of 1750 gadgets, leading to a randomness-assignment time of roughly 55 hours. This results in an optimization/time trade-off since larger clusters allow better optimization but require more time.

In general, we see that strict clustering works much better than relaxed clustering, except for compositions using HPC_2 gadgets. This is in contrast to the already discussed data from smaller designs, where the best clustering method was more dependent on the overall structure and used composition strategy. The reason is that while the randomness reduction opportunities increase with the cluster size, the number of required unique randomness increases with the number of gadgets. SNI gadgets remove dependencies in the simplified computation tree and, hence, a higher number of SNI gadgets generally results in larger clusters. However, even as the relaxed clustering results in larger clusters than the strict clustering, the difference is not large enough to compensate for the required unique randomness. The reverse is true when using HPC_2 gadgets since PINI does not remove any dependencies and, hence, more gadgets do not necessarily increase the cluster size. In fact, due to the permutation layer of AES, nearly all gadgets depend on all other gadgets, resulting in an average cluster size of 1.2 gadgets. In contrast, HPC_2 in combination with relaxed clustering can leverage the SNI property of HPC_2^+.

Interestingly, the Canright S-box, in combination with strict clustering and *DoubleSNI* gadgets, achieves the best result, highlighting the fact that the most efficient implementation, in terms of area and latency, is not necessarily best suited for optimizing randomness.

Table 4: Randomness requirements in number of bits for AES-128 (without key schedule) for different optimization strategies.

			Probing Security		SNI		PINI - *DoulbeSNI*		PINI - HPC_2	
		t	Std	Opt	Std	Opt	Std	Opt	Std	Opt
strict	[Can05]*	1	†	N/A	31 712	N/A	12 800	N/A	6 400	N/A
		2	†	†	95 136	2 596	38 400	2 169	19 200	16 782
		3	†	†	190 272	6 205	76 800	5 235	38 400	34 860
		4	†	†	3 171 120	13 302**	128 000	10 405	64 000	59 388
	[BP12]*	1	5 440	N/A	31 392	N/A	10 880	N/A	5 440	N/A
		2	16 320	2 329	94 176	2 389	32 640	2 725	16 320	14 250
		3	32 640	5 822	188 352	5 731	65 280	6 713	32 640	29 559
		4	54 400	12 483	313 920	11 927**	108 800	14 204	54 400	50 286
relaxed	[Can05]*	1	†	N/A	31 712	N/A	12 800	N/A	6400	N/A
		2	†	†	95 136	63 446	38 400	25 612	19 200	12 803
		3	†	†	190 272	97 064	76 800	39 382	38 400	19 636
		4	†	†	3 171 120	131 499	128 000	53 578	64 000	29 270
	[BP12]*	1	5 440	N/A	31 392	N/A	10 880	N/A	5 440	N/A
		2	16 320	10 898	94 176	62 806	32 640	21 775	16 320	10 886
		3	32 640	17 159	188 352	96 067	65 280	33 634	32 640	16 903
		4	54 400	23 791	313 920	130 142	108 800	45 943	54 400	25 165

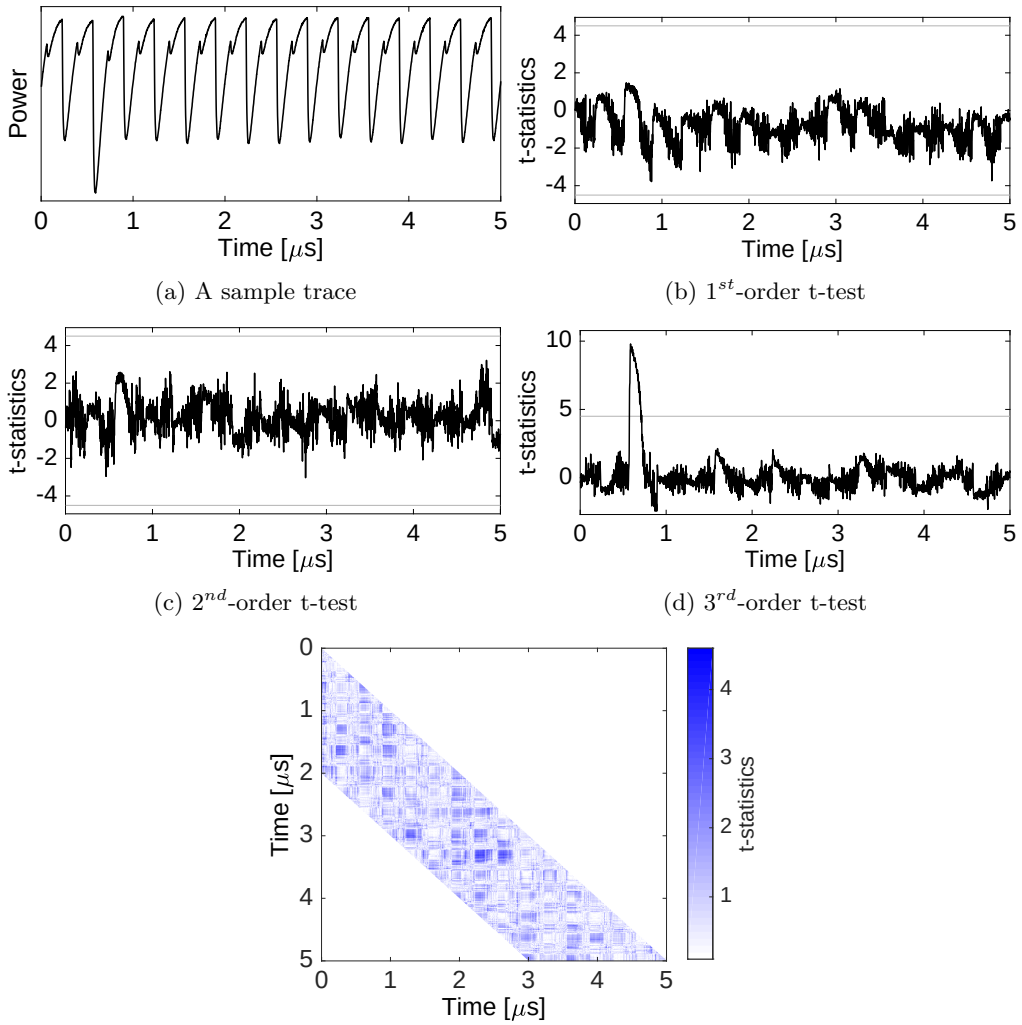
*Referenced S-box is embedded in an unrolled AES-128; † `tightPROVE` did not finish in 48h.
**Maximum cluster size restricted to 1750 gadgets.

The most efficient implementation, considering area or latency, results from the Boyar and Peralta S-box, in combination with the probing-secure composition or HPC_2 gadgets, respectively. However, those implementations have a higher 1^{st}-order randomness demand than the best-optimized 3^{rd}-order implementation. The best optimization of the version using the Boyar and Peralta S-box is achieved by the combination of strict clustering with the composition achieving probing security via the `tightPROVE` algorithm for 2^{nd}-order security and the SNI composition for 3^{rd}- and 4^{th}-order security.

6.5 Experimental Analysis

For the sake of completeness in assessing the security of our optimized construction, we further performed experimental analysis by conducting a leakage assessment on SCA traces measured from an FPGA prototype. To this end, and in order to be consistent with the state-of-the-art, we mainly followed the procedure explained by Knichel et al. [KMMS22]. More precisely, we implemented our designs on an SCA-evaluation toolkit board and measured the power consumption traces when the target design was fed with either fixed or random input (in both cases, masked inputs). Then, we analyzed the collected 100 million traces with fixed-versus-random t-tests in both univariate and multi-variate forms for first and higher orders.

As the setup, we used a SAKURA-X [HKSS12], where a Kintex-7 FPGA is embedded. The measurements of the power consumption traces were conducted with a digital oscilloscope at the sampling rate of 500 MS/s and an FPGA clocked at a frequency of 3 MHz. For this case study, we analyzed the 2^{nd}-order design consisting of 16 parallel AES S-boxes with relaxed clustering in combination with HPC_2 gadgets. This represents the entire SubBytes operation suitable for a round-based implementation of the AES encryption. The design requires 1092 fresh random bits and has a latency of 8 clock cycles. Hence, we again followed Knichel et al. [KMMS22] for an FPGA-friendly implementation



(a) A sample trace

(b) 1^{st}-order t-test

(c) 2^{nd}-order t-test

(d) 3^{rd}-order t-test

(e) 2^{nd}-order multi-variate t-test

Figure 9: Experimental analysis, fixed-versus-random t-test, of 2^{nd}-order 16 parallel AES S-boxes with relaxed clustering in combination with HPC_2 gadgets, using 100 million traces.

of a 31-bit Linear Feedback Shift Register (LFSR) for each required fresh random bit, updated at every clock cycle. A sample power trace (covering the entire 8 clock cycles) and the result of t-tests are depicted in Figure 9. The results indeed confirm our claim, i.e., security up to 2^{nd} order. More precisely, our optimization methods, re-using randomness between gadgets, do not degrade the security order. Note that for the univariate t-tests, we continued the analyses up to the 3^{rd}-order. However, as 3^{rd}-order leakage is already detected in the univariate t-test, we did omit the univariate analyzes for 3^{rd}-order.

7 Related and Future Works

To the best of our knowledge, this is the first work elaborating global randomness optimization for side-channel-secure gadget compositions through cross-gadget randomness re-usage targeting hardware. Existing work analyses randomness reduction in software through different means as described below. We give a quantitative comparison based on

Table 5: Comparison of randomness requirements for AES-128 with different optimization techniques for software (S) and hardware (H) implementations.

	Bits of Randomness						
	$t=1$	$t=2$	$t=3$	$t=4$	S/H	Base	Approach
[RP10]	7 680	23 040	46 080	76 800	S	-	Hand-crafted
[Can05]*	6 400	19 200	38 400	64 000	S,H	-	Tower field, PINI
[BP12]*	5 440	16 320	32 640	54 400	S,H	-	Depth reduction, PINI
[BBP+16]	7 680	20 480	40 960	64 000	S	[RP10]	Optimized gadgets
[BGR18]	5 440	16 320	32 640	54 400	S,H	[BP12]	Gadget reduction, SNI
[WGS+20]	3 456	9 088	16 896	26 880	S	[RP10]	Multiple-operation gadgets
[FPS17]	16	3 984	7 968	27 888	S	[RP10]	Randomness reuse
This work	N/A	2 329	5 822	12 483	H	[BP12]	Randomness reuse
This work	N/A	2 169	5 235	10 405	H	[Can05]	Randomness reuse
[CGZ20]	N/A	384	864	1 536	S	[RP10]	Randomness expansion

*Referenced S-box is masked using HPC_2 gadgets and embedded in an unrolled AES-128.

an AES-128 implementation without key schedule in Table 5. As a baseline for comparison, we use the standard masked software implementation from Rivain and Prouff [RP10] and two unrolled hardware versions using the S-box implementations from Canright [Can05] and Boyar and Peralta [BP12], both with HPC_2 gadgets [CGLS21] for side-channel security.

Gadget Reduction. In the first line of research, Belaïd et al. [BGR18] reduce the number of SNI-refresh gadgets required to achieve probing security. The proposed algorithm `tightPROVE` either verifies probing security or indicates positions within the circuit where SNI-refresh gadgets are required. We evaluated the combination of `tightPROVE` with our approach in Section 6. The introduction of PINI [CS20] can also be seen as a contribution in this direction, as it allows a trivial implementation of linear gadgets, removing the need for SNI-refresh gadgets entirely. PINI gadgets are used to derive the two baseline hardware implementations in Table 5.

Gadget Optimization. Gadgets themselves can be optimized with regard to randomness as well. Belaïd et al. [BBP+16] analyze the theoretical minimum of required randomness for a t-order probing-secure multiplication gadget ($t+1$ bit for $t \geq 3$, and t bit otherwise) and give concrete instances for $t = 2, 3, 4$. In addition, they provide a multiplication gadget for arbitrary order that requires $\frac{t^2}{4} + t$ random bits when t is even and $\frac{t^2-1}{4} + t$ random bits when t is odd, compared to $\frac{t(t+1)}{2}$ random bits for the DOM gadget. However, their gadgets are only NI instead of SNI, and special care needs to be taken when composing them. Similarly, Cassiers and Standaert present a PINI gadget with a reduced randomness requirement for higher-order masking [CS19]. Their gadget requires $\lfloor \frac{t^2}{4} \rfloor + 2t + 1$ bits of randomness instead of $\frac{t(t+1)}{2}$ bits for HPC_2, which is beneficial for $t \geq 7$. We omit this work in Table 5 as it is not beneficial for the compared security orders. Wang et al. [WGS+20] construct gadgets that perform multiple multiplications in parallel and thus are able to optimize the gadget even further based on randomized linear codes. For n parallel multiplications their scheme requires $2t^2 + \frac{t(t+1)}{2}$ random bits, however, the resulting gadget supports a weaker composability notion than MO-SNI (similar but not equivalent to RMO-SNI). Currently, there exists no glitch-robust version of those randomness-reduced gadgets, and, hence, the cost in terms of area and latency is yet unknown.

We expect our approach to also apply to gadgets with reduced randomness requirements but leave a thorough analysis for future work.

Global Randomness Reuse. The work most similar to ours was provided by Faust et al. [FPS17], in which they also reuse randomness in different gadgets. For this, they introduce a new composability notion (Secure with Common Randomness (SCR)), where randomness can be shared entirely between gadgets (or blocks of gadgets), however, requiring independent inputs and only achieving NI. Hence, additional refreshing is required. To fulfill SCR, they construct a new multiplication gadget based on non-completeness. The resulting t-order gadget requires $\lfloor \frac{t}{2} \rfloor^2 + \lfloor \frac{t}{2} \rfloor + 1$ shares for $t > 3$ and $t+1$ shares otherwise. In addition, $\frac{t \cdot n}{2} + n$ random elements are required. Again, there exists no glitch-robust version yet, and, hence, the cost in terms of area and latency is unknown. In contrast, we rely on the widely used DOM and HPC_2 gadgets without additional refreshing, however, also introduce more complex rules.

In addition, and in contrast to our work, they also analyzed the special case of $t = 1$ and provided a scheme for 1^{st}-order security of arbitrary circuits with only two random elements. For this, all gadgets need to be adapted, including linear ones. Please note, that their AES implementation is based on $\mathbb{F}_{2^8}$, resulting in 16 bits of randomness. In theory, an implementation based on $\mathbb{F}_2$ could achieve 1^{st}-order security with only 2 bits of randomness.

Randomness Expansion. All the above methods (including ours) focus on reducing the number of random elements used to refresh the intermediate values of the computation. Ishai et al. [IKL+13] added an entirely different layer to the analysis by including the quality of randomness, stating that not all required randomness needs to be *true randomness*. By introducing a locality parameter ℓ, which indicates the maximum number of random elements any wire depends on, they observe that security is given if all subsets of $\ell \cdot t$ random elements are uniformly and independently distributed. Hence, the quality of the randomness can be reduced to $(\ell \cdot t)$-wise independent pseudo randomness, as long as the Pseudo-Random Number Generator (PRNG) is secure against a t-probing attack, i.e., is *robust*. To reduce the locality of a design, they introduce a locality refreshing placed after each multiplication gadget, achieving a locality of $\ell = \mathcal{O}(t^2)$ and a total true-randomness requirement of $\tilde{\mathcal{O}}(t^{3+\epsilon})$. We omit this work in Table 5 as they do not provide concrete numbers for an AES-128 implementation. This approach was further improved by Coron et al. [CGZ20] in two steps: (i) They reduce the locality of any given private circuit to $\ell = \mathcal{O}(t)$ by doing gadget-internal locality refreshing. (ii) They use multiple PRNGs instead of one, effectively removing the requirement for the robustness of the PRNG and reducing the locality with respect to a single PRNG to $\ell = \mathcal{O}(1)$. In combination, this reduces the number of required true randomness to $\tilde{\mathcal{O}}(t^2)$. Only recently, Goyal et al. [GIS22] further improved the asymptotic randomness requirement of private circuits to $\tilde{\mathcal{O}}(t)$ by shifting the view from secret sharing to masking of values carried by wires. However, this approach increases the circuit size from $\mathcal{O}(t^2 s)$ to $\mathcal{O}(t^3 s)$, where s is the size of the original (unprotected) circuit. Again, we omit this work in Table 5 as no concrete numbers for an AES-128 implementation are provided.

Arguably, this line of research establishes the state-of-the-art when it comes to the reduction of required true randomness of a complete design. More specific, our work requires roughly 6× more randomness than Coron et al. [CGZ20] for an AES-128 implementation. However, we see this line of research as orthogonal to our work in that it reduces the number of random elements that can influence any given set of probes. In contrast, our work shows that not all randomness influencing a set of probes need to be independent. Therefore, we expect additional gains when combining inter-gadget randomness sharing with locality reduction and randomness expansion but leave this for future work.

Future Work. In this work, we analyzed optimization opportunities in a spatial dimension only, i.e., optimizing the randomness required to evaluate a single input once by each

hardware component. When using a round-based implementation or processing multiple inputs, a fresh set of random values is required. Hence, a natural extension of our work is an optimization across rounds or inputs, essentially adding the dimension of time to the optimization. With regards to gadgets, the focus of this work lies on the widely used DOM and HPC_2 gadgets, which both require the same amount of randomness. We believe our rules to be applicable also to other gadgets, however, currently security proofs need to be made one-by-one. Therefore, an interesting direction for future research is to first apply (and potentially extend) the presented rules to gadgets using more or less randomness, and then to prove the security for arbirary gadgets, i.e., make a general statement for SNI and PINI. We also focused on the reuse of randomness within clusters and – as described in Section 5.2.1 – restricted our considerations to a simple clustering algorithm. However, as already mentioned, building optimal clusters is an optimization problem in its own right, and utilizing better clustering algorithms is expected to improve the overall result. Going one step further, the design representation itself can be optimized to enable more optimal clusters, as the possible clusters are highly restricted by the specific design representation. A different representation may lead to larger clusters and, hence, reduced randomness requirements. Ideally, the entire process described in this paper works in an integrated manner, i.e., a clustering algorithm should already consider the resulting randomness reduction and construct clusters that lead to a global optimum. Whether this can be reached by always searching for the largest clusters is an open question. Of course, randomness optimization is not the only constraint when designing side-channel secure hardware, and integration in the larger context of EDA requires the combination with other constraints like area, latency, or speed. For this, it is essential to provide a cost function of each possible optimization step and to consider the necessary notion of composition. Currently, our optimization strategies have no additional costs, with the only exception being the relaxed clustering in combination with HPC_2 gadgets, where additional registers are required. Further optimization may change this, e.g., when a design is dynamically changed to yield an optimal clustering. An important but still unknown piece is the cost function of fresh randomness with regard to other design constraints. A dense integration of side-channel-related security awareness into the EDA design flow ultimately requires that function.

In general, our techniques and proofs also hold in the standard model, which is usually associated with software implementations, since it is strictly weaker than the glitch-extended model. Nevertheless, the specific leakage behavior of the microarchitecture under consideration needs to be taken into account to guarantee the security of software running on a Central Processing Unit (CPU). Hence, an interesting line of research could evaluate and potentially adopt the presented optimization techniques for software targets.

8 Conclusions

In this work, we introduced different methodologies for reducing the randomness consumption in composed, masked circuits. By identifying well-defined clusters of gadgets with opportunities of cross-gadget randomness re-use, we mitigate the disadvantage of higher-order masking compositions, rooted in the atomic nature of composable gadgets and their (often unnecessary) requirement of individual fresh randomness.

We gave a thorough theoretic proof of security in the glitch-robust t-probing model for all outlined optimization techniques. Furthermore, we presented an extensive comparison of all our optimization methods with respect to their randomness reduction when applied to a variety of real-world cryptographic designs. Our results indicate a significant optimization of randomness consumption for compositions of SNI gadgets. PINI compositions seem to have no benefit for optimizing randomness other than minimizing the number of gadgets through trivial composition. In general, optimization opportunities are highly dependent

on the design structure and the required level of security.

Finally, we developed MLIR-based EDA passes, which are easily extendable and well equipped for integration into a security-aware EDA framework.

Acknowledgments

The work described was funded 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 Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, 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 Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

[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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129, 2016.

[BBP+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness Complexity of Private Circuits for Multiplication. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 616–648, 2016.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 208–225, 2012.

[BDPvA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche. The Keccak Reference. https://keccak.team/files/Keccak-reference-3.0.pdf, 2011.

[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 Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, pages 343–372, 2018.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, pages 450–466, 2007.

[BP12] Joan Boyar and René Peralta. A Small Depth-16 Circuit for the AES S-Box. In *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*, pages 287–298, 2012.

[Can05] David Canright. A Very Compact S-Box for AES. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, pages 441–455, 2005.

[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 Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2017.

[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 Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-Channel Masking with Pseudo-Random Generator. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, pages 342–375, 2020.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CS19] Gaëtan Cassiers and François-Xavier Standaert. Towards Globally Optimized Masking: From Low Randomness to Low Noise Rate: or Probe Isolating Multiplications with Reduced Randomness and Security against Horizontal Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):162–198, Feb. 2019.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[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.

[FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing Randomness Complexity in Private Circuits. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 781–810, 2017.

[GIS22] Vipul Goyal, Yuval Ishai, and Yifan Song. Private Circuits with Quasilinear Randomness. *IACR Cryptol. ePrint Arch.*, 2022, 2022.

[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 Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, 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 Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[HKSS12] Y. Hori, T. Katashita, A. Sasaki, and A. Satoh. SASEBO-GIII: A hardware security evaluation board equipped with a 28-nm FPGA. In *IEEE Global Conference on Consumer Electronics*, pages 657–660, 2012.

[HS13] Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2013.

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust Pseudorandom Generators. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 576–588, 2013.

[IKL+17] Adam M. Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, and Jonathan Bachrach. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*, pages 209–216, 2017.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, 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 Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KMMS22] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1), 2022.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[LAB+21] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques A. Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2021, Seoul, South Korea, February 27 - March 3, 2021*, pages 2–14, 2021.

[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 Alfred Menezes, editor, *Topics in*

Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings, 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 Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2006.

[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.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 413–427, 2010.

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

[WGS+20] Weijia Wang, Chun Guo, François-Xavier Standaert, Yu Yu, and Gaëtan Cassiers. Packed Multiplication: How to Amortize the Cost of Side-Channel Masking? In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, pages 851–880, 2020.

[yos] Yosys Open Synthesis Suite. http://www.clifford.at/yosys/. Accessed: 13.10.2021.