Low-Latency Hardware Private Circuits

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

Amir Moradi
firstname.lastname@uni-koeln.de
University of Cologne
Institute for Computer Science
Cologne, Germany

ABSTRACT

Over the last years, the rise of the Internet of Things (IoT), and the connection of mobile – and hence physically accessible – devices, immensely enhanced the demand for fast and secure hardware implementations of cryptographic algorithms which offer thorough protection against Side-Channel Analysis (SCA) attacks. Among a variety of proposed countermeasures against SCA, *masking* has transpired to be a promising candidate, attracting significant attention in both, academia and industry. Here, abstract adversary models have been derived, aiming to accurately model real-world attack scenarios, while being sufficiently simple to enable formally proving the SCA resilience of masked implementations on an algorithmic level. In the context of hardware implementations, the *robust probing model* has become highly relevant for proving SCA resilience due to its capability to model physical defaults like glitches and data transitions. As constructing a correct and secure masked variant of large and complex circuits is a challenging task, a new line of research has recently emerged, aiming to design small, masked subcircuits – realizing for instance a simple AND gate – which still guarantee security when composed to a larger circuit. Although several designs realizing such composable subcircuits – commonly referred to as *gadgets* – have been proposed, negligible research was conducted in order to find trade-offs between different overhead metrics, like randomness requirement, latency, and area consumption.

In this work, we present HPC3, a hardware gadget which is trivially composable under the notion of PINI in the glitch-extended robust probing model. HPC3 realizes a two-input AND gate in one clock cycle which is generalized for any arbitrary security order. Existing state-of-the-art PINI-gadgets either require a latency of two clock cycles or are limited to first-order security. In short, HPC3 enables the designer to trade double the randomness for half the latency compared to existing gadgets, providing high flexibility and enabling the designer to gain significantly more speed in real-time applications.

1 INTRODUCTION

Since the first seminal description of Side-Channel Analysis (SCA) in [35, 36], a significant increase in physically accessible devices has further driven demand for effective protection against physical attacks. At the same time, a wide variety of different exploitable side channels have been revealed, including timing [35], power consumption [36], electromagnetic (EM) emanations [24], or temperature and heat dissipation [30], posing a wide-ranging and divers threat to cryptographic implementations. Despite the increasing effort which has been committed to developing countermeasures against SCA attacks during the last decades, it remains a challenging and complex task to design secure hardware implementations offering a sufficient security level on the one hand, and high efficiency on the other.

Among a wide range of emerged SCA countermeasures, *masking* has proven to be a highly promising candidate due to its well-founded theoretical background based on secret sharing and its well-understood security requirements [20]. Although extensive research in this area has been conducted and several masking schemes were proposed over the last years [26, 28, 31, 41, 43, 51], many of these schemes have been proven to suffer from either design flaws or invalid assumptions [39].

As a consequence, researchers started focusing on the establishment of formal models in order to abstractly define an SCA attacker's capabilities and to realistically model circuit behavior [6, 23, 31, 42]. The advantage of these models is twofold. On the one hand, they enable security verification of masked implementations on an abstract level, aiming to detect security flaws in an early stage of the design process. Here, a variety of tools, working at different abstraction levels and offering various accuracy, have been proposed [3–5, 11, 12, 14, 17, 33]. On the other hand, this formalization enables the design of masked constructions that are provably secure with respect to the defined adversary model. The ISW d-probing model [31] – and its extension to modeling physical defaults in hardware implementations [23] – has proven to be well suited to find such secure constructions, due to its convenient level of abstraction and its existing reduction to the *Noisy Leakage Model* [22], which is considered to be the closest to reality with respect to accurately modeling leakage behavior.

Although different models for formal security verification have been established in the context of SCA, finding provably secure masking schemes remains a hard task for complex circuits and high security orders. As a remedy, different composability notions have been proposed which define properties that aim to guarantee security even when circuits are composed. Following a divide-and-conquer approach, this reduces the task of finding large secure circuits to the task of finding small subcircuits which are in conformity with the composability notions. In the context of the d-probing model, Non-Interference (NI)/Strong Non-Interference (SNI) [4, 5] and Probe-Isolating Non-Interference (PINI) [18] have been proposed, where SNI was introduced since the restrictions defined by NI were not sufficient to guarantee composability. As the scope of SNI was originally limited to single-output gadgets, Cassiers et al. provided an extension to cover multiple-output gadgets, but at the same time proposed PINI which further reduces the overhead necessary to construct composable gadgets [18]. In their recent work [19], Cassiers and Standaert further extended the adversary model to

accurately cope with data transitions between different clock cycles and formally defined the notion of Output Probe-Isolating Non-Interference (O-PINI), which aims to achieve trivial composability under (i) transitions and (ii) transitions + glitches. In the course of the introduction of these wide variety of notions, several concrete gadgets have been presented, either realizing atomic logic functions, like AND or refresh gates [5, 11, 17–19, 27, 31, 38], or even arbitrary logic functions [34].

As embedded real-time applications become increasingly relevant – especially in the context of the rapidly growing IoT – and due to the high accessibility of the involved devices, designing fast and protected cryptographic hardware implementations is the key to guarantee a high level of security and sufficiently fast data processing. As a consequence, reducing the latency, introduced by masking a cryptographic hardware implementation, is an important task whose difficulty is mainly rooted in omitting leakage through physical defaults like *glitches* and *transitions* [19, 23]. As composable hardware gadgets for arbitrary security orders are restricted to atomic logic functions, reducing the latency of a gadget would have a significant effect on the latency of the over-all composed circuit. It is hence highly beneficial if we can further reduce the latency of existing schemes.

Contributions. In this work, we present HPC3, a low-latency hardware gadget, that offers d-th order security and is trivially composable under the notion of PINI in the glitch-extended robust probing model. Similar to HPC1 and HPC2 [17], it realizes a masked AND gate that can be instantiated for arbitrary security orders d. In contrast to HPC2, our newly introduced HPC3 only needs a single register stage instead of two, regardless of the security order, while doubling the randomness requirements. It hence enables the designer to trade double the randomness for half the latency, while sustaining the same security level and providing significantly more flexibility with respect to different use cases – for example in the context of fast memory encryption. To the best of our knowledge, HPC3 is the first glitch-robust composable gadget which is settled in the PINI framework and can be instantiated utilizing only one register stage for any security order. We further show that, compared to HPC2, our construction itself leads to less chip area overhead – at the cost of a higher demand for fresh randomness.

Furthermore, we present HPC3$^+$, a gadget that is directly based on HPC3 and offers trivial composability in the simultaneous presence of both transition and glitches. Eventually, we prove and formally verify our constructions, compare our work to state-of-the-art hardware gadgets, and explore multiple case studies and conduct experimental leakage assessments.

Outline. We start by elaborating all necessary theoretical concepts in Section 2. This includes a summary of all notations used throughout this work, a definition of the circuit and adversary model, and recapitulating different security and composability notions. In Section 3, we present our new gadget HPC3 and prove its conformity to the PINI notion in the glitch-extended robust probing model. Here, we further present HPC3$^+$ and prove its trivial composability under transitions paired with glitches, before we compare our designs to state-of-the-art trivially-composable hardware gadgets in Section 4. After we conduct analyses on an extensive list of case studies, i.e., different cryptographic implementations, and perform experimental leakage assessments in Section 5, we conclude our work in Section 6.

2 BACKGROUND

2.1 Notations

We denote random variables by capital letters, e.g., $X \in \mathbb{F}_2$ denotes a binary random variable. Further, we use bold letters like $\mathbf{X}$ to denote sets. Initializations of random variables are denoted by small letters, while the probability that X takes x is written as $Pr[X = x]$. Moreover, we indicate the i-th input to a function with subscript i while superscripts identify shares of random variables. Hence, the s-th share of the i input to a function is denoted as X_i^s. Let further $|\mathbf{X}|_i$ denote the number of shares in a set $\mathbf{X}$ corresponding to X_i. When masking a function $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ with t shares per input, the set containing all input shares is given as $Sh(\mathbf{X}) = [X_0^0, X_0^1, \ldots, X_0^{t-1}, X_1^0, \ldots, X_{n-1}^{t-1}]$. In the same manner, for a set of share indices $\mathbf{I} \subseteq [0, \ldots, t-1]$, $Sh(\mathbf{X})^{\mathbf{I}}$ denotes the set of all input shares X_i^s, with $0 \leq i < n$ and $s \in \mathbf{I}$. Eventually, P_W denotes the (extended) probe on a wire W, while drawing a value X uniformly and at random from a set S is denoted as $X \xleftarrow{\$} S$.

2.2 Circuit Model

As originally considered in [31] and later extended in [23], any stateful and deterministic circuit C is modeled as a Directed Acyclic Graph (DAG) $\mathsf{G_C} = \{\mathcal{V}, \mathcal{E}\}$ with $\mathcal{V}$ being the set of vertices and $\mathcal{E}$ the set of edges in $\mathsf{G_C}$. The edges represent wires carrying elements of $\mathbb{F}_2$ while the vertices are combinational gates such as AND and XOR, or memory gates, i.e., registers. Memory gates will output the previous input to the gate on any circuit invocation while storing the input for the next invocation. Eventually, $\mathsf{G_C}$ realizes a Boolean function $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$.

As this model lacks the notion of cycling connections in a circuit, and the ability to model different executions of the same physical gate, it has been extended towards a more specific circuit model in [19]. This model introduces so-called *structural gates* which among its functionality, public and secret parameters, also captures the latency of a gate. *Structural wires* are then defined as wires connecting structural gates, and finally a *structural circuit* is defined as a directed graph whose nodes are structural gates and whose edges are structural wires.

The definition of a structural circuit can then be extended to cover states in different clock cycles by defining a state of each wire and gate for every clock cycle:

Definition 2.1 (Circuit Execution). *A Circuit Execution of a structural circuit* $\mathsf{C} = (\mathbf{G}, \mathbf{W})$ *for the set of cycles* $\mathbf{T}$ *is a directed graph* $\mathsf{G_C} = (\mathcal{V}, \mathcal{E})$ *with* $\mathcal{V} \in \mathbf{G} \times \mathbf{T}$ *and* $\mathcal{E} \in \mathbf{W} \times \mathbf{T}$ *where wires connect the gates according to their latency. Here,* $\mathbf{G}$ *and* $\mathbf{W}$ *denote the structural gates and wires of* C.

Note that in this model, *combinational gates* – like AND and XOR – are structural gates with latency $l = 0$, realizing the corresponding Boolean function, while registers are structural gates realizing the identity function and having latency $l = 1$. Nevertheless, structural gates are defined in a more general way, possibly containing several register stages.

Thanks to the previous research conducted in [19] and due to the share isolating property of PINI, we can conveniently limit all our considerations to the simpler original circuit model while achieving secure composition under transitions in the more complex model of circuit executions.

Encoded Circuit Model. In order to restrict the adversarial probing solely to the masked circuit, and to exclude the masking and unmasking of the input data, the process of secure computation is divided into three steps. As defined in [1], a *circuit compiler* is defined by a tuple of three algorithms (COMPILE, ENCODE, DECODE), which are defined as follows.

- The COMPILE algorithm is deterministic and takes as input a structural circuit C and outputs a randomized (masked) circuit $\tilde{\mathsf{C}}$.
- ENCODE is a probabilistic algorithm that takes as input $\mathbf{X}$ and outputs the encoded input $\tilde{\mathbf{X}}$. This encoded input corresponds to the shared representation of the data, i.e., $\tilde{\mathbf{X}} = Sh(\mathbf{X})$, which – in our case – is derived by means of Boolean masking.
- Eventually, DECODE is a deterministic algorithm that takes encoded data $\tilde{\mathbf{Y}}$ and decodes/unshares it to achieve $\mathbf{Y}$.

As a result, these algorithms enable sharing of the initial input data (through ENCODE), a computation on the shared representation of the input (through $\tilde{\mathsf{C}}$) and an unsharing of the result (through DECODE), such that $\mathbf{Y} \leftarrow \mathsf{DECODE} \circ \tilde{\mathsf{C}} \circ \mathsf{ENCODE}(\mathbf{X})$ for $\mathbf{Y} \leftarrow \mathsf{C}(\mathbf{X})$, while the adversary is restricted to only make observations within $\tilde{\mathsf{C}}$. It means that the algorithms ENCODE and DECODE are known to the adversary, but neither the output of ENCODE nor the input of DECODE.

2.3 Boolean Masking

Following the approach of secret sharing, a *Boolean masking* of a secret $X \in \mathbb{F}_2^n$ is a set $\mathbf{X} \in \mathbb{F}_2^{n \times s}$ of s independent secret shares $X^i \in \mathbb{F}_2^n$, $0 \leq i < s$, such that $X = \bigoplus_{i=0}^{s-1} X^i$. This is commonly derived by independently drawing $X^i \xleftarrow{\$} \mathbb{F}_2^n$, for $0 \leq i < s-1$ and calculating the remaining share as the XOR sum of all other shares: $X^{s-1} = \bigoplus_{i=0}^{s-2} X^i$. Following the definition described in Section 2.2, this step describes the ENCODE algorithm in the case of Boolean masking. A direct implication of Boolean masking necessitates splitting sensitive data in at least $d+1$ shares in order to achieve d-th order security.

2.4 Adversary Model

Following the definition in Section 2.2, for the remainder of this work, we assume that an adversary's access to computations is limited to $\tilde{\mathsf{C}}$ and that the execution of ENCODE and DECODE remain unavailable.

d-Probing Model. The standard d-probing model, introduced by Ishai et al. in [31], grants an adversary the ability to probe up to d wires of a circuit. Here, every logic gate acts as a synchronization element, so every wire is considered stable, carrying only the result of the driving gate under the current assignment of the primary inputs. Intuitively, security in this model is given, if an adversary is not able to receive any information about sensitive (unshared) data by observing the joint distribution over maximal d probed wires.

Glitch-Extended Probing. Since the standard probing model does not cover the modeling of physical defaults occurring in hardware implementations, it was extended by Faust et al. in [23]. In their work, the authors introduced the *robust probing model* which aims to extend the probes enabling them to capture leakage originating from (i) data transitions at registers, (ii) coupling effects, i.e. dependencies between adjacent wires, and (iii) glitches.

The concept of *glitches* describes signal recombination caused by different delay paths in a digital circuit. Hence, in practice, a single probe on a wire is not only able to observe the intended and stable output signal of its driving gate, but possibly a recombination of several upstream signals, up to the last synchronization point, i.e., registers output or primary inputs. To capture the worst case in this context, the glitch extension of a probe is hence the set of probes placed on all these synchronization points.

This is formally defined through Algorithm 1. The glitch extension of a probe is defined recursively and will either return the union over all glitch-extended probes placed on the input of the driving combinational gate, or return the probe itself in case the probe is placed on an output of a register or a primary input. Eventually, the extension of a set of standard probes is simply derived by uniting all corresponding glitch-extended probes.

Transition-Extended Probing. Considering the definition of a circuit execution, given in Definition 2.1, the transition-extended probes on a *structural gate* – which is formally defined in [19] by the fact that all its executions are identical except for a shift in time, i.e., in clock cycles – are all non-extended probes P_W and additionally the sets $\{(W, t-1), (W, t)\}$. This means the joint distribution of probes in two consecutive clock cycles while $\{(W, t-1), (W, t)\}$ belongs to the same execution of the probed structural gate. Here, t refers to a clock cycle and (W, t) refers to the state of W during this specific clock cycle, i.e., a node in the circuit execution given in Definition 2.1. For example, a non-extended probe on a register output would extend to two probes: on its input and output wires.

Transition+Glitch-Extended Probing. Combining transition- and glitch-based probe extensions is straightforward. First, all non-extended probes are glitch-extended, before every resulting probe becomes transition-extended. This corresponds to the physical property that a propagation of a glitch may depend on both the new and the previous value of the wire.

d-Probing Security. In the (robust) probing model, an adversary is granted the ability to probe up to d wires of a masked circuit $\tilde{\mathsf{C}}$. Naturally, d-probing security is achieved iff an adversary with these capabilities, i.e., an adversary in the d-probing model, does not learn anything about the processed secrets [31]:

Definition 2.2 (d-Probing Security). *A masked circuit $\tilde{\mathsf{C}}$ – derived by compiling C with secret input $\mathbf{X}$ – achieves d-probing security iff for any set of probes $\mathbf{P}, |\mathbf{P}| \leq d$, the joint distribution over all observations $\mathbf{Q}$ made by (extended) probes $\mathbf{P}$ is statistically independent of any secret $\mathbf{X}$, i.e., the following holds for all possible assignments to variables in $\mathbf{Q}$ and $\mathbf{X}$:*

$$Pr[\mathbf{Q}|\mathbf{X}] = Pr[\mathbf{Q}]$$

Algorithm 1: Glitch Extension

Input : Non-extended probe $P \in \mathbb{F}_2$

Output : Glitch-extended probes $\mathbf{P}^E \in \mathbb{F}_2^p, p > 0$

```
if P is placed on an output of a combinational gate then
    P^E ← ⋃_{0≤i<n} glitch-extend(P_i)      // where P_i, 0 ≤ i < n
                                               are all inputs to
                                               the driving gate
else
    if P is placed on an output of a register or on a primary input
     then
        P^E ← {P}
    end
end
```

2.5 Circuit Composition

Several security and composability notions have been proposed in recent years, aiming to enable efficient masking of large and complex circuits. Before we give an overview of these notions, we start by introducing the concepts of *probe simulation* and *probe propagation*, paving the ground for the subsequent definitions.

Probe Simulatability. The concept of *probe simulatability* [18, 31] helps to argue about dependencies between probes and input shares to a masked/encoded circuit. Its definition is given in the following.

Definition 2.3 (Perfect Probe Simulation). *Given a set of (extended) probes* $\mathbf{P} \in \mathbb{F}_2^t$ *on an encoded circuit* $\tilde{\mathrm{C}}$, $\mathbf{P}$ *is said to be* perfectly simulatable *by a set* $\mathbf{S}$ *of input shares iff there exist a simulator* SIM, *such that for any values of the inputs to* $\tilde{\mathrm{C}}$, *the joint probability distribution over* $\mathbf{P}$ *and* $\mathsf{SIM}(\mathbf{S})$ *are equal, where* $\mathsf{SIM}(\mathbf{S}) : \mathbb{F}_2^{|\mathbf{S}|} \mapsto \mathbb{F}_2^t$ *with input* $\mathbf{S} \subseteq Sh(\mathbf{X})$ *is a probabilistic polynomial time (p.p.t.) simulator.*

Probe Propagation. The concept of *Probe Propagation* – initially introduced by Cassiers et al. in [18] – is closely related to simulatability, and defines which wires are necessary to perfectly simulate a probe placed on $\tilde{\mathrm{C}}$. Intuitively, it describes, how leakage is traversed backwards throughout a circuit, beginning from the point where the probe is placed. Restricting probe propagation has proven to be a key factor to guarantee composability.

Considering a probe $P \in \mathbb{F}_2$ on a subcircuit , P is said to propagate into an input of the subcircuit, if the input is required to perfectly simulate P. When considering composability notions, propagation of internal and output probes of a gadget are restricted following a well defined set of rules and allowing to make statements of the overall circuit. For this, the propagated probes can be derived by iteratively substituting a probe by its propagated variant on the gadgets' inputs until the overall circuit's input is reached.

Non-Interference (NI). As directly designing masked circuits which achieve d-probing security has proven to be hard for large and complex functions, different composability notions have been introduced as a remedy. These notions aim to define sufficient properties which a masked circuit must fulfill in order to provide d-th order security in the probing model when composed to a larger circuit. Following a divide-and-conquer approach and utilizing atomic logic functions – typically AND and XOR – large circuits can be derived by composing masked versions of these atomic gates – commonly referred to as *gadgets*.

Utilizing the concept introduced above, and as initially elaborated in [4], NI aims to restrict probe propagation to a certain set of input shares:

Definition 2.4 (d-Non-Interference (NI)). *A masked circuit* $\tilde{\mathrm{C}}$ *provides d-Non-Interference iff for any probe set* $|\mathbf{P}|$ *of* $t \leq d$ *probes, there exists a set* $\mathbf{S}$ *of input shares with* $|\mathbf{S}|_{\forall i} \leq t$ *such that* $\mathbf{P}$ *can be perfectly simulated by* $\mathbf{S}$.

Strong Non-Interference (SNI). The notion of SNI [5] has been introduced in order to correct composability flaws regarding NI and restricts probe propagation even further:

Definition 2.5 (d-Strong Non-Interference (SNI)). *A masked circuit* $\tilde{\mathrm{C}}$ *provides d-Strong Non-Interference iff for any probe set* $\mathbf{P}$, *containing* $t = t_1 + t_2 \leq d$ *probes, where* t_1 *probes are placed on internal wires and* t_2 *on output wires, there exists a simulation set* $\mathbf{S}$ *of input shares with* $|\mathbf{S}|_{\forall i} \leq t_1$, *such that* $\mathbf{P}$ *can be perfectly simulated by* $\mathbf{S}$.

Here, output probes are not allowed to propagate into any input shares, hence resolving the flaws with respect to composability discovered for NI [5], but implication of secure composition was originally still restricted to single-output gadgets.

Probe-Isolating Non-Interference (PINI). In [18], it has been shown that the original definition of SNI can be extended to cover multiple-output gadgets as well. Nonetheless, in the same work, the authors introduced the notion of PINI, which elegantly isolates probe propagation within single share domains, enabling trivial implementations of linear functions and reducing entropy and area requirements compared to SNI, while guaranteeing straightforward composability.

Similar to Domain-Oriented Masking (DOM), in the context of PINI, a specific share domain is assigned to every input and output share. Now, in order to be PINI, every output probe is only allowed to propagate within its own share domain (share index) while propagation of every internal probe is limited to a single (but arbitrary) share domain:

Definition 2.6 (d-Probe-Isolating Non-Interference (PINI)). *Let* $\mathbf{P_I}$ *be the set of internal probes with* $|\mathbf{P_I}| = t_1$. *Let further* $\mathbf{I_O}$ *be the index set assigned to the output wires probed by* $\mathbf{P_O}$ *with* $|\mathbf{I_O}| = t_2$.

A masked circuit $\tilde{\mathrm{C}}$ *provides d-Probe-Isolating Non-Interference iff for every* $\mathbf{P} = \mathbf{P_I} \cup \mathbf{P_O}$ *with* $t_1 + t_2 \leq d$, *there exists a set* $\mathbf{I_I}$ *of circuit indices with* $|\mathbf{I_I}| \leq t_1$ *such that* $\mathbf{P}$ *can be perfectly simulated by* $\mathbf{S} = Sh(\mathbf{X})^{\mathbf{I_I} \cup \mathbf{I_O}}$.

Conveniently, this definition directly implies that the trivial implementation of linear functions with d+1 shares, i.e., the share-wise application of the unshared function, is d-PINI even in the glitch-extended robust probing model. Consequently, linear functions do not introduce any additional latency or entropy overhead into the design.

PINI is invariant under composition, i.e., a circuit composed of gadgets fulfilling d-PINI is itself d-PINI. This is true, if gadgets are carefully connected, i.e., if an output with a certain share index is only connected to an input with the same share index. Therefore, since d-PINI implies d-probing security, the resulting circuit

will be secure in the d-probing security model and will reveal no information about any secret value under d (extended) probes.

In order to achieve trivial composition under transitions, an extension to the original definition of PINI, i.e., O-PINI, was recently introduced in [19]. It deals with adjacent executions of the same gadget, i.e., if the input of a gadget depends on the output of itself. As for simulating the second execution of the gadget, the simulator may need to also simulate outputs of the first gadget execution, this leads to an effective probe extension of internal probes to additional probes on the output. As a consequence, the probed wires may propagate into more circuit shares than allowed, causing a reduction in the security level of the design. Naturally, a possible solution would be to avoid adjacent execution of gadgets, for example by means of an 'empty/dummy' clock cycle between executions. As this would introduce composability restrictions, which make additional design verifications necessary, O-PINI was introduced in order to guarantee trivial composability. O-PINI copes with the issue of adjacent executions by additionally enforcing simulatability of every output wire whose share index is also a share index in the simulation set:

Definition 2.7 (d-Output Probe-Isolating Non-Interference (O-PINI)). *Let $\mathbf{P_I}$ be the set of internal probes with $|\mathbf{P_I}| = t_1$. Let further $\mathbf{I_O}$ be the index set assigned to the output wires probed by $\mathbf{P}_O$ with $|\mathbf{I_O}| = t_2$.*

A (unrolled) masked circuit $\tilde{\mathrm{C}}$ provides d-Probe-Isolating Non-Interference iff for every $\mathbf{P}$ with $t_1 + t_2 \leq d$, there exists a set $\mathbf{I_I}$ of circuit indices with $|\mathbf{I_I}| \leq t_1$ such that $\mathbf{P} = \mathbf{P_I} \cup \mathbf{P}_O \cup \mathbf{O}^{\mathbf{I_I}}$ can be perfectly simulated by $\mathbf{S} = Sh(\mathbf{X})^{\mathbf{I_I} \cup \mathbf{I_O}}$.

Note that, for the sake of simplicity and in contrast to [19], we restrict the definition of d-O-PINI to unrolled designs (which in [19] is called *pipelined*), i.e., designs where no feedback loops exist. Due to their implication of trivial composability, all our constructions in this work are based on the PINI and O-PINI notions.

2.6 Automated Masking

Gadgets fulfilling composability notions like PINI are well suited for automated generation of masked hardware. By simply synthesizing the implementation based on a library that only includes logic gates for which a corresponding gadgets exist, the implementation can be masked by substituting every gate in the resulting netlist with those gadgets. Since composability notions elegantly guarantee (robust) probing security when gadgets are composed, the overall design will be provable secure.

The recently introduced software tool AGEMA [32] realizes such a transformation by first representing the netlist as a graph, before translating it into a Mealy machine consisting of a combinational circuit and a single main register stage. The designer can then select which parts of the netlist should be masked and specify the gadget family to apply. In the *naive* approach, every gate in the netlist is then simply substituted by the corresponding gadget as explained. As these gadgets introduce additional latency into the design, the correct functionality of the circuit is ensured by automatically applying pipelining or clock gating. We utilized AGEMA in this work in order to construct our case studies and to offer a fair comparison to state-of-the-art composable gadgets.

Algorithm 2: HPC3 Multiplication

Input : $Sh(X) = [X^0, \ldots, X^d]$, $Sh(Y) = [Y^0, \ldots, Y^d] \in \mathbb{F}_2^{d+1}$
Output: $Sh(Z) = [Z^0, \ldots, Z^d] \in \mathbb{F}_2^{d+1}$
/* valid sharings of $X, Y, Z = X \cdot Y \in \mathbb{F}_2$ */

for $i = 0$ *to* $d-1$ **do**
  for $j = i+1$ *to* d **do**
    $R'_{ij} \xleftarrow{\$} \mathbb{F}_2$, $R''_{ij} \xleftarrow{\$} \mathbb{F}_2$
    $R'_{ji} \leftarrow R'_{ij}$
    $R''_{ji} \leftarrow R''_{ij}$
  end
end
for $i = 0$ *to* d **do**
  for $j = 0$ *to* d **do**
    if $i \neq j$ **then**
      $U'_{ij} \leftarrow Reg[Y^j \oplus R'_{ij}]$
      $U''_{ij} \leftarrow Reg[\overline{X^i} \cdot R'_{ij} \oplus R''_{ij}]$
      $C_{ij} \leftarrow Reg[X^i] \cdot U'_{ij} \oplus U''_{ij}$
    end
  end
end
for $i = 0$ *to* d **do**
  $Z^i \leftarrow Reg[X^i \cdot Y^i] \oplus \bigoplus_{0 \leq j \leq d} \left(C_{ij}\right)$
end

3 LOW-LATENCY HARDWARE PRIVATE CIRCUITS

In this section, we present the algorithm describing our low-latency Hardware Private Circuits (HPC3), formally prove its security and composability and give exemplary schematics for the first and second security orders, before we extend it to be composable under transitions and glitches.

3.1 Glitch-Robust Variant

HPC3 is described in Algorithm 2. Here, every cross-domain term $X^i \cdot Y^j$ is blinded by the sum of two freshly drawn random masks $R'_{ij} \oplus R''_{ij}$, where $R'_{ij} = R'_{ji}$ and $R''_{ij} = R''_{ji}$, resulting in a randomness requirement of $2 \cdot \big(d \cdot (d+1)\big)/2 = d \cdot (d+1)$ and a latency of a single clock cycle to achieve d-th order security in the glitch-extended probing model. The composability under the notion of PINI – utilizing only a single register stage – is achieved through blinding Y^j by R'_{ij} before multiplying X^i, and at the same time blinding the corresponding correction term $\overline{X^i} \cdot R'_{ij}$ by another fresh randomness R''_{ij}. This way, a single extended output probe is never able to reveal X^i and Y^j.

Intuitively, this means that probes on different cross-domain terms can always be simulated as they are independently blinded and an adversary would always need two probes on $X^i \cdot Y^j$ and $X^j \cdot Y^i$ to reveal information about domain i and j. Yet, these two probes would always be placed on share domain i and share domain j, which is in conformity with the PINI security notion. We formalize this argument by proving Theorem 3.1 as follows.

Theorem 3.1. *HPC3– with security parameter d – provides a correct and PINI-secure circuit in the glitch-extended d-probing model.*

Proof.

Correctness. First, we prove the correctness of the construction. For this, we discard all registers in Algorithm 2. Since

$$\begin{aligned} C_{ij} &= X^i \cdot (Y^j \oplus R'_{ij}) \oplus \overline{X^i} \cdot R'_{ij} \oplus R''_{ij} \\ &= X^i \cdot Y^j \oplus X^i \cdot R'_{ij} \oplus \overline{X^i} \cdot R'_{ij} \oplus R''_{ij} \\ &= X^i \cdot Y^j \oplus R'_{ij} \oplus R''_{ij} \end{aligned}$$

and $R'_{ij} = R'_{ji}$, $R''_{ij} = R''_{ji}$, it holds that

$$\begin{aligned} Z &= \bigoplus_{0 \le i \le d} Z^i \\ &= \bigoplus_{0 \le i \le d} \left(X^i \cdot Y^i \bigoplus_{0 \le j \le d, j \ne i} \left(X^i \cdot Y^j \oplus R'_{ij} \oplus R''_{ij} \right) \right) \\ &= \bigoplus_{0 \le i \le d} \left(\bigoplus_{0 \le j \le d} X^i \cdot Y^j \right) \oplus \bigoplus_{0 \le i \le d} \left(\bigoplus_{0 \le j \le d, i \ne j} R'_{ij} \oplus R''_{ij} \right) \\ &= \bigoplus_{0 \le i \le d} \left(\bigoplus_{0 \le j \le d} X^i \cdot Y^j \right) \\ &= \left(\bigoplus_{0 \le i \le d} X^i \right) \cdot \left(\bigoplus_{0 \le j \le d} Y^j \right) = X \cdot Y. \end{aligned}$$

PINI. Next, we prove PINI security in the glitch-extended d-probing model by considering every relevant case of probe placement and arguing about simulatability.

I. A glitch-extended probe on the input to U'_{ij}, i.e., $P_{U'_{ij}} = [Y^j, R'_{ij}]$, can be perfectly simulated by Y^j and tossing a fair coin $R'_{ij} \xleftarrow{\$} \mathbb{F}_2$.

II. The glitch-extended probes $P_{U''_{ij}} = [X^i, R'_{ij}, R''_{ij}]$ can be simulated by X^i and drawing two random bits $R'_{ij}, R''_{ij} \xleftarrow{\$} \mathbb{F}_2$. If additionally $P_{U''_{ji}}$ needs to be simulated, this can straightforwardly be done by adding X^j to the simulation set.

III. A glitch-extended probe on C_{ij}, i.e.,

$$P_{C_{ij}} = [X^i, Y^j \oplus R'_{ij}, \overline{X^i} \cdot R'_{ij} \oplus R''_{ij}]$$

can be simulated by $[X^i, R'_{ij}, R''_{ij}]$, where $R'_{ij}, R''_{ij} \xleftarrow{\$} \mathbb{F}_2$. If additionally $P_{C_{ji}}$ needs to be simulated – which lies in share domain j – this can be done by adding X^j, Y^j and Y^i to the input set of the simulator. As now two probes are considered, this is in conformity to the PINI notion. All other probes can be handled in the same manner but completely independent, as every share domain is blinded with completely fresh (and hence independent) randomness.

IV. A glitch-extended output probe

$$P_{Z_i} = [X^i, Y^i] \cup \left\{ \bigcup_{0 \le j \le d, i \ne j} C_{ij} \right\}$$

can be simulated by X^i and Y^i and drawing d times $R'_{ij} \xleftarrow{\$} \mathbb{F}_2$ and $R''_{ij} \xleftarrow{\$} \mathbb{F}_2$ (to simulate C_{ij}). As by construction, two outputs have at most one cross domain in common, and following the same argument as given above in step III, adding another probe would result in requiring (next to some additional random bits) inputs from only one other domain. This is in conformity to the PINI notion. □

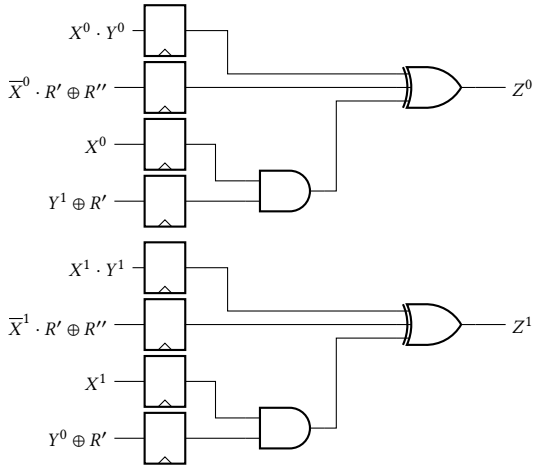


Figure 1: Schematics for HPC3 instantiated for $d = 1$.

In Figure 1, we provide a schematic overview of our construction, configured for the first security order and requiring two freshly drawn random bits. In order to highlight at which points the same randomness is introduced, we simply denote $R'_{01} = R'_{10}$ and $R''_{01} = R''_{10}$ by R' and R'', respectively. Furthermore, we give the schematics for the first circuit share of our second-order ($d = 2$) design in Figure 2.

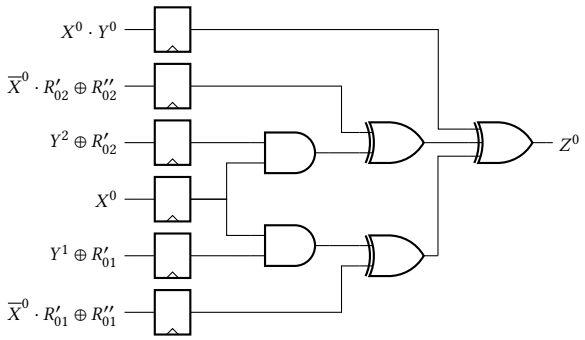


Figure 2: Schematics for HPC3 instantiated for $d = 2$, only the part of the circuit required to generate an output share Z^0.

In addition to proving PINI security in the glitch-extended probing model for general security order d, we further formally verified our construction up to the fourth order utilizing SILVER [33] – an open-source Binary Decision Diagram (BDD)-based software tool for formally verifying different security and composability notions

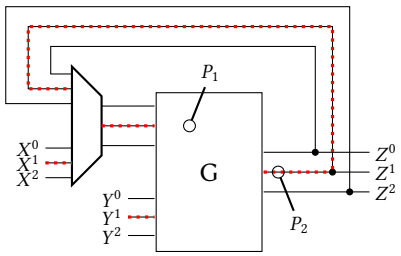


Figure 3: Schematic probe propagation between different iterations of the same gadget instantiation

under the glitch-extended probing model. We choose SILVER over other existing verification tools as, at the time we conducted our research, it was the only tool available for verifying the notion of PINI. In parallel to the publication of this work, IronMask [10] became available as an alternative tool.

3.2 Iterated Glitch+Transition-Robust Variant

As elaborated in [19], fulfilling the notion of PINI may not be sufficient to guarantee security under transitions in an iterative design, i.e., in a design where the same gadget instantiation is executed several times and the input of one iteration depends on the output of a preceding one. The reason is depicted in Figure 3. During an execution of gadget G, an adversary places an internal, transition-extended probe P_1 on the gadget, which w.l.o.g. propagates into share domain 1. As depicted in Figure 3, in order to simulate the probe P_1, the output Z^1 of the prior execution is needed, which is equivalent to placing a second probe P_2 onto Z^1. Now PINI only guarantees that the new probe set $\mathbf{P} = \{P_1, P_2\}$ can be simulated utilizing two input shares instead of one (while only one probe is counted), possibly decreasing the design's security order. As a remedy, the authors introduced the notion O-PINI which, similar to the methodology described in [19], enables us to extend HPC3 to also be trivially composable under (iterated) transitions and glitches. The algorithm of our iterated transition+glitches-robust variant is given as HPC3$^+$ in Algorithm 3. Here, each output is blinded by an individual fresh mask $M_i, 0 \le i \le d$ before saved in an output register, intuitively enabling simpler simulation of the output wires. For the sake of visualization, Figure 4 depicts the schematic of a part of the circuit (for an output share) of HPC3$^+$ configured for $d = 2$.

Due to the share-separating nature of O-PINI, and based on Lemma 1 and Lemma 2 given in [19], we can leverage the fact that our glitch+transition-robust HPC3$^+$ gadget is unrolled, which means that there are no loops within the gadget, enabling us to prove *iterated glitch+transition-robust* composability, i.e., composability under transitions+glitches and subsequent executions of the same hardware gadget, by conveniently staying in our simplified circuit model and proving perfect simulatability in conformity with O-PINI, as given in Definition 2.7.

We now prove the security and trivial composability of our HPC3$^+$ gadget in the presence of both, iterated transitions *and* glitches:

Theorem 3.2. *HPC3$^+$ is iterated glitch+transition-robust t-O-PINI for $t \le d$, where d is the security order.*

Algorithm 3: HPC3$^+$ Multiplication

Input : $Sh(X) = [X^0, \ldots, X^d]$, $Sh(Y) = [Y^0, \ldots, Y^d] \in \mathbb{F}_2^{d+1}$
Output: $Sh(Z) = [Z^0, \ldots, Z^d] \in \mathbb{F}_2^{d+1}$
/* valid sharings of $X, Y, Z = X \cdot Y \in \mathbb{F}_2$ */

$W \leftarrow$ HPC3(X, Y)
for $i = 0$ *to* $d - 1$ **do**
  $M_i \xleftarrow{\$} \mathbb{F}_2$
end
$M_d \leftarrow \bigoplus_{0 \le i < d} M_i$
for $i = 0$ *to* d **do**
  $Z_i \leftarrow Reg[W_i \oplus Reg[M_i]]$
end

Proof.

Correctness. The correctness of the construction is directly implied by the correctness of HPC3 and the fact that $\bigoplus_{0 \le i \le d} M_i = 0$.

O-PINI. The proof follows the same argument as the proof given in [19]. Based on this and the fact that our design is unrolled, we only have to prove that one execution of HPC3$^+$ is glitch-robust O-PINI and we can stay in our simple circuit model. Without loss of generality, we restrict our analysis to the most powerful probes and use the probe simulator for HPC3 considered in the proof of Theorem 3.1. Intuitively, the additional refresh after executing HPC3 enables a simulator to simulate additional output probes possibly caused by subsequent executions of the gadget.

I. Glitch-extended internal probes on W_i are simulated correctly by the simulator for HPC3.
II. Glitch-extended probes on $P_{W_i \oplus M_i} = [W_i, M_i]$ can be simulated by running the simulator of HPC3 to generate W_i – which can be simulated using only shares from share domain i – and by drawing a fresh random bit $M_i \xleftarrow{\$} \mathbb{F}_2$.
III. For glitch-extended probes $P_{Z_i} = [W_i \oplus M_i]$ which are added to the output due to the propagation of an internal probe, i.e., $i \in \mathbf{O^I}$ given in Definition 2.7 of Section 2.5, we consider the following cases:
  i. If not all intermediate values of W_i have been simulated, the simulator simply flips a fair coin and outputs the result.
  ii. If all intermediate values of W_i have already been simulated (for example if there is another probe on W_i), use these values and perform as specified by the algorithm.

The simulation of output probes is correct, as for the worst case where there is a glitch-extended probe on M_d (and hence all $M_i, 0 \le i \le d$ are observed enabling to unblind all P_{C_i}), there are at most $d - 1$ other probes. If one of these probes is P_{W_i} or $P_{W_i \oplus S_i}$, we can trivially simulate it with shares from share domain i and using the simulator of HPC3. If the probes are internal probes on HPC3, no glitch-extended probe on a register stage other than P_{W_i} or $P_{W_i \oplus S_i}$ can observe more than one $R'_{ij} \oplus R''_{ij}$ at once. As a consequence, maximal $d - 1$ of the d terms $R'_{ij} \oplus R''_{ij}$ are observed per i, and Z_i is

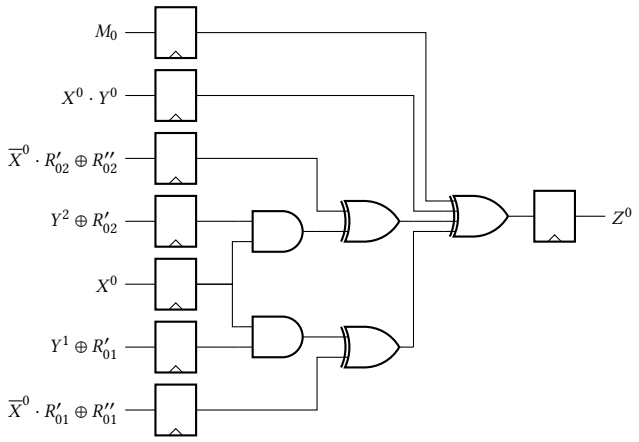


Figure 4: Exemplary circuit share of HPC3$^+$ instantiated for $d = 2$

blinded by one fresh random bit. The simulator can hence always simulate it by tossing a fair coin. □

4 COMPARISON TO STATE-OF-THE-ART HPCs AND LOW-LATENCY DESIGNS

A wide variety of Hardware Private Circuits (HPCs), i.e., trivially composable hardware gadgets, have been recently introduced, differing with respect to their supported security order, their robustness against glitches and transitions, their randomness requirements and their latency. In the following, we introduce all related designs, before we compare them with our low-latency variants. To ensure comparability with respect to composability guarantees, we restrict our comparison to gadgets that are trivially composable in (i) the presence of glitches and/or (ii) transitions.

4.1 Glitch-Robust Hardware Private Circuits

HPC1 *[17].* The HPC1 gadget – introduced by Cassiers et al. – realizes a shared version of a two-input AND gate – generic for any security order d. It is built based on a DOM-AND gadget and additionally refreshing one of the inputs. As a consequence, additional randomness has to be introduced into the design in order to guarantee SNI-conform refreshing. Here, the number of additional random bits compared to the DOM-AND depends on the number of shares, i.e., the security order d.

HPC2 *[17].* In the same work, Cassiers et al. introduced HPC2, which, in contrast to HPC1, further reduces randomness requirements. It is based on blinding terms with fresh randomness depending on their cross domain and guaranteeing that no single glitch-extended probe can observe two terms involving the same randomness at the same time.

GHPC *[34].* In their work, Knichel et al. presented a methodology for transforming any vectorial Boolean function $\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ into a Hardware Private Circuit providing security and composability under the PINI notion in the glitch-extended probing model. As a result, GHPC gadgets enable, for example, the creation of a single gadget realizing entire SBOXes, while requiring only a low amount of fresh randomness, but are limited to the first security order.

GHPC$_{\mathsf{LL}}$ *[34].* In the same line of work, Knichel et al. additionally introduced GHPC$_{\mathsf{LL}}$, enabling the realization of any vectorial Boolean function in one clock cycle – at the cost of significantly higher randomness requirements. Similar to GHPC, GHPC$_{\mathsf{LL}}$ is restricted to the first security order.

4.2 Iterated Transition-Robust Hardware Private Circuits

O-PINI1 [19]. O-PINI1 – recently introduced by Cassiers and Standaert – realizes a masked two-input AND gate that is provably composable under iterated transitions, i.e., when there exist a feedback loop in a design and one input of a gadget depends on the output of the same gadget, and which can be instantiated for arbitrary security orders d. Their construction is directly based on HPC2, only adding an additional refresh of its output. Note that this gadget is either glitch-robust or iterated transition-robust, but not both at the same time. In this work, we focus our consideration on the case where both, glitches and transitions, happen simultaneously, as this is the relevant case for hardware implementations.

O-PINI2 [19]. For the purpose of achieving trivial composability in the presence of iterated transitions *and* glitches, Cassiers and Standaert introduced O-PINI2 in the same work. In comparison to O-PINI1, it introduces an additional register stage at the end of the design, subsequent to the refresh of HPC2's output, further restricting propagation of output probes and hence enabling trivial composition, even in the presence of both transitions and glitches.

4.3 Low-Latency Hardware Designs

CMS$_{\mathsf{LL}}$ *[38].* In this work, Molteni et al. adapted the original Consolidating Masking Schemes (CMS) multiplier to be effectively realized in a single clock cycle by adding pre-computed sums of random bits to the cross domains. Although the achieved design is glitch-robust SNI and not PINI – and hence linear operations cannot be trivially (share-wise) implemented – it is the only other existing composable gadget which realizes an AND operation within a single clock cycle for arbitrary security orders. Hence, we give a detailed comparison between this scheme and our construction later in this section.

LMDPL *[45].* In their work, Sasdrich et al. presented a first-order secure, masked AES with a latency of a single cycle per round by utilizing the concept of LUT-based masked dual-rail pre-charge logic (LMDPL). Although the scheme can be seen as a gadget-based approach, the application of LMDPL gadgets requires circuit-specific signals, generated by a dedicated module called "masked table generator". The authors themselves alleviated the scheme's generality by stating that the underlying technique might not guarantee the same level of SCA resistance when used in ciphers with smaller/fewer SBOXes. As it is only restricted to the first security order and may offer lower levels of security when applied in other contexts, we omit a further, more detailed comparison to this design scheme.

To the best of our knowledge, other non-linear, composable hardware gadgets like DOM [23, 27] offer strictly worse latency than our newly proposed gadget. In [25], Gross et al. proposed an

Table 1: Comparison of existing trivially composable Hardware Private Circuits

Scheme	Latency	Randomness	Function	Ref.
HPC1	2	$d(d+1)/2 + r[d]^*$	AND	[17]
HPC2	2	$d(d+1)/2$	AND	[17]
GHPC†	2	m	$\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$	[34]
GHPC$_{\mathsf{LL}}{}^\dagger$	1	$2^n \cdot m$	$\mathsf{F} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$	[34]
HPC3	**1**	$d(d+1)$	AND	[new]
O-PINI2‡	3	$d(d+1)/2 + d$	AND	[19]
HPC3$^+$ ‡	**2**	$d(d+1) + d$	AND	[new]

* $r[d] = [1, 2, 4, 5, 7, 9, 11, 12, 15, 17]$ for $d \leq 10$
† restricted to $d = 1$
‡ with robustness against iterative transitions+glitches

Table 2: Comparison of single-cycle glitch-robust AND gadgets

Scheme	Framework	d	Randomness	Ref.
CMS$_{\mathsf{LL}}$	SNI	1	2	[38]
		2	6	
		3	16	
		≥ 4	$2(d+1)^2$	
HPC3	PINI	1	2	[new]
		2	6	
		3	12	
		≥ 4	$d(d+1)$	

algorithmic-level hardware masking scheme to achieve low latency for arbitrary security orders which is rooted in the idea of skipping the compression layer of non-linear DOM gadgets whenever possible while ensuring that the sharing of inputs to every non-linear function are independent. Albeit its good results in terms of latency reduction, no formal security proof is given in the robust probing model and, as elaborated in [39], the scheme may suffer from composability issues for higher security orders when instantiated in different settings.

4.4 Comparison

A comparison of our designs with other state-of-the-art PINI gadgets is given in Table 1. Here, the latency is given as the number of register stages, and the required number of random bits is listed. Note that GHPC and GHPC$_{\mathsf{LL}}$ are the only gadgets limited to the first security order. Similar to HPC2, our newly presented HPC3 has a constant latency regardless of the security order, while requiring only a single register stage instead of two. On the other hand, the number of required random bits for HPC3 is doubled compared to HPC2. We would like to highlight that due to the mandatory blinding of one share domain in order to derive the cross terms $X^i \cdot Y^j$, our construction achieves an optimal latency with respect to achieving trivial composability under the notion of PINI in the glitch-extended probing model. As a consequence, HPC3 allows the designer to trade double the randomness for half of the latency, or even combining HPC3 and HPC2 gadgets (as they are both composable under the PINI notion), providing significantly more flexibility with respect to adjusting the latency and randomness requirements of a masked implementation in different use cases.

We further compare the existing transition-robust gadget with our newly introduced one. Applying the same methodology as described in [19] to our HPC3, we derived HPC3$^+$. Similar to O-PINI2, our HPC3$^+$ construction offers trivial composability under the simultaneous presence of both, iterated transitions *and* glitches, while again trading one clock cycle less for $d(d+1)/2$ additional fresh random bits.

In Table 2, a comparison between the required fresh masks of the aforementioned single-cycle robust-SNI gadget [38] and our HPC3 is given. While our design is already advantageous for $d = 3$, it needs less than half of their randomness requirements for $d \geq 4$. Another advantage stems from simpler composition conditions of PINI in comparison to SNI. As shown in [5, 17], trivial, i.e., share-wise, implementations of linear operations are not SNI. This possibly - depending on the circuit structure - demands for additional robust-SNI refresh gadgets to be inserted into the circuit, further increasing the introduced overhead. As the original composition approach by Barthe et al. [5] was over-conservative with respect to the insertion of refresh gadgets, Belaïd et al. introduced a new method in [12], drastically reducing the number of required refresh gadgets. They even utilize their method to generate a secured SNI-based AES implementation without any need for additional refresh gadgets. Although this approach immensely reduces the number of necessary refresh gadgets in the SNI context, the overhead still has a higher dependency on the circuit's structure than in the context of PINI where each non-linear gate can simply be substituted by its gadget variant and the latency is hence simply determined by the maximum number of non-linear gates in any signal path.

To sum up, while there exist certain circuit structures where our approach will achieve similar results for low security orders when compared to CMS$_{\mathsf{LL}}$, the latency, the randomness requirements, and the area (additional registers, larger sources to generate randomness) of an arbitrary design (including all linear operations and layers of the cipher) composed of HPC3 gadgets will be favorable to using CMS$_{\mathsf{LL}}$ in most of the cases - and for all cases when $d \geq 3$.

4.5 Algorithmic-Level Masking vs. Composable Gadgets

Instead of utilizing composable gadgets, there exist several works aiming to derive highly optimized masked version for a certain cipher and design architecture by manually and carefully constructing each block/module in order to ensure security of the entire overarching design [45, 47, 48, 50, 50]. We refer to these approaches as *algorithmic-level* masking. The main advantage of these approaches is that they typically lead to a highly optimized implementation with respect to the introduced overhead. On the downside, these approaches result in implementations that are commonly restricted to a low (mostly the first or second) security order. Extending them to

higher orders and translating them to other ciphers or design architectures is not trivially possible, or not possible at all. They further require a high expertise while the top-level designs' security cannot be easily proved in the formal d-probing model which is rooted in the fact that current formal verification tools like SILVER [33] and IronMask [10] are not able to cope with large circuits. Although these works typically present a leakage assessment of the resulting implementation, we would like to highlight that these leakage assessments are limited to making security statements about the implementation in a very specific setup, while the d-probing model aims to generalize the SCA resilience of an implementation.

Following a divide-and-conquer approach, composable gadgets, on the other hand, enable automated construction of masked circuits [32] by leveraging composability notions like PINI in order to ensure a provably secure top-level implementation, while typically coming at the cost of higher overheads. These schemes are not bounded to any cipher or design architecture but enable straightforward transformation of any circuit into its masked variant at any security order (with GHPC/GHPC$_{\mathsf{LL}}$ being an exception restricted to the first order). In order to provide a fair comparison and as algorithmic-level masking and composable gadgets are conceptually different techniques, we omit any further benchmark comparison with algorithmic-level masking schemes in this work.

4.6 Discussion on Latency-Optimized Logic Representations of Masked Circuits

In [17], Cassiers et al. described an optimized design strategy for small SBOXes using a SAT solver to find a Boolean representation such that one input to each HPC2 gadget in a later circuit stage is a linear combination of inputs to an earlier stage. Due to the unbalance with respect to the input latency of HPC2, this leads to additional overall latency reduction. They showed, for example, that they can realize some 4-bit SBOXes utilizing only 3 register stages instead of trivially using 4 in a binary-tree multiplication of each monom in the respective coordinate functions. Despite the fact that this optimization approach is not practical for more complex functions, e.g., the AES SBOX, our HPC3 gadget still outperforms HPC2 latency-wise even in these cases where the SBOX representation is optimized in favor of HPC2. Note that only 2 clock cycles are necessary to realize a 4-bit SBOX in a trivial way. Hence, in favor of trivial composability, this optimization step is not required anymore when employing our HPC3 gadgets.

5 EXPERIMENTS AND EVALUATIONS

In order to give an overview of the performance figures of our constructed gadgets and the circuits composed of them, in this section, we present several case studies followed by experimental evaluation results.

5.1 Case Studies

Before we explain the details of our covered case studies, we refer to the recently introduced tool for automated generation of masked hardware implementations, AGEMA [32], which is publicly available through GitHub[1]. As given in Section 2.6, it receives the gate-level netlist of the unprotected implementation and generates a masked version of the same design using the specified gadgets. Although different processing methods are offered by AGEMA, we limit our comparisons to the **naive** approach which does not re-synthesize the given netlist and just replaces the gates with their corresponding gadgets. For the sake of comparability, we focus only on utilizing HPC2, O-PINI2, HPC3, and HPC3$^+$ gadgets which can all be instantiated for arbitrary security orders. To this end, we adopted the customized library of AGEMA and specified the gates of the aforementioned gadgets. We further constructed generic VHDL code of these gadgets (for all 2-input gates), where the desired security order d can be arbitrary adjusted. For area comparison, we utilized Synopsys Design Compiler, together with the publicly-available standard library Nangate 45 for synthesizing the design.

Table 3: Performance figures, 2-input AND.

Scheme	Security [order]	Latency [cycle]	Rand. [bit]	Area [GE]	Ref.
HPC2	1	2	1	53	[17]
	2	2	3	156	
	3	2	6	311	
HPC3	1	1	2	43	[new]
	2	1	6	125	
	3	1	12	249	
O-PINI2	1	3	2	137	[19]
	2	3	5	301	
	3	3	9	529	
HPC3$^+$	1	2	3	69	[new]
	2	2	8	167	
	3	2	15	306	

Small Circuits. As the first case study, we compare single realizations of each of the considered AND gadgets, instantiated for the first three security orders $d \in \{1, 2, 3\}$. The corresponding results are shown in Table 3, clearly indicating the benefits of our constructed gadgets with respect to latency and area requirements. Our newly introduced HPC3 requires 20% less area compared to their counterpart HPC2 gadget. This advantage reaches even to around 55% in case of HPC3$^+$ compared to O-PINI2.

As a more realistic case study, we consider the 4-bit SBOX of the SKINNY-64 cipher [8]. The authors of [17] have provided a netlist for the SKINNY SBOX, which is optimized for HPC2 gadgets. The same netlist has been used for benchmarking purposes of AGEMA in [32]. Hence, we have taken the same netlist and constructed the SKINNY SBOX using our constructed gadgets. The results, which are shown in Table 4, reflect roughly the same conclusion as for the 2-input AND. The benefit of our gadgets with respect to latency is highly visible, but the area advantage is slightly mitigated, which is justified by the presence of the same XOR gadgets in all designs independent of the employed gadget family. More precisely, the application of our gadgets would still lead to a lower area overhead by a magnitude of 10%–20% while halving the latency. Note that for the SBOX and full-cipher case studies, we compare the results

[1] https://github.com/Chair-for-Security-Engineering/AGEMA

Table 4: Performance figures, Skinny SBOX [17] and AES SBOX [16].

Scheme	Security [order]	Latency [cycle]	Rand. [bit]	Area [GE]	Ref.
Skinny SBOX					
HPC2	1	4	4	281	[17]
	2	4	12	730	
	3	4	24	1384	
HPC3	1	2	8	241	[new]
	2	2	24	606	
	3	2	48	1133	
AES SBOX					
HPC2	1	8	34	2189	[17]
	2	8	102	5923	
	3	8	204	11469	
HPC3	1	4	68	1849	[new]
	2	4	204	4855	
	3	4	408	9261	

corresponding to only HPC2 and HPC3, since we believe that O-PINI2 and HPC3$^+$ are mainly relevant for the iterative designs shown in Figure 3 which we already compared in Table 3.

For the AES SBOX, we refer to [16], where a description for the inversion in GF(2^8) is given which needs a low number of cascaded 2-input gates. The same design has been used in the case studies of AGEMA as well. Hence, our AES SBOX case study is made based on this netlist, which has 4 layers of cascaded 2-input AND gates. Since our HPC3 gadgets make use of only one register stage, this naturally leads to 4 clock cycles latency for the entire masked AES SBOX at any arbitrary order. The corresponding results are illustrated in Table 4, indicating 15%−20% area reduction in addition to a naturally lower latency compared to the equivalent state-of-the-art designs. As a side note, in all results presented above, we have not considered extra registers required to construct pipeline designs, which is a feature of AGEMA.

Full Ciphers. In order to have a better overview on the benefit and overhead of our constructed gadgets, we took all full-cipher designs used by AGEMA as the case study, including SKINNY-64-64 round-based encryption, AES-128 byte-serial encryption, AES-128 round-based encryption function, round-based encryption of CRAFT [9], nibble-serial encryption of PRESENT [15], round-based encryption of LED-64 [29], and round-based encryption/decryption design of Midori-64 [2]. The corresponding results of the first two case studies (round-based SKINNY and AES) are given in Table 5 and Table 6. We give the remaining results in Tables 7 to 11 in Appendix A.

We provided the comparative results for the first three security orders $d \in \{1, 2, 3\}$ as well as for the pipeline and non-pipeline designs. The pipeline designs, which naturally have higher area footprints, can process multiple sequentially-given inputs. For example, we refer to Table 5, where HPC2 designs have 165 clock cycles latency and HPC3 designs 99 clock cycles. The non-pipeline

Table 5: Synthesis results, SKINNY-64-64 round-based encryption.

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	1494	0.52	-	33
HPC2		1	6895	0.55	64	165
		2	15193	0.61	192	165
		3	26777	0.65	384	165
	✓	1	20210	0.53	64	165
	✓	2	36147	0.59	192	165
	✓	3	56096	0.63	384	165
HPC3		1	6467	1.06	128	99
		2	13517	1.18	384	99
		3	23171	1.19	768	99
	✓	1	13462	0.59	128	99
	✓	2	23956	0.64	384	99
	✓	3	37051	0.66	768	99

Table 6: Synthesis results, AES-128 round-based encryption.

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	9906	1.85	-	11
HPC2		1	52597	2.04	680	99
		2	131631	2.39	2040	99
		3	246924	2.53	4080	99
	✓	1	161440	0.82	680	99
	✓	2	305274	0.89	2040	99
	✓	3	492077	0.93	4080	99
HPC3		1	44581	2.11	1360	55
		2	108476	2.34	4080	55
		3	200307	2.33	8160	55
	✓	1	94450	0.79	1360	55
	✓	2	182883	0.83	4080	55
	✓	3	299013	0.86	8160	55

designs, receive a single plaintext and key and perform the encryption after 165 (resp. 99) clock cycles, what the unprotected design does in 33 clock cycles. The HPC2 pipeline design can process 4 individual plaintext-key pairs in 165 clock cycles, and the HPC3 pipeline design 2 individual inputs in those 99 clock cycles.

The results are indeed along the same line as those given for SBOX case studies. More precisely, the area overhead and latency of the circuits made by HPC3 gadgets are less than those made by HPC2 gadgets while demanding for double amount of fresh randomness. The area advantage becomes more obvious at higher orders – particularly for pipeline designs. We should highlight that, in contrast to the SBOX cases studies, the latency of HPC3 circuits is not exactly half of the latency of HPC2 circuits since registers already exist in the unprotected designs. AGEMA models the given unprotected design as a Mealy machine made by a single-stage register and a fully combinational circuit whose inputs consists the circuit's primary input and the output of the registers provided by a feedback loop (for more details see [32, § 2.6]). The role of HPC2, HPC3 or any other gadget lies in how the combinational circuit is converted into a secure one. Therefore, the register stage

of the Mealy machine stays as it is; just being extended based on the number shares, i.e., the desired security order d. As a result, if we denote the latency of the unprotected circuit by η, the latency of the converted circuit becomes $(l+1)\eta$ cycles, where l stands for the latency of the combinational circuit realized by HPC2 gadgets. This means that the latency of the same circuit implemented by HPC3 gadgets becomes $(l/2+1)\eta$ clock cycles which can also be seen in Tables 5 to 11. It is worth to highlight that all our HDL designs of the gadgets and the case studies are provided in the GitHub: https://github.com/Chair-for-Security-Engineering/HPC3.

5.2 Leakage Assessment

Although (robust) probing security is implied by our gadget's conformity to the PINI notion, we conducted further analysis for the purpose of presenting a complete work and showing our methodologies final practical security. By means of SILVER [33], we have verified the security of the constructed SBOXes (reported in Section 5.1) under the glitch-extended probing model. In short, all our constructions are reported secure up to the desired security order. However, evaluation of larger designs, e.g., a cipher round, is out of the feasibility limits of such verification tools. Therefore, the remaining choice to evaluate a full cipher implementation is to conduct experimental analyses. To this end, we employed a Spartan-6 FPGA-based evaluation platform (SAKURA-G [44]), and collected power consumption traces of different designs to conduct various fixed-versus-random t-tests [7].

Setup. We monitored the output of the embedded AC-amplifier of the SAKURA-G which senses the voltage drop over a 1 Ω shunt resistor placed on the VDD path of the target FPGA, and collected power consumption traces by sampling such an amplified signal at a sampling frequency of 500 MS/s. During this time, the underlying design under test was supplied by a stable and jitter-free clock at a frequency of 6 MHz.

For the sake of comparability with the state of the art, we choose the SKINNY-64-64 encryption function for experimental analysis, since the same has been used in [32]. This choice has indeed neither an effect on the validity of our experiments nor on the security of our constructions. An obvious choice is to examine the AES implementations. However, the third-order round-based AES can hardly fit into our FPGA setup, while all variants of SKINNY-64-64 easily fit.

Nevertheless, we have taken the first- to third-order pipeline designs reported in Table 5. In short, our designs require 99 clock cycles to accomplish the encryption, in contrast to 33 clock cycles for the unprotected design and 165 clock cycles for the designs protected by HPC2 gadgets. Further, our designs require 128, 384, and 768 fresh mask bits, for first- to third-order security respectively, which should be updated at every clock cycle (the same when using HPC3 gadgets).

When measuring the power consumption of our designs, we made sure to cover all 99 clock cycles of each entire encryption process. In order to supply the required fresh masks, we made use of the FPGA-optimized construction illustrated in [37], which realizes an individual 31-bit Linear Feedback Shift Register (LFSR)[2] for every fresh mask bit, seeded randomly at the power-up of the device and updated at every clock cycle.

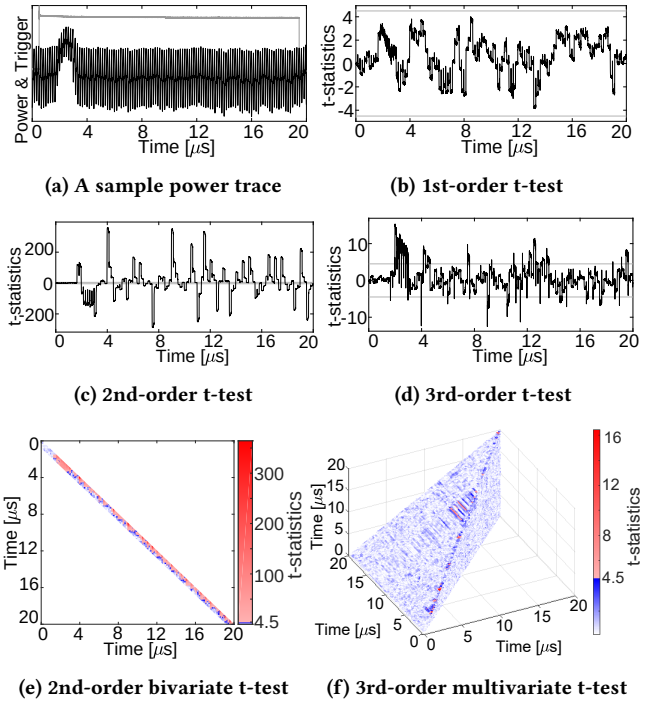
Results. To analyze the implementations, we conducted various forms of a fixed-versus-random t-test, commonly referred to as TVLA [7, 46]. It is a well-known leakage assessment technique that is able to detect SCA leakage in measurements collected from cryptographic implementations by examining whether the leakages associated to two groups are distinguishable; one group with a fixed plaintext and the other one with randomly-chosen plaintexts, while a constant and identical key is used in both groups. In all cases, the entire inputs (plaintext + key) are given to the circuit in a masked form with respect to the security order of the underlying design.

Starting with our first-order design (d=1), we performed the ordinary t-test on each sample point individually (i.e., univariate), whose corresponding results – shown in Figure 5b – confirm its first-order security. As expected, this does not hold true when conducting high-order t-tests on this implementation. For higher orders t-tests, we made the traces mean-free (for each group of fixed and random individually). Afterwards, each mean-free sample point is squared (resp. cubed), before calculating the t-statistics for univariate second-order (resp. univariate third-order) t-tests (see Figure 5c and Figure 5d).

For the bivariate second-order t-tests, we should perform an individual t-test for each combination of every two possible sample points by multiplying the corresponding mean-free power values. In our experiments, each power trace contains 10 000 sample points, which translates to $10\,000 \times 10\,000/2 = 50\,000\,000$ individual t-tests, hence a very time-intensive computation even using large CPU clusters. Since our constructions utilize fresh masks (updated every clock cycle), the bivariate leakage is expected to be present for a combination between sample points in close proximity. Therefore, we limited our bivariate analysis to sample points within a distance of at most five clock cycles. This strongly reduces the amount of computations and allows us to accomplish the evaluations in a reasonable time frame. As expected, the corresponding results – shown in Figure 5e – confirm the existence of second-order bivariate leakages.

For the third-order multivariate analysis, the processing is even more complex. If we limit the maximum distance between the sample points to, e.g., five clock cycles, the number of possible combinations of three sample points is way above the feasibility threshold. In order to find an alternative approach, we refer to Figure 5c and common knowledge indicating that the amount of leakage (in power traces) associated to a clock cycle is approximately the same for the entire clock cycle. Therefore, an appropriate sample point per clock cycle should suffice for such analyses. Indeed, this is a known concept referred to as memory effect in power consumption measurements due to the low-pass filter inherently built by the components involved in the measurement setup, e.g., the shunt resistor, the chip package, and the Printed Circuit Board (PCB) [40]. Therefore, we down-sampled the traces by taking a sample point for each clock cycle (carefully selected at the middle of the cycle). Note that such a down sampling and restricting the multivariate analysis to a small period of time has been done in the sate of the art as well [13, 21, 49, 52]. The result of this analysis is shown by a 3D pyramid in Figure 5f, indicating some tuples (of three points)

[2] With feedback polynomial $x^{31} + x^{28} + 1$.



(a) A sample power trace (b) 1st-order t-test

(c) 2nd-order t-test (d) 3rd-order t-test

(e) 2nd-order bivariate t-test (f) 3rd-order multivariate t-test

Figure 5: Experimental analysis of SKINNY-64-64 encryption round-based design, masked with first-order HPC3 gadgets using 100 million traces.

whose combination (mean-free product) leads to a detectable leakage. Note that higher-order univariate and multivariate leakages are expected in case of this first-order design; we just showed the detailed results of such analyses as a proof of functionality of our setup.

The same procedure has been repeated on the second- and third-order designs (d=2 and d=3 respectively). The corresponding results are depicted in Figure 6 and Figure 7 in Appendix B, confirming the expected security levels. More precisely, no first- and second-order univariate/multivariate leakage has been detected from the traces of the second-order design, and no first-, second- and third-order univariate/multivariate leakage from the third-order design. Note that we have verified the correctness of our setup and its ability to detect univariate and multivariate higher-order leakages using our first-order design (Figure 5).

6 CONCLUSIONS

Summary. In this work, we presented HPC3, a low-latency Hardware Private Circuit that is trivially composable under the PINI notion in the glitch-extended robust probing model. To the best of our knowledge, this is the first PINI-based hardware gadget that realizes a masked AND-gate in a single clock cycle for any arbitrary security order. We further gave the algorithm for its construction and formally proved its conformity to the PINI notion in the glitch-extended d-probing model. We should stress that the achieved latency of a single clock cycle is the lowest bound for trivial composability of an AND gadget under the notion of PINI. This is due to the necessary computation of cross terms, i.e., multiplication of shares from different share domains. Hence, HPC3 is the first proposed gadget achieving this lowest bound for arbitrary security orders. We additionally presented HPC3$^+$, a masked AND-gate that is trivially composable under the presence of both, iterated transitions and glitches and can be instantiated for arbitrary security orders d. Moreover, we practically verified the security of our constructions by means of various case studies and leakage assessments.

Compared to existing state-of-the-art Hardware Private Circuits, HPC3 enables the designers to make use of more fresh randomness to halve the latency while maintaining trivial composability (PINI) in the glitch-extended probing model. This offers a significant speedup of cryptographic implementations for different use cases and paving the ground for secure, low-latency applications. A similar trade-off can be made with HPC3$^+$, which – compared to OPINI-2 – enables the designers to omit one register stage and hence reducing the design's latency by a factor of 2/3.

Future Works. It remains an interesting open topic, if such a trade-off between the demand for fresh randomness and latency can be also found for larger gates, i.e., with more inputs than two. Constructing composable and generic hardware gadgets for 3- or 4-input non-linear gates can highly increase the efficiency of masked implementation with respect to latency. Further, there still exists yet no detailed cost function to examine the overhead introduced by different metrics. For example, it is not clear at which point area consumption required to generate additional fresh randomness compensates the area gain by requiring fewer register stages. It would be hence beneficial for the research community to have detailed and realistic cost functions for these factors.

ACKNOWLEDGMENTS

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

A PERFORMANCE COMPARISON RESULTS

Table 7: Synthesis results, AES-128 byte-serial encryption.

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	3263	0.83	-	227
HPC2		1	10090	2.11	34	2043
		2	17649	2.66	102	2043
		3	27026	2.71	204	2043
	✓	1	42146	0.98	34	2043
	✓	2	65583	1.41	102	2043
	✓	3	91149	1.01	204	2043
HPC3		1	9140	2.27	68	1135
		2	15661	2.55	204	1135
		3	23597	2.47	408	1135
	✓	1	24852	0.89	68	1135
	✓	2	38875	0.99	204	1135
	✓	3	54304	1.07	408	1135

Table 8: Synthesis results, CRAFT round-based encryption.

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	1066	0.58	-	32
HPC2		1	15680	0.94	256	288
		2	43172	1.03	768	288
		3	84024	1.12	1536	288
	✓	1	42367	0.55	256	288
	✓	2	87291	0.57	768	288
	✓	3	148316	0.50	1536	288
HPC3		1	13072	1.05	512	160
		2	35145	1.18	1536	160
		3	67550	1.43	3072	160
	✓	1	27036	0.55	512	160
	✓	2	55896	0.57	1536	160
	✓	3	95257	0.50	3072	160

Table 9: Synthesis results, PRESENT nibble-serial encryption.

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	1613	0.59	-	543
HPC2		1	4160	0.99	4	2715
		2	6478	1.13	12	2715
		3	8977	1.18	24	2715
	✓	1	12103	0.59	4	2715
	✓	2	18270	0.55	12	2715
	✓	3	24692	0.67	24	2715
HPC3		1	3421	0.97	8	1629
		2	5307	1.02	24	1629
		3	7345	1.05	48	1629
	✓	1	7212	0.56	8	1629
	✓	2	10916	0.56	24	1629
	✓	3	14785	0.56	48	1629

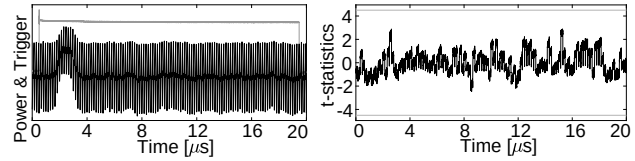
Table 10: Synthesis results, LED-64 round-based encryption.

Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	2056	150	-	33
HPC2		1	7691	1.98	64	165
		2	16375	2.07	192	165
		3	28322	2.33	384	165
	✓	1	20743	1.07	64	165
	✓	2	36890	1.14	192	165
	✓	3	57021	1.18	384	165
HPC3		1	7029	2.19	128	99
		2	14367	2.16	384	99
		3	24290	2.46	768	99
	✓	1	13697	1.08	128	99
	✓	2	24300	1.12	384	99
	✓	3	37483	1.15	768	99

Table 11: Synthesis results, Midori-64 round-based encryption/decryption.

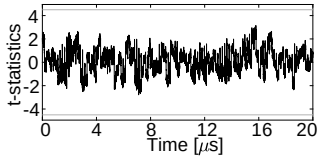
Gadget	Pipel.	Security [order]	Area [GE]	Delay [ns]	Rand. [bits]	Latency [cycles]
unprotected	-	0	2035	0.97	-	17
HPC2		1	17801	1.10	256	153
		2	46371	1.21	768	153
		3	88246	1.27	1536	153
	✓	1	54309	0.95	256	153
	✓	2	105198	0.67	768	153
	✓	3	172179	0.69	1536	153
HPC3		1	14910	1.18	512	85
		2	37892	1.32	1536	85
		3	71207	1.55	3072	85
	✓	1	31718	0.67	512	85
	✓	2	62944	0.64	1536	85
	✓	3	104657	0.65	3072	85

B LEAKAGE ASSESSMENT RESULTS

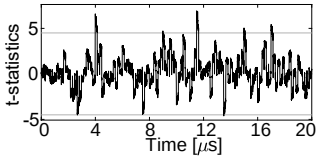


(a) A sample power trace

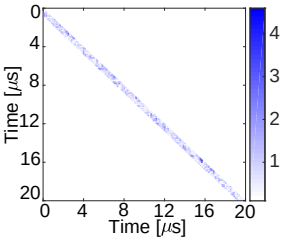
(b) 1st-order t-test



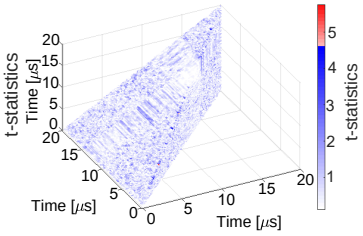
(c) 2nd-order t-test



(d) 3rd-order t-test

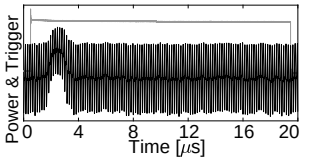


(e) 2nd-order bivariate t-test

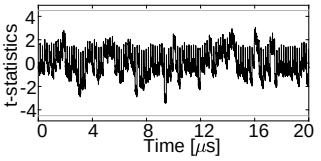


(f) 3rd-order multivariate t-test

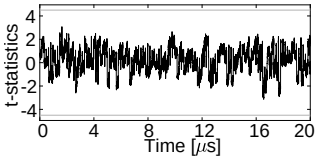
Figure 6: Experimental analysis of SKINNY-64-64 encryption round-based design, masked with second-order HPC3 gadgets using 100 million traces.



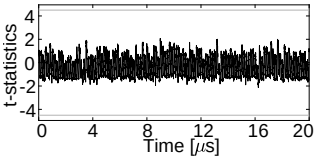
(a) A sample power trace



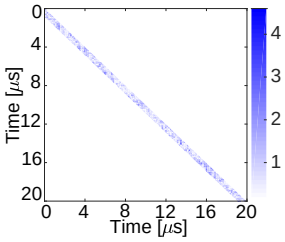
(b) 1st-order t-test



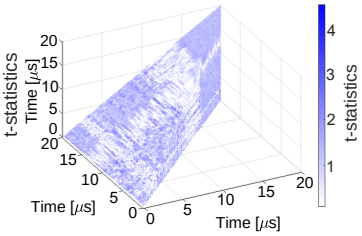
(c) 2nd-order t-test



(d) 3rd-order t-test



(e) 2nd-order bivariate t-test



(f) 3rd-order multivariate t-test

Figure 7: Experimental analysis of SKINNY-64-64 encryption round-based design, masked with third-order HPC3 gadgets using 100 million traces.

REFERENCES

[1] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. 2018. Private Circuits: A Modular Approach. In *CRYPTO 2018 (Lecture Notes in Computer Science)*, Vol. 10993. Springer, 427–455.

[2] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. 2015. Midori: A Block Cipher for Low Energy. In *ASIACRYPT 2015 (Lecture Notes in Computer Science)*, Vol. 9453. Springer, 411–436.

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

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

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

[6] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. 2017. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT 2017 (Lecture Notes in Computer Science)*, Vol. 10210. Springer, 535–566.

[7] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi, and S. Saab. 2013. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*.

[8] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. 2016. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO 2016 (Lecture Notes in Computer Science)*, Vol. 9815. Springer, 123–153.

[9] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. 2019. CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks. *IACR Trans. Symmetric Cryptol.* 2019, 1 (2019), 5–45.

[10] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. 2022. IronMask: Versatile Verification of Masking Security. In *IEEE SP 2022*. IEEE.

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

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

[13] Tim Beyne, Siemen Dhooghe, Amir Moradi, and Aein Rezaei Shahmirzadi. 2022. Cryptanalysis of Efficient Masked Ciphers: Applications to Low Latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 679–721.

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

[15] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. 2007. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007 (Lecture Notes in Computer Science)*, Vol. 4727. Springer, 450–466.

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

[17] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. 2021. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers* 70, 10 (2021), 1677–1690.

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

[19] Gaëtan Cassiers and François-Xavier Standaert. 2021. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 2 (2021), 136–158.

[20] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999 (Lecture Notes in Computer Science)*, Vol. 1666. Springer, 398–412.

[21] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2016. Masking AES with d+1 Shares in Hardware. In *CHES 2016 (Lecture Notes in Computer Science)*, Vol. 9813. Springer, 194–212.

[22] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. 2014. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In *EUROCRYPT 2014 (Lecture Notes in Computer Science)*, Vol. 8441. Springer, 423–440.

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

[24] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *CHES 2001 (Lecture Notes in Computer Science)*, Vol. 2162. Springer, 251–261.

[25] Hannes Groß, Rinat Iusupov, and Roderick Bloem. 2018. Generic Low-Latency Masking in Hardware. *TCHES 2018* 2018, 2 (2018), 1–21.

[26] Hannes Groß and Stefan Mangard. 2018. A unified masking approach. *J. Cryptogr. Eng.* 8, 2 (2018), 109–124.

[27] Hannes Groß, Stefan Mangard, and Thomas Korak. 2016. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS@CCS 2016*. ACM, 3.

[28] Hannes Groß, Stefan Mangard, and Thomas Korak. 2017. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *CT-RSA 2017 (Lecture Notes in Computer Science)*, Vol. 10159. Springer, 95–112.

[29] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. 2011. The LED Block Cipher. In *CHES 2011 (Lecture Notes in Computer Science)*, Vol. 6917. Springer, 326–341.

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

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

[32] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. 2022. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 589–629.

[33] David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020 (Lecture Notes in Computer Science)*, Vol. 12491. Springer, 787–816.

[34] David Knichel, Pascal Sasdrich, and Amir Moradi. 2022. Generic Hardware Private Circuits Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 323–344.

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

[36] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO 1999 (Lecture Notes in Computer Science)*, Vol. 1666. Springer, 388–397.

[37] Lauren De Meyer, Amir Moradi, and Felix Wegener. 2018. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 596–626.

[38] Maria Chiara Molteni, Jürgen Pulkus, and Vittorio Zaccaria. 2022. On robust strong-non-interferent low-latency multiplications. *IET Inf. Secur.* 16, 2 (2022), 127–132.

[39] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. 2019. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 2 (2019), 256–292.

[40] Amir Moradi and Oliver Mischke. 2013. On the Simplicity of Converting Leakages from Multivariate to Univariate - (Case Study of a Glitch-Resistant Masking Scheme). In *CHES 2013 (Lecture Notes in Computer Science)*, Vol. 8086. 1–20.

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

[42] Emmanuel Prouff and Matthieu Rivain. 2013. Masking against Side-Channel Attacks: A Formal Security Proof. In *EUROCRYPT 2013 (Lecture Notes in Computer Science)*, Vol. 7881. Springer, 142–159.

[43] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. 2015. Consolidating Masking Schemes. In *CRYPTO 2015 (Lecture Notes in Computer Science)*, Vol. 9215. Springer, 764–783.

[44] SAKURA. 2022. Side-channel Attack User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html. (2022).

[45] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. 2020. Low-Latency Hardware Masking with Application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020, 2 (2020), 300–326.

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

[47] Aein Rezaei Shahmirzadi, Dusan Bozilov, and Amir Moradi. 2021. New First-Order Secure AES Performance Records. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 2 (2021), 304–327.

[48] Aein Rezaei Shahmirzadi and Amir Moradi. 2021. Re-Consolidating First-Order Masking Schemes Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 1 (2021), 305–342.

[49] Aein Rezaei Shahmirzadi and Amir Moradi. 2021. Second-Order SCA Security with almost no Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 3 (2021), 708–755.

[50] Takeshi Sugawara. 2019. 3-Share Threshold Implementation of AES S-box without Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 1 (2019), 123–145.
[51] Elena Trichina. 2003. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.* 2003 (2003), 236.
[52] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. 2021. Low-Latency Keccak at any Arbitrary Order. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 4 (2021), 388–411.