Constant-Cycle Hardware Private Circuits

Daniel Lammers[1], Nicolai Müller[1], Siemen Dhooghe[2] and Amir Moradi[3]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
firstname.lastname@rub.de
[2] COSIC, ESAT, KU Leuven, Leuven, Belgium
siemen.d@hotmail.com
[3] Technische Universität Darmstadt, Darmstadt, Germany
amir.moradi@tu-darmstadt.de

Abstract. The efficient implementation of Boolean masking with minimal overhead in terms of latency has become a critical topic due to the increasing demand for physically secure yet high-performance cryptographic primitives. However, achieving low latency in masked circuits while ensuring that glitches and transitions do not compromise their security remains a significant challenge. State-of-the-art multiplication gadgets, such as the recently introduced HPC4 (CHES 2024), offer composable security against glitches and transitions, as proven under the robust d-probing model. However, these gadgets require at least one clock cycle per computation, resulting in a latency overhead that increases with the algebraic degree. In contrast, LMDPL gadgets (CHES 2014 & CHES 2020) can achieve fixed latency independent of the algebraic degree, effectively addressing this issue. However, they are limited to two shares, and extending them to guarantee composable security at order d with $d+1$ shares is considered an open challenge.
In this work, we introduce Constant-Cycle Hardware Private Circuits (CCHPC), a novel hardware masking scheme built on the concept of LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL). Specifically, CCHPC achieves a fixed latency of d clock cycles by masking a Boolean function of arbitrary algebraic degree with $d+1$ shares. CCHPC gadgets are secure and trivially composable, as formally proven under the Robust but Relaxed d-probing model (CHES 2024). Using CCHPC gadgets, we design a masked Advanced Encryption Standard (AES) encryption core which can be instantiated for an arbitrary number of $d+1$ shares with a total latency of $11+d$ clock cycles.

Keywords: Composable Gadget, Hardware, Low-Latency, Masking, Probing Security, Side-Channel Security

1 Introduction

Boolean masking is one of the most popular and extensively studied countermeasures against Side-Channel Analysis (SCA) attacks [KJJ99], owing to two key factors. First, the security guarantees and limitations of masking are well-understood and supported by formal proofs. Second, implementing masking securely and efficiently is relatively straightforward, as any algorithm can be securely realized using gadgets, i.e., small circuits that perform atomic operations on a shared representation of sensitive variables [CJRR99]. This approach simplifies the application of masking by reducing the problem to finding a set of efficient and secure gadgets. However, constructing these gadgets is a complex task. Physical defaults like glitches and transitions pose serious threats to the security of masked implementations by potentially compromising the independence of shares. Moreover, each

gadget must not only ensure its own security but also guarantee that its composition with other gadgets does not compromise the overall security of the system. To streamline the design of secure gadgets and establish a formal framework for assessing their security and composability, formal adversary models, abstracting an adversary's ability to observe a circuit, have been developed. The basic d-probing model [ISW03], along with its extension to account for leakage from glitches and transitions, referred to as the robust d-probing model [FGP+18], has proven to offer a realistic and practical abstraction of actual leakage behavior. Within these models, security notions ensuring the security and composability can be formally proven. The Probe-Isolating Non-Interference (PINI) security notion is particularly notable for addressing the composability of gadgets in the presence of glitches [CS20], while its extension O-PINI further accounts for transitions. Both have become the de facto standards for proving gadget's composability. In recent years, a variety of gadgets optimized for different objectives, including circuit size, latency, and the requirement for fresh randomness, have been developed and shown to be provably PINI. However, gadget-based masking still comes with the drawback of increased overhead, which often makes it more efficient to mask a complex function in its entirety. Especially when the goal is to minimize latency, i.e. to keep the number of clock cycles required for an operation as low as possible, the gadget-based masking proves ineffective. Even the latest non-linear gadgets optimized for low latency still require at least one register stage to prevent glitch propagation necessary for achieving PINI, making it impractical to mask functions with a high algebraic degree [KM22, CSV24].

Consequently, researchers in the field of low-latency masking are increasingly exploring alternatives, with LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL) [LMW14], originally presented at CHES 2014, emerging as a promising candidate. LMDPL enables the construction of gadgets that can be composed by running their registers in parallel, achieving a constant latency of two clock cycles, regardless of the complexity of the masked function. However, security and composability of LMDPL gadgets cannot be proven using standard security notions, as the robust probing model does not accurately consider the partially glitch-free behavior of LMDPL. Consequently, research has focused on investigating the security and composability of LMDPL through analytical approaches and practical experiments [SBHM20, MLM24]. To bridge this gap in formal security proofs, a relaxed version of the robust probing model, referred to as the Robust but Relaxed (RR) probing model, has been introduced at CHES 2024 [MM24]. This model specifically focuses on glitches that can manifest in practice based on the processed data. It has been shown that this model can be used to formally prove the security and composability of LMDPL and enables the construction of new gadgets that can be proven secure under this model, which is closer to reality and keeps a distance to the conservative nature of the robust probing model. We should also refer to Low-Latency version of Generic Hardware Private Circuits (GHPC_{LL}) [KSM22] and Time Sharing Masking (TSM) [VDB+24], where approaches are introduced to construct the masked target function, e.g., an S-box entirely, in conformity with the PINI notion, but with a fixed latency. Similar to LMDPL, they are limited to first order, i.e., only 2 shares, and their composition – unlike LMDPL – adds up to latency. In short, they are not to construct small gadgets, whose compositions result in a low-latency circuit.

1.1 Our Contributions

Building on the current state-of-the-art – which includes on the one hand various gadgets proven under the robust probing model with high latency, and on the other hand LMDPL which implements a first-order secure gadget with constant latency under the RR probing model – our contributions are as follows.

1. We introduce Constant-Cycle Hardware Private Circuits (CCHPC), a composable

gadget that, like LMDPL, supports the masking of arbitrary computations with constant latency, meaning that the composition of CCHPC gadgets does not increase overall latency. Contrary to LMDPL, CCHPC can be instantiated for any security order d. Formally, we prove that CCHPC gadgets satisfy (O-)PINI under the RR d-probing model.

2. We examine the application of CCHPC at the architectural level to design a low-latency Advanced Encryption Standard (AES)-128 architecture that is secure under the RR d-probing model. The underlying architecture is similarly structured as the LMDPL-protected AES encryption architecture presented in [SBHM20] and achieves a total latency of $11 + d$ clock cycles for d-order security.

3. We use the PROLEAD [MM22] tool to demonstrate the composable security of the proposed CCHPC gadgets under the RR d-probing model and further reinforced our claims through experimental leakage assessments conducted on an Field Programmable Gate Array (FPGA), targeting our low-latency AES-128 architecture.

2 Background

2.1 Notation

We denote elements in a field with lowercase letters such as $x \in \mathbb{F}_2$, and sets with capital letters such as $X \in \mathbb{F}_2^n$, where x_i is the i-th element in X. Further, we denote matrices (sets of sets) by calligraphic letters such as $\mathcal{X}$. We use bold letters for the dual-rail representations of signals, i.e., $\boldsymbol{x} = (x_\mathrm{t}, x_\mathrm{f})$ and $\boldsymbol{X}$ for sets accordingly. Consequently, $\widetilde{\boldsymbol{X}}$ denotes a set of shares in their dual-rail representation, i.e., $\widetilde{\boldsymbol{X}} = \left\{\left(\boldsymbol{x}^0, \ldots, \boldsymbol{x}^d\right)\right\} = \left\{\left(x_\mathrm{t}^0, x_\mathrm{f}^0\right), \ldots, \left(x_\mathrm{t}^d, x_\mathrm{f}^d\right)\right\}$. If $x \in \mathbb{F}_2$ represents the signal on a particular wire, we omit the time instance for better readability when it is not relevant. However, if we assume that x represents the value of a signal during clock cycle t, we use x' to denote the value of the same signal during clock cycle $t-1$, i.e., the preceding signal. Finally, we use sans-serif fonts to denote Boolean functions with n inputs, like $\mathsf{f} : \mathbb{F}_2^n \mapsto \mathbb{F}_2$.

2.2 Constraint Automaton

Constraint automata serve as acceptors for Timed Data Streams (TDSs) [AR02] associated with the input and output variables of a circuit [BSAR06]. A TDS of length n is defined as a tuple (A, T) consisting of a data stream $A = (a_0, \ldots, a_{n-1}) \in \mathbb{F}_2^n$ and a time stream $T = (t_0, \ldots, t_{n-1}) \in \mathbb{N}^n$ with $t_0 < \ldots t_{n-1}$. Hence, it holds that $(a_i, t_i), i \in [0, n)$ denotes a data value a_i which is assigned during clock cycle t_i. A constraint automaton is defined as a quadruple $\mathcal{A} = (Q, \Sigma, \rightarrow, q_\mathsf{start})$ over a data space, where: Q is a finite set of states, Σ is a set of input/output variables, $\rightarrow$ is a set of transition relations, and q_start is the initial state. Each transition relation is of the form $q_\mathrm{now} \xrightarrow{N,g} q_\mathrm{next}$, where $N \subseteq \Sigma$, $N \neq \emptyset$, represents the set of input/output variables that may change to satisfy a constraint g. Assuming that the constraint automaton is currently in state q_now, it waits for new values to be assigned to variables in Σ in an upcoming clock cycle of T. We denote the current value assigned to the variable x as d_x, and the previously assigned value as d'_x. A transition is valid if the new values are assigned only to variables in N, and these assignments satisfy the constraint g, triggering a transition from q_now to q_next. If no valid transition emerges, the constraint automaton is said to *reject*. Otherwise, if no rejection occurs during the entire time span T, the automaton is said to *accept*. In this work, we exclude probabilistic constraint automata, assuming that each automaton has a single initial state and, at most, one valid transition per state.

2.3 Circuit Model

We adopt the structural circuit model introduced in [CS21], where a structural circuit is modeled as a directed graph $C = (V, E)$. In this model, the vertices in V correspond to structural gates and the edges in E represent structural wires. Structural gates are represented as quadruples, consisting of a set of inputs, public and secret parameters, the Boolean function that defines the gate's output value, and the associated latency. Structural wires connect these gates by forwarding elements in $\mathbb{F}_2$ across different time instances, i.e., clock cycles. In this model, combinational gates perform their function with zero latency, while sequential gates, such as registers, introduce a latency of one, propagating their input data to the output after one clock cycle. Therefore, we can model the execution of arbitrary circuits, as well as iterative circuits – those that evaluate the same gate across multiple clock cycles – by defining circuit execution in the following way.

Definition 1 (Circuit Execution)**.** The execution of a structural circuit $C = (V, E)$ over a set of cycles T is a directed graph $C_T = (V_T, E_T)$ with $V_T \in V \times T$ and $E_T \in E \times T$. Specifically, each wire $(e, t) \in E_T$ with $e \in E$ and $t \in T$ connects a source gate to a destination gate according to their latency.

2.4 Circuit Compiler

A circuit compiler [AIS18] is defined as a triple of algorithms $(\mathsf{CC}, \mathsf{ENC}, \mathsf{DEC})$, where:

- The circuit compiler algorithm CC is a deterministic algorithm that takes a structural circuit C as input and transforms it into a randomized (masked) structural circuit $\widetilde{C}$.
- The encode algorithm ENC is a probabilistic algorithm that takes the inputs X of C and returns the shared inputs $\widetilde{X}$ of $\widetilde{C}$ with respect to Boolean Masking.
- The decode algorithm DEC is a deterministic algorithm that takes the shared outputs $\widetilde{Y}$ of $\widetilde{C}$ and reverses the process of ENC to obtain Y, i.e., the outputs of C.

We note that every circuit compiler must be correct in the sense that the following condition holds: $Y = \mathsf{DEC} \circ \widetilde{C} \circ \mathsf{ENC}(X)$ with $Y = C(X)$.

2.5 Boolean Masking

Masking has proven to be highly effective for hardware implementations as a countermeasure against SCA attacks [CJRR99]. Similar to a secret sharing scheme [Sha79], masking splits every sensitive Boolean variable $x \in \mathbb{F}_2$ into a set of $d + 1 \geq 2$ shares $\widetilde{X} = \{x^0, \ldots, x^d\}$, where the first d shares are sampled randomly to be uniformly distributed. In the case of Boolean masking, we treat every variable x as an element in $\mathbb{F}_2$ and $\widetilde{x}$ as an element in $\mathbb{F}_2^{d+1}$. Consequently, the last share x^d is computed as $x^d = \left(\bigoplus_{i=0}^{d-1} x^i\right) \oplus x$ ensuring that $x = x^0 \oplus \ldots \oplus x^d$. Therefore, the hardware implementation must be adjusted to perform all logic operations on $\widetilde{x}$ rather than on x itself.

2.6 Adversary Model

We consider a passive adversary who can observe a masked circuit $\widetilde{C}$ but does not have access to $\widetilde{X}$ or $\widetilde{Y}$ entirely, i.e. all outputs of ENC or all inputs of DEC. However, the circuit compiler itself is known to the adversary. In the following sections, we introduce the relevant security models to analyze the security of $\widetilde{C}$.

2.6.1 Robust d-Probing Model

In the d-probing model [ISW03], an adversary is limited to jointly observing up to d stable signals of a circuit. These observations are modeled by d so-called standard probes that the adversary can place on arbitrary structural wires within $\widetilde{C}$. A circuit is considered secure under the d-probing model if the joint distribution of intermediates observed by the adversary is statistically independent of any sensitive information. However, the d-probing model assumes an ideal circuit where all physical gates and wires are updated simultaneously, and each wire carries the output of the driving gate based on the current assignment of primary inputs with no delay, i.e., atomic gates. Thus, a standard probe captures one stable signal at a specific clock cycle. In practice, the assumption of an ideal circuit does not hold, prompting the development of the robust d-probing model [FGP+18]. This model extends the d-probing model by accounting for the impact of physical phenomena, namely glitches [MPG05], transitions [CGP+12], and couplings [CBG+17] on the security of $\widetilde{C}$. This is achieved by modeling the information gain from these effects individually through so-called extended probes, which can record multiple signals potentially revealed by such phenomena. Consequently, an extended probe abstracts a set of multiple standard probes. A transition-extended probe models the information an adversary gains from the energy consumed when overwriting a wire, depending on the values carried by the wire both before and after the transition. According to Definition 1, a transition-extended probe on a structural wire (e, t) extends to a tuple of standard probes placed on the preceding wire $(e, t-1)$ and the subsequent wire (e, t). Glitches refer to unintended signal transitions caused by different delay paths in a digital circuit. When observing a signal, an adversary may detect a series of transient changes before the signal stabilizes. This sequence can reveal information about multiple elements that contribute to the probed signal. Consequently, a glitch-extended probe traces back to the last synchronization point. If a glitch-extended probe is placed on a structural wire (e, t), it extends to a set of standard probes on all structural wires carrying register outputs or primary inputs required to compute the value carried by (e, t). Both types of extended probes can be combined to account for glitches and transitions simultaneously. A similar extension procedure can also address the information gained from coupling effects. However, since our circuit model does not include details on the circuit's placement and routing, we exclude coupling from this work.

2.6.2 Robust but Relaxed (RR) d-Probing Model

The RR d-probing model ensures the same level of security as the robust d-probing model including glitches and transitions while offering a less conservative approach to cover glitches [MM24]. In particular, the RR d-probing model accounts for as possible occurrences of both potential glitches in a circuit and the effects of early evaluation. Similarly to the precise handling of glitches in [HHB24], this model conditionally extends probes to target only those glitches that may occur based on the processed data. This is represented by the so-called relaxed probes, which are glitch- and transition-extended probes that undergo conditional extension. The formal definition of RR d-probing security is as follows:

Definition 2 (RR d-Probing Security)**.** A masked circuit execution $\widetilde{C}$ with a set of secret input vectors $\mathcal{X}$ is RR d-probing secure iff for any set of relaxed probes P, with $|P| \leq d$, it holds that

$$\forall Q \in \mathcal{Q}, X \in \mathcal{X}, \quad Pr[Q|X] = Pr[Q], \tag{1}$$

while $\mathcal{Q}$ denotes all observations made by the relaxed probes P.

The exact probe-extension procedure is detailed in Algorithm 1, which builds upon Algorithm 3 from [MM24]. Specifically, it focuses on combinational gates with a fan-in of 2, such as an AND gate computing $z = a \wedge b$. Moreover, the probe-extension process is based on a sequence of inputs processed by the combinational gates. For instance, in the case of an AND gate, we assume it first computes $z' = a' \wedge b'$ before transitioning to $z = a \wedge b$.

Algorithm 1 Relaxed Probe-Extension (relaxed_extend)

Input: P_z ▷ Relaxed probe on wire z
Output: $\boldsymbol{P}$ ▷ The set of extensions on P_z
1: **if** P_z is placed on the output of a combinational gate with a fan-in of 2 **then**
2: **if** $F_z = 1$ **then**
3: $\boldsymbol{P} \leftarrow \texttt{relaxed_extend}(P_a) \cup \texttt{relaxed_extend}(P_b)$ ▷ P_a and P_b denote the relaxed probes on both inputs a and b of the combinational gate
4: **else**
5: $\boldsymbol{P} \leftarrow \{P_{z'}, P_z\}$ ▷ Do the transition extension
6: **end if**
7: **else if** P_z is placed on the output of a NOT-gate **then**
8: $\boldsymbol{P} \leftarrow \texttt{relaxed_extend}(P_a)$ ▷ P_a denotes the relaxed probe on the input a of the NOT-gate
9: **else if** P_z is placed on an output of a sequential gate or a primary input **then**
10: $\boldsymbol{P} \leftarrow \{P_{z'}, P_z\}$ ▷ Do the transition extension
11: **end if**

Algorithm 1 illustrates the distinction between the RR d-probing model and the robust d-probing model. In the robust d-probing model, probes are always extended through combinational gates, whereas in the RR d-probing model, this extension occurs only under a specific condition (cf. Line 2). The conditional value $F_z \in \mathbb{F}_2$ is determined by the probe-extension function, which is unique for every atomic logic gate type. Based on the data-dependent gate behavior, it indicates whether probing z reveals information about the inputs. The probe-extension function takes the previous input states a' and b', the current input states a and b, and another value, $G_z \in \mathbb{F}_2$, as inputs. G_z is determined by the glitch-propagation function, which determines whether the previous and current input states could cause a glitch on the probed wire and whether an existing glitch at the input can propagate to the output. This means that the model extends probes backward while propagating glitches forward. We demonstrate the computation of F_z and G_z for the atomic logic gates AND and OR, which are relevant in the context of this work, as examples.

Example 1 (AND (resp. NAND))**.** Consider an AND (resp. NAND) gate computing $z = a \wedge b$ (resp. $z = \overline{a \wedge b}$) and a relaxed probe P_z placed on its output z. P_z extends to both inputs, i.e. $P_z \rightarrow \{P_a, P_b\}$ only if $F_z = 1$ while the value of F_z is determined as follows:

$$F_z = (a' \vee a \vee G_a) \wedge (b' \vee b \vee G_b)$$

$$G_z = (a' \wedge \overline{a} \wedge \overline{b'} \wedge b) \vee (\overline{a'} \wedge a \wedge b' \wedge \overline{b}) \vee (G_a \wedge (b' \vee b \vee G_b)) \vee (G_b \wedge (a' \vee a \vee G_a))$$

Example 2 (OR (resp. NOR))**.** Consider an OR (resp. NOR) gate computing $z = a \vee b$ (resp. $z = \overline{a \vee b}$) and a relaxed probe P_z placed on its output z. P_z extends to both inputs, i.e. $P_z \rightarrow \{P_a, P_b\}$ only if $F_z = 1$ while the value of F_z is determined as follows:

$$F_z = (\overline{a}' \vee \overline{a} \vee G_a) \wedge (\overline{b'} \vee \overline{b} \vee G_b)$$

$$G_z = (a' \wedge \overline{a} \wedge \overline{b'} \wedge b) \vee (\overline{a'} \wedge a \wedge b' \wedge \overline{b}) \vee (G_a \wedge (\overline{b'} \vee \overline{b} \vee G_b)) \vee (G_b \wedge (\overline{a'} \vee \overline{a} \vee G_a))$$

Additionally, we usually assume that $G_z = 0$ if z is a primary input or the output of a register. The conditional probe-extension makes the RR d-probing model well-suited for evaluating the security of masking schemes that are inherently glitch-free, such as LMDPL [LMW14] which would be trivially insecure under the robust d-probing model [SBHM20, MLM24].

2.7 Composable Gadgets

However, the computational complexity of proving the security of a masked circuit under the robust (but relaxed) d-probing model increases exponentially with the circuit size and the number of shares, making it impractical for larger circuits handling multiple shares. To mitigate this increasing complexity, each gate of the circuit can be replaced by a gadget – a securely masked component that adheres to specific composability properties. This method facilitates proving the security of the entire masked circuit by reducing it to individual, easier-to-manage security proofs for each gadget. To formally analyze the composability of these gadgets, we restate the concept of *simulatability*, which defines the information that a probe can access from the input shares [BBD+16]. Furthermore, we refer to a gadget execution as defined in Definition 1.

Definition 3 (Simulatability)**.** Let P be a set of robust (resp. relaxed) probes in a gadget execution. P can be simulated by a simulation set S, i.e. a set of input shares, iff it holds that

$$\forall Q \in \mathcal{Q}, \widetilde{X} \in \widetilde{\mathcal{X}}, \quad Pr[Q|S] = Pr[Q|\widetilde{X}], \tag{2}$$

while $\mathcal{Q}$ denotes all observations made by the probes P, and $\widetilde{\mathcal{X}}$ the set of all possible shared inputs.

To ensure the secure composability of gadgets at any security order, the simulatability of each probe must be guaranteed based on a restricted simulation set, as formally defined by the PINI security notion in [CS20].

Definition 4 (Probe-Isolating Non-Interference (PINI))**.** Let C be a gadget execution, I a set of robust (resp. relaxed) probes on its internal wires, O a set of robust (resp. relaxed) probes on its output shares, and A the set of indices of the shares probed by probes in O. The gadget is robust (resp. relaxed) d-PINI iff for all probes in $I \cup O$ with $|I| + |A| \leq d$ there exists a set of indices B with $|B| \leq |I|$ such that probes in $I \cup O$ can be simulated using only input shares with indices in $A \cup B$.

The O-PINI security notion is ultimately used to capture trivial composition in the presence of glitches and transitions [CS21]. It can be seen as a stronger version of PINI, where the simulator must simulate the outputs that share the same index as the input share being probed. O-PINI security is required when the circuit's output directly provides its next input. If there are at least two register layers between the circuit's input and output, PINI notion is enough to guarantee security of the composed circuit.

Definition 5 (Output Probe-Isolating Non-Interference (O-PINI))**.** Let C be a gadget execution, I a set of robust (resp. relaxed) probes on its internal wires, O a set of robust (resp. relaxed) probes on its output shares, and A the set of indices of the shares probed by probes in O. The gadget is robust (resp. relaxed) d-O-PINI iff for all probes $I \cup O$ with $|I| + |A| \leq d$, there exists a set of indices B with $|B| \leq |I|$ such that probes $I \cup O$ and output shares with index in B can be simulated using only input shares with indices in $A \cup B$.

2.8 Dual-Rail Pre-charge (DRP) Logic

DRP logic was originally introduced as a power-equalization technique to maintain a constant number of toggles in each clock cycle, thereby mitigating power analysis attacks. To achieve this, a circuit using DRP logic represents each Boolean variable $x \in \mathbb{F}_2$ as a tuple $\boldsymbol{x} = (x_\mathrm{t}, x_\mathrm{f})$. Here, x_t is equal to x, while x_f denotes the complementary value. For instance, the dual-rail representation $\boldsymbol{x} = (1, 0)$ represents $x = 1$, while $\boldsymbol{x} = (0, 1)$ represents $x = 0$. Additionally, a circuit implementing DRP logic is usually designed to satisfy two additional requirements:

- Each dual-rail encoded variable $\boldsymbol{x}$ must be initially set to a null state $\boldsymbol{x} = (0, 0)$ before evaluation. This ensures that every valid transition, whether $(0, 0) \rightarrow (1, 0)$ or $(0, 0) \rightarrow (0, 1)$, results in a fixed amount of one toggle per dual-rail encoded signal. To achieve this, the circuit alternates between an evaluation phase and a pre-charge phase, during which all dual-rail signals are set to null.
- Every combinational gate in the circuit must be monotonic positive, as defined in Definition 6. This means that a gate output can only toggle from zero to one during the evaluation phase and from one to zero during the pre-charge phase. This ensures that the circuit is glitch-free.

Definition 6 (Monotonic)**.** A circuit is monotonic positive (resp. negative) iff it computes a Boolean function $\mathsf{f} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, where for every $X, Y \in \mathbb{F}_2^n, X \leq Y \implies \mathsf{f}(X) \leq \mathsf{f}(Y)$ (resp. $\mathsf{f}(X) \geq \mathsf{f}(Y)$) while $X \leq Y$ means that $x_i \leq y_i$, $\forall\, i \in \{0, \ldots, n-1\}$.

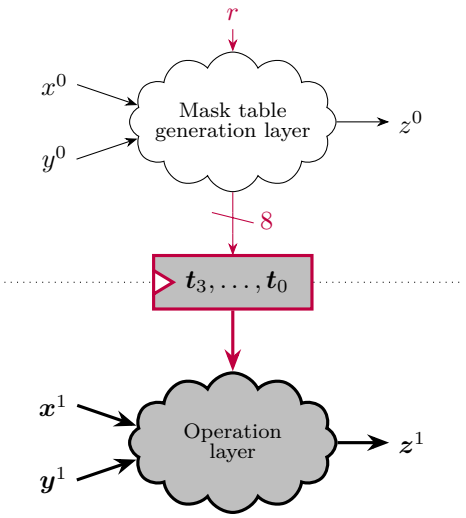
2.9 LUT-based Masked Dual-Rail with Pre-charge Logic (LMDPL)

LMDPL defines a circuit compiler algorithm that transforms a circuit implementing an arbitrary Boolean function into a first-order secure masked LMDPL gadget, processing two shares per element. This transformation enables the construction of LMDPL gadgets from atomic gates, which can then be utilized in another circuit compiler. This second compiler replaces all gates in an unmasked circuit with LMDPL gadgets that achieve the same functionality [LMW14]. Importantly, every masked circuit composed solely of LMDPL gadgets achieves a constant latency of two clock cycles per execution. Each LMDPL gadget is structured in two layers, as illustrated in Figure 1(a).

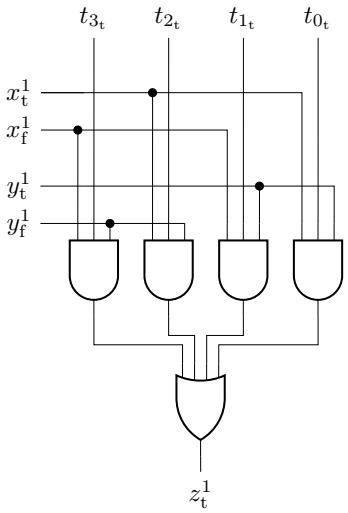
The mask table generation layer of a non-linear gadget pre-processes all single-rail input elements with share index 0, i.e. x^0 and y^0 along with a fresh mask r to compute $\boldsymbol{t}_0, \ldots, \boldsymbol{t}_3$. The output share z^0 is derived solely from the fresh mask, resulting in $z^0 = r$. The operation layer takes the remaining shares $\boldsymbol{x}^1$ and $\boldsymbol{y}^1$ in their dual-rail representation to compute output share $\boldsymbol{z}^1$. Since this layer is constructed using DRP logic, it guarantees no glitches. In the case of a non-linear gadget implementing the function $\mathsf{f}(.,.)$ over two variables (e.g., 2-input AND gate), a register stage separates both layers, while the fresh mask blinds all masked table outputs $\boldsymbol{t}_0, \ldots, \boldsymbol{t}_3$, which are computed as follows.

$$t_{(2i+j)_\mathrm{t}} = \mathsf{f}(x^0 \oplus j, y^0 \oplus i) \oplus r, \qquad t_{(2i+j)_\mathrm{f}} = t_{(2i+j)_\mathrm{t}} \oplus 1, \qquad 1 \leq i, j \leq 0$$

With one clock cycle offset, the operation layer processes all $\boldsymbol{t}_0, \ldots, \boldsymbol{t}_3$ using two 4-to-1 multiplexers, with $\boldsymbol{x}^1$ and $\boldsymbol{y}^1$ serving as a 2-bit select signal (as visualized in Figure 1(b)). We remark that all elements with share index 1 and all $\boldsymbol{t}_0, \ldots, \boldsymbol{t}_3$, which are stored in registers, follow a pre-charge/evaluation fashion per clock cycle to guarantee the absence of glitch propagation from the mask table generation layer into the operation layer. In this way, one clock cycle is required by the mask table generation layer to compute and store the corresponding values in $\boldsymbol{t}_0, \ldots, \boldsymbol{t}_3$, while they are reset in the former clock cycle together with $\boldsymbol{x}^1$ and $\boldsymbol{y}^1$ to be in the pre-charge phase. One more clock cycle is required by the operation layer to evaluate its inputs, i.e., to be in the evaluation phase.



(a) LMDPL gadget layers. DRP logic in gray and **bold**. Non-linear only parts colored in red.



(b) One output rail of a non-linear LMDPL gadget operation layer. z_f^1 is generated analogously.

Figure 1: An overview of LMDPL gadgets [SBHM20].

3 Basic DRP Components

In this work, we introduce DRP circuits that do not fully satisfy the requirements outlined in Section 2.8. The reason is that our construction's security does not rely on power equalization for which DRP logic was originally introduced. Formally, a DRP circuit evaluates a Boolean function $\mathsf{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m$, where $\mathsf{f}(a_0, \ldots, a_{n-1}) = (z_0, \ldots, z_{m-1})$. f is implemented in dual-rail logic as $\mathsf{h} : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2m}$, with $\mathsf{h}(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1}) = (\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{m-1})$, using only combinational gates. We apply DRP logic to ensure that, in specific parts of our final construction, glitches cannot occur or propagate, provided that only certain transitions are allowed on the primary inputs. To accommodate this approach, we model the constraints on the primary signals using a constraint automaton $\mathcal{A}^{n,m} = (Q, \Sigma, \to, q_{\text{start}})$, where $Q = \{q_{\text{start}}, q_{\text{toggle}}\}$, $\Sigma = \{\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1}, \boldsymbol{z}_0, \ldots, \boldsymbol{z}_{m-1}\}$ over a global data domain, i.e., $\forall \boldsymbol{v} \in \Sigma, d_{\boldsymbol{v}} \in \{(0,0),(0,1),(1,0)\}$. In addition, we specify the following transition relations in $\to = \{\to_{\text{start}}, \to_{\text{eval}}, \to_{\text{prech}}\}$.

$$
\begin{aligned}
&\to_{\text{start}} = q_{\text{start}} \xrightarrow{\Sigma, g_{\text{start}}} q_{\text{toggle}}, && g_{\text{start}} = \forall \boldsymbol{v} \in \Sigma, d_{\boldsymbol{v}} = (0,0) \\
&\to_{\text{eval}} = q_{\text{toggle}} \xrightarrow{\Sigma, g_{\text{eval}}} q_{\text{toggle}}, && g_{\text{eval}} = \forall \boldsymbol{v} \in \Sigma, d'_{\boldsymbol{v}} = (0,0) \vee d'_{\boldsymbol{v}} = d_{\boldsymbol{v}} \\
&\to_{\text{prech}} = q_{\text{toggle}} \xrightarrow{\Sigma, g_{\text{prech}}} q_{\text{toggle}}, && g_{\text{prech}} = \forall \boldsymbol{v} \in \Sigma, d_{\boldsymbol{v}} = (0,0) \vee d'_{\boldsymbol{v}} = d_{\boldsymbol{v}}
\end{aligned}
$$

Based on the constraint automaton $\mathcal{A}^{n,m}$, we now define the correctness for DRP circuits.

Definition 7 (Correctness for DRP circuits)**.** Let $\mathsf{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with $\mathsf{f}(a_0, \ldots, a_{n-1}) = (z_0, \ldots, z_{m-1})$ be a Boolean function. Let C be a DRP circuit that implements $\mathsf{h} : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2m}$ with $\mathsf{h}(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1}) = (\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{m-1})$. Then, C is a correct DRP circuit for f iff $\forall\ (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1}) \in \{(0,1),(1,0)\}^n \implies \mathsf{f}(a_{0_t}, \ldots, a_{n-1_t}) = (z_{0_t}, \ldots, z_{m-1_t}) \wedge \forall j \in [0, m), z_{j_f} = \overline{z_{j_t}}$.

Following Section 2.8, we assume that every $\boldsymbol{a}_i$ does not undergo two consecutive evaluation phases without an intervening pre-charge phase. Furthermore, we restrict the set of valid transitions on $(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1})$ to all input transitions where each signal either

holds its value or all $\boldsymbol{a}_i$ perform the same phase transition (either pre-charge or evaluation), but never a mix of both phases. Based on $\mathcal{A}^{n,m}$ we define the glitch-freeness of a DRP circuit based on the glitch-propagation function of the RR d-probing model.

Definition 8 (Glitch-Freeness)**.** Let $C = (V, E)$ be a DRP circuit execution over a series of clock cycles in T where $\mathsf{h} : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2m}$ with $\mathsf{h}(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1}) = (\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{m-1})$ and let $\mathcal{A}^{n,m}$ be the constraint automaton. Suppose that $\forall\ i \in \{a_{0_\mathrm{t}}, a_{0_\mathrm{f}} \ldots, a_{n-1_\mathrm{t}}, a_{n-1_\mathrm{f}}\} \times T, G_i = 0$. Then, C is glitch-free iff for any set of TDSs accepted by $\mathcal{A}^{n,m}$ it holds that $\forall\ e \in E \times T, G_e = 0$.

In the following, we prove that arbitrary compositions of glitch-free DRP circuits remain glitch-free.

Lemma 1. *A DRP circuit execution C on n inputs, m outputs, and the set of clock cycles T composed of glitch-free DRP circuits C_i with n_i inputs and m_i outputs is glitch-free for any set of TDSs accepted by $\mathcal{A}^{n,m}$.*

Proof. Suppose an arbitrary set of TDSs for C over T, which $\mathcal{A}^{n,m}$ accepts. Then, it follows that $\mathcal{A}^{n,m}$ applies a series of transition relations from $\rightarrow$ without rejecting. Suppose that, during an arbitrary clock cycle $t \in T$, $\mathcal{A}^{n,m}$ applies a transition relation, i.e., the corresponding $g \in \{g_\mathrm{start}, g_\mathrm{eval}, g_\mathrm{prech}\}$ is satisfied. Since satisfaction of g implies that all inputs and outputs must fulfill the same property and due to the definition of $\mathcal{A}^{n_i,m_i}$ for C_i, it holds that, for all C_i, g is also satisfied. Hence, it follows that if $\mathcal{A}^{n,m}$ accepts, all $\mathcal{A}^{n_i,m_i}$ for C_i also accept. Since all C_i are glitch-free and all $\mathcal{A}^{n_i,m_i}$ accept, it follows that $\forall C_i = (V_i, E_i), \forall e \in E_i \times T, G_e = 0$. Finally, for $C = (V, E)$ it holds that $\forall e \in E \times T, G_e = 0$ what completes the proof that the composed circuit is glitch-free. □

3.1 DRP-AND

We refer to the gadget that performs a single AND operation on two inputs in DRP logic, as defined in the context of Wave Dynamic Differential Logic (WDDL) [TV04], as DRP-AND. DRP-AND receives two dual-rail signals $\boldsymbol{a}, \boldsymbol{b}$ to compute $\boldsymbol{z} = \boldsymbol{a} \cdot \boldsymbol{b} = (a_\mathrm{t} \wedge b_\mathrm{t}, a_\mathrm{f} \vee b_\mathrm{f})$ by instantiating one AND and one OR gate.

Proposition 1. *DRP-AND is a correct DRP circuit for $z = a \cdot b$.*

Proof. Suppose that $(\boldsymbol{a}, \boldsymbol{b}) \in \{(0,1),(1,0)\}^2$. Then, $\boldsymbol{z} = (1,0) \iff \boldsymbol{a} = (1,0) \wedge \boldsymbol{b} = (1,0)$ and $\boldsymbol{z} = (0,1) \iff \boldsymbol{a} = (0,1) \vee \boldsymbol{b} = (0,1)$. Thus, we conclude that DRP-AND is a correct DRP circuit for $z = a \cdot b$. □

Proposition 2. *DRP-AND is glitch-free.*

Proof. Suppose that there are two arbitrary consecutive input states $(\boldsymbol{a}', \boldsymbol{b}')$ and $(\boldsymbol{a}, \boldsymbol{b})$ in a set of TDSs accepted by $\mathcal{A}^{2,1}$ and $(G_{a_\mathrm{t}}, G_{a_\mathrm{f}}, G_{b_\mathrm{t}}, G_{b_\mathrm{f}}) = (0,0,0,0)$. Substituting this into the equations for G_{z_t} (cf. Example 1) and G_{z_f} (cf. Example 2), we get the following:

$$G_{z_\mathrm{t}} = (a'_\mathrm{t} \wedge \overline{a_\mathrm{t}} \wedge \overline{b'_\mathrm{t}} \wedge b_\mathrm{t}) \vee (\overline{a'_\mathrm{t}} \wedge a_\mathrm{t} \wedge b'_\mathrm{t} \wedge \overline{b_\mathrm{t}}), \quad G_{z_\mathrm{f}} = (a'_\mathrm{f} \wedge \overline{a_\mathrm{f}} \wedge \overline{b'_\mathrm{f}} \wedge b_\mathrm{f}) \vee (\overline{a'_\mathrm{f}} \wedge a_\mathrm{f} \wedge b'_\mathrm{f} \wedge \overline{b_\mathrm{f}})$$

Further, suppose that $G_{z_\mathrm{t}} \vee G_{z_\mathrm{f}} = 1$. It must follow that $G_{z_\mathrm{t}} \vee G_{z_\mathrm{f}} = 1 \implies (a'_\mathrm{t} \neq a_\mathrm{t} \wedge a'_\mathrm{t} \neq b'_\mathrm{t} \wedge a'_\mathrm{t} = b_\mathrm{t}) \vee (a'_\mathrm{f} \neq a_\mathrm{f} \wedge a'_\mathrm{f} \neq b'_\mathrm{f} \wedge a'_\mathrm{f} = b_\mathrm{f})$, but this contradicts that $\mathcal{A}^{2,1}$ accepts. Therefore, it holds that $G_{z_\mathrm{t}} \wedge G_{z_\mathrm{f}} = 0$, which proves that DRP-AND is glitch-free. □

3.2 DRP-MUX

Figure 2 illustrates the construction of a multiplexer, termed DRP-MUX, which selects between two dual-rail input signals, $\boldsymbol{a}$ and $\boldsymbol{b}$, based on a dual-rail select signal $\boldsymbol{s}$.



(a) True rail. (b) False rail.

Figure 2: The gate representation of a DRP-MUX computing $\boldsymbol{z} = \text{DRP-MUX}_{\boldsymbol{s}}(\boldsymbol{a}, \boldsymbol{b})$.

Proposition 3. *DRP-MUX is a correct DRP circuit for* $z = \begin{cases} a, & \text{if } s = 1 \\ b, & \text{if } s = 0 \end{cases}$.

Proof. Since $z_t = \overline{\overline{(a_t \wedge s_t)} \wedge \overline{(b_t \wedge s_f)}}$ and $z_f = \overline{\overline{(a_f \wedge s_t)} \wedge \overline{(b_f \wedge s_f)}}$ it follows that $\boldsymbol{s} = (0,1) \implies \boldsymbol{z} = \boldsymbol{b}$, and $\boldsymbol{s} = (1,0) \implies \boldsymbol{z} = \boldsymbol{a}$. Thus, we conclude that DRP-MUX is a correct DRP circuit for $z = a$ if $s = 1$, and b if $s = 0$. □

Proposition 4. *DRP-MUX is glitch-free.*

Proof. Suppose two arbitrary consecutive input states $(\boldsymbol{a}', \boldsymbol{b}', \boldsymbol{s}')$ and $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{s})$ in a set of TDSs accepted by $\mathcal{A}^{3,1}$ and $(G_{a_t}, G_{a_f}, G_{b_t}, G_{b_f}, G_{s_t}, G_{s_f}) = (0,0,0,0,0,0)$. Substituting this into the equations for G_c to G_f (cf. Example 1), we get the following:

$$G_c = (a'_t \wedge \overline{a_t} \wedge \overline{s'_t} \wedge s_t) \vee (\overline{a'_t} \wedge a_t \wedge s'_t \wedge \overline{s_t}), \quad G_d = (s'_f \wedge \overline{s_f} \wedge \overline{b'_t} \wedge b_t) \vee (\overline{s'_f} \wedge s_f \wedge b'_t \wedge \overline{b_t}),$$
$$G_e = (a'_f \wedge \overline{a_f} \wedge \overline{s'_t} \wedge s_t) \vee (\overline{a'_f} \wedge a_f \wedge s'_t \wedge \overline{s_t}), \quad G_f = (s'_f \wedge \overline{s_f} \wedge \overline{b'_f} \wedge b_f) \vee (\overline{s'_f} \wedge s_f \wedge b'_f \wedge \overline{b_f})$$

Further, suppose that $G_c \vee G_d \vee G_e \vee G_f = 1$. It must follow that $G_c \vee G_d \vee G_e \vee G_f = 1 \implies (a'_t \neq a_t \wedge a'_t \neq s'_t \wedge a'_t = s_t) \vee (s'_f \neq s_f \wedge s'_f \neq b'_t \wedge s'_f = b_t) \vee (a'_f \neq a_f \wedge a'_f \neq s'_t \wedge a'_f = s_t) \vee (s'_f \neq s_f \wedge s'_f \neq b'_f \wedge s'_f = b_f)$ but this contradicts the fact that $\mathcal{A}^{3,1}$ accepts. Thus, it holds that $G_c \wedge G_d \wedge G_e \wedge G_f = 0$, which we substitute into the equations of G_{z_t} and G_{z_f}.

$$G_{z_t} = (c' \wedge \overline{c} \wedge \overline{d'} \wedge d) \vee (\overline{c'} \wedge c \wedge d' \wedge \overline{d}), \quad G_{z_f} = (e' \wedge \overline{e} \wedge \overline{f'} \wedge f) \vee (\overline{e'} \wedge e \wedge f' \wedge \overline{f})$$

Now, suppose that $G_{z_t} \vee G_{z_f} = 1$. It follows that $G_{z_t} \vee G_{z_f} = 1 \implies (c' \neq c \wedge c' \neq d' \wedge c' = d) \vee (e' \neq e \wedge e' \neq f' \wedge e' = f) \implies (s'_t = s_f = 1) \vee (s'_f = s_t = 1)$, but this again contradicts the fact that $\mathcal{A}^{3,1}$ accepts. Therefore, it holds that $G_{z_t} \wedge G_{z_f} = 0$, which proves that DRP-MUX is glitch-free. □

3.3 DRP-XOR

Similar to DRP-MUX, the XOR operation on the dual-rail representation of two input signals $\boldsymbol{a}$ and $\boldsymbol{b}$ can be performed using NAND gates, illustrated in Figure 3. We follow the same structure of argumentation for DRP-XOR as for DRP-MUX.



(a) True-rail. (b) False-rail.

Figure 3: The gate representation of a DRP-XOR computing $\boldsymbol{z} = \text{DRP-XOR}(\boldsymbol{a}, \boldsymbol{b})$.

Proposition 5. *DRP-XOR is a correct DRP circuit for* $z = a \oplus b$.

Proof. Since $(\boldsymbol{a},\boldsymbol{b}) \in \{(0,1),(1,0)\}^2 \implies ((\boldsymbol{z}=(1,0) \iff \boldsymbol{a}\neq\boldsymbol{b}) \wedge (\boldsymbol{z}=(0,1) \iff \boldsymbol{a}=\boldsymbol{b}))$, we conclude that DRP-XOR is a correct DRP circuit for $z = a \oplus b$. □

Proposition 6. *DRP-XOR is glitch-free.*

Proof. Suppose that there are two arbitrary consecutive input states $(\boldsymbol{a}',\boldsymbol{b}')$ and $(\boldsymbol{a},\boldsymbol{b})$ in a set of TDSs accepted by $\mathcal{A}^{2,1}$ and $(G_{a_\mathsf{t}}, G_{a_\mathsf{f}}, G_{b_\mathsf{t}}, G_{b_\mathsf{f}}) = (0,0,0,0)$. Substituting this into the equations for G_c to G_f (cf. Example 1), we get the following:

$$G_c = (a'_\mathsf{t} \wedge \overline{a_\mathsf{t}} \wedge \overline{b'_\mathsf{f}} \wedge b_\mathsf{f}) \vee (\overline{a'_\mathsf{t}} \wedge a_\mathsf{t} \wedge b'_\mathsf{f} \wedge \overline{b_\mathsf{f}}), \quad G_d = (a'_\mathsf{f} \wedge \overline{a_\mathsf{f}} \wedge \overline{b'_\mathsf{t}} \wedge b_\mathsf{t}) \vee (\overline{a'_\mathsf{f}} \wedge a_\mathsf{f} \wedge b'_\mathsf{t} \wedge \overline{b_\mathsf{t}})$$
$$G_e = (a'_\mathsf{t} \wedge \overline{a_\mathsf{t}} \wedge \overline{b'_\mathsf{t}} \wedge b_\mathsf{t}) \vee (\overline{a'_\mathsf{t}} \wedge a_\mathsf{t} \wedge b'_\mathsf{t} \wedge \overline{b_\mathsf{t}}), \quad G_f = (a'_\mathsf{f} \wedge \overline{a_\mathsf{f}} \wedge \overline{b'_\mathsf{f}} \wedge b_\mathsf{f}) \vee (\overline{a'_\mathsf{f}} \wedge a_\mathsf{f} \wedge b'_\mathsf{f} \wedge \overline{b_\mathsf{f}})$$

Further, suppose that $G_c \vee G_d \vee G_e \vee G_f = 1$. It must follow that $G_c \vee G_d \vee G_e \vee G_f = 1 \implies (a'_\mathsf{t} \neq a_\mathsf{t} \wedge a'_\mathsf{t} \neq b'_\mathsf{t} \wedge a'_\mathsf{t} = b_\mathsf{t}) \vee (a'_\mathsf{t} \neq a_\mathsf{t} \wedge a'_\mathsf{t} \neq b'_\mathsf{f} \wedge a'_\mathsf{t} = b_\mathsf{f}) \vee (a'_\mathsf{f} \neq a_\mathsf{f} \wedge a'_\mathsf{f} \neq b'_\mathsf{t} \wedge a'_\mathsf{f} = b_\mathsf{t}) \vee (a'_\mathsf{f} \neq a_\mathsf{f} \wedge a'_\mathsf{f} \neq b'_\mathsf{f} \wedge a'_\mathsf{f} = b_\mathsf{f})$ but this contradicts the fact that $\mathcal{A}^{2,1}$ accepts. Thus, $G_c \wedge G_d \wedge G_e \wedge G_f = 0$ which we substitute into the equations of G_{z_t} and G_{z_f}.

$$G_{z_\mathsf{t}} = (c' \wedge \overline{c} \wedge \overline{d'} \wedge d) \vee (\overline{c'} \wedge c \wedge d' \wedge \overline{d}), \quad G_{z_\mathsf{f}} = (e' \wedge \overline{e} \wedge \overline{f'} \wedge f) \vee (\overline{e'} \wedge e \wedge f' \wedge \overline{f})$$

Now, suppose that $G_{z_\mathsf{t}} \vee G_{z_\mathsf{f}} = 1$. It follows that $G_{z_\mathsf{t}} \vee G_{z_\mathsf{f}} = 1 \implies (c' \neq c \wedge c' \neq d' \wedge c' = d) \vee (e' \neq e \wedge e' \neq f' \wedge e' = f) \implies (a'_\mathsf{t} = a_\mathsf{f} = 1) \vee (a'_\mathsf{f} = a_\mathsf{t} = 1)$, but this again contradicts the fact that $\mathcal{A}^{2,1}$ accepts. Therefore, it holds that $G_{z_\mathsf{t}} \wedge G_{z_\mathsf{f}} = 0$, which proves that DRP-XOR is glitch-free. □

4 CCHPC

In the following, we introduce the Constant-Cycle Hardware Private Circuits (CCHPC) scheme. Each CCHPC gadget is constructed by $d+1$ layers, where the first layer operates on the single-rail representation of the first share inputs. Every layer $l \in [1,d]$ operates on the dual-rail representation of variables and alternately switches between pre-charge and evaluation phases, with the input values separated in the time domain for each layer. Note that DRP logic is not utilized for hiding purposes in our approach (and also in LMDPL), but rather to mitigate the propagation of glitches and signal delays in certain areas of the designed CCHPC gadgets. In particular, the approach benefits from DRP properties to achieve a low-latency masked circuit. For simplicity, $\widehat{\boldsymbol{X}}$ now indicates signals in CCHPC representation, where $\widehat{\boldsymbol{X}} = (\widehat{\boldsymbol{X}} \setminus \boldsymbol{x}^0) \cup x^0$. In other words, the share x^0 is the only share that is not in dual-rail form. A CCHPC circuit implements a Boolean function $\mathsf{f}(a_0,\ldots,a_{n-1},s_0\ldots,s_{p-1}) = (z_0,\ldots,z_{m-1})$, with n sensitive inputs, p public inputs, and m outputs. Every sensitive a_i with $i \in [0,n)$ is processed in the CCHPC representation $\widehat{\boldsymbol{A}}_i = \{a_i^0, \boldsymbol{a}_i^1, \ldots, \boldsymbol{a}_i^d\}$. Every public s_i with $i \in [0,p)$ follows the same representation through duplication at each layer. Specifically, $\widehat{\boldsymbol{S}}_i = \{s_i^0, \boldsymbol{s}_i^1, \ldots, \boldsymbol{s}_i^d\}$, where $s_i^0 = s_i$ and $\forall l \in [1,d], \boldsymbol{s}_i^l = (s_i, \overline{s_i})$. A CCHPC circuit may receive additional random bits, such as $\boldsymbol{R} = \{\boldsymbol{r}_0,\ldots,\boldsymbol{r}_{k-1}\}$, provided in dual-rail representation. Therefore, we express a CCHPC circuit by a Boolean function $\mathsf{h} : \mathbb{F}_2^{(n+p)(2d+1)+2k} \to \mathbb{F}_2^{m(2d+1)}$, where $\mathsf{h}(\widehat{\boldsymbol{A}}_0,\ldots,\widehat{\boldsymbol{A}}_{n-1},\widehat{\boldsymbol{S}}_0,\ldots,\widehat{\boldsymbol{S}}_{p-1},\boldsymbol{R}) = (\widehat{\boldsymbol{Z}}_0,\ldots,\widehat{\boldsymbol{Z}}_{m-1})$. To capture constraints w.r.t input and output signals, we define a constraint automaton $\mathcal{B}^{n,p,k,m} = (Q,\Sigma,\to,q_{\text{start}})$, where $Q = \{q_{\text{start}}, q_0,\ldots,q_{d+1}\}$, $\widehat{\Sigma} = \{\widehat{\boldsymbol{A}}_0,\ldots,\widehat{\boldsymbol{A}}_{n-1},\widehat{\boldsymbol{S}}_0,\ldots,\widehat{\boldsymbol{S}}_{p-1},\widehat{\boldsymbol{Z}}_0,\ldots,\widehat{\boldsymbol{Z}}_{m-1}\}$, and $\Sigma = \widehat{\Sigma} \cup \boldsymbol{R}$ over a global data domain, i.e., $\forall \widehat{\boldsymbol{V}} \in \widehat{\Sigma}, d_{v^0} \in \{0,1\}, \forall l \in [1,d], d_{\boldsymbol{v}^l} \in \{(0,0),(0,1),(1,0)\}$ and $\forall q \in [0,k), d_{\boldsymbol{r}_q} \in \{(0,0),(0,1),(1,0)\}$. We denote the subset of random bits given to layer l is $\boldsymbol{R}_l$ with $\boldsymbol{R} = \bigcup \boldsymbol{R}_l$, the subset of all primary signals from layer $l \in [1,d]$ is $\Sigma^l = \{\boldsymbol{a}_0^l,\ldots,\boldsymbol{a}_{n-1}^l,\boldsymbol{s}_0^l,\ldots,\boldsymbol{s}_{p-1}^l,\boldsymbol{R}_l,\boldsymbol{z}_0^l,\ldots,\boldsymbol{z}_{m-1}^l\}$, and $\Sigma^0 =$

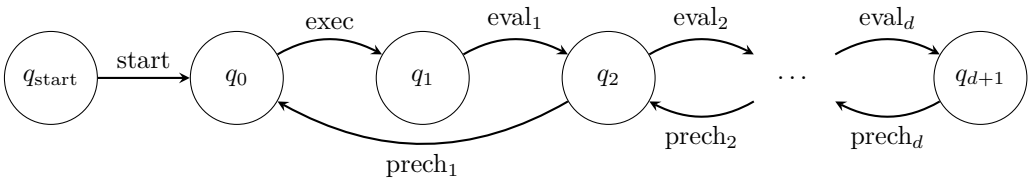


Figure 4: Constraint automaton transition relations for CCHPC primary signals.

$\{a^0_0, \dots a^0_{n-1}, s^0_0, \dots s^0_{p-1}, \boldsymbol{R}_0, z^0_0, \dots z^0_{n-1}\}$. Further, we specify the following transition relations in $\rightarrow = \{\rightarrow_{\text{start}}, \rightarrow_{\text{exec}}, \rightarrow_{\text{eval}_1}, \dots, \rightarrow_{\text{eval}_d}, \rightarrow_{\text{prech}_1}, \dots, \rightarrow_{\text{prech}_d}\}$.

$$
\begin{aligned}
& \rightarrow_{\text{start}} = q_{\text{start}} \xrightarrow{\Sigma, g_{\text{start}}} q_0, && g_{\text{start}} = \forall \boldsymbol{v} \in \Sigma \setminus \Sigma^0,\ d_{\boldsymbol{v}} = (0,0) \\
& \rightarrow_{\text{exec}} = q_0 \xrightarrow{\Sigma^0, g_{\text{exec}}} q_1, && g_{\text{exec}} = \forall \boldsymbol{r} \in \boldsymbol{R}_0,\ d_{\boldsymbol{r}} \neq (0,0) \\
\forall l \in [1,d], \quad & \rightarrow_{\text{eval}_l} = q_l \xrightarrow{\Sigma^l, g_{\text{eval}_l}} q_{l+1}, && g_{\text{eval}_l} = \forall \boldsymbol{v} \in \Sigma^l,\ d_{\boldsymbol{v}} \neq (0,0) \\
\forall l \in (1,d], \quad & \rightarrow_{\text{prech}_l} = q_{l+1} \xrightarrow{\Sigma^l, g_{\text{prech}_l}} q_l, && g_{\text{prech}_l} = \forall \boldsymbol{v} \in \Sigma^l,\ d_{\boldsymbol{v}} = (0,0) \\
& \rightarrow_{\text{prech}_1} = q_2 \xrightarrow{\Sigma^1, g_{\text{prech}_1}} q_0, && g_{\text{prech}_1} = \forall \boldsymbol{v} \in \Sigma^1,\ d_{\boldsymbol{v}} = (0,0)
\end{aligned}
$$

The transition relations of $\mathcal{B}^{n,p,k,m}$ are visualized in Figure 4, highlighting the valid sequences of states, triggered by changes of primary inputs accepted by the constraint automaton. Although the automaton mostly exhibits a generic structure, where each state q_l is reached either via $\rightarrow_{\text{eval}_{l-1}}$ from the preceding state q_{l-1} or via $\rightarrow_{\text{prech}_l}$ from the succeeding state q_{l+1}, the single-rail representation at layer $l = 0$ causes a deviation from this pattern. Specifically, it does not require a transition relation that encompasses precharging of that layer. Hence, the transition relation of q_0 to q_1 is captured by $\rightarrow_{\text{exec}}$, while $\rightarrow_{\text{prech}_1}$ from q_2 bypasses q_1 and directly transitions to state q_0. Based on the constraint automaton $\mathcal{B}^{n,p,k,m}$, we define the correctness for CCHPC circuits in Definition 9.

Definition 9 (Correctness for CCHPC circuits)**.** Let $\mathsf{f} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ with $\mathsf{f}(a_0, \dots, a_{n-1}) = (z_0, \dots, z_{m-1})$ be a Boolean function and C be a CCHPC circuit where $\mathsf{h} : \mathbb{F}_2^{(n+p)(2d+1)+2k} \rightarrow \mathbb{F}_2^{m(2d+1)}$ with $\mathsf{h}(\widehat{\boldsymbol{A}}_0, \dots, \widehat{\boldsymbol{A}}_{n-1}, \widehat{\boldsymbol{S}}_0, \dots, \widehat{\boldsymbol{S}}_{p-1}, \boldsymbol{r}_0, \dots, \boldsymbol{r}_{k-1}) = (\widehat{\boldsymbol{Z}}_0, \dots, \widehat{\boldsymbol{Z}}_{m-1})$. C is a correct CCHPC circuit for f iff $\forall (i,l,u,q) \in [0,n) \times [1,d] \times [0,p) \times [0,k), \boldsymbol{a}^l_i, \boldsymbol{s}^l_u, \boldsymbol{r}_q \in \{(0,1),(1,0)\} \implies \forall j \in [0,m), \forall l \in [1,d], z^l_{j_\mathsf{f}} = \overline{z^l_{j_\mathsf{t}}} \wedge f(a^0_0 \oplus (\bigoplus_{l=1}^{l \leq d} a^l_{0_\mathsf{t}}), \dots, a^0_{n-1} \oplus (\bigoplus_{l=1}^{l \leq d} a^l_{n-1_\mathsf{t}})) = ((z^0_0 \oplus (\bigoplus_{l=1}^{l \leq d} z^l_{0_\mathsf{t}})), \dots, (z^0_{m-1} \oplus (\bigoplus_{l=1}^{l \leq d} z^l_{m-1_\mathsf{t}})))$.

It follows that a CCHPC circuit that receives valid dual-rail inputs computes valid dual-rail outputs that fulfill functional correctness of the (unshared) variables. Now, contrary to the (entirely) glitch-free operating DRP components (cf. Definition 8), CCHPC circuits allow the occurrence of glitches in certain areas, as long as the absence of glitch propagation holds. We utilize $\mathcal{B}^{n,p,k,m}$ and the glitch-propagation function of the RR d-probing model for the according definition.

Definition 10 (Glitch-Propagation-Freeness)**.** Let C be a CCHPC circuit execution over a series of clock cycles in T that implements $\mathsf{h} : \mathbb{F}_2^{(n+p)(2d+1)+2k} \rightarrow \mathbb{F}_2^{m(2d+1)}$, where $\mathsf{h}(\widehat{\boldsymbol{A}}_0, \dots, \widehat{\boldsymbol{A}}_{n-1}, \widehat{\boldsymbol{S}}_0, \dots, \widehat{\boldsymbol{S}}_{p-1}, \boldsymbol{r}_0, \dots, \boldsymbol{r}_{k-1}) = (\widehat{\boldsymbol{Z}}_0, \dots, \widehat{\boldsymbol{Z}}_{m-1})$ and let $\mathcal{B}^{n,p,k,m}$ be the constraint automaton. Suppose that $\forall e \in \{a^l_{i_\mathsf{t}}, a^l_{i_\mathsf{f}} \mid \forall i \in [0,n), \forall l \in [1,d]\} \times T, G_e = 0$ and $\forall e \in \{s^l_{i_\mathsf{t}}, s^l_{i_\mathsf{f}} \mid \forall i \in [0,p), \forall l \in [1,d]\} \times T, G_e = 0$ and $\forall e \in \{r_{q_\mathsf{t}}, r_{q_\mathsf{f}} \mid \forall q \in [0,k)\} \times T, G_e = 0$. Then, C is glitch-propagation-free iff for any set of TDSs accepted by $\mathcal{B}^{n,p,k,m}$ it holds that $\forall e \in \{z^l_{j_\mathsf{t}}, z^l_{j_\mathsf{f}} \mid \forall j \in [0,m), \forall l \in [1,d]\} \times T, G_e = 0$.

Building on Definition 10, we prove that arbitrary compositions of glitch-propagation-free CCHPC circuits remain glitch-propagation-free. This enables us to then further derive the CCHPC d-PINI definition.

Lemma 2. *A CCHPC circuit execution C on n sensitive inputs, p public inputs, m outputs, k random bits, and the set of clock cycles T composed of glitch-propagation-free CCHPC circuits C_i with n_i sensitive inputs, p_i public inputs, m_i outputs, and k_i random bits is glitch-propagation-free for any set of TDSs accepted by $\mathcal{B}^{n,p,k,m}$.*

Proof. Suppose an arbitrary set of TDSs for C over T, which $\mathcal{B}^{n,p,k,m}$ accepts. Then, it follows that $\mathcal{B}^{n,p,k,m}$ applies a series of transition relations from $\rightarrow$ without rejecting. Suppose that, during an arbitrary clock cycle $t \in T$, $\mathcal{B}^{n,p,k,m}$ applies a transition relation, i.e., the corresponding $g \in \{g_{\text{start}}, g_{\text{exec}}, g_{\text{eval}_1}, \ldots, g_{\text{eval}_d}, g_{\text{prech}_1}, \ldots, g_{\text{prech}_d}\}$ is satisfied. Since satisfaction of g implies that all inputs and outputs must fulfill the same property and due to the definition of $\mathcal{B}^{n_i,p_i,k_i,m_i}$ for C_i, it holds that, for all C_i, g is also satisfied. Hence, it follows that if $\mathcal{B}^{n,p,k,m}$ accepts, all $\mathcal{B}^{n_i,p_i,k_i,m_i}$ for C_i also accept. Since all C_i are glitch-propagation-free and all $\mathcal{B}^{n_i,p_i,k_i,m_i}$ accept, it follows that for all output signals e of any C_i it holds that $G_e = 0$. This implies that $G_e = 0$ for all output signals e of C what completes the proof of glitch-propagation-freenes for the composed circuit. □

Definition 11 (CCHPC d-PINI)**.** Let C be a CCHPC circuit execution over a set of clock cycles T that implements the Boolean function $\mathsf{h} : \mathbb{F}_2^{(n+p)(2d+1)+2k} \rightarrow \mathbb{F}_2^{m(2d+1)}$, where $\mathsf{h}(\widehat{\boldsymbol{A}}_0, \ldots, \widehat{\boldsymbol{A}}_{n-1}, \widehat{\boldsymbol{S}}_0, \ldots, \widehat{\boldsymbol{S}}_{p-1}, \boldsymbol{r}_0, \ldots, \boldsymbol{r}_{k-1}) = (\widehat{\boldsymbol{Z}}_0, \ldots, \widehat{\boldsymbol{Z}}_{m-1})$ and let $\mathcal{B}^{n,p,k,m}$ be the constraint automaton. Then, C is CCHPC d-PINI for any set of of TDSs accepted by $\mathcal{B}^{n,p,k,m}$ iff C is glitch-propagation-free and C is RR d-PINI for any set of of TDSs accepted by $\mathcal{B}^{n,p,k,m}$.

4.1 CCHPC Gadgets

Based on the DRP circuits introduced in Section 3, we construct different CCHPC gadget types below. We show correctness, fulfillment of glitch-propagation-freeness (cf. Definition 10), and security satisfaction of the PINI notion under the RR d-probing model.

4.1.1 CCHPC-NOT Gadget

We define the CCHPC-NOT gadget suitable for the CCHPC representation of variables, i.e., the share a^0 of a sharing $\widehat{\boldsymbol{A}}$ is in single-rail, while the remaining d shares are available in their dual-rail form. The dual-rail state of a single bit variable is inverted without any overhead by simply swapping the true and false rails. For a shared representation, it is sufficient to invert only one share. Therefore, $\widehat{\boldsymbol{Z}} = \mathsf{CCHPC\text{-}NOT}(\widehat{\boldsymbol{A}})$ computes as follows.

$$z^0 = a^0, \qquad \boldsymbol{z}^l = \boldsymbol{a}^l, \ \forall\, l \in [1, d), \qquad \boldsymbol{z}^d = (a_{\text{f}}^d, a_{\text{t}}^d)$$

We chose to invert the last share without a specific reason. The properties can be proven analogously, even if a different share is inverted.

Proposition 7. *CCHPC-NOT is a correct CCHPC circuit for $z = \overline{a}$.*

Proof. Suppose that $\forall l \in [1, d], \boldsymbol{a}^l \in \{(0,1),(1,0)\}$. Then, $\forall l \in [1, d], \boldsymbol{a}^l \in \{(0,1),(1,0)\} \implies a_{\text{f}}^l = \overline{a_{\text{t}}^l} \implies y_{\text{f}}^l = \overline{y_{\text{t}}^l}$. Further, it follows that $z^0 \oplus (\bigoplus_{l=1}^{l \leq d} z_{\text{t}}^l) = a^0 \oplus (\bigoplus_{l=1}^{l<d} a_{\text{t}}^l) \oplus a_{\text{f}}^d = a^0 \oplus (\bigoplus_{l=1}^{l \leq d} a_{\text{t}}^l)$ concluding that CCHPC-NOT is a correct CCHPC circuit for $z = \overline{a}$. □

Proposition 8. *CCHPC-NOT is glitch-propagation-free.*

Proof. Suppose that $\forall l \in [1,d], G_{a^l_\mathrm{t}} = G_{a^l_\mathrm{f}} = 0$. Since, $\forall l \in [1,d), \boldsymbol{z}^l = \boldsymbol{a}^l$ and $\boldsymbol{z}^d = (a^d_\mathrm{f}, a^d_\mathrm{t})$ it holds that $\forall l \in [1,d], G_{a^l_\mathrm{t}} = G_{a^l_\mathrm{f}} = 0 \implies G_{z^l_\mathrm{t}} = G_{z^l_\mathrm{f}} = 0$ which proves that CCHPC-NOT is glitch-propagation-free for all sets of TDSs accepted by $\mathcal{B}^{1,0,0,1}$. □

Proposition 9. *CCHPC-NOT with $d+1$ shares is CCHPC d-PINI.*

Proof. Suppose that CCHPC-NOT is glitch-propagation-free for all sets of TDSs accepted by $\mathcal{B}^{1,0,0,1}$ (cf. Proposition 8). As CCHPC-NOT is a share-isolating gadget, it follows that CCHPC-NOT is RR d-PINI for all sets of TDSs accepted by $\mathcal{B}^{1,0,0,1}$. Therefore, CCHPC-NOT with $d+1$ shares is CCHPC d-PINI. □

4.1.2 Linear CCHPC Gadgets

We define linear CCHPC gadgets, inspired by LMDPL [SBHM20] but utilizing NANDs, capable of computing the XOR and XNOR operations on $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$. CCHPC-XOR applies DRP-XOR on each dual-rail share separately, while a single XOR is sufficient for a^0 and b^0. Formally, CCHPC-XOR operates as follows.

$$z^0 = a^0 \oplus b^0, \qquad \boldsymbol{z}^l = \mathsf{DRP\text{-}XOR}(\boldsymbol{a}^l, \boldsymbol{b}^l),\ \forall\, l \in [1,d]$$

CCHPC-XNOR combines CCHPC-XOR and CCHPC-NOT gadgets as shown below.

$$\widehat{\boldsymbol{Z}} = \mathsf{CCHPC\text{-}XNOR}(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}) = \mathsf{CCHPC\text{-}NOT}(\mathsf{CCHPC\text{-}XOR}(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}))$$

Proposition 10. *CCHPC-XOR (resp. CCHPC-XNOR) is a correct CCHPC circuit for $z = a \oplus b$ (resp. $z = \overline{a \oplus b}$).*

Proof. Since $\forall l \in [1,d], (\boldsymbol{a}^l, \boldsymbol{b}^l) = \{(0,1),(1,0)\}^2 \implies ((\boldsymbol{z}^l = (0,1) \iff \boldsymbol{a}^l = \boldsymbol{b}^l) \wedge (\boldsymbol{z}^l = (1,0) \iff \boldsymbol{a}^l \neq \boldsymbol{b}^l))$, it follows that $z^l_\mathrm{t} = a^l_\mathrm{t} \oplus b^l_\mathrm{t}$ and $z^l_\mathrm{f} = \overline{z^l_\mathrm{t}}$. Therefore, $z^0 \oplus (\bigoplus_{l=1}^{l\leq d} z^l_\mathrm{t}) = a^0 \oplus b^0 \oplus (\bigoplus_{l=1}^{l<d} a^l_\mathrm{t} \oplus b^l_\mathrm{t})$ concluding correctness of CCHPC-XOR for $z = a \oplus b$. The correctness of CCHPC-XNOR follows from composing CCHPC-NOT and CCHPC-XOR. □

Proposition 11. *CCHPC-XOR (resp. CCHPC-XNOR) is glitch-propagation-free.*

Proof. Suppose that $\forall l \in [1,d], G_{a^l_\mathrm{t}} = G_{a^l_\mathrm{f}} = G_{b^l_\mathrm{t}} = G_{b^l_\mathrm{f}} = 0$. Further, suppose an arbitrary set of TDSs accepted by $\mathcal{B}^{2,0,0,1}$. Then, each layer l receives a set of TDSs for $(\boldsymbol{a}^l, \boldsymbol{b}^l)$ such that a constraint automaton $\mathcal{A}^{2,1}$ accepts. From the proven Proposition 6 it follows that $\forall l \in [1,d], G_{a^l_\mathrm{t}} = G_{a^l_\mathrm{f}} = G_{b^l_\mathrm{t}} = G_{b^l_\mathrm{f}} = 0 \implies G_{z^l_\mathrm{t}} = G_{z^l_\mathrm{f}} = 0$, which proves that CCHPC-XOR is glitch-propagation-free for all sets of TDSs accepted by $\mathcal{B}^{2,0,0,1}$. The glitch-propagation-freeness of CCHPC-XNOR follows from composing CCHPC-NOT and CCHPC-XOR. □

Proposition 12. *CCHPC-XOR (resp. CCHPC-XNOR) with $d+1$ shares is CCHPC d-PINI.*

Proof. Suppose that CCHPC-XOR is glitch-propagation-free for all sets of TDSs accepted by $\mathcal{B}^{2,0,0,1}$ (cf. Proposition 11). As CCHPC-XOR is a share-isolating gadget, it follows that CCHPC-XOR is RR d-PINI for all sets of TDSs accepted by $\mathcal{B}^{2,0,0,1}$. Therefore, CCHPC-NOT with $d+1$ shares is CCHPC d-PINI. CCHPC d-PINI of CCHPC-XNOR follows from composing CCHPC-NOT and CCHPC-XOR. □

CCHPC-MUX Gadget. Iterative, e.g. round-based, designs often face the utilization of multiplexers, where the select signal is not sensitive (e.g. for the AES last round selection). The multiplexer gadget CCHPC-MUX, inspired by LMDPL [SBHM20] but utilizing NANDs, is implemented as follows, where $\widehat{\boldsymbol{S}}$ denotes the unmasked select signal in CCHPC representation.

$$z^0 = b^0 \oplus \left(s^0 \wedge \left(a^0 \oplus b^0\right)\right), \qquad \boldsymbol{z}^l = \mathsf{DRP\text{-}MUX}_{\boldsymbol{s}^l}(\boldsymbol{a}^l, \boldsymbol{b}^l),\ \forall\, l \in [1,d]$$

Proposition 13. *CCHPC-MUX is a correct CCHPC circuit for* $z = \begin{cases} a, & \text{if } s = 1 \\ b, & \text{if } s = 0 \end{cases}$.

Proof. From Proposition 3 follows $\forall l \in [1,d], (\boldsymbol{a}^l, \boldsymbol{b}^l, \boldsymbol{s}^l) = \{(0,1),(1,0)\}^3 \implies z_{\mathrm{f}}^l = \overline{z_{\mathrm{t}}^l}$. Further, since $\forall l \in [1,d], (\boldsymbol{a}^l, \boldsymbol{b}^l) = \{(0,1),(1,0)\}^2 \implies (\boldsymbol{s}^l = (1,0) \implies \boldsymbol{z}^l = \boldsymbol{a}^l) \wedge (\boldsymbol{s}^l = (0,1) \implies \boldsymbol{z}^l = \boldsymbol{b}^l)$. Since $s^0 = s_{\mathrm{t}}^1 = \ldots = s_{\mathrm{t}}^d$, it either holds that $s^0 = 1 \wedge \boldsymbol{s}^1 = \ldots = \boldsymbol{s}^d = (1,0) \implies z^0 \oplus (\bigoplus_{l=1}^{l\leq d} z_{\mathrm{t}}^l) = a^0 \oplus (\bigoplus_{l=1}^{l\leq d} a_{\mathrm{t}}^l)$ or $s^0 = 0 \wedge \boldsymbol{s}^1 = \ldots = \boldsymbol{s}^d = (0,1) \implies z^0 \oplus (\bigoplus_{l=1}^{l\leq d} z_{\mathrm{t}}^l) = b^0 \oplus (\bigoplus_{l=1}^{l\leq d} b_{\mathrm{t}}^l)$, concluding correctness of CCHPC-MUX for $z = a$ if $s = 1$, and b if $s = 0$. □

Proposition 14. *CCHPC-MUX is glitch-propagation-free.*

Proof. Suppose that $\forall l \in [1,d], G_{a_{\mathrm{t}}^l} = G_{a_{\mathrm{f}}^l} = G_{b_{\mathrm{t}}^l} = G_{b_{\mathrm{f}}^l} = G_{s_{\mathrm{t}}^l} = G_{s_{\mathrm{f}}^l} = 0$. Further, suppose an arbitrary set of TDSs accepted by $\mathcal{B}^{2,1,0,1}$. It holds that if $\mathcal{B}^{2,1,0,1}$ accepts the set of TDSs then $\mathcal{A}^{3,1}$ accepts the set of TDSs for all $(\boldsymbol{a}^l, \boldsymbol{b}^l, \boldsymbol{s}^l)$ with $l \in [1,d]$. Since Proposition 4 has already been proven, it holds that $\forall l \in [1,d], G_{a_{\mathrm{t}}^l} = G_{a_{\mathrm{f}}^l} = G_{b_{\mathrm{t}}^l} = G_{b_{\mathrm{f}}^l} = G_{s_{\mathrm{t}}^l} = G_{s_{\mathrm{f}}^l} = 0 \implies G_{z_{\mathrm{t}}^l} = G_{z_{\mathrm{f}}^l} = 0$, which proves that CCHPC-MUX is glitch-propagation-free for all sets of TDSs accepted by $\mathcal{B}^{2,1,0,1}$. □

Proposition 15. *CCHPC-MUX with $d+1$ shares is CCHPC d-PINI.*

Proof. Suppose that CCHPC-MUX is glitch-propagation-free for all sets of TDSs accepted by $\mathcal{B}^{2,1,0,1}$ (cf. Proposition 14). As CCHPC-MUX is a share-isolating gadget, it follows that CCHPC-MUX is RR d-PINI for all sets of TDSs accepted by $\mathcal{B}^{2,1,0,1}$. Therefore, CCHPC-MUX with $d+1$ shares is CCHPC d-PINI. □

4.1.3 Non-linear CCHPC Gadgets

We first define the non-linear CCHPC gadget type, called CCHPC-AND, capable of computing the AND operation on $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$. To facilitate further proofs, we first address the CCHPC-AND gadget for $d = 1$, and then generalize the construction to support multiple types of non-linear gadgets for arbitrary security orders.

First-order CCHPC-AND. The first-order gadget pre-processes $\boldsymbol{p}_{(0,1,0)}, \ldots, \boldsymbol{p}_{(3,1,0)}$, i.e., four intermediates in dual-rail representation, using three (single-rail) inverters. The original rail computation of $p_{(0,1,0)_{\mathrm{t}}}, \ldots, p_{(3,1,0)_{\mathrm{t}}}$ is defined as follows, while $\boldsymbol{r}_0$ denotes a uniformly distributed random bit.

$$p_{(0,1,0)_{\mathrm{t}}} = r_{0_{\mathrm{t}}} \qquad p_{(1,1,0)_{\mathrm{t}}} = a^0 \oplus r_{0_{\mathrm{t}}} \qquad p_{(2,1,0)_{\mathrm{t}}} = b^0 \oplus r_{0_{\mathrm{t}}} \qquad p_{(3,1,0)_{\mathrm{t}}} = a^0 \oplus b^0 \oplus r_{0_{\mathrm{t}}}$$

The intermediate values $\boldsymbol{p}_{(0,1,0)}, \ldots, \boldsymbol{p}_{(3,1,0)}$ are passed into a two-stage DRP-MUX, where the correct intermediate is selected based on $(\boldsymbol{a}^1, \boldsymbol{b}^1)$. Initially, $(\boldsymbol{a}^1, \boldsymbol{b}^1)$ are set to $(0,0)$, effectively blocking signal propagation through the multiplexers. In a subsequent clock

cycle, $(\boldsymbol{a}^1, \boldsymbol{b}^1)$ evaluate and the actual selection is applied. Finally, the first and second output share generations can be expressed as follows.

$$z^0 = a^0 b^0 \oplus r_0, \quad \boldsymbol{z}^1 = \begin{cases} (0,0), \text{ if } \boldsymbol{a}^1 = (0,0) \vee \boldsymbol{b}^1 = (0,0) \quad \text{(cf. proof of Proposition 3)} \\ \mathsf{DRP\text{-}XOR}(\mathsf{DRP\text{-}AND}(\boldsymbol{a}^1, \boldsymbol{b}^1), \boldsymbol{p}_{(3,1,0)}), \text{ if } \boldsymbol{a}^1 = (1,0) \wedge \boldsymbol{b}^1 = (1,0) \\ \mathsf{DRP\text{-}XOR}(\mathsf{DRP\text{-}AND}(\boldsymbol{a}^1, \boldsymbol{b}^1), \boldsymbol{p}_{(2,1,0)}), \text{ if } \boldsymbol{a}^1 = (1,0) \wedge \boldsymbol{b}^1 = (0,1) \\ \mathsf{DRP\text{-}XOR}(\mathsf{DRP\text{-}AND}(\boldsymbol{a}^1, \boldsymbol{b}^1), \boldsymbol{p}_{(1,1,0)}), \text{ if } \boldsymbol{a}^1 = (0,1) \wedge \boldsymbol{b}^1 = (1,0) \\ \mathsf{DRP\text{-}XOR}(\mathsf{DRP\text{-}AND}(\boldsymbol{a}^1, \boldsymbol{b}^1), \boldsymbol{p}_{(0,1,0)}), \text{ if } \boldsymbol{a}^1 = (0,1) \wedge \boldsymbol{b}^1 = (0,1) \end{cases}$$

Conceptually, the first-order gadget construction can be separated into two layers $l \in [0,1]$, where each layer generates the corresponding output share z^l. Similarly, each layer processes input shares whose indices are also equal to the layer index, i.e., layer $l = 0$

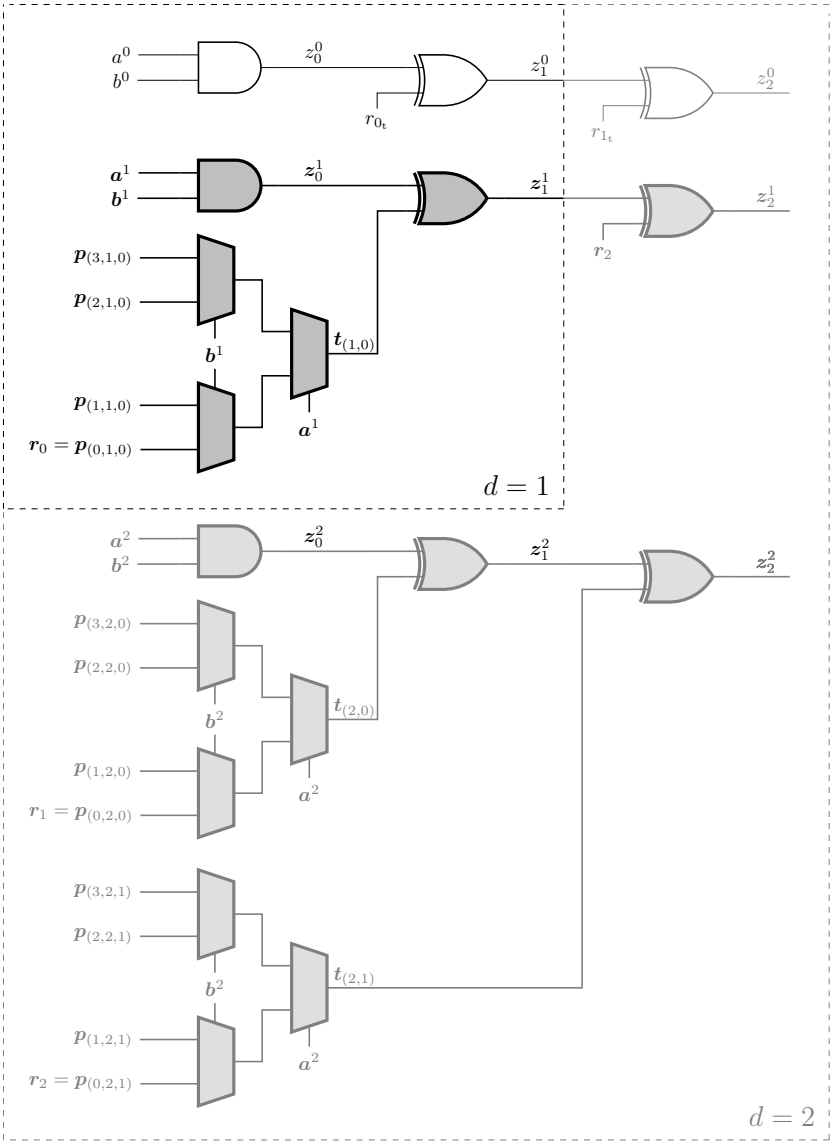


Figure 5: Non-linear CCHPC gadget for security order $d = 1$ (resp. $d = 2$). Pre-processed $\boldsymbol{p}_{(i,l,k)}$ in layer l with inputs from layer k (cf. Algorithm 2). DRP logic in gray and **bold**.

processes a^0 and b^0, while layer $l = 1$ processes a^1 and b^1. Additionally, layer $l = 1$ pre-processes intermediates $\boldsymbol{p}_{(0,1,0)}, \ldots, \boldsymbol{p}_{(3,1,0)}$ based on a^0 and b^0. We express this by the introduction of two additional indices l and k for the pre-processed intermediates. The index l denotes the destination layer to which the pre-processed intermediates are provided, while k indicates the source layer of the pre-processing inputs. In the first-order design, $\boldsymbol{p}_{(0,k,l)}, \ldots, \boldsymbol{p}_{(3,k,l)}$ is fixed to $\boldsymbol{p}_{(0,1,0)}, \ldots, \boldsymbol{p}_{(3,1,0)}$. For consistency, we assume that every random bit is available in dual-rail form and always refer to it as such. To avoid introducing additional variable names, we reuse the output shares as intermediates in higher-order gadget constructions. An additional index in the subscript of the output shares indicates the security order d it provides, i.e., for $d = 1$, z_1^0 and $\boldsymbol{z}_1^1$ correspond to the output shares. The first-order (resp. second-order) gadget construction is shown in Figure 5.

Algorithm 2 CCHPC-AND gadget with $d + 1$ shares

Parameters: d, config ▷ Security order, Configuration $\in \{\texttt{default}, \texttt{RS}, \texttt{PL}\}$
Inputs: $\widehat{\boldsymbol{A}}$, $\widehat{\boldsymbol{B}}$, $\boldsymbol{R}$ with $|\boldsymbol{R}| = \frac{d(d+1)}{2}$ ▷ $\boldsymbol{R}$ is fresh randomness
Output: $\widehat{\boldsymbol{Z}} = \{z_d^0, \boldsymbol{z}_d^1, \ldots, \boldsymbol{z}_d^d\}$

$z_0^0 \leftarrow a^0 \wedge b^0$
for $i = 0$ to $d - 1$ **do**
  $z_{i+1}^0 \leftarrow z_i^0 \oplus r_{\left(\frac{i(i+1)}{2}\right)_\mathrm{t}}$
end for

for $l = 1$ to d **do** ▷ Generate consecutive layer l
  $\boldsymbol{z}_0^l \leftarrow \mathsf{DRP\text{-}AND}(\boldsymbol{a}^l, \boldsymbol{b}^l)$
  for $k = 0$ to $l - 1$ **do** ▷ Pre-process from source layer k
    $\boldsymbol{p}_{0,l,k} \leftarrow \boldsymbol{r}_{\left(\frac{l(l-1)}{2}+k\right)}$
    $p_{(1,l,k)_\mathrm{t}} \leftarrow a_\mathrm{t}^k \oplus r_{\left(\frac{l(l-1)}{2}+k\right)_\mathrm{t}}$ ▷ if $k = 0$: $a_\mathrm{t}^k = a^k$
    $p_{(1,l,k)_\mathrm{f}} \leftarrow \overline{p_{(1,l,k)_\mathrm{t}}}$
    $p_{(2,l,k)_\mathrm{t}} \leftarrow b_\mathrm{t}^k \oplus r_{\left(\frac{l(l-1)}{2}+k\right)_\mathrm{t}}$ ▷ if $k = 0$: $b_\mathrm{t}^k = b^k$
    $p_{(2,l,k)_\mathrm{f}} \leftarrow \overline{p_{(2,l,k)_\mathrm{t}}}$
    $p_{(3,l,k)_\mathrm{t}} \leftarrow a_\mathrm{t}^k \oplus b_\mathrm{t}^k \oplus r_{\left(\frac{l(l-1)}{2}+k\right)_\mathrm{t}}$ ▷ if $k = 0$: $b_\mathrm{t}^k = b^k$
    $p_{(3,l,k)_\mathrm{f}} \leftarrow \overline{p_{(3,l,k)_\mathrm{t}}}$
    if config is `RS` **then** ▷ Insert $\mathrm{CCHPC_{RS}}$ register stage
      for $i = 0$ to 3 **do**
        $\boldsymbol{p}_{(i,l,k)} \leftarrow \mathsf{REG}(\boldsymbol{p}_{(i,l,k)})$
      end for
    else if config is `PL` **then** ▷ Insert $\mathrm{CCHPC_{PL}}$ register chain of depth $l - k$
      for $i = 0$ to 3 **do**
        $\boldsymbol{p}_{(i,l,k)} \leftarrow \mathsf{REG\text{-}PIP}_{(l-k)}(\boldsymbol{p}_{(i,l,k)})$
      end for
    end if
    $\boldsymbol{t}_{(l,k)} \leftarrow \mathsf{DRP\text{-}MUX}_{\boldsymbol{a}^l}(\mathsf{DRP\text{-}MUX}_{\boldsymbol{b}^l}(\boldsymbol{p}_{(3,l,k)}, \boldsymbol{p}_{(2,l,k)}), \mathsf{DRP\text{-}MUX}_{\boldsymbol{b}^l}(\boldsymbol{p}_{(1,l,k)}, \boldsymbol{p}_{(0,l,k)}))$
    $\boldsymbol{z}_{k+1}^l \leftarrow \mathsf{DRP\text{-}XOR}(\boldsymbol{z}_k^l, \boldsymbol{t}_{(l,k)})$
  end for
  for $i = l$ to $d - 1$ **do**
    $\boldsymbol{z}_{i+1}^l \leftarrow \mathsf{DRP\text{-}XOR}\left(\boldsymbol{z}_i^l, \boldsymbol{r}_{\left(\frac{i(i+1)}{2}+l\right)}\right)$
  end for
end for

General Construction. Based on the introduced indices, we extend the first-order gadget design to the generic CCHPC-AND gadget definition for arbitrary order d in Algorithm 2. A configuration parameter is set to `default`, but supports the advanced options `RS` and `PL`, which can reduce latency and enable gadget utilization in more complex architectures, i.e., round-based designs, discussed further in Section 4.2. The algorithm appends each layer l in ascending order up to layer $l = d$ to the construction and assigns pre-processed intermediate variables based on inputs from previously generated layers $k \in [0, l-1]$ within the current layer generation. As for $d = 1$, the pre-processed values are then 'selected' by DRP-MUX components in each layer (cf. Line 24). The selection signals correspond to the dual-rail shares whose index equals the layer generation index in progress. Note that the pre-processing in every layer is realized in single-rail logic to reduce the gadget's overhead. This is possible as the processing of the shares with index $l > 0$ is realized in DRP logic. Thus, the shares with index l keep the null state of layer l intact until its evaluation starts, with layers executed sequentially in ascending order across consecutive clock cycles. Further, necessary fresh randomness is added for each order increment to each output share, building a DRP-XOR chain of depth $d - l$, which finalizes the output share l in layer l. (cf. Lines 2 to 4 and 27 to 29).

Additionally, we define more non-linear CCHPC gadget types capable of computing the NAND, NOR, and OR operations by inverting the inputs and/or output of a CCHPC-AND gadget as follows.

$$\widehat{\boldsymbol{Z}} = \mathsf{CCHPC\text{-}NAND}(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}) = \mathsf{CCHPC\text{-}NOT}(\mathsf{CCHPC\text{-}AND}(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}))$$

$$\widehat{\boldsymbol{Z}} = \mathsf{CCHPC\text{-}NOR}(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}) = \mathsf{CCHPC\text{-}AND}(\mathsf{CCHPC\text{-}NOT}(\widehat{\boldsymbol{A}}), \mathsf{CCHPC\text{-}NOT}(\widehat{\boldsymbol{B}}))$$

$$\widehat{\boldsymbol{Z}} = \mathsf{CCHPC\text{-}OR}(\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}) = \mathsf{CCHPC\text{-}NOT}(\mathsf{CCHPC\text{-}AND}(\mathsf{CCHPC\text{-}NOT}(\widehat{\boldsymbol{A}}), \mathsf{CCHPC\text{-}NOT}(\widehat{\boldsymbol{B}})))$$

Proposition 16. *CCHPC-AND is a correct CCHPC circuit for $z = a \cdot b$.*

Proof. We prove Proposition 16 by induction on d.

Base Case: We show that Proposition 16 holds for $d = 1$.

Suppose a CCHPC-AND gadget for $d = 1$, as depicted in Figure 5. We reiterate that $z_1^0 = a^0 b^0 \oplus r_{0_t}$ (cf. Lines 1-4 of Algorithm 2), $z_{0_t}^1 = a_t^1 b_t^1$ (cf. Line 6 of Algorithm 2), and the following pre-processed variables:

$$p_{(0,1,0)_t} = r_{0_t} \qquad p_{(1,1,0)_t} = a^0 \oplus r_{0_t} \qquad p_{(2,1,0)_t} = b^0 \oplus r_{0_t} \qquad p_{(3,1,0)_t} = a^0 \oplus b^0 \oplus r_{0_t}$$

From Proposition 5 and Proposition 3, it follows that the computation in Line 25 can be simplified.

$$z_1^0 \oplus z_{1_t}^1 = \begin{cases} z_1^0 \oplus z_{0_t}^1 \oplus p_{(3,1,0)_t} = \overline{a^0 b^0 \oplus a^0 \oplus b^0}, & \text{if } \boldsymbol{a}^1 = (1,0) \wedge \boldsymbol{b}^1 = (1,0) \\ z_1^0 \oplus z_{0_t}^1 \oplus p_{(2,1,0)_t} = a^0 b^0 \oplus b^0, & \text{if } \boldsymbol{a}^1 = (1,0) \wedge \boldsymbol{b}^1 = (0,1) \\ z_1^0 \oplus z_{0_t}^1 \oplus p_{(1,1,0)_t} = a^0 b^0 \oplus a^0, & \text{if } \boldsymbol{a}^1 = (0,1) \wedge \boldsymbol{b}^1 = (1,0) \\ z_1^0 \oplus z_{0_t}^1 \oplus p_{(0,1,0)_t} = a^0 b^0, & \text{if } \boldsymbol{a}^1 = (0,1) \wedge \boldsymbol{b}^1 = (0,1) \end{cases}$$

From this, we conclude that $z_1^0 \oplus z_{1_t}^1 = a^0 b^0 \oplus a^0 b_t^1 \oplus a_t^1 b^0 \oplus a_t^1 b_t^1$ and by the same procedure for inverted rails, it holds that $z_{1_f}^1 = \overline{z_{1_t}^1}$ thereby proving correctness.

Induction: Suppose that Proposition 16, holds for d. We prove that it also holds for $d+1$ with $d > 0$. To facilitate the proof, we focus on the evaluation of $z_{(d+1)_t}^{d+1}$ while $z_{(d+1)_f}^{d+1}$ behaves analogously. By the inductive hypothesis, the following equation holds.

$$z_d^0 \oplus \bigoplus_{l=1}^{d} z_{d_t}^l = a^0 b^0 \oplus \bigoplus_{i=1}^{d} (a_t^i b^0) \oplus \bigoplus_{i=1}^{d} (a^0 b_t^i) \oplus \bigoplus_{i=1}^{d} \bigoplus_{j=1}^{d} (a_t^i b_t^j)$$

According to Line 28 and by Proposition 5, it holds that:

$$z^0_{d+1} = z^0_d \oplus r_{\left(\frac{d(d+1)}{2}\right)_\mathrm{t}}, \quad z^i_{(d+1)_\mathrm{t}} = z^i_{d_\mathrm{t}} \oplus r_{\left(\frac{d(d+1)}{2}+i\right)_\mathrm{t}}, \ \forall\, i \in [1,d]$$

Additionally, for $d+1$ an extra share, $\boldsymbol{z}^{d+1}_{d+1}$, is required. Referring to Lines 8-14, the pre-processed variables to calculate $\boldsymbol{z}^{d+1}_{d+1}$ are as follows.

$$\begin{aligned}
p_{(3,d+1,k=0)_\mathrm{t}} &= a^0 \oplus b^0 \oplus r_{\left(\frac{d(d+1)}{2}\right)_\mathrm{t}} & p_{(3,d+1,k>0)_\mathrm{t}} &= a^k_\mathrm{t} \oplus b^k_\mathrm{t} \oplus r_{\left(\frac{d(d+1)}{2}+k\right)_\mathrm{t}} \\
p_{(2,d+1,k=0)_\mathrm{t}} &= b^0 \oplus r_{\left(\frac{d(d+1)}{2}\right)_\mathrm{t}} & p_{(2,d+1,k>0)_\mathrm{t}} &= b^k_\mathrm{t} \oplus r_{\left(\frac{d(d+1)}{2}+k\right)_\mathrm{t}} \\
p_{(1,d+1,k=0)_\mathrm{t}} &= a^0 \oplus r_{\left(\frac{d(d+1)}{2}\right)_\mathrm{t}} & p_{(1,d+1,k>0)_\mathrm{t}} &= a^k_\mathrm{t} \oplus r_{\left(\frac{d(d+1)}{2}+k\right)_\mathrm{t}} \\
p_{(0,d+1,k)_\mathrm{t}} &= r_{\left(\frac{d(d+1)}{2}+k\right)_\mathrm{t}}
\end{aligned}$$

As stated in Lines 6, 24, and 25, it holds that $z^{d+1}_{(d+1)_\mathrm{t}}$ is defined in the following manner.

$$z^{d+1}_{(d+1)_\mathrm{t}} = \begin{cases}
a^{d+1}_\mathrm{t} b^{d+1}_\mathrm{t} \oplus \bigoplus_{k=0}^{d} p_{(3,d+1,k)_\mathrm{t}}, & \text{if } \boldsymbol{a}^{d+1} = (1,0) \wedge \boldsymbol{b}^{d+1} = (1,0) \\
a^{d+1}_\mathrm{t} b^{d+1}_\mathrm{t} \oplus \bigoplus_{k=0}^{d} p_{(2,d+1,k)_\mathrm{t}}, & \text{if } \boldsymbol{a}^{d+1} = (1,0) \wedge \boldsymbol{b}^{d+1} = (0,1) \\
a^{d+1}_\mathrm{t} b^{d+1}_\mathrm{t} \oplus \bigoplus_{k=0}^{d} p_{(1,d+1,k)_\mathrm{t}}, & \text{if } \boldsymbol{a}^{d+1} = (0,1) \wedge \boldsymbol{b}^{d+1} = (1,0) \\
a^{d+1}_\mathrm{t} b^{d+1}_\mathrm{t} \oplus \bigoplus_{k=0}^{d} p_{(0,d+1,k)_\mathrm{t}}, & \text{if } \boldsymbol{a}^{d+1} = (0,1) \wedge \boldsymbol{b}^{d+1} = (0,1)
\end{cases}$$

Simplifying, we conclude that the following equation holds.

$$z^{d+1}_{(d+1)_\mathrm{t}} = a^{d+1}_\mathrm{t} b^0 \oplus \bigoplus_{k=1}^{d} (a^{d+1} b^k_\mathrm{t}) \oplus a^0 b^{d+1}_\mathrm{t} \oplus \bigoplus_{k=1}^{d+1} (a^k_\mathrm{t} b^{d+1}) \oplus \bigoplus_{k=0}^{d} r_{\left(\frac{d(d+1)}{2}+k\right)_\mathrm{t}}$$

Substituting all output shares for $d+1$ leads to the final equation, completing the proof.

$$\begin{aligned}
z^0_{d+1} \oplus \bigoplus_{l=1}^{d+1} z^l_{(d+1)_\mathrm{t}} &= z^0_d \oplus \bigoplus_{l=1}^{d} z^l_{d_\mathrm{t}} \oplus \bigoplus_{k=0}^{d} r_{\left(\frac{d(d+1)}{2}+k\right)_\mathrm{t}} \oplus z^{d+1}_{(d+1)_\mathrm{t}} \\
&= z^0_d \oplus \bigoplus_{l=1}^{d} z^l_{d_\mathrm{t}} \oplus a^{d+1} b^0 \oplus \bigoplus_{k=1}^{d} (a^{d+1} b^k_\mathrm{t}) \oplus a^0 b^{d+1} \oplus \bigoplus_{k=1}^{d+1} (a^k_\mathrm{t} b^{d+1})
\end{aligned}$$

□

The correctness of all other non-linear CCHPC gadget types follow from the composition of CCHPC-AND with linear CCHPC gadgets. To prove that the presented non-linear CCHPC gadgets with $d+1$ shares satisfy CCHPC d-PINI, we first analyze the occurrence of glitches in these gadgets.

Proposition 17. *CCHPC-AND with $d+1$ shares is glitch-propagation-free.*

Proof. Suppose that $\forall l \in [1,d], G_{a^l_\mathrm{t}} = G_{a^l_\mathrm{f}} = G_{b^l_\mathrm{t}} = G_{b^l_\mathrm{f}} = 0$ and $\forall \boldsymbol{r} \in \boldsymbol{R}, G_{r_\mathrm{t}} = G_{r_\mathrm{f}} = 0$. Further, suppose an arbitrary set of TDSs accepted by $\mathcal{B}^{2,0,\frac{d(d+1)}{2},1}$ from which we conclude that $\forall k \in [0,l), (\boldsymbol{p}'_{(0,l,k)} \neq \boldsymbol{p}_{(0,l,k)}) \wedge (\boldsymbol{p}'_{(1,l,k)} \neq \boldsymbol{p}_{(1,l,k)}) \wedge (\boldsymbol{p}'_{(2,l,k)} \neq \boldsymbol{p}_{(2,l,k)}) \wedge (\boldsymbol{p}'_{(3,l,k)} \neq \boldsymbol{p}_{(3,l,k)}) \implies \boldsymbol{a}^{l'} = \boldsymbol{b}^{l'} = \boldsymbol{a}^l = \boldsymbol{b}^l = (0,0)$. Substituting this into the equations for $G_{c_{(l,k)}}, G_{d_{(l,k)}}, G_{e_{(l,k)}}, G_{f_{(l,k)}}$ for $\mathsf{DRP\text{-}MUX}_{\boldsymbol{b}^1}(\boldsymbol{p}_{(1,l,k)}, \boldsymbol{p}_{(0,l,k)})$ (cf. Figure 2), where

$$c_{(l,k)} = \overline{p_{(1,l,k)_\mathrm{t}} \wedge b^l_\mathrm{t}}, \quad d_{(l,k)} = \overline{b^l_\mathrm{f} \wedge p_{(0,l,k)_\mathrm{t}}}, \quad e_{(l,k)} = \overline{p_{(1,l,k)_\mathrm{f}} \wedge b^l_\mathrm{t}}, \quad f_{(l,k)} = \overline{b^l_\mathrm{f} \wedge p_{(0,l,k)_\mathrm{f}}}$$

We receive $G_{c_{(l,k)}} = G_{d_{(l,k)}} = G_{e_{(l,k)}} = G_{f_{(l,k)}} = 0$. The same holds for the four intermediates computed by $\mathsf{DRP\text{-}MUX}_{\boldsymbol{b}^1}(\boldsymbol{p}_{(3,l,k)}, \boldsymbol{p}_{(2,l,k)})$ what therefore implies $G_{t_{(l,k)_\mathrm{t}}} = G_{t_{(l,k)_\mathrm{f}}} = 0$. Further, if $\mathcal{B}^{2,0,\frac{d(d+1)}{2},1}$ accepts, we can conclude that $(\boldsymbol{a}^{l'} \neq \boldsymbol{a}^l) \wedge (\boldsymbol{b}^{l'} \neq \boldsymbol{b}^l) \implies (\boldsymbol{p}'_{(0,l,k)} = \boldsymbol{p}_{(0,l,k)}) \wedge (\boldsymbol{p}'_{(1,l,k)} = \boldsymbol{p}_{(1,l,k)}) \wedge (\boldsymbol{p}'_{(2,l,k)} = \boldsymbol{p}_{(2,l,k)}) \wedge (\boldsymbol{p}'_{(3,l,k)} = \boldsymbol{p}_{(3,l,k)})$. Substituting this into the same equations again implies that $G_{t_{(l,k)_\mathrm{t}}} = G_{t_{(l,k)_\mathrm{f}}} = 0$. Since, acceptance of $\mathcal{B}^{2,0,\frac{d(d+1)}{2},1}$ implies that the constraint automata associated with all DRP circuits also accept and due to Proposition 2, Proposition 4, and Proposition 6, it follows that $\forall l \in [1,d], G_{z^l_{d_\mathrm{t}}} = G_{z^l_{d_\mathrm{f}}} = 0$, which proves that CCHPC-AND with $d+1$ shares is glitch-propagation-free. □

The glitch-propagation-freeness of all other non-linear CCHPC gadget types follow from the composition of CCHPC-AND with linear CCHPC gadgets.

Proposition 18. *CCHPC-AND with $d+1$ shares is CCHPC d-PINI.*

Before proving Proposition 18, we focus on the probe-extension through a DRP-MUX according to Section 3.2. To enhance readability, two probes on dual-rail values, such as $(P_{a_\mathrm{t}}, P_{a_\mathrm{f}})$ are represented as a single probe on the dual-rail variable, like $P_{\boldsymbol{a}}$. Suppose an adversary places probes $P_{\boldsymbol{z}}$ on the DRP-MUX output and a set of TDSs accepted by $\mathcal{A}^{3,1}$. Then, the probes extend as follows according to Algorithm 1:

$$P_{\boldsymbol{z}} \rightarrow \begin{cases} \{P_{a'_\mathrm{t}}, P_{a_\mathrm{t}}, P_{s'_\mathrm{t}}, P_{s_\mathrm{t}}, P_{d'}, P_d, P_{e'}, P_e, P_{f'}, P_f\}, & \text{if } (a'_\mathrm{t}{=}1 \vee a_\mathrm{t}{=}1) \wedge (s'_\mathrm{t}{=}1 \vee s_\mathrm{t}{=}1) \\ \{P_{a'_\mathrm{f}}, P_{a_\mathrm{f}}, P_{s'_\mathrm{t}}, P_{s_\mathrm{t}}, P_{c'}, P_c, P_{d'}, P_d, P_{f'}, P_f\}, & \text{if } (a'_\mathrm{f}{=}1 \vee a_\mathrm{f}{=}1) \wedge (s'_\mathrm{t}{=}1 \vee s_\mathrm{t}{=}1) \\ \{P_{b'_\mathrm{t}}, P_{b_\mathrm{t}}, P_{s'_\mathrm{f}}, P_{s_\mathrm{f}}, P_{c'}, P_c, P_{e'}, P_e, P_{f'}, P_f\}, & \text{if } (b'_\mathrm{t}{=}1 \vee b_\mathrm{t}{=}1) \wedge (s'_\mathrm{f}{=}1 \vee s_\mathrm{f}{=}1) \\ \{P_{b'_\mathrm{f}}, P_{b_\mathrm{f}}, P_{s'_\mathrm{f}}, P_{s_\mathrm{f}}, P_{c'}, P_c, P_{d'}, P_d, P_{e'}, P_e\}, & \text{if } (b'_\mathrm{f}{=}1 \vee b_\mathrm{f}{=}1) \wedge (s'_\mathrm{f}{=}1 \vee s_\mathrm{f}{=}1) \end{cases} \tag{3}$$

For better understanding, let us exemplary assume that $(a'_\mathrm{t} = 1 \vee a_\mathrm{t} = 1) \wedge (s'_\mathrm{t} = 1 \vee s_\mathrm{t} = 1)$. In this case, we have $F_c = 1$, $F_d = 0$, $F_e = 0$, $F_f = 0$, $F_{z_\mathrm{t}} = 1$, and $F_{z_\mathrm{f}} = 1$, which implies that P_{z_t} (respectively, P_{z_f}) extends to (P_c, P_d) (respectively, (P_e, P_f)), with only P_c being further extended to $(P_{a_\mathrm{t}}, P_{s_\mathrm{t}})$. For simplicity, we omit the probe extensions if inputs do not change, as they are strictly less informative than those in Equation (3). This can be verified by constructing the corresponding extensions using the same method. Moreover, all probes highlighted in gray can be removed from the sets, as they provide no additional information to the adversary. For instance, consider P_e in the first case of Equation (3): since one extension is P_{a_t} and $a_\mathrm{t} = 1$ implies $a_\mathrm{f} = 0$, the adversary can deduce that $e = 1$ rendering P_e redundant.

Proof. Suppose that CCHPC-AND is glitch-propagation-free for all sets of TDSs accepted by $\mathcal{B}^{2,0,\frac{d(d+1)}{2},1}$ (cf. Proposition 17). First, we focus on the computation of z^0_d which is not glitch-free. Hence, a relaxed probe on z^0_d, denoted as $P_{z^0_d}$, extends as in the robust probing model and can be simulated using only shares with index 0 and by tossing a fair coin.

$$P_{z^0_d} \rightarrow \left\{ P_{a^{0\prime}}, P_{b^{0\prime}}, P_{r'_{\left(\frac{i(i+1)}{2}\right)_\mathrm{t}}}, P_{a^0}, P_{b^0}, P_{r_{\left(\frac{i(i+1)}{2}\right)_\mathrm{t}}} \mid i \in [0,d) \right\}$$

Similarly, $\forall\, j \in [0,d)$, $P_{z^0_j}$ can be simulated like $P_{z^0_d}$ as it holds that $P_{z^0_j} \subset P_{z^0_d}$, meaning that an internal probe on the computation z^0_j is strictly less informative than its output probe. Next, suppose a probe on $z^l_{d_\mathrm{t}}$ (resp. $z^l_{d_\mathrm{f}}$) where $l \in [1,d]$. We assume no restricted extension through DRP-XOR gadgets. Hence, a probe on $z^l_{d_\mathrm{t}}$ (resp. $z^l_{d_\mathrm{f}}$) extends back to the output of the cascaded DRP-MUX gadgets.

$$P_{z^l_{d_\mathrm{t}}} \rightarrow \left\{ P_{\boldsymbol{a}^{l\prime}}, P_{\boldsymbol{b}^{l\prime}}, P_{\boldsymbol{r}'_{\left(\frac{i(i+1)}{2}+l\right)}}, P_{\boldsymbol{t}'_{(l,k)}}, P_{\boldsymbol{a}^l}, P_{\boldsymbol{b}^l}, P_{\boldsymbol{r}_{\left(\frac{i(i+1)}{2}+l\right)}}, P_{\boldsymbol{t}_{(l,k)}} \mid (i,k) \in [l,d) \times [0,l) \right\}$$

Now, we apply the probe-extensions established in Equation (3) to further extend the probes on $\boldsymbol{t}_{(l,k)}$ $\forall\, k \in [0, l)$.

$$P_{\boldsymbol{t}_{(l,k)}} \rightarrow \begin{cases} \left\{P_{\boldsymbol{a}^{l'}}, P_{\boldsymbol{b}^{l'}}, P_{\boldsymbol{p}'_{(3,l,k)}}, P_{\boldsymbol{a}^{l}}, P_{\boldsymbol{b}^{l}}, P_{\boldsymbol{p}_{(3,l,k)}}\right\}, & \text{if } \boldsymbol{a}^l = (1,0) \wedge \boldsymbol{b}^l = (1,0) \\ \left\{P_{\boldsymbol{a}^{l'}}, P_{\boldsymbol{b}^{l'}}, P_{\boldsymbol{p}'_{(2,l,k)}}, P_{\boldsymbol{a}^{l}}, P_{\boldsymbol{b}^{l}}, P_{\boldsymbol{p}_{(2,l,k)}}\right\}, & \text{if } \boldsymbol{a}^l = (1,0) \wedge \boldsymbol{b}^l = (0,1) \\ \left\{P_{\boldsymbol{a}^{l'}}, P_{\boldsymbol{b}^{l'}}, P_{\boldsymbol{p}'_{(1,l,k)}}, P_{\boldsymbol{a}^{1}}, P_{\boldsymbol{b}^{l}}, P_{\boldsymbol{p}_{(1,l,k)}}\right\}, & \text{if } \boldsymbol{a}^l = (0,1) \wedge \boldsymbol{b}^l = (1,0) \\ \left\{P_{\boldsymbol{a}^{l'}}, P_{\boldsymbol{b}^{l'}}, P_{\boldsymbol{p}'_{(0,l,k)}}, P_{\boldsymbol{a}^{l}}, P_{\boldsymbol{b}^{l}}, P_{\boldsymbol{p}_{(0,l,k)}}\right\}, & \text{if } \boldsymbol{a}^l = (0,1) \wedge \boldsymbol{b}^l = (0,1) \end{cases}$$

It is already shown in the proof of Proposition 17 that glitches that occur during the computation of $\boldsymbol{p}_{(0,l,k)}, \ldots, \boldsymbol{p}_{(3,l,k)}$ cannot propagate through the DRP-MUX gadgets. Hence, $P_{\boldsymbol{p}_{(0,l,k)}}, \ldots, P_{\boldsymbol{p}_{(3,l,k)}}$ cannot be extended further. Therefore, $P_{\boldsymbol{z}^l_d}$ can be simulated using only shares with index l (to simulate $\{P_{\boldsymbol{a}^{l'}}, P_{\boldsymbol{b}^{l'}}, P_{\boldsymbol{a}^{l}}, P_{\boldsymbol{b}^{l}}\}$) and by tossing a fair coin (to simulate $P_{\boldsymbol{p}_{(0,l,k)}}, \ldots, P_{\boldsymbol{p}_{(3,l,k)}}$ whose probed values are blinded by $\boldsymbol{r}_{\left(\frac{l(l-1)}{2}+k\right)}$). Internal probes on the computation of $\boldsymbol{z}^l_d$ can be simulated in the same way as we assume $P_{z^l_{i_\mathrm{t}}} \subseteq P_{z^l_{d_\mathrm{t}}}$ $\forall\, i \in [0, d)$ (resp. $P_{z^l_{i_\mathrm{f}}} \subseteq P_{z^l_{d_\mathrm{f}}}$ $\forall\, i \in [0, d)$), i.e., $P_{z^l_{i_\mathrm{t}}}$ (resp. $P_{z^l_{i_\mathrm{f}}}$) is less informative than $P_{z^l_{d_\mathrm{t}}}$ (resp. $P_{z^l_{d_\mathrm{f}}}$). However, the situation changes if an adversary probes the pre-processing, i.e., places one of the following probes:

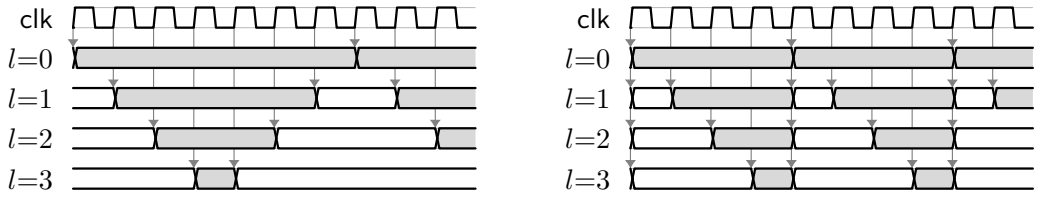
$$\forall\,(i,l,k) \in [0,3] \times [1,d] \times [0,l), P_{\boldsymbol{p}_{(i,l,k)}} \subseteq \left\{P_{\boldsymbol{a}^{k'}}, P_{\boldsymbol{b}^{k'}}, P_{\boldsymbol{r}'_{\left(\frac{l(l-1)}{2}+k\right)}}, P_{\boldsymbol{a}^{k}}, P_{\boldsymbol{b}^{k}}, P_{\boldsymbol{r}_{\left(\frac{l(l-1)}{2}+k\right)}}\right\}$$

Hence, the observation set made by a probe $P_{\boldsymbol{p}_{(i,l,k)}}$ can be simulated using only shares with index k. Finally, suppose an arbitrary set O of relaxed probes on output shares with A denoting the set of corresponding share indices, and an arbitrary set I of relaxed probes on intermediates for which it holds that $|I| + |A| \leq d$. We construct the set of required input shares B by adding the share index l for every internal probe placed in layer l, provided that $l \notin A \cup B$. The only exception are probes on the pre-processing: for every probe of the form $P_{p_{(i,l,k)_\mathrm{t}}}$ or $P_{p_{(i,l,k)_\mathrm{f}}}$, where $(l,k) \in [1,d] \times [0,l)$, we add the share index k to B, again provided that $k \notin A \cup B$. As no condition allows more than one share index to be added to B per probe, it follows that $|B| \leq |I|$. Since every random value $\boldsymbol{r} \in \boldsymbol{R}$ is distributed to exactly two layers — specifically, $\boldsymbol{r}_{\left(\frac{l(l-1)}{2}+k\right)}$ is shared between layers l and k — any probe in layer l that accesses a pre-processed value blinded using such a random value can only be unblinded if there is a corresponding probe in layer k, and that probe is not on a pre-processed value. However, by construction, B includes k for every such probe in layer k (not on a pre-processed value), ensuring that $A \cup B$ suffices to simulate $I \cup O$, even in the case of (partial) unblinding. This concludes the proof that CCHPC-AND is RR d-PINI, and therefore CCHPC d-PINI. □

CCHPC d-PINI of all other non-linear CCHPC gadget types follow from the composition of CCHPC-AND with linear CCHPC gadgets.

Execution Sequence. The CCHPC-AND gadget does not inherently include register stages. Therefore, to ensure probing security, the share domains must be manually separated in time. When used in practice, this separation is achieved by sequentially applying the shared inputs across different clock cycles and therefore, shifted DRP executions in the layers (previously validated by the constraint automaton). Figure 6(a) exemplarily visualizes the phases in every layer for order $d = 3$. More concretely, the input sequence of a CCHPC-AND gadget with $d + 1$ shares is as follows.

1. a^0 and b^0 are fed to the gadget, while $\boldsymbol{a}^i, \boldsymbol{b}^i$ $\forall\, i \in [1, d]$ convey the null state. The layer $l = 0$ pre-processes intermediate values for layers $l > 0$ and completes the output share z^0 computation.



(a) Without register stage, $d = 3$. (b) CCHPC$_{\text{RS}}$ (with register stage), $d = 3$.

Figure 6: DRP phases across CCHPC layers, with and without a register stage in all non-linear gadgets of the circuit. ▢ corresponds to the pre-charge and ▢ to the evaluation phase (resp. execution phase for layer $l = 0$). The ↓ indicates the trigger for the transition to the next phase.

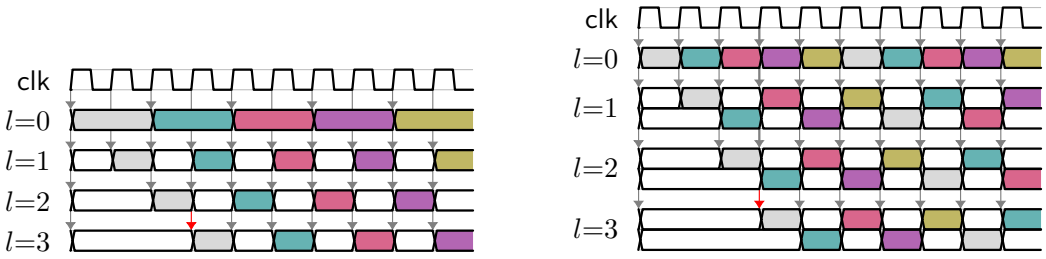
2. During the next clock cycle, $\boldsymbol{a}^1$ and $\boldsymbol{b}^1$ are fed to the circuit. As a consequence, the evaluation of layer $l = 1$ issues pre-processed intermediate values to layers $l > 1$ and generates $\boldsymbol{z}^1$.
3. In every following clock cycle, for layer $l = [2, d]$ the same procedure as in layer $l = 1$ is followed. After the d-th share evaluation, the primary inputs are pre-charged in descending share order, resulting in the null state in each layer consecutively, except for layer $l = 0$.
4. Eventually, when layer $l = 1$ is pre-charged, the execution finishes and the next shared primary inputs can again be applied consecutively.

Since applying the primary input shares with index l to layer l, which issues the output share l, cascaded CCHPC gadgets achieve a constant cycle count for arbitrary gadget compositions. All output shares are generated with a latency of d clock cycles. However. the next primary input can be applied after $2d + 1$ clock cycles, as each layer must be pre-charged in descending order across different clock cycles.

4.2 CCHPC-composed Circuits

At this point, the provably-secure gadget-based CCHPC scheme achieves a constant number of cycles for arbitrary circuit sizes by the replacement of each logic gate with the corresponding CCHPC gadget counterpart, which already showcases its low-latency capabilities, especially with large designs. Though, the latency doubles for increasing security orders due to the sequential execution of DRP phases (cf. Figure 6(a)). This can be reduced by the re-introduction of registers and improved CCHPC-composed circuit architectures.

CCHPC$_{\text{RS}}$. An improved variant, namely CCHPC$_{\text{RS}}$, comprises an additional register stage between the layers. Similar to CCHPC, the layers are executed in ascending order. However, the propagation of layer crossing sets $\{\boldsymbol{p}_{(0,l,k)}, \ldots, \boldsymbol{p}_{(3,l,k)}\}$ passes controlled registers placed between the source layer k and destination layers l during the evaluation phases. By this, after layer d is evaluated, we can reset every register and primary input, effectively pre-charging all layers $l > 0$ at once. By such a trick, the total execution latency of a CCHPC$_{\text{RS}}$ gadget increases by one clock cycle per increment in the security order. Specifically, $d + 1$ clock cycles are required between two consecutive primary inputs to a CCHPC-protected circuit. The according execution sequence is visualized in Figure 6(b) for $d = 3$ and a configuration flag is available in Algorithm 2 that enables instantiating the optional register stage (cf. Line 15). We note that CCHPC$_{\text{RS}}$ with $d + 1$ shares is still CCHPC d-PINI. Formal proofs are given in Appendix A.



(a) CCHPC_{PL} (with register pipeline), $d = 3$. (b) CCHPC_{PL} & duality, $d = 3$.

Figure 7: DRP phases across CCHPC_{PL} layers. ▢ corresponds to the pre-charge and the filled ones (▢ , ▢ , ▢ , ▢ , ▢) to the evaluation phases. Different filled colors highlight the evaluation of different inputs across the layers of the entire circuit. The ↓ indicates the trigger for the transition to the next phase. Randomness pre-processing ends with ↓ .

Pipelined CCHPC Layers. Gadget-based masking schemes typically use registers for each non-linear function, which adds one or more clock cycles to the total latency of a design. Such schemes can be utilized in a pipelined fashion, where primary inputs are applied consecutively to the circuit. While the increased latency remains for this approach, it enables a high throughput. The advantage of high throughput is also achieved by a pipelined CCHPC design. However, CCHPC generally has the special property of layers being executed with an offset in clock cycles, where the execution of layer l requires available fresh randomness and intermediate values pre-processed by layers $< l$ within the same non-linear gadget. From this, it follows that the required pre-processed values from each source layer k should be preserved in additional register stages, such that primary inputs can be applied consecutively to the circuit. Therefore, the register stage of CCHPC_{RS}, that stores intermediates of layer k to be used in layer $l > k$, is replaced by a register pipeline of depth $l - k$, where pre-charge and evaluation phases are carried through the chain of registers, such that the DRP phases are synchronized between the layers and security properties are maintained. Based on the configuration parameter, Algorithm 2 instantiates the register pipeline $\mathsf{REG\text{-}PIP}_{\text{depth}}$ of according depth within the condition of Line 19. Consequently, although layer $l = 0$ executes in a single clock cycle, the DRP phases in layers $l > 0$ demand for the minimum amount of two cycles per layer. In other words, layer $l = 0$ must hold its state every other clock cycle. We refer to the concept of CCHPC with register pipelines between layers as CCHPC_{PL}. Compared to CCHPC and CCHPC_{RS}, the evaluation latency remains unchanged, but consecutive primary inputs can be fed to the circuit every two clock cycles. The DRP phases over time are exemplarily shown in Figure 7(a) for order $d = 3$. We note that CCHPC_{PL} with $d + 1$ shares is also CCHPC d-PINI. Formal proofs are given in Appendix A.

Round-based vs. Unrolled. Similar to LMDPL [SBHM20], the latency of CCHPC-protected circuits, in terms of clock cycles, also remains constant for round-based designs, regardless of the circuit size. However, both DRP phases must be carried through the round function. Specifically, the round function must alternate between pre-charge and evaluation phases to complete each execution, i.e., the processing of a round input. This can be achieved by the insertion of two register stages, which preserve both DRP states, also in an alternating manner. These register stages can be placed at arbitrary locations in the circuit, but it must be ensured that one register maintains the null state, allowing the succeeding circuit to perform the pre-charge phase up to the second register stage. This second register stage holds the evaluation state and enables the evaluation for the remaining part of the circuit, thereby completing the round function implementation. A

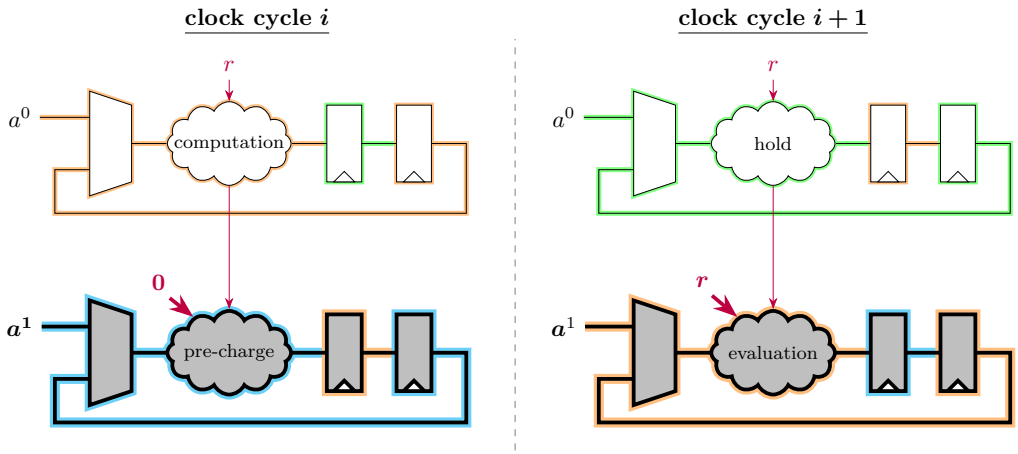


Figure 8: Sequence execution in a first-order CCHPC-masked iterative design with cascaded registers exemplarily placed at the round function output. DRP logic in gray and **bold**, non-linear parts colored in red, logic parts in pre-charge phase shaded blue, evaluating logic parts shaded orange, and latency compensation in layer $l = 0$ shaded green.

straightforward approach is to append two cascaded register stages to the round function output, as shown in Figure 8. Note that layer $l = 0$ also implements both register stages, even though it does not operate in DRP. This compensates the latency introduced by the pre-charge phases in layers $l > 0$, ensuring synchronization with the shifted DRP executions throughout the CCHPC-composed circuit.

Furthermore, a d-th order m-round algorithm can be (partially) unrolled with an n-round CCHPC_{PL} implementation, such that the total evaluation takes $\lceil \frac{2m}{n} \rceil + d - 1$ clock cycles. Alternating DRP phases for each round execution introduce the factor of 2, leading to $2m$ phases in total. The number of clock cycles required is further determined by dividing these phases into partial groups of n rounds, given by the ceiling of $\lceil \frac{2m}{n} \rceil$, while $d - 1$ reflects the added latency by sequential execution of d layers. The accompanying high circuit depth, that is inevitably caused by this approach, leads to a trade-off between area, timing and latency considerations.

Duality. The aforementioned limitation of two clock cycles per execution in each CCHPC_{PL} layer can furthermore be avoided by the concept of duality, originally proposed in [SBHM20] for a low-latency focused LMDPL-protected AES implementation. The concept considers a duplication of the round function for the parts that operate in DRP logic. More precisely, a two-round function is partitioned in two parts. Both partitions are intended to alternately pass the processing of DRP phases to each other, ensuring evaluation in each consecutive clock cycle. Consider a two-round function f partitioned into f_0 and f_1. If f_0 evaluates, f_1 pre-charges and vice-versa. Round results processed by layer $l = 0$ are alternately provided to f_0 and f_1. The state of fresh randomness is synchronized to the DRP phases according to the targeted round instance. As a result, every clock cycle a new input can be given to the circuit. We extend this concept for d-order secure implementations. Every layer $l > 0$ is duplicated, such that a CCHPC_{PL}-protected design with applied duality incorporates pipelines in two dimensions: (1) across layers, and (2) between duplicated rounds. This results in a total of $11 + d$ clock cycles for the full AES-128 encryption function. The DRP phases are traced for both dimensions, as shown in Figure 7(b) for $d = 3$. Note that the duality concept can be combined with (partial) unrolling to further reduce the total number of clock cycles, naturally with higher area overhead. More precisely, a d-th order m-round algorithm can be (partially) unrolled with duplicated n-round CCHPC_{PL}

instances, to reduce its evaluation latency by half, to $\lceil \frac{m}{n} \rceil + d - 1$ clock cycles.

Pre-processing. Focusing on primary inputs of a masked circuit, since the first d Boolean masked shares of a $d+1$ sharing are sampled from randomness, the structure of CCHPC gadgets enables the pre-processing of d input shares. Each layer l requires shares with index $\leq l$, i.e., the first d layers require shares with index $\leq (d-1)$, and can be pre-processed independent of the last input share d. This approach saves up to d clock cycles of latency for any of the above presented CCHPC gadget, resulting in a total added latency of one remaining clock cycle for the result to be available at the output. Afterwards, the pre-charge phases are performed according to the gadget's configuration. For a CCHPC_{PL}-protected AES design with duality, pre-processing of randomness takes a single clock for each round computation, i.e., no additional latency is added compared to the latency of a similar unprotected design. The start of execution after pre-processing for CCHPC_{PL} with and without duality is highlighted in Figure 7 with ↓.

Transitional Leakage at the Architecture Level. The non-linear CCHPC gadgets are not RR d-O-PINI. We note that this also applies to the CCHPC_{RS} and CCHPC_{PL} variants, which implies that, in theory, CCHPC may be vulnerable to transitional leakage. Specifically, if certain gates are evaluating multiple times during circuit operation, indicating the presence of loops in the design. Therefore, integrating CCHPC gadgets into a round-based design requires an iterative architecture, similar to Figure 8, that imposes two cascaded registers per round function output. Further, it holds that every register alternates between pre-charge and evaluation phases. As a consequence, any transition-extended probe capable of recording over two consecutive clock cycles in such a circuit provides no more information than a probe which is not extended by transitions. Therefore, if a CCHPC-protected circuit is implemented using a round-based architecture as described above and illustrated in Figure 8, it achieves RR d-probing security, effectively capturing robustness against transitional leakage.

4.3 CCHPC Performance Metrics

Here, we summarize the CCHPC gadget overheads when constructing a masked circuit considering the corresponding unmasked design as the baseline. Regardless of the non-linear gadget configuration and independent of advanced architectures, the linear gadget (including the CCHPC-MUX) add d layers to the unmasked circuit, which scales by a constant factor for increasing security orders. In contrast, the non-linear gadget grows quadratically. An overview of exact gate counts for $d \in [1, 3]$ and a general expression for arbitrary security orders is provided in Table 1. We also list each gadget's critical path depth in the same table, i.e., the maximum number of single-rail gates cascaded in the gadget. As no register is necessary to be placed between consecutive gadgets in the circuit to fulfill SCA security, the critical path naturally lengthens with each concatenated gadget in a row. Thus, a high circuit depth might cause a compromise in the maximum frequency provided for execution on a physical device, i.e., for an unrolled cipher implementation. Of course, a register stage between gadgets conventionally avoids a low clock frequency, at the cost of increased clock cycles.

We further classify different non-linear CCHPC gadget configurations to examine the register count. Table 2 presents the corresponding results as well as the latency in two aspects. The 'evaluation latency' indicates when the output shares of the gadget become available, while the 'execution latency' refers to the clock cycles required to wait until being able to give a new input to the gadget. These are shown for different non-linear CCHPC gadgets and for security orders $d = [1, 2, 3]$ as well as for arbitrary orders. Similar to the gate count, the register count increases quadratically with the security order for all

Table 1: CCHPC gadget gate counts and critical path depth.

Type	Gadget	Order	Gate Count					Critical Path Depth
			#NOT	#XOR	#AND	#NAND	#OR	
Linear	CCHPC-X(N)OR	1	-	1	-	6	-	2
		2	-	1	-	12	-	2
		3	-	1	-	18	-	2
		d	-	1	-	$6d$	-	2
	CCHPC-MUX	1	-	2	1	6	-	3
		2	-	2	1	12	-	3
		3	-	2	1	18	-	3
		d	-	2	1	$6d$	-	3
Non-linear	CCHPC-(N)AND/(N)OR	1	3	4	2	12	1	6
		2	9	11	3	39	2	8
		3	18	21	4	81	3	10
		d	$\frac{3d(d+1)}{2}$	$\frac{3d(d+1)}{2} + d$	$d+1$	$15d^2 + 9d$	d	$2d+4$

Table 2: Non-linear CCHPC gadget register count. Latency stated for 'evaluation' (the delay when the entire circuit's output is available) and 'execution' (the number of clock cycles between every two consecutive inputs).

Configuration	Order	#REG	Latency	
			eval.	exec.
CCHPC	1	-	1	3
	2	-	2	5
	3	-	3	7
	d	-	d	$2d+1$
CCHPC_{RS}	1	8	1	2
	2	24	2	3
	3	48	3	4
	d	$4d(d+1)$	d	$d+1$
CCHPC_{PL}	1	8	1	2
	2	32	2	2
	3	72	3	2
	d	$8d^2$	d	2
CCHPC_{PL} & Duality[1]	1	16	1	1
	2	64	2	1
	3	144	3	1
	d	$16d^2$	d	1

[1]Requires a register stage for layers $l > 0$ between duplicated rounds.

gadget configurations, except for the default CCHPC. Every variant sequentially evaluates its layers, such that the evaluation latency is equal for all cases. While CCHPC_{RS} and CCHPC_{PL} behave equally for security order $d = 1$, the number of registers in CCHPC_{PL} increases more rapidly at higher orders. However, it additionally achieves a constant execution latency independent of the security order. The lowest execution latency is achieved, however, by the combination of CCHPC_{PL} and an architecture that implements duality, but the gate and register count are doubled in that case.

As a side note, the instantiation of any CCHPC variant requires control logic to manage the DRP phases correctly across all circuit parts. We omit this consideration from our performance metrics, as an efficient implementation of such logic results in only a very slight area overhead particularly for higher security orders and is generally negligible.

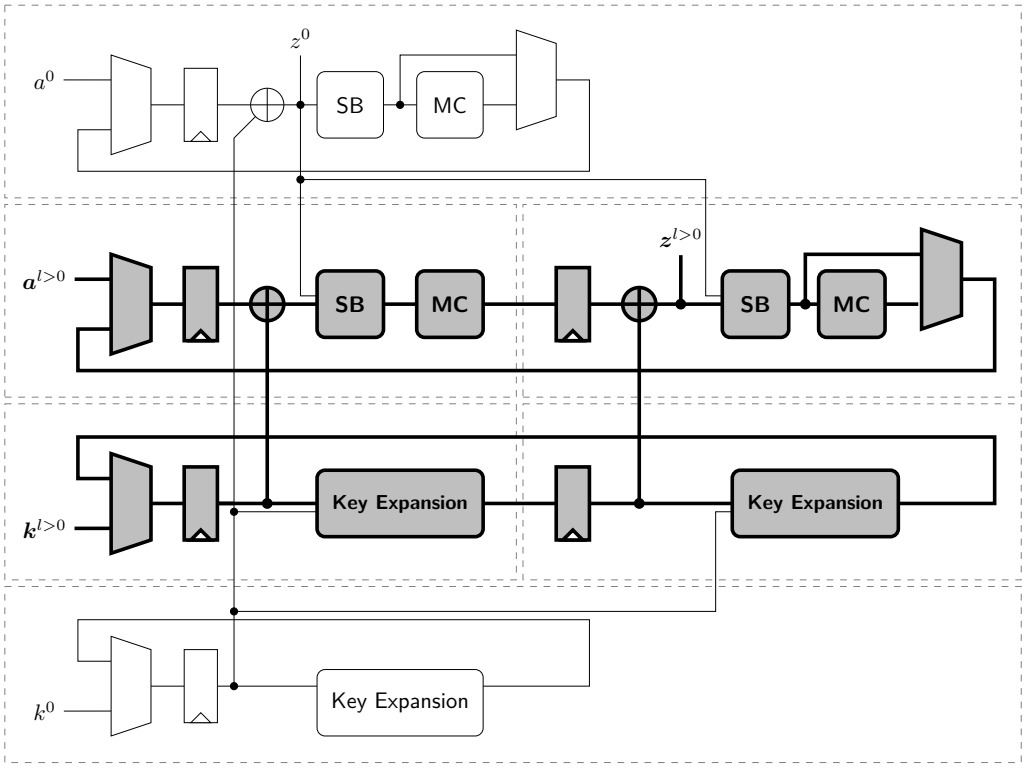


Figure 9: CCHPC$_{\text{PL}}$ & duality, d-th order AES architecture. DRP logic in gray and **bold**.

5 Case Studies

In this section, we put CCHPC to the test. We present a low-latency AES architecture that incorporates the duality concept and interpret our synthesis results compared to a similar LMDPL implementation. Furthermore, we consider low-latency gadget-based masking methods that follow different approaches in our comparison. Lastly, we provide detailed insights into the tool-assisted theoretic evaluation and discuss the results of experimental analyses.

5.1 Low-Latency CCHPC Round-based AES Architecture

In accordance with our explanations in Section 4.2, we implemented a CCHPC$_{\text{PL}}$-protected AES encryption engine with duality that balances the trade-off between area and latency. As outlined in the d-th order architecture schematic in Figure 9, the round function and key expansion are both masked and layers $l > 0$ are duplicated. A minor exception to the duplication is the input and last round selections, where the latter discards the MixColumns operation. As the number of AES rounds is even, the last round is always computed in the second round function instance for $l > 0$. Upon reset, both round function instances are pre-charged and once the cipher core is enabled, the DRP phases across the layers are executed accordingly. The control instance, implemented as a Moore machine, generates these glitch-free signals and operates in single-rail internally. We would like to highlight that – as stated before – the DRP-MUX instances in both cases require stable dual-rail select signals. However, the outgoing signals from the control unit to the DRP-MUX instances are unmasked dual-rail signals, which are synchronized with the phases of the destination layer l. Hence, the select signals are pre-charged and shifted by one clock cycle

for each layer in ascending order, provided by the controller. For primary inputs, our AES core loading phase eliminates glitches through the registers, i.e., proper masked dual-rail representation is accounted for at the primary IOs and not on the gadget level. Once a proper (glitch-free) DRP state is reached, the circuit operates in DRP and no glitches can be introduced at a later stage. Since at least one of the target layers is in the pre-charge phase at any time, the glitch propagation of layer $l = 0$ is stopped by the pipeline registers within each gadget, whose first instance is reset. Additionally, note that layer $l = 0$ stores its state in only one state register and performs the share-isolated XOR operation, for which glitch propagation is permitted. The resulting intermediate value is provided to the linear and non-linear gadgets in both round function instances in parallel.

The duplication almost doubles the area requirements, but ensures continuous evaluation without sacrificing clock cycles for pre-charging phases. Therefore, a total runtime of $11 + d$ clock cycles is required to obtain the ciphertext for the given plaintext. We utilize this architecture for the following evaluations and make it publicly available on GitHub[1]

5.2 Synthesis Results

The synthesis results for a CCHPC_{RS} AES S-box instantiated for all $d \in [1, 3]$ are shown in Table 3. These results are compared to the performance of relevant low-latency masked AES S-box implementations reported in the literature, all of which utilize a single register stage. Specifically, we only consider masking schemes that offer composable secure masking of arbitrary Boolean functions by using only a single register stage. Broadly, these methods can be categorized into two groups: (1) masking schemes that employ DRP logic to mitigate glitches, such as LMDPL [LMW14] and CCHPC, and (2) those which do not utilize DRP logic, such as GHPC_{LL} [KSM22] and TSM [VDB+24]. A key distinction lies in latency. For LMDPL and CCHPC, the composition of the scheme does not increase the overall latency of the circuit. In contrast, GHPC_{LL} and TSM schemes accumulate latency due to their design principle. To avoid increased latency in these cases, the entire AES S-box must be implemented as a single GHPC_{LL} or TSM gadget.

Conversely, LMDPL and CCHPC allow the creation of a masked S-box with fixed-latency, e.g. '1' for LMDPL and d for CCHPC, by assembling smaller gadgets. We implemented the LMDPL- and CCHPC-protected S-boxes based on the publicly available implementation of the Boyar-Peralta AES S-box [BP12][2]. The same design framework was utilized for the GHPC_{LL}-protected S-box, which was generated using AGEMA [KMMS22][3]. For the TSM-protected S-box, we used an optimized S-box design from [VDB+24], which is also available online[4]. To synthesize all designs, we used Synopsis Design Compiler and the Nangate45 standard cell library.

The area and randomness requirements for GHPC_{LL} scale with the number of function inputs n while for TSM, these requirements grow with the algebraic degree k. Since these gadgets cannot be composed without increasing latency, masking an AES S-box with $n = 8$ and $k = 7$ results in overhead in both area and randomness requirements, as demonstrated in Table 3. In contrast, LMDPL and CCHPC do not face these limitations, as their overhead remains constant with respect to n and k, making them more efficient for such designs. Consequently, our comparison focuses on CCHPC and LMDPL, where we find that the overhead for both schemes is nearly identical. However, CCHPC stands out as the only scheme in this comparison capable of being instantiated at arbitrary security orders, whereas all other schemes are restricted to first-order security. Naturally, the overhead of

[1] https://github.com/ChairImpSec/CCHPC

[2] https://github.com/Chair-for-Security-Engineering/AGEMA/tree/main/CaseStudies/05_AES_Sbox_Optimized/behavioral

[3] https://github.com/Chair-for-Security-Engineering/AGEMA/tree/main/CaseStudies/05_AES_Sbox_Optimized/netlists/sbox_GHPCLL_ANF_Pipeline_d1

[4] https://artifacts.iacr.org/tches/2024/a20/

Table 3: Synthesis results of masked low-latency AES S-box implementations.

Ref.	Method	Proof	Order	Area [GE]	Delay [ns]	Rand [bits]	Latency [cycles]
Our Work	CCHPC_{RS}	*relaxed*	1	4,062	2.22	34	1
			2	11,231	1.86	102	2
			3	21,749	2.19	204	3
[SBHM20]	LMDPL	*relaxed*	1	3,980	2.22	34	1
[VDB+24]	TSM	*robust*	1	14,351	4.07	262	1
[KSM22]	GHPC_{LL}	*robust*	1	134,992	1.51	2048	1

Table 4: Synthesis results of masked round-based AES implementations.

Ref.	Method	Proof	Order	Area [kGE]	Delay [ns]	Rand [bits]	Latency [cycles]
Our Work	CCHPC_{PL} & duality	*relaxed*	1	213	3.87	680	12
			2	664	4.67	2040	13
			3	1,438	5.61	4080	14
[SBHM20]	LMDPL & duality	*relaxed*	1	218	3.16	680	12
[CGLS21]	HPC2	*robust*	1	149	1.23	680	99
	HPC2		2	287	1.33	2040	99
	HPC2		3	468	1.39	4080	99
[KM22]	HPC3	*robust*	1	96	1.25	1360	55
	HPC3		2	186	1.36	4080	55
	HPC3		3	302	1.33	8160	55
[CGM+24]	COMPRESS	*robust*	1	66	0.8	920	44
	COMPRESS		2	152	0.92	2760	44
	COMPRESS		3	279	1.07	5520	44

CCHPC, in terms of area and randomness requirements, increases with the security order. The main drawback of LMDPL-, CCHPC_{RS}- and CCHPC_{PL}-protected designs is that their gadget's internal registers cannot function as the state register in round-based cipher designs, since both DRP states must be preserved. Therefore, the regular stage register cannot be omitted, and the latency increases due to the underlying two-phase protocol of the employed DRP. As previously explained, this issue can be resolved using the concept of duality. For example, a first-order secure round-based CCHPC_{PL} design can achieve single-cycle latency per round by instantiating a second instance of the round function. While this approach approximately doubles the overhead in terms of area requirements, it is important to note that LMDPL and CCHPC_{PL} still remain more efficient compared to the other designs. Since the area requirements of a masked round-based AES implementation protected with TSM, and especially with GHPC_{LL}, would be impractically high, we instead compare the overhead of round-based AES implementations protected by LMDPL and CCHPC against classical gadget-based masked designs generated using AGEMA and the recently presented COMPRESS tool, as shown in Table 4. Specifically, we made the synthesis with Synopsis Design Compiler (Version O-2018.06-SP4) and a NanGate 45 nm standard cell library. The synthesizer is configured with area constraints that are unattainable, causing optimization efforts to minimize area requirements. Moreover, we focus on gadgets that are provably PINI, including HPC2 [CGLS21] and HPC3 [KM22]. However, since the generated designs consist exclusively of PINI gadgets and are fully pipelined, they are inherently resistant to transitional leakages [CS21].

We conclude that both LMDPL and CCHPC exhibit comparable overheads when instantiated for first-order protection. However, only CCHPC supports secure instanti-

ation at higher orders. Furthermore, even when compared to the design generated by COMPRESS, which achieves a latency 4 clock cycles per round, CCHPC remains preferable in terms of latency. This advantage stems from the fact that using HPC gadgets incurs a minimum latency overhead of four clock cycles per round due to the underlying S-box architecture, which requires four cascaded masked non-linear gadgets. Nevertheless, the improved latency of CCHPC comes at the cost of increased circuit size compared to classical gadgets. This increase, however, is justifiable for the following reasons:

- First, the application of duality in CCHPC doubles the circuit size while halving the latency. If a CCHPC-protected round-based AES implementation were realized without duality, the resulting circuit would be smaller than comparable circuits protected with classical gadgets, while still achieving a total latency of 24 clock cycles, an improvement over classical approaches.

- Second, at first order, the CCHPC fresh mask demand is 50% lower than that of HPC3 and approximately 26% lower than that of the design produced by COMPRESS. This trend continues at higher orders. Consequently, the size of the Pseudo Random Number Generator (PRNG) required to securely support CCHPC is also smaller compared to that needed for classical gadgets, further narrowing the gap in overall circuit size.

5.3 Tool-assisted Evaluation

The RR d-probing model, introduced at CHES 2024, is a relatively new adversary model, and currently the only tool supporting evaluation under this model is the latest version of PROLEAD [MM22, MM24]. PROLEAD enables the assessment of RR d-probing security for medium-sized circuits, such as masked S-boxes and serialized AES designs at low security orders, through logic simulations. Using an incomplete set of simulated intermediates, PROLEAD applies conditional probe extension and glitch propagation, followed by statistical independence tests to evaluate security. However, PROLEAD currently does not support composability notions, including the critical PINI. To address this limitation, we extended PROLEAD to support composability notions within the RR d-probing model. Since such an extension is not directly in the scope of this work, the corresponding procedure which we followed is given in Appendix B.

Results. We conducted all tool-assisted evaluations using our extended version of PROLEAD on an Ubuntu 22.04 server equipped with an Intel(R) Xeon(R) Gold 5320 CPU operating at 2.2 GHz with 2 TB of physical RAM. PROLEAD was configured to leverage parallel execution across 52 hyper-threading cores. Through these evaluations, we confirmed the composable security of our non-linear CCHPC gadget (with and without configured register stage) for all $d \in [1, 3]$. Specifically, we verified that the non-linear CCHPC gadget, implemented with $d + 1$ shares as described in Section 4.1.3 is RR d-PINI. Finally, we attempted to evaluate the fully protected CCHPC AES core, including duality, as presented in Section 5. However, PROLEAD was unable to produce any results due to an infeasibly long runtime.

5.4 Experimental Evaluations

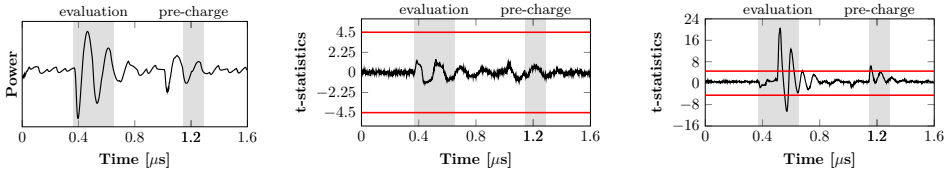
To further substantiate our theoretical claims, we performed several analyses on physical devices. The experimental setup, our testing approach to different designs and the results are presented below.

5.4.1 Measurement Setup

Depending on their resource demands, the designs under evaluation were implemented on the Kintex-7 target FPGA of a SAKURA-X side-channel evaluation board or, if the design exceeded the platform's capacity, on the larger Kintex UltraScale KU095 of the CW341 board. We measured dynamic power consumption using a digital oscilloscope at a sampling rate of 625 MS/s, with measurements taken at a shunt resistor inserted in the Vdd path of the target FPGA. Throughout the measurements, the designs were supplied with a stable 6 MHz clock. In combination, the sampling rate and clock frequency lead to approximately 104 measured sample points per clock cycle. For each design under test, we collected 100 million traces while the circuit processed either a fixed or random plaintext with a constant key, if used. The plaintext and key are provided in a freshly shared form to the target FPGA for each measurement. Fresh randomness is provided as required by the tested design by PRNG that is located on the target board as well and is randomly seeded when the device is powered up. Randomness generation runs permanently and is internally realized by 32 - bit Linear Feedback Shift Register (LFSR) instances for each random bit, which is a commonly known efficient solution for FPGA platforms [MMW18]. To evaluate the results, we conducted non-specific (fixed vs. random) t-tests. This test compares the statistical properties of the two groups of traces and detects information leakage by estimating t-statistic at each sample point based on Student's t-distribution [SM15]. To further examine higher-order security, we conducted bivariate t-tests. This method considers sample points of two points in time by multiplying their mean-free power values and performs an individual t-test on the result [SM15].

5.4.2 S-box Evaluation

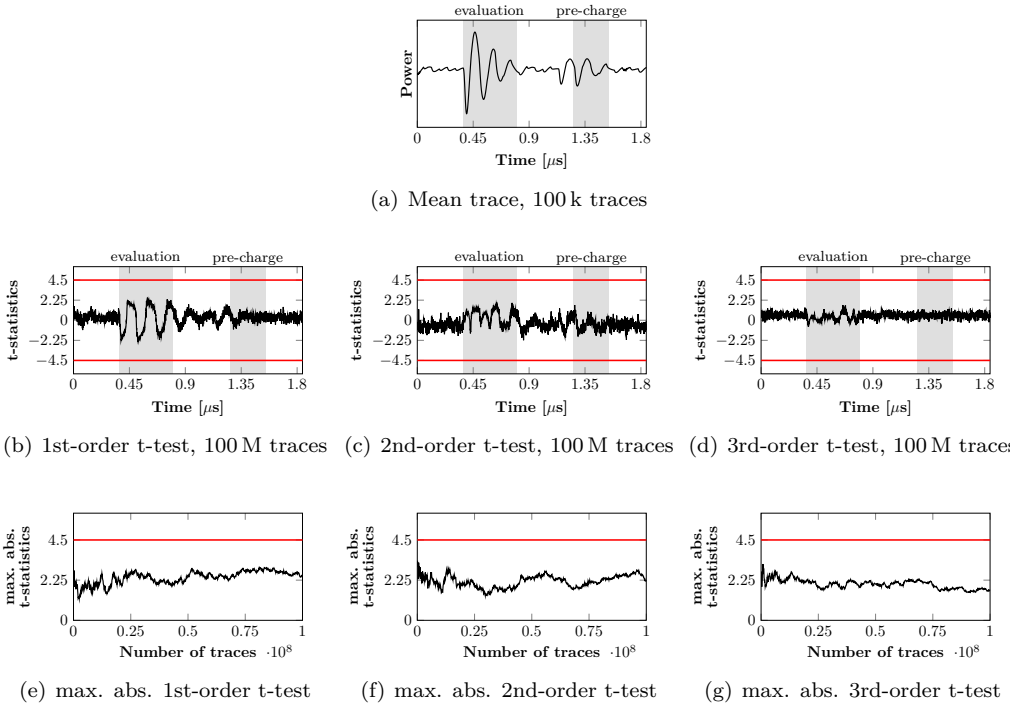
We begin by analyzing the power SCA leakage of a CCHPC_{RS} masked Boyar-Peralta AES S-box implementation for $d = 1$ with the Sakura-X board. The CCHPC non-linear configuration employs a register stage between the layers and the primary inputs are fed into the S-box one after another. Within our measurement setup, we place two parallel registers, both before and after the S-box implementation, which are reset before every trace recording. The input registers store one share each and are enabled in successive



(a) Mean trace, 100 k traces (b) 1st-order t-test, 100 M traces (c) 2nd-order t-test, 100 M traces

(d) max. abs. 1st-order t-test (e) max. abs. 2nd-order t-test

Figure 10: Univariate fixed vs. random t-test results for a first-order CCHPC_{RS}-protected Boyar-Peralta AES S-box, 1300 sample points per trace.



(a) Mean trace, 100 k traces

(b) 1st-order t-test, 100 M traces (c) 2nd-order t-test, 100 M traces (d) 3rd-order t-test, 100 M traces

(e) max. abs. 1st-order t-test (f) max. abs. 2nd-order t-test (g) max. abs. 3rd-order t-test

Figure 11: Univariate fixed vs. random t-test results for a second-order CCHPC_{RS}-protected Boyar-Peralta AES S-box, 1500 sample points per trace.

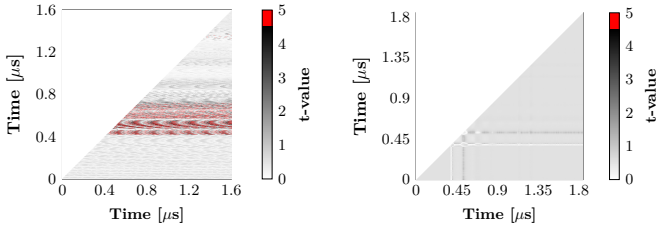
clock cycles, accordingly, to emulate the required input sequence as shown in Figure 7(b). We intentionally separated the evaluation and pre-charge phases by three clock cycles to observe emerging leakage of both DRP phases, individually. Figure 10(a) shows a sample mean power trace over 100 k traces for both parts of the S-box execution. The plot visualizes the two evaluating clock cycles, clearly. After three clock cycles, the result of layer $l = 0$ is stored in one output register. In the next cycle, layer $l = 1$ is pre-charged and the second output share is stored.

The t-statistic curves, shown in Figure 10(b), align with our theoretical expectations, as no first-order information leakage is detected for the first-order design. Figure 10(d) demonstrates that the absolute t-statistics remain significantly below the 4.5 threshold over the number of measurements. We also conducted a second-order t-test on this first-order design, depicted in Figure 10(c) and Figure 10(e). Information leakage is detected during both the evaluation and pre-charge of layer $l = 1$. The results correspond to our expectations, as second-order leakage only occurs when layer $l = 1$ is active, where the pre-processed intermediate results of layer $l = 0$ are further processed and combined with the second share inputs in layer $l = 1$.

We followed the same approach for the second-order S-box design. Three registers are located at the primary inputs and outputs, all of which are reset before every trace recording and enabled in consecutive clock cycles. DRP phases and storing outputs takes place after a three clock cycle offset in each layer. Compared to the first-order design, one clock cycle is added to the evaluation and pre-charge phases, as visible in the exemplary mean trace in Figure 11(a). As expected, the first- and second-order t-statistic show no signs of leakage (cf. Figure 11). Similar to the first-order t-test result, the absolute maximum t-statistic is constantly below the threshold. The third-order t-statistic of the second-order design also remains significantly under the 4.5 boundary. Presumably, 100

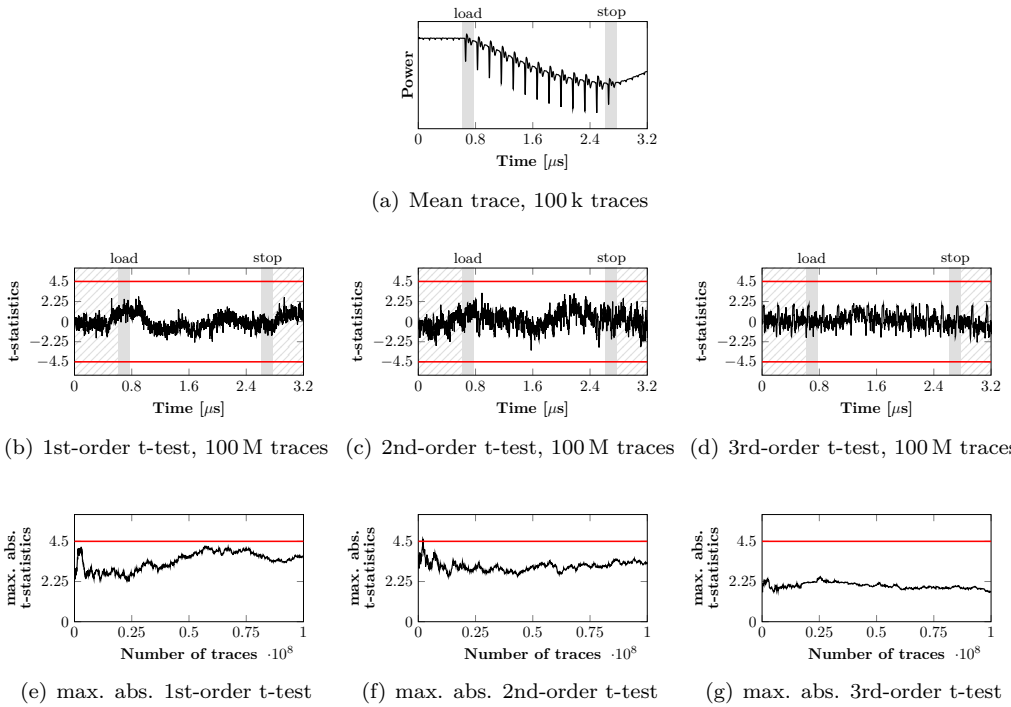
million traces are not sufficient to observe such a leakage.

To further investigate second-order leakage, bivariate fixed vs. random t-tests were conducted on the collected power traces of the first- and second-order CCHPC$_{\mathrm{RS}}$-masked Boyar-Peralta AES S-box designs. The results align with expectations, as Figure 12(b) indicates the absence of bivariate information leakage for $d = 2$. Consistent with the univariate second-order t-test on the provable first-order design, bivariate information leakage is detected (cf. Figure 12(a)).



(a) $d = 1$, 100 M traces. (b) $d = 2$, 100 M traces.

Figure 12: Bivariate fixed vs. random t-test results for first- and second-order CCHPC$_{\mathrm{RS}}$-protected Boyar-Peralta AES S-box designs, 1300 sample points per trace for $d = 1$ (approximately 846 k unique pairs) and 1500 sample points per trace for $d = 2$ (approximately 1.1 M unique pairs).

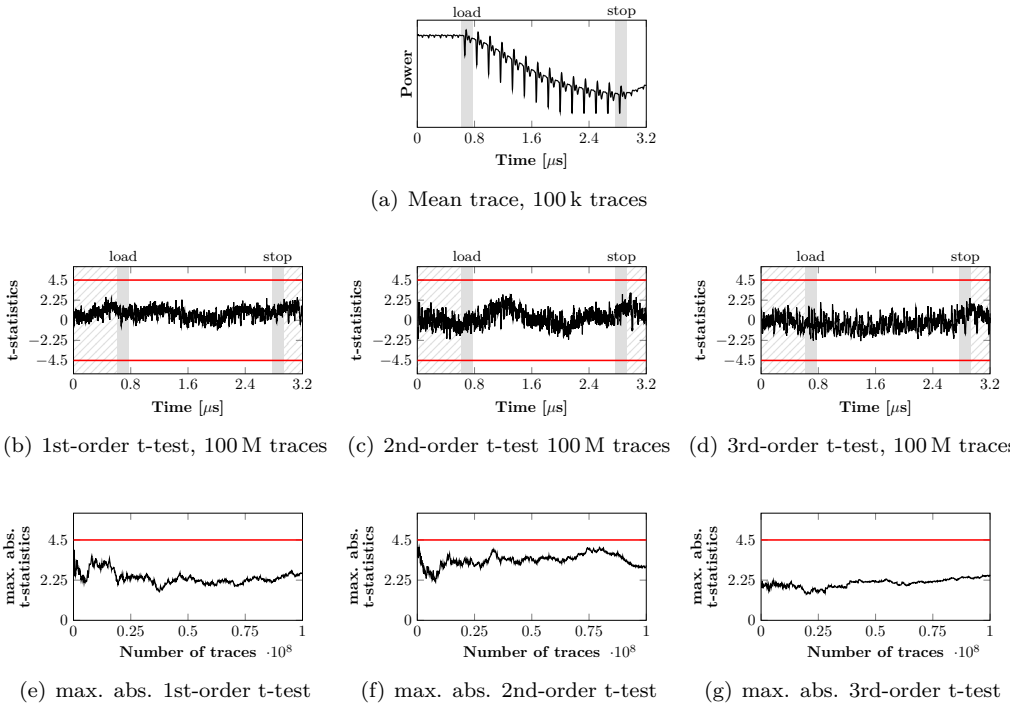


(a) Mean trace, 100 k traces

(b) 1st-order t-test, 100 M traces (c) 2nd-order t-test, 100 M traces (d) 3rd-order t-test, 100 M traces

(e) max. abs. 1st-order t-test (f) max. abs. 2nd-order t-test (g) max. abs. 3rd-order t-test

Figure 13: Univariate fixed vs. random t-test results for a first-order CCHPC$_{\mathrm{PL}}$-protected AES with duality applied, 2000 sample points per trace.



(a) Mean trace, 100 k traces

(b) 1st-order t-test, 100 M traces (c) 2nd-order t-test 100 M traces (d) 3rd-order t-test, 100 M traces

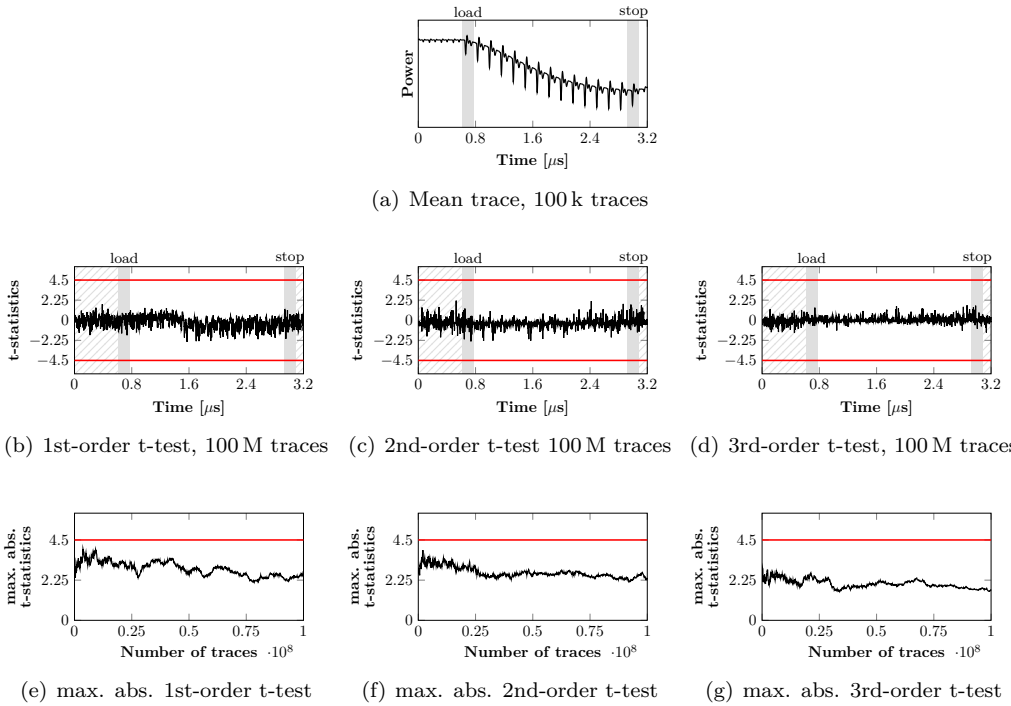
(e) max. abs. 1st-order t-test (f) max. abs. 2nd-order t-test (g) max. abs. 3rd-order t-test

Figure 14: Univariate fixed vs. random t-test results for a second-order CCHPC_{PL}-protected AES with duality applied, 2000 sample points per trace.

5.4.3 AES-128 Evaluation.

We continued our analyses with examining d-order CCHPC_{PL}-protected full AES encryption implementation that incorporates duality, explained in Section 4.2. Since the design could not fit into the Sakura-X FPGA, we instantiate the AES core for each order $d \in [1, 3]$ on the UltraScale KU095 FPGA of the CW241 board. In short, for $d = 1$ the second layer $l = 1$ operates with one clock cycle offset and is duplicated, such that the latency per round is reduced to one clock cycle. Hence, a full AES encryption is accomplished in 12 clock cycles. Similar to the former case study (S-box evaluation), we provided fresh randomness in proper DRP form. When the core is executed once, the primary inputs are loaded into the internal state register before the actual computation of the first round starts. According to the correct application of the duality concept, the duplicated part of the round function in layer $l = 1$ needs to be pre-charged again after both ciphertext shares are generated. Both aforementioned specifications of our deployed architecture are observable in the mean power trace (cf. Figure 13(a)). We examined the first three statistical orders over 100 million traces and none of the t-tests indicates information leakage, as shown in Figure 13.

To further validate the security of higher-order design instantiations, we repeated the measurements for $d \in [2, 3]$. As visible in Figure 14(a), the provable second-order secure AES cipher core requires an additional clock cycle due to the increase in order. Again, no leakage is detected in any of the tested first three statistical orders over 100 million traces (cf. Figure 14). The observations for $d = 3$ are consistent, i.e., the mean power trace reveals an additonal clock cycle for the cipher core execution (compared to $d = 2$), and no leakage is detected in any of the first three statistical orders (cf. Figure 15). We further conducted bivariate t-tests for $d \in [1, 3]$, and no t-value for any pair of sample



(a) Mean trace, 100 k traces

(b) 1st-order t-test, 100 M traces (c) 2nd-order t-test 100 M traces (d) 3rd-order t-test, 100 M traces

(e) max. abs. 1st-order t-test (f) max. abs. 2nd-order t-test (g) max. abs. 3rd-order t-test

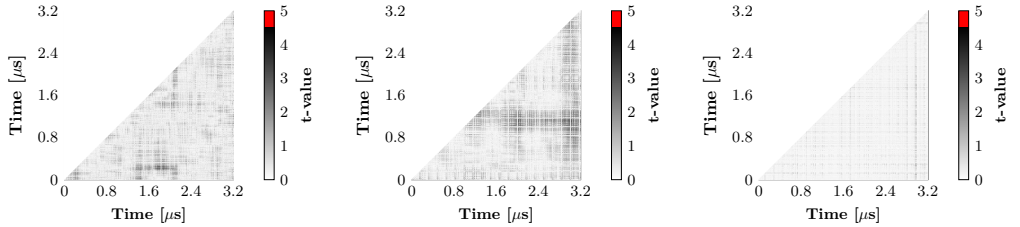
Figure 15: Univariate fixed vs. random t-test results for a third-order CCHPC_{PL}-protected AES with duality applied, 2000 sample points per trace.

points exceeded the threshold of 4.5 (cf. Figure 16).

To examine our setup, we repeated the measurements on the first-order design without fresh randomness, i.e., we turned the PRNG off, meaning that its output is constant for all measurements. The results depicted in Figure 17 show first-order leakage, as expected. Nevertheless, our experimental evaluation demonstrates practical robustness, even at statistical orders greater than the provable security order of the design under test. This is due to three facts: (1) Most of the circuit is realized following the DRP protocol. This potentially reduces the leakage as glitches, which amplify the leakage, are prevented. (2) Shares of a variable belonging to different share indices are mainly processed in different clock cycles. This potentially reduces univariate leakages. (3) The design is considerably large as two full AES rounds are active in parallel (half in pre-charge and half in evaluation phase). This adds noise to the measurements and should decrease the Signal-to-Noise Ratio (SNR). These results are indeed in line with those presented for LMDPL in [SBHM20], where no second-order leakage has been detected from a first-order design.

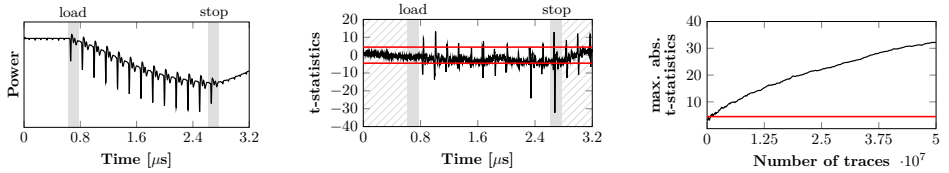
6 Conclusions

In this work, we introduced CCHPC, an innovative low-latency focused gadget-based masking approach that achieves constant-cycle latency independent of the algebraic degree and the circuit size, but increases with the security order d, specifically to d clock cycles. Arbitrary gadget compositions retain this property also for iterative designs by application of duality. We extensively elaborated our foundational decisions that exploit particular characteristics of DRP logic to design the proposed CCHPC gadgets. We discussed how the



(a) $d = 1$, 100 M traces. (b) $d = 2$, 100 M traces. (c) $d = 3$, 100 M traces.

Figure 16: Bivariate fixed vs. random t-test results for first-, second-, and third-order CCHPC_{PL}-protected AES designs with duality applied, 2000 sample points per trace resulting in approximately 2 M unique pairs.



(a) Mean trace, 100 k traces (b) 1st-order t-test, 50 M traces (c) max. abs. 1st-order t-test

Figure 17: Univariate fixed vs. random t-test results for a first-order CCHPC_{PL}-protected AES with duality applied, but no fresh randomness provided, 2000 sample points per trace.

resulting constructions can be utilized in hardware architectures in contrast to conventional approaches. We further demonstrated its practical applicability by presenting a dth-order AES-128 encryption core for an arbitrary d with $11+d$ clock cycle latency. The SCA security of CCHPC was rigorously proven under the RR d-probing model and the state-of-the-art PINI notion while tool-assisted evaluations of isolated gadget instances confirmed our theoretical claims. For composed circuits, we also showed its resistance against transitional leakage. Practical assessments further validate the efficacy of a CCHPC-protected AES S-box implementation and the aforementioned AES encryption function, utilizing the concept of duality, on an FPGA platform. We compared CCHPC with the existing low-latency oriented gadget-based approaches showing that it achieves competitive performance in terms of latency and the associated area overhead. More significantly, it stands out as the only approach in the literature capable of scaling to arbitrary security orders, while maintaining a low latency constant for the desired security order.

Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and project SSIMA - 535533866.

References

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018*, volume 10993 of *LNCS*, pages 427–455. Springer, 2018.

[AR02] Farhad Arbab and Jan J. M. M. Rutten. A coinductive calculus of component connectors. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *WADT 2002*, volume 2755 of *LNCS*, pages 34–55. Springer, 2002.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *ACM CCS*, pages 116–129. ACM, 2016.

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the AES s-box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *IFIP TC SEC*, volume 376 of *IFIP*, pages 287–298. Springer, 2012.

[BSAR06] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan Rutten. Modeling component connectors in reo by constraint automata. *Science of Computer Programming*, 61:75–113, 07 2006.

[CBG+17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does Coupling Affect the Security of Masked Implementations? In *COSADE 2017*, volume 10348 of *LNCS*, pages 1–18. Springer, 2017.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. on Computers*, 2021.

[CGM+24] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. Compress: Generate small and fast masked pipelined circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):500–529, Jul. 2024.

[CGP+12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In *COSADE 2012*, volume 7275 of *LNCS*, pages 69–81. Springer, 2012.

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

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

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR TCHES*, 2021(2):136–158, 2021.

[CSV24] Gaëtan Cassiers, François-Xavier Standaert, and Corentin Verhamme. Low-latency masked gadgets robust against physical defaults with application to ascon. *IACR TCHES*, 2024(3):603–633, 2024.

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

[HHB24] Johannes Haring, Vedad Hadzic, and Roderick Bloem. Closing the gap: Leakage contracts for processors with transitions and glitches. *IACR TCHES*, 2024(4):110–132, 2024.

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

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACMCCS 2022*, pages 1799–1812. ACM, 2022.

[KMMS22] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated generation of masked hardware. *IACR TCHES*, 2022(1):589–629, 2022.

[KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic hardware private circuits towards automated generation of composable secure gadgets. *IACR TCHES*, 2022(1):323–344, 2022.

[LMW14] Andrew J. Leiserson, Mark E. Marson, and Megan A. Wachs. Gate-level masking under a path-based leakage metric. In *CHES 2014*, volume 8731 of *LNCS*, pages 580–597. Springer, 2014.

[MLM24] Nicolai Müller, Daniel Lammers, and Amir Moradi. A deep analysis of two glitch-free hardware masking schemes SESYM and LMDPL. *IACR TCHES*, 2024(3):76–98, 2024.

[MM22] Nicolai Müller and Amir Moradi. PROLEAD A Probing-Based Hardware Leakage Detection Tool. *IACR TCHES*, 2022(4):311–348, 2022.

[MM24] Nicolai Müller and Amir Moradi. Robust but relaxed probing model. *IACR TCHES*, 2024(4):451–482, 2024.

[MMW18] Lauren De Meyer, Amir Moradi, and Felix Wegener. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES. *IACR TCHES*, 2018(3):596–626, 2018.

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

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR TCHES*, 2020(2):300–326, 2020.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

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

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *(DATE*, pages 246–251. IEEE Computer Society, 2004.

[VDB+24] Dilip Kumar S. V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Time sharing - A novel approach to low-latency masking. *IACR TCHES*, 2024(3):249–272, 2024.

A Security Proof of CCHPC$_{\mathsf{RS}}$ and CCHPC$_{\mathsf{PL}}$

For CCHPC$_{\mathrm{RS}}$ and CCHPC$_{\mathrm{PL}}$, we add the public global control signal rst, which allows a simultaneous reset of all registers in layers $l \in [1,d]$. We express this by a Boolean function $\mathsf{h} : \mathbb{F}_2^{(n+p)(2d+1)+2k+1} \to \mathbb{F}_2^{m(2d+1)}$, where $\mathsf{h}(\widehat{\boldsymbol{A}}_0, \ldots, \widehat{\boldsymbol{A}}_{n-1}, \widehat{\boldsymbol{S}}_0, \ldots, \widehat{\boldsymbol{S}}_{p-1}, \boldsymbol{R}, rst) = (\widehat{\boldsymbol{Z}}_0, \ldots, \widehat{\boldsymbol{Z}}_{m-1})$. For CCHPC$_{\mathrm{RS}}$, we define the constraint automaton $\mathcal{C}^{n,p,k,m}$ based on $\mathcal{B}^{n,p,k,m}$ (cf. Section 4) with $\Sigma = \widehat{\Sigma} \cup \boldsymbol{R} \cup rst$ and $d_{rst} \in \{0,1\}$. Further, the transition relations $\to_{\mathrm{prech}_l}$ are adjusted, such that $\forall l \in [1,d], \to_{\mathrm{prech}_l} = q_{l+1} \xrightarrow{\Sigma, g_{\mathrm{prech}}} q_0$, where $g_{\mathrm{prech}} = \forall \boldsymbol{v} \in \Sigma \setminus \Sigma^0, d_{\boldsymbol{v}} = (0,0) \wedge d_{rst} = 1$.

To express the pipelined execution of CCHPC$_{\mathrm{PL}}$, we define another constraint automaton $\mathcal{D}^{n,p,k,m} = (Q^*, \Sigma, \to^*, q_{\mathrm{start}})$ with $Q^* = \{q_{\mathrm{start}}, q_{\mathrm{even}}, q_{\mathrm{odd}}\}$ and $\to^*$ as follows.

$$\to_{\mathrm{start}} = q_{\mathrm{start}} \xrightarrow{\Sigma, g_{\mathrm{start}}} g_{\mathrm{even}},\ g_{\mathrm{start}} = \forall \boldsymbol{v} \in \Sigma \setminus \Sigma^0, d_{\boldsymbol{v}} = (0,0)$$

$$\to_{\mathrm{even}} = q_{\mathrm{even}} \xrightarrow{\Sigma, g_{\mathrm{even}}} q_{\mathrm{odd}},\ g_{\mathrm{even}} = d_{\mathrm{rst}} = 0 \wedge \forall l \in [1,d], \forall \boldsymbol{v} \in \Sigma^l, d_{\boldsymbol{v}} \begin{cases} \neq (0,0) \text{ if } l \text{ even} \\ = (0,0) \text{ if } l \text{ odd} \end{cases}$$

$$\to_{\mathrm{odd}} = q_{\mathrm{odd}} \xrightarrow{\Sigma \setminus \Sigma^0, g_{\mathrm{odd}}} q_{\mathrm{even}},\ g_{\mathrm{odd}} = d_{\mathrm{rst}} = 0 \wedge \forall l \in [1,d], \boldsymbol{v} \in \Sigma^l, d_{\boldsymbol{v}} \begin{cases} \neq (0,0) \text{ if } l \text{ odd} \\ = (0,0) \text{ if } l \text{ even} \end{cases}$$

$$\to_{\mathrm{reset}} = q_{\mathrm{even/odd}} \xrightarrow{\Sigma, g_{\mathrm{reset}}} g_{\mathrm{even}},\ g_{\mathrm{reset}} = d_{\mathrm{rst}} = 1 \wedge \forall \boldsymbol{v} \in \Sigma \setminus \Sigma^0, d_{\boldsymbol{v}} = (0,0)$$

We note that $\mathcal{D}^{n,p,k,m}$ does not enforce the necessary requirement that all primary input shares provided during the same clock cycle must originate from independent sharings. The following proofs leverage the fact that, instead of pre-charging each $\boldsymbol{z}^l$ individually over separate clock cycles, the use of registers effectively blocks glitch propagation (caused by pre-processing) without compromising simulatability.

Proposition 19. *CCHPC$_{\mathsf{RS/PL}}$-AND is a correct CCHPC circuit for $z = a \cdot b$ if $rst = 0$.*

Proof. Suppose that $rst = 0$, then the insertion of registers does not affect the evaluation. Since CCHPC-AND is a correct CCHPC circuit for $z = a \cdot b$, it follows that CCHPC$_{\mathsf{RS/PL}}$-AND is a correct CCHPC circuit for $z = a \cdot b$ if $rst = 0$. □

Proposition 20. *CCHPC$_{\mathsf{RS/PL}}$-AND with $d+1$ shares is glitch-propagation-free.*

Proof. Suppose that $\forall l \in [1,d], G_{a_{\mathrm{t}}^l} = G_{a_{\mathrm{f}}^l} = G_{b_{\mathrm{t}}^l} = G_{b_{\mathrm{f}}^l} = 0$ and $\forall \boldsymbol{r} \in \boldsymbol{R}, G_{r_{\mathrm{t}}} = G_{r_{\mathrm{f}}} = 0$. Further, suppose an arbitrary set of TDSs accepted by $\mathcal{C}^{2,0,\frac{d(d+1)}{2},1}$ (resp. $\mathcal{D}^{2,0,\frac{d(d+1)}{2},1}$). The following three properties hold: (1) for every register output e, it holds that $G_e = 0$, (2) rst is only used as control signal for registers, and (3) CCHPC-AND with $d+1$ shares is glitch-propagation-free. It follows that CCHPC$_{\mathsf{RS/PL}}$-AND with $d+1$ shares is glitch-propagation-free. □

Proposition 21. *CCHPC$_{\mathsf{RS/PL}}$-AND with $d+1$ shares is CCHPC d-PINI.*

Proof. Suppose that $\forall l \in [1,d], G_{a_{\mathrm{t}}^l} = G_{a_{\mathrm{f}}^l} = G_{b_{\mathrm{t}}^l} = G_{b_{\mathrm{f}}^l} = 0$ and $\forall \boldsymbol{r} \in \boldsymbol{R}, G_{r_{\mathrm{t}}} = G_{r_{\mathrm{f}}} = 0$. Further, suppose an arbitrary set of TDSs accepted by $\mathcal{C}^{2,0,\frac{d(d+1)}{2},1}$ (resp. $\mathcal{D}^{2,0,\frac{d(d+1)}{2},1}$). The following three properties hold: (1) for every register output e, it holds that $G_e = 0$, (2) rst is only used as control signal for registers, and (3) CCHPC-AND with $d+1$ shares is CCHPC d-PINI. It follows that CCHPC$_{\mathsf{RS/PL}}$-AND with $d+1$ shares is CCHPC d-PINI. □

B Extension of PROLEAD

B.1 RR d-Probing Security

We begin by revisiting the evaluation of the RR d-probing security using PROLEAD, as introduced in [MM24]. For each RR d-probing adversary, defined by the set of relaxed probes they can place, PROLEAD simulates a set of observations made by the adversary, denoted as $\mathcal{Q}$. These observations, along with the set of corresponding unshared input vectors that produce them, denoted as $\mathcal{X}$, are stored in a two-way contingency table as illustrated in Table 5.

Table 5: Contingency table for two variables $\mathcal{X}$ and $\mathcal{Q}$, with F_{X_i,Q_j} denoting the frequency of observed samples for a particular observation $Q_j \in \mathcal{Q}$ under input vector $X_i \in \boldsymbol{X}$.

F_{X_i,Q_j}	Q_0	...	Q_{n-1}	**Total**
X_0	F_{X_0,Q_0}	...	$F_{X_i,Q_{n-1}}$	$F_{X_i,*}$
...	...	...	...	...
X_{m-1}	F_{X_{m-1},Q_j}	...	$F_{X_{m-1},Q_{n-1}}$	$F_{X_{m-1},*}$
Total	F_{*,Q_0}	...	$F_{*,Q_{n-1}}$	$F_{*,*}$

To evaluate RR d-probing security, PROLEAD verifies whether Equation (1) holds. This is done by confirming the equality of each row in the contingency table. To enable this verification procedure, we made the following changes to PROLEAD.

- Originally, PROLEAD employed random sampling of an incomplete set of input vectors. This approach allowed the verification of larger designs using feasible computational resources but introduced a small probability of false negatives. However, since the number of distinct inputs for gadgets is relatively small, it is now practical to simulate all possible inputs. By doing so, we can eliminate the risk of false negatives. Accordingly, we have modified PROLEAD to exhaustively evaluate all possible input vectors.
- PROLEAD was originally designed to evaluate full cipher cores in a fixed-vs-fixed or fixed-vs-random setting, considering only two distinct groups. We extend PROLEAD to treat each input as its own group, aligning with the exhaustive approach described earlier. This modification enables the assessment of RR d-probing security with zero false-negative probability.

B.2 RR d-PINI

Instead of verifying the statistical independence of observations from the unshared input, assessing common composability notions requires proving the existence of a simulation set satisfying Equation (2). Since Equation (2) utilizes shared input vectors rather than unshared ones, we create a contingency table with the shared input values instead of the corresponding unshared input values. The resulting contingency table is presented in Table 6. Next, we verify whether a given simulation set satisfies Equation (2) by converting the contingency table into an $m \times n$ matrix, denoted as $\mathcal{T}$ which contains all $F_{\widetilde{X}_i,Q_j}$. The matrix $\mathcal{T}$ is then used as input to Algorithm 3, which checks whether the simulation set, provided to the algorithm as an index set I, satisfies the required conditions. For every value the simulation set can take, denoted as S (cf. Line 1), Algorithm 3 builds a new contingency table encompassing all rows of $\mathcal{T}$ whose associated shared input vector $\widetilde{X}_q$ matches S (cf. Lines 4-11). Specifically, in Line 5, we check if particular shared inputs of $\widetilde{X}$, selected by I are equal to the values expected from S. The result of this step is a new matrix $\mathcal{S}$ which encompasses the frequencies of all rows of $\mathcal{T}$ matching to S. Now, it holds that Equation (2) is violated if the rows of $\mathcal{S}$ are not equal which is checked

Table 6: Contingency table for two variables $\widetilde{\mathcal{X}}$ and $\mathcal{Q}$, with $F_{\widetilde{X}_i,Q_j}$ denoting the frequency of observed samples for a particular observation $Q_j \in \mathcal{Q}$ under shared input vector $\widetilde{X}_i \in \widetilde{\mathcal{X}}$.

$F_{\widetilde{X}_i,Q_j}$	Q_0	...	Q_{n-1}	**Total**
$\widetilde{X}_0$	$F_{\widetilde{X}_0,Q_0}$	...	$F_{\widetilde{X}_i,Q_{n-1}}$	$F_{\widetilde{X}_i,*}$
...	...	...	...	...
$\widetilde{X}_{m-1}$	$F_{\widetilde{X}_{m-1},Q_j}$	...	$F_{\widetilde{X}_{m-1},Q_{n-1}}$	$F_{\widetilde{X}_{m-1},*}$
Total	F_{*,Q_0}	...	$F_{*,Q_{n-1}}$	$F_{*,*}$

Algorithm 3 Simulatability Evaluation

Input: $\mathcal{T} \in \mathbb{N}^{m \times n}$ ▷ A matrix containing the frequencies for variables $\widetilde{\mathcal{X}}$ and $\mathcal{Q}$
Input: $\widetilde{\mathcal{X}}, \mathcal{Q}$ ▷ Sorted sets of shared input vectors and observations
Input: I ▷ The set of indices in the shared input vector belonging to the simulation set
Output: $b \in \{\texttt{true}, \texttt{false}\}$ ▷ True if Equation (2) holds, false otherwise

1: **for** $S \in \mathbb{F}_2^{|I|}$ **do**
2: $\quad\mathcal{S} \leftarrow 0_{\frac{m}{2^{|I|}} \times n}$ ▷ Initialize a matrix with zeros
3: $\quad a \leftarrow 0$
4: $\quad$**for** $\widetilde{X}_q \in \widetilde{\mathcal{X}}$ **do**
5: $\quad\quad$**if** $\forall j \in \{0, \ldots |I| - 1\} : s_i = \widetilde{x}_{Ii_j}, s_i \in S, \widetilde{x}_{I_j} \in \widetilde{X}$ **then** ▷ Check if $\widetilde{X}$ matches S
6: $\quad\quad\quad$**for** $c \in \{0, \ldots, n-1\}$ **do**
7: $\quad\quad\quad\quad\mathcal{S}_{a,c} \leftarrow F_{\widetilde{X}_q,Q_c}$ ▷ Add matching rows to a new contingency table
8: $\quad\quad\quad$**end for**
9: $\quad\quad\quad a \leftarrow a + 1$
10: $\quad\quad$**end if**
11: $\quad$**end for**
12: $\quad$**for** $r \in \{0, \ldots, \frac{m}{2^{|I|}} - 1\}$ **do**
13: $\quad\quad$**for** $c \in \{0, \ldots n-1\}$ **do**
14: $\quad\quad\quad$**if** $\mathcal{S}_{0,c} \neq \mathcal{S}_{r,c}$ **then** ▷ Check all rows of $\mathcal{S}$ for equivalence
15: $\quad\quad\quad\quad$**return** `false`
16: $\quad\quad\quad$**end if**
17: $\quad\quad$**end for**
18: $\quad$**end for**
19: **end for**
20: **return** `true`

in Lines 12-18. If such a violation is detected, Algorithm 3 returns `false`. However, if Equation (2) holds for all possible S the algorithm will return `true` indicating that the given simulation set can indeed simulate the adversary. We note that Algorithm 3 must be executed with different index sets I until a suitable simulation set is identified. If no such simulation set exists, Algorithm 3 will return `false` that satisfy the requirements defined in Definition 4. PROLEAD can independently identify all possible I by iterating through the index sets derived from $\widetilde{\mathcal{X}}$ and verifying whether they meet the requirements specified in Definition 4. If a given index set satisfies these requirements, it is passed to Algorithm 3 for further evaluation until a valid simulation set is found.

B.3 Discussion

Finally we want to clarify some points with respect to PROLEAD.

- In this work, we focus exclusively on evaluating the PINI notion under the RR

d-probing model. However, the same methodology can be adapted to assess other composability notions, such as Non-Interference (NI) and Strong Non-Interference (SNI), by modifying the generation of appropriate simulation sets. Moreover, these evaluations can also be done under the robust d-probing model.

- We do not claim that verifying composability with PROLEAD is the most efficient approach. However, since no other tool currently supports evaluation under the RR d-probing model, there is no basis for direct comparison. We believe that integrating verification under the RR d-probing model into widely used formal verification tools, such as VERICA or IronMask, would be a valuable direction for future research, though it is non-trivial and beyond the scope of this work. Nevertheless, PROLEAD's runtime and memory requirements proved feasible for our experiments.

- In our explanation, we focused on exhaustive verification, where all input vectors are simulated exactly once. However, when this approach becomes infeasible – such as for gadgets with high security orders – PROLEAD can alternatively generate a randomly selected, incomplete set of inputs for evaluation. In such cases, checking the equality of all rows is insufficient. Instead, each $\mathcal{S}$ can be analyzed using the G-test, as originally proposed by the PROLEAD authors [MM22]. This approach makes it possible to evaluate larger designs at higher orders, albeit with a small false-negative probability as a trade-off.