IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 3, pp. 603–633. DOI:10.46586/tches.v2024.i3.603-633

Low-Latency Masked Gadgets Robust against Physical Defaults with Application to Ascon

Gaëtan Cassiers[1†], François-Xavier Standaert[1] and Corentin Verhamme[1]

[1] Crypto Group, ICTEAM, UCLouvain, Louvain-la-Neuve, Belgium.

Abstract. Low-latency masked hardware implementations are known to be a difficult challenge. On the one hand, the propagation of glitches can falsify their independence assumption (that is required for security) and can only be stopped by registers. This implies that glitch-robust masked AND gates (maintaining a constant number of shares) require at least one cycle. On the other hand, Knichel and Moradi's only known single-cycle multiplication gadget that ensures (composable) security against glitches for any number of shares requires additional care to maintain security against transition-based leakages. For example, it cannot be integrated in a single-cycle round-based architecture which is a natural choice for low-latency implementations. In this paper, we therefore describe the first single-cycle masked multiplication gadget that is trivially composable and provides security against transitions and glitches, and prove its security in the robust probing model. We then analyze the interest of this new gadget for the secure implementation of the future lightweight cryptography standard Ascon, which has good potential for low-latency. We show that it directly leads to improvements for uniformly protected implementations (where all computations are masked). We also show that it is can be handy for integration in so-called leveled implementations (where only the key derivation and the tag generation are masked, which provides integrity with leakage in encryption and decryption and confidentiality with leakage in encryption only). Most importantly, we show that it is very attractive for implementations that we denote as multi-target, which can alternate between uniformly protected and leveled implementations, without latency overheads and at limited cost. We complete these findings by evaluating different protected implementations of Ascon, clarifying its hardware design space.

Keywords: Lightweight Authenticated Encryption · Masking · Probing Security · Glitches · Transitions · Leakage-Resistance · Leveled Implementations

1 Introduction

The National Institute of Standards and Technology (NIST) recently selected Ascon as a new standard for lightweight cryptography [DEMS21].[1] Ascon is aimed at efficient and low-cost implementations, but is also anticipated to be easier to protect against side-channel analysis than authenticated encryption based on the AES Rijndael, thanks to two relevant features. First, its 5-bit S-box has a simple bit-level representation that only takes five AND gates with an AND-depth of one, which is especially suitable for masking [BMD+20]. As a result, efficient masked implementations have been the focus of a sequence of works, exploiting either Threshold Implementations (TIs) with low security order [GWDE15, GWDE17, KBG+23] or generic solutions with arbitrary security order [GM17, GIB18, VCS22]. Second, its mode of operation enables so-called *leveled*

† Work performed in parts during a post-doctoral stay at IAIK, TU Graz, Graz, Austria.
[1] https://csrc.nist.gov/Projects/lightweight-cryptography

Received: 2024-01-15 Accepted: 2024-03-15 Published: 2024-07-18

implementations, where different parts of the implementations have different security requirements and can therefore be protected with different (more or less expensive) countermeasures [GPPS20, BBC+20, VCS22]. More precisely, this second sequence of works shows that a leveled implementation of Ascon can reach the highest possible integrity with leakage, coined Ciphertext Intergrity with Misuse and Leakage in encryption and decryption (CIML2), and confidentiality against Chosen Ciphertext Attacks with misuse-resilience in case Leakage can only be observed in encryption (CCAmL1). Concretely, such an implementation combines a masked implementation of the Ascon permutation for key derivation and tag generation, and an unprotected (parallel) one for the message processing. By contrast, if confidentiality with leakage in decryption is needed, the only solution is to step back to a (more expensive) *uniformly protected implementation.*

These good features of Ascon, and especially its low AND-depth, are naturally appealing for low-latency implementations. Yet, it is well-known that hardware implementations of cryptographic algorithms combining low latency and side-channel security are difficult to design. This is because such implementations have to resist against physical defaults like glitches [MPG05, MPO05], and established methodologies to prevent the propagation of glitches require register stages [NRS11, FGP+18]. Briefly summarized, most of the known gadgets that provide higher-order composable security require two cycles per AND gate [CGLS21]; first attempts to design low-latency gadgets were limited to low number of shares [ABP+17, MMM21, KSM22, BDMS22]; and the generalizations to arbitrary numbers of shares proposed in [GIB18] and [ZSS+21] were shown to suffer from security flaws [MMSS19] or to leak at reduced security orders when transition-based leakages can be observed [MM22], respectively. As a result, and to the best of our knowledge, the only published masked gadgets combining resistance against glitches and low latency are the low-latency Hardware Private Circuits (HPC3) of Knichel and Moradi [KM22]. Formally, these gadgets are Probe-Isolating Non-Interferent (PINI) [CS20] in the glitch-robust probing model [FGP+18]. It was shown in [CS21] that they ensure composable security against transitions and glitches as long as transitions only occur between parallel evaluations of a gadget. This concretely means that in round-based implementations, there must be a bubble in the pipeline between the execution of consecutive rounds. Unfortunately, when aiming for a low-latency implementation of Ascon with a single-cycle round-based architecture, this condition prevents from using PINI gadgets such as HPC3.[2]

Based on this state-of-the-art, our contributions are twofold.

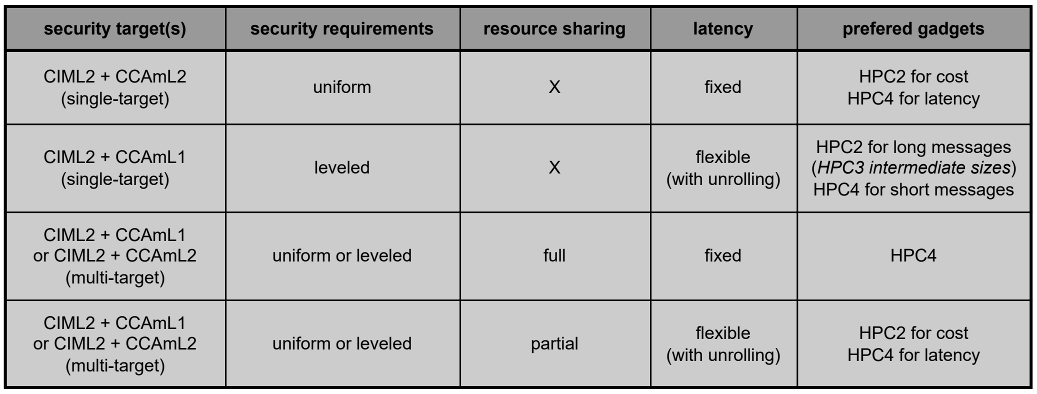
On the one hand, we design and prove the first low-latency masked hardware gadgets that offer unconditional composable security against transitions and glitches, which we denote as HPC4. Formally, these gadgets ensure the Output-Probe-Isolating Non-Interference (O-PINI) security notion [CS21], which guarantees that they can be trivially composed, even in the case of single-cycle per round implementations. The only known O-PINI gadgets have a latency of (at least) two cycles [CS21, KM22]. The HPC4 gadgets have a latency of a single cycle. We argue that they are directly useful for uniformly protected implementations of Ascon with CCAmL2 security, since they halve their latency.

On the other hand, we investigate the interest of such low-latency gadgets at a higher architectural level. So far, the literature has shown that compared to uniformly protected implementations, leveled implementations of Ascon can bring significant performance (e.g., latency, energy) gains at limited area cost if only CIML2 and CCAmL1 security have to be guaranteed [VCS22]. But if one additionally requires CCAmL2 security (i.e., confidentiality with leakage in decryption), Ascon implementations must be protected uniformly.

This naturally leads to the question whether a single implementation could provide both at limited cost, thanks to resource sharing. We show that such *multi-target* designs can

[2] By this, we mean an architecture that performs a single round of the Ascon permutation per cycle.

highly benefit from HPC4 gadgets, which can come in two flavors. In a fully shared design, the same (masked) Ascon permutation is re-used for both CCAmL1- and CCAmL2-secure implementations of the mode, and the masking's randomness generation is frozen during the message processing in case only CCAmL1 is required. In a partially shared design, we still add an unprotected parallel (possibly unrolled) permutation to the architecture for the CCAmL1 part. These investigations lead us to simplify the design space of masked hardware implementations of Ascon providing composable security against both transitions and glitches for arbitrary number of shares, as depicted in Figure 1.



security target(s)	security requirements	resource sharing	latency	prefered gadgets
CIML2 + CCAmL2 (single-target)	uniform	X	fixed	HPC2 for cost HPC4 for latency
CIML2 + CCAmL1 (single-target)	leveled	X	flexible (with unrolling)	HPC2 for long messages (*HPC3 intermediate sizes*) HPC4 for short messages
CIML2 + CCAmL1 or CIML2 + CCAmL2 (multi-target)	uniform or leveled	full	fixed	HPC4
CIML2 + CCAmL1 or CIML2 + CCAmL2 (multi-target)	uniform or leveled	partial	flexible (with unrolling)	HPC2 for cost HPC4 for latency

Figure 1: Masked hardware implementations of Ascon: design space of architectures with composable security against transitions & glitches for arbitrary number of shares.

In summary, if targeting CIML2 and CCAmL2 security with a uniformly protected implementation, the HPC2 gadgets are preferable for cost and HPC4 ones are preferable for latency. If targeting CIML2 and CCAmL1 security with a leveled implementation, the HPC2 gadgets are preferable for long messages (i.e., when the latency of Ascon's masked key derivation and tag generation is amortized) and the HPC4 gadgets are preferable for short messages (i.e., when such an amortization does not take place). These leveled implementations offer a flexible latency, because their message processing is performed by an unprotected parallel implementations of the Ascon permutation, for which it is possible to adapt the level of unrolling. Eventually, in case a multi-target design (that can either guarantee CIML2+CCAmL1 or CIML2+CCAmL2) is desired, the HPC4 gadget is the best solution. This is because its low latency makes it the best option for resource sharing. In the fully shared case, the key derivation, tag generation and message processing can then be performed using the same masked architecture, with limited latency penalty (i.e., one cycle per round, only loosing the flexibility that the single-target leveled implementation enables thanks to unrolling) and energy gains for the CIML2+CCAmL1 mode, for which the masking's randomness generation can be frozen during the message processing part. In the partially shared version, flexible latency is restored by using an additional unprotected parallel (possibly unrolled) implementation of the Ascon permutation and the choice of gadgets follows the same tradeoff as in the single-target uniformly protected case.

We further observe that the HPC3 gadgets in [KM22] provide a combination between local low-latency features and a need to be integrated in an implementation with more than one cycle per round (with a pipeline bubble) to maintain their security guarantees. As such, they stand between the single-cycle HPC4-based implementations and the area/randomness-efficient (but slower) HPC2-based ones. They are therefore most relevant when the masked permutation does not have to achieve very low latency, but should not have a too high latency either. This may for example happen in the case where a single-target leveled implementation is used for encrypting messages of intermediate length. Given the relatively narrow nature of this sweet spot, we leave it as a scope for further investigations. We also note that the table excludes optimizations that specifically target low number of shares (in which case performance improvements are possible – see the aforementioned references).

We finally mention the related works [SBHM20, NGPM22, SBB+23, SBB+22] which propose alternative methodologies for low-latency masked implementation by leveraging different technologies (e.g., asynchronous or and/or dual-rail logic syles) and stronger physical assumptions, recently discussed (and in part falsified) in [LMM23]. As such, they are hardly comparable with the standard CMOS-based solutions we propose. We additionally discuss other possible applications of HPC4 gadgets in Section 7.

2 Background

This section adopts a top-down approach by showcasing how modes of operation can contribute to side-channel security and describing masking as a primitive-level countermeasure. We use this structure in order to provide a global picture on the security requirements that cryptographic designs put on implementers, and on how to ensure these requirements in practice, while also motivating the design of low-latency masked gagdets.

2.1 Mode-level security against leakage

Authenticated Encryption (AE) aims at ensuring both message integrity and confidentiality. This can be formalized both with an "all-in-one" definition as proposed by Rogaway and Shrimpton [RS06] or with the composite definitional framework of Guo et al. [GPPS19]. We use the second option which allows leveraging the fact that integrity with leakage and confidentiality with leakage impose quite different security requirements, which is the basis of leveled implementations [BBC+20]. It relies on the following notions:

Definition 1 (Ciphertext Integrity with Leakage (informal))**.** In the Ciphertext Integrity with Leakage (CIL) security game, the adversary can perform a number of queries to encryption and decryption oracles enhanced with leakage functions, that capture the implementation of an AE scheme. Her goal is to produce a valid fresh ciphertext. The implementation is secure if the adversary can only succeed with negligible probability.

Definition 2 (Confidentiality with leakage (informal))**.** In the Chosen Ciphertext Attack with Leakage (CCAL) security game, the adversary can perform a number of queries to encryption and decryption oracles enhanced with leakage functions, that capture the implementation of an AE scheme. During a so-called "challenge query", she picks up two fresh messages X_0 and X_1 and receives a ciphertext Y_b encrypting X_b for $b \in 0, 1$, with the corresponding leakage. Her goal is to guess the bit b. The implementation is secure if the adversary can only succeed with negligible advantage over a random guess.

These two games can be played in different variants, corresponding to more or less powerful adversaries. A first axis of variants specifies whether leakage can be observed in encryption only (which is reflected by a "1" in the notations) or in encryption and decryption (which is reflected by a "2" in the notations). For example, CIL1 and CCAL1 stand for integrity and confidentiality with leakage in encryption only, CIL2 and CCAL2 stand for their counterpart with additional leakage in decryption. Another axis of variants specifies whether the implementation is assumed to be nonce-respecting or if some kind of nonce control is allowed. We consider the possibility of nonce misuse-resistance [RS06] (which is reflected by a capital M in the notations) when targeting integrity with leakage, and the possibility of nonce misuse-resilience [ADL17] (which is reflected by a small m in the notations) when targeting confidentiality with leakage. In the first case, the nonce can be misused even in the challenge query. In the second case, misuse is excluded for the challenge query, which aims to ensure that although confidentiality may be lost when the nonce is under adversarial control, it is restored as soon as fresh nonces are used. For example, CIML1 and CIML2 stand for integrity with misuse-resistance and leakage, CCAmL1 and CCAmL2 stand for confidentiality with misuse-resilience and leakage.[3]

Thanks to this definitional framework, it is possible to analyze the security of a mode of operation with leakage under different physical assumptions for its different parts, in turn leading to leveled implementations. Ascon's leveling opportunities have been formally investigated in [GPPS20] and are summarized in Figure 2. Informally, CIML2 security only requires to protect the key derivation and tag generation against Differential Power Analysis (DPA), leaving the message processing part unprotected; CCAmL1 security additionally requires that the message processing part is protected against Simple Power Analysis (SPA); CCAmL2 security finally requires a uniform protection against DPA. By DPA (resp., SPA), we mean side-channel attacks where the adversary can observe the encryption of many plaintexts (resp., a few plaintexts) with the same key. We can already observe that taking advantage of Ascon's leveling opportunities requires knowing the security target (e.g., for confidentiality) in advance. This raises the question whether an implementation with a security target selected at execution time can be built more efficiently than with the naive solution where multiple designs are just implemented independently.

2.2 Masking as primitive-level countermeasure

Figure 2 shows the different security targets that can be ensured by Ascon, with the security requirements they imply for the implementation of its permutation. In our hardware implementation context, security against SPA can be obtained quite efficiently, by just using an architecture with a sufficient level of parallelism [USS+20, BMPS21]. As a result, the main challenge to implement Ascon securely lies in the permutations that must resist against DPA. For this purpose, the standard approach in the literature is to rely on masking [CJRR99]. We next provide the background necessary to introduce our new low-latency gadgets that provide composable security against transitions and glitches.

From now on, we denote a random variable by lowercase letter : $x \in \mathbb{F}_2$ denotes a binary random variable. In the particular case of masking we will denote by x_i the i-th share of the x variable such that $\bigoplus_{\forall i} x_i = x, i \in \{0, \dots, d-1\}$ where d denotes the number of shares. Besides, drawing a value uniformly at random from a set is denoted as $x \xleftarrow{\$} S$.

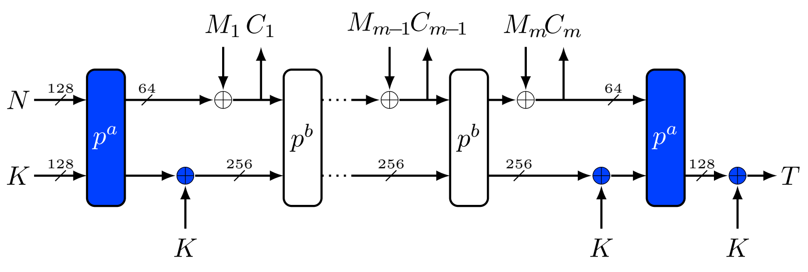
2.2.1 Circuit definition

The circuit model we use comes from [CS21]. We keep its formalism but restrain some of the notions to fit our needs that are limited to a single-cycle pipelined gadgets. This model first designates physical circuits as sets of gates and wires:

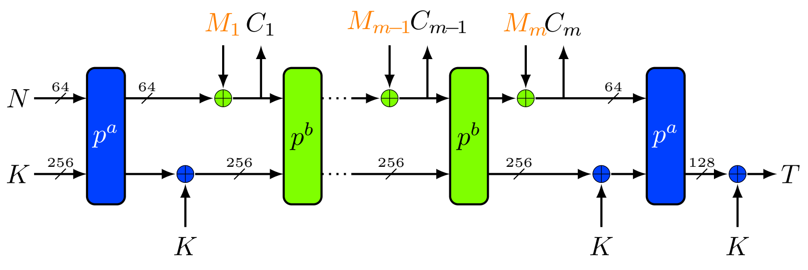
Definition 3 (Structural gate)**.** A structural gate is a tuple (I, P, f, lat), where

- I is the set of inputs of the gate,
- P is the set of parameters and the union of public and private parameters P^P, P^S,
- $f : (I \to \mathbb{F}_q) \times (P \to S) \to \mathbb{F}_q$ (where S is any set) is the evaluation function,
- $\mathsf{lat} : I \to \mathbb{N}$ is the latency of the inputs (i.e., one if the input is required one clock cycle before the output is produced, zero if it is required at the same cycle).

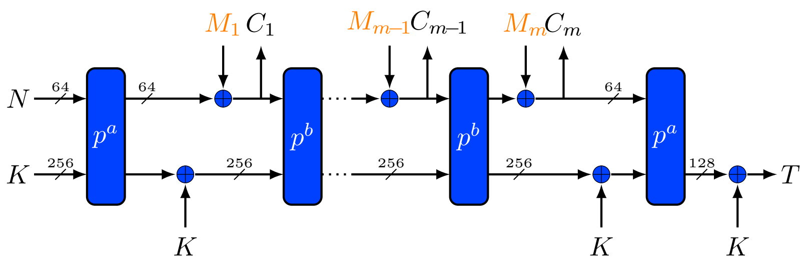
[3] As discussed in [GPPS19], this restriction is needed because confidentiality with leakage and nonce misuse-resistance can only be obtained with unrealistic (leak-free) requirements for the implementers.



(a) CIML2-secure implementation.



(b) CCAmL1-secure implementation.



(c) CCAmL2-secure implementation.

Figure 2: Leveled implementation of Ascon for different security targets. The blue blocks have to be protected against DPA, the green ones have to be protected against SPA and the white ones do not require protection against side-channel leakage.

Definition 4 (Structural wire)**.** A structural wire is a pair $(g, (g', i))$ which connects a (source) gate g to the input i of a (destination) gate g', denoted as (g', i).

Merging these two notions, one can define structural circuits :

Definition 5 (Structural circuit)**.** A structural circuit is a directed graph whose nodes are structural gates and whose edges are structural wires. A wire connects its source to its destination. In a structural circuit, there must be no combinational loop (i.e., there must be no cycle for which all the wires have a destination with latency 0).

We then need to define the execution of a structural circuit:

Definition 6 (Circuit execution)**.** The execution of a structural circuit $C = (G, W)$ for the set of cycles $T = \{t_0, \dots, t_{L-1}\}$ is a directed graph whose set of nodes is $G \times T$ (the gates) and whose set of edges is $W \times T$ (the wires). Wires connect gates according to their latency: let $w = (g, (g', i))$ be a structural wire and (g', t) a gate, the wire (w, t) connects its source $(g, t - \mathsf{lat}_{g'}(i))$ to its destination (g', t), where $\mathsf{lat}_{g'}$ is the latency function of the gate g'. If the source does not exist, then the wire is connected to a fresh "initial state" source gate (i.e., a no-input gate having as output a public parameter). Each gate may be annotated with a parametrization function, mapping the set of all (resp., public) parameters of the underlying structural gate to values, in which case the execution is full (resp., partial).

Now that objects are formally defined, we next introduce a security notion over them.

2.2.2 Probing model and robust probing model

We will use the following two definitions in our contributions.

Definition 7 (Probing security [ISW03])**.** A set of probes P in a gadget execution G is a subset of its gates and input wires. In an evaluation of G with input values x, the values of the probes are denoted as $G_P(x)$. The sensitive values are defined as the sums of the values on the wires of each input sharing. A gadget execution G is secure against a set of probes P if $G_P(x)$ is independent of the sensitive values when the inputs x are uniformly distributed. G is t-probing secure if it is secure against any P such that $|P| \leq t$.

A more generic model exists that models physical defaults thanks to extended probes:

Definition 8 (Robust probing security [FGP+18])**.** A probe expansion scheme is a function that maps a gadget execution to a larger set of extended probes, which are sets of probes in the gadget execution. The probes in those sets are named expanded probes. A gadget execution is t-robust probing secure with respect to a probe expansion scheme if it is secure against all the expanded probes from any set of t extended probes.

In this work, we will be interested in glitch- and transition-extended probes. Informally, glitch-extended probes turn any probe on combinatorial wire into all its (registered) inputs (for registered input wires, there is no extension); and transition-extended probes turn any probe on a register into a pair of values that correspond to consecutive invocations. When combining both, the set of extended probes is obtained by first computing the set transition-extended probes and then replacing every expanded probe in a transition-extended probe with all the expanded probes contained in the glitch-extended probe that correspond to it. Physically, this models the fact that the propagation of a glitch (e.g., its timing) depends on both the new and the previous values on the wire.

2.2.3 Circuit composition

Proving the robustness to probing becomes computationally hard when the size of the circuit to verify and the number of shares increase [BBD+15, BGI+18, BBC+19, KSM20]. A standard approach to enable the more efficient verification of complex circuits is to require the gadgets to ensure some (stronger) composability properties [BBD+16].

We first introduce the simulatability framework that we will use in our results:

Definition 9 (Simulatability [BBP+16])**.** A set of probes P in a gadget execution G can be simulated by a set of input shares $I = (i_1, j_1), \ldots, (i_k, j_k)$ if there exists a randomized simulator algorithm S such that the distributions $G_P(x_{*,*})$ (the values of the probes) and $S(x_{i_1,j_1}, \ldots, x_{i_k,j_k})$ are equal for any value of the inputs $x_{*,*}$.

Definition 10 (Glitch+Transition robust Simulatability [CS21])**.** A set of extended probes P in a gadget execution G can be simulated by a set of input shares $I = (i_1, j_1), \ldots, (i_k, j_k)$ if there exists a randomized simulator algorithm S such that the distributions $G_P(x_{*,*})$ and $S(x_{i_1,j_1}, \ldots, x_{i_k,j_k})$ are equal for any value of the inputs $x_{*,*}$.

Next, the PINI property is used to capture trivial composition:

Definition 11 (Probe-Isolating Non-Interference [CS20])**.** Given a gadget execution G, let I be a set of at most t_1 probes on its internal wires and O a set of probes on its output shares. Let A be the set of the share indexes of the shares in O, and $t_2 = |A|$. Let I and O be chosen such that $t_1 + t_2 \leq t$. The gadget G is t-PINI iff for all I and O there exists a set of at most t_1 share indexes B such that observations corresponding to I and O can be simulated using only the shares with indexes $A \cup B$ of each input sharing.

Algorithm 1 HPC2 AND gadget with d shares.

Require: Sharings $\mathbf{x}$, $\mathbf{y}$
Ensure: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

```
1:  for $i = 0$ to $d-1$ do
2:      for $j = i+1$ to $d-1$ do
3:          $r_{ij} = r_{ji} \xleftarrow{\$} \mathbb{F}_2$
4:      end for
5:  end for
6:  for $i = 0$ to $d-1$ do
7:      for $j = 0$ to $d-1, j \neq i$ do
8:          $u_{ij} \leftarrow (x_i \oplus 1) \otimes \mathsf{Reg}\,[r_{ij}]$
9:          $v_{ij} \leftarrow y_j \oplus r_{ij}$
10:     end for
11: end for
12: for $i = 0$ to $d-1$ do
13:     $z_i \leftarrow \mathsf{Reg}\,[x_i \otimes \mathsf{Reg}\,[y_i]] \oplus \bigoplus_{j=0,j\neq i}^{d-1}(\mathsf{Reg}\,[u_{ij}] \oplus \mathsf{Reg}\,[x_i \otimes \mathsf{Reg}\,[v_{ij}]])$
14: end for
```

Eventually, the O-PINI property is used to capture trivial composition with glitches and transitions. It can be viewed as a strengthening of PINI, where the simulator must simulate the outputs with same share index as the one on which an input share index is probed:

Definition 12 (Output Probe-Isolating Non-Interference [CS21])**.** Given a gadget execution G, let I be a set of at most t_1 probes on its internal wires and O a set of probes on its output shares. Let A be the set of the share indexes of the shares in O, and $t_2 = |A|$. Let I and O be chosen such that $t_1 + t_2 \leq t$. The gadget G is t-O-PINI iff for all I and O there exists a set of at most t_1 share indexes B such that observations corresponding to I, O and output shares with index in B can be simulated using only the shares with indexes $A \cup B$ of each input sharing. We say a gadget is O-PINI if it is t-O-PINI for any t.

It has been proven in that such gadgets can be trivially composed:

Theorem 1 (Trivial composition of O-PINI gadgets [CS21])**.** *Let S_i be a set of pipeline t-O-PINI structural gadgets, and let G_i be their executions. If the structural gadget S is a composition of S_i gadgets, then it is t-O-PINI.*

2.2.4 Hardware Private Circuits

Hardware Private Circuits are adaptations of Ishai et al.'s seminal private circuits [ISW03], adapted in order to better cope with physical defaults that are typical from hardware implementations. This generally comes at the cost of stronger contraints on their latency (since registers are instrumental to avoid the propagation of glitches) and larger randomness requirements. We next describe the two main gadgets that will drive our discussions (while there exists other HPC gadgets, they are not relevant for our use-case).

First, the HPC2 gadget introduced in [CGLS21] is described in Algorithm 1. It achieves trivial composition against glitches in two clock cycles (with respect to one of its inputs, and only one cycle with respect to the other one) and requires $d\frac{(d-1)}{2}$ fresh random bits per evaluation (with d shares). Since HPC2 is glitch-robust PINI but not glitch-robust O-PINI, it is not trivially composable in presence of transitions, although a more restricted form of composition is possible, which may come with added latency [CS21].

Second, the HPC3 multiplication gadget introduced in [KM22] is described in Algorithm 2. It achieves trivial composition against glitches in one clock cycle and requires

Algorithm 2 HPC3 AND gadget with d shares.

Require: Sharings $\mathbf{x}$, $\mathbf{y}$
Ensure: Sharing $\mathbf{z}$ such that $z = x \cdot y$.

```
for $i = 0$ to $d-1$ do
    for $j = i+1$ to $d-1$ do
        $r_{ij} = r_{ji} \xleftarrow{\$} \mathbb{F}_2$
        $r'_{ij} = r'_{ji} \xleftarrow{\$} \mathbb{F}_2$
    end for
end for
for $i = 0$ to $d-1$ do
    for $j = 0$ to $d-1, j \neq i$ do
        $u_{ij} \leftarrow \mathsf{Reg}\,[y_j \oplus r_{ij}]$
        $v_{ij} \leftarrow \mathsf{Reg}\,\left[(x_i \oplus 1) \otimes r_{ij} \oplus r'_{ij}\right]$
        $w_{ij} \leftarrow \mathsf{Reg}\,[x_i] \otimes u_{ij} \oplus v_{ij}$
    end for
end for
for $i = 0$ to $d-1$ do
    $z_i \leftarrow \mathsf{Reg}\,[x_i \otimes y_i] \oplus \bigoplus_{j=0,j\neq i}^{d-1}(w_{ij})$
end for
```

$d(d-1)$ fresh random bits per evaluation. Similarly to HPC2, HPC3 is PINI but not O-PINI, with the same impact on composability with transition-based leakages.

Let us remark that HPC2 and HPC3 have O-PINI variants (O-PINI1/2 and HPC3+ respectively), which cost one additional clock cycle of latency and $d-1$ additional random bits. Hence, no existing solution allows single-cycle round-based implementations.

3 The HPC4 gadget

In this section, we introduce the new single-cycle HPC4 AND gadget, prove that it is glitch-robust O-PINI and discuss its performance figures compared to HPC2 and HPC3.

Let us first recall why the HPC2 and HPC3 gadgets are not O-PINI, which will give some motivation for the design choices of HPC4. For HPC2, and looking at Algorithm 1, a probe on $u_{ij} = (x_i \oplus 1) \otimes r_{ij}$requires knowledge of x_i and r_{ij} to be simulated.[4] Therefore, following the O-PINI definition, z_i should be simulated which, due to glitches, means that the value $x_i \otimes (y_j \oplus r_{ij})$ should be simulated. This in turn means that $y_j \oplus r_{ij}$ must be known to the simulator, which, combined with the simulation of r_{ij} means that y_j must be known as well. This contradicts the definition of O-PINI: it should be possible to simulate one probe using only one input share. For HPC3, the issue is similar: if there is a probe in the computation of v_{ij}, then x_i, r_{ij} and r'_{ij} are observed. Therefore, z_i must be simulated. With glitches, it leads to the same requirement on the simulation of $y_j \oplus r_{ij}$.

From these examples, we can see that the core issue comes from (*i*) y_j being masked by a single random value, which (*ii*) appears in a computation with x_i, and that (*iii*) glitches on the output shares z_i leak a lot of information about the intermediate computations in the gadget. We may try to fix the issue by removing any of these conditions. Eliminating (*iii*) is the approach of existing O-PINI gadgets: they essentially refresh the shares z_i, but this has a latency cost. We therefore look at the other solutions: eliminating (*ii*) appears difficult while preserving the correctness of the gagdet. By contrast, for (*i*), we mask y_j

[4] When discussing the values of wires, registers are irrelevant and therefore omitted for readability.

using two independent random values, which appear each in a separate computation with x_i. Hence, probing one of these computations does not immediately reveal y_j.

For the rest of the design, we take then inspiration from HPC3, leading to computations:

$$z_i = \mathsf{Reg}\left[x_i \otimes y_i\right] \oplus \bigoplus_{i \neq j} \mathsf{Reg}\left[x_i\right] \otimes \mathsf{Reg}\left[y_j \oplus r_{ij} \oplus r'_{ij}\right] \oplus \mathsf{Reg}\left[x_i \otimes r_{ij} \oplus r''_{ij}\right] \oplus \mathsf{Reg}\left[x_i \otimes r'_{ij} \oplus r'''_{ij}\right]. \quad (1)$$

Let us note that, compared to HPC3, we remove the NOT gate on x_i (which can also be done for HPC3 without impacting its security [CGM+23]). The values r''_{ij} and r'''_{ij} are sampled uniformly at random from $\mathbb{F}_2$ and are symmetric: $r''_{ij} = r''_{ji}$, $r'''_{ij} = r'''_{ji}$, which is needed to ensure the correctness of the gadget. The values r_{ij} and r'_{ij} are also sampled uniformly at random from $\mathbb{F}_2$, but r_{ij} and r_{ji} are independent, while $r'_{ij} = r'_{ji}$ (these two values could be independent, but this would lead to a higher randomness usage). The motivation for having independent r_{ij} and r_{ji} comes from a probe on $u_{ji} = y_i \oplus r_{ji} \oplus r''_{ji}$. Indeed, simulating such a probe requires knowledge of y_i, which means that z_i, and therefore $\mathsf{Reg}\left[y_j \oplus r_{ij} \oplus r''_{ij}\right]$ must be simulated. If the random values in u_{ji} and u_{ij} were equal, then simulation would require knowledge of y_j, and the gadget would not be O-PINI.

3.1 Security proof

HPC4 is described in Algorithm 3, for a finite field $\mathbb{F}_q$. We now prove that it is O-PINI.

Proposition 1. *The gadget HPC4 is a pipeline and glitch-robust t-O-PINI for $t = d - 1$.*

Proof. First of all, we trivially observe that HPC4 is a pipeline. Next, we assume without loss of generality that the extended probes are placed at the input of registers or on output shares, since the set of wires observed by any other extended probe is always a subset of the one observed by one of these probes. For the sake of simplicity, we also ignore the presence of the register $\mathsf{Reg}\left[x_i y_i\right]$, which is added only for synchronization, and is not needed for security. We therefore obtain for any i and for all $j \neq i$ the following probes. The probe z_i yields $\{x_i, y_i, x_i \mathsf{Reg}\left[y_j + r_{ij} + r'_{ij}\right], x_i r_{ij} + r''_{ij}, x_i r'_{ij} + r'''_{ij}\}$; the probe u_{ij} provides $\{y_j, r_{ij}, r'_{ij}\}$; the probe v_{ij} reveals $\{x_i, r_{ij}, r''_{ij}\}$; and the probe w_{ij} gives $\{x_i, r'_{ij}, r'''_{ij}\}$.

Given a set of extended probes P on intermediates values and probed output shares A, the set of required additional input shares' indices B is computed as follows:

1. Initialize $X \leftarrow \emptyset$.
2. For each probed output z_i in A, add i to X.
3. For every pair $i < j$:
   (a) If at least two of v_{ij}, w_{ij}, u_{ij}, v_{ji}, w_{ji}, u_{ji} belong to P, add i and j to X,
   (b) Else, if i (resp., j) already belongs to X, and one of the above probes belongs to P, add j (resp., i) to X,
   (c) Else, if u_{ji}, v_{ij} or w_{ij} belongs to P, add i to X,
   (d) Else, if u_{ij}, v_{ji} or w_{ji} belongs to P, add j to X.
4. Let $B = X \setminus A$.

By construction we observe that $|B| \leq |P|$.

We now build a simulator that uses the input shares with index in $X = A \cup B$ to simulate all the required values. First, we observe that by construction of X, any probe

v_{ij}, w_{ij} or u_{ij} in P yields the knowledge of the input share needed to compute it, and can therefore be perfectly simulated by following Algorithm 3. Next, we simulate all the probes z_i for all $i \in X$. For every such z_i, since x_i and y_i are known to the simulator, it remains to simulate $\mathsf{Reg}\,[u_{ij}]$ for all $j \neq i, j \notin X$ (for the other values, this is trivially done by following Algorithm 3), which we can do with fresh randomness.

Only the last step of simulating $\mathsf{Reg}\,[u_{ij}]$ for $i \in X$, $j \notin X$ is not trivially correct. By construction of X, we know that for such $\mathsf{Reg}\,[u_{ij}]$, at most one of v_{ij}, w_{ij} and u_{ji} is probed, while u_{ij}, v_{ji} and w_{ji} are not probed. If v_{ij} (resp., w_{ij}) is probed, then, thanks to the fresh random r'''_{ij} (resp., r''_{ij}), the random r'_{ij} (resp., r_{ij}) is independent of all the observations of the adversary except $\mathsf{Reg}\,[u_{ij}]$, therefore $\mathsf{Reg}\,[u_{ij}]$ appears as a fresh random. If u_{ji} if probed, then r_{ij} is not observed except through $\mathsf{Reg}\,[u_{ij}]$. □

Algorithm 3 HPC4 AND gadget with d shares.

```
Require: shares $(x_i)_{0\le i\le d-1}$ and $(y_i)_{0\le i\le d-1}$, such that $\sum_i x_i = x$ and $\sum_i y_i = y$.
Ensure: shares $(z_i)_{0\le i\le d-1}$, such that $\sum_i z_i = xy$.
  for $i = 0$ to $d-1$ do
    for $j = i+1$ to $d-1$ do
      $r_{ij} \overset{\$}{\leftarrow} \mathbb{F}_q$
      $r_{ji} \overset{\$}{\leftarrow} \mathbb{F}_q$
      $r'_{ij} = r'_{ji} \overset{\$}{\leftarrow} \mathbb{F}_q$
      $r''_{ij} = -r''_{ji} \overset{\$}{\leftarrow} \mathbb{F}_q$
      $r'''_{ij} = -r'''_{ji} \overset{\$}{\leftarrow} \mathbb{F}_q$
    end for
  end for
  for $i = 0$ to $d-1$ do
    for $j = 0$ to $d-1$, $j \neq i$ do
      $u_{ij} \leftarrow y_j + r_{ij} + r'_{ij}$
      $v_{ij} \leftarrow x_i r_{ij} + r''_{ij}$
      $w_{ij} \leftarrow x_i r'_{ij} + r'''_{ij}$
    end for
  end for
  for $i = 0$ to $d-1$ do
    $z_i \leftarrow \mathsf{Reg}[x_i y_i] + \sum_{j=0,j\neq i}^{d-1} (\mathsf{Reg}[x_i]\,\mathsf{Reg}[u_{ij}] - \mathsf{Reg}[v_{ij}] - \mathsf{Reg}[w_{ij}])$
  end for
```

As a sanity check, and to avoid mistakes in our verilog implementation of HPC4, we ran SILVER [KSM20] on it, for $d = 2, 3$ (it did not complete quickly for higher number of shares). This ensured that our implementation of the gadget is PINI. Unfortunately, to the best of our knowledge, there is no public verification tool for O-PINI.

3.2 Comparison of HPC multiplication gadgets

We conclude this section on the HPC4 gadget with a high-level evaluation of its performance figures. We also compare it with natural competitors, namely HPC2, HPC3 and HPC3+. First, Table 1 summarizes the security guarantees, the latency and the randomness complexity of these gadgets. Next, Table 2 provides the gate counts of the same gadgets. Finally, we make these results more concrete by providing post-synthesis results in Table 3. Overall, these results essentially confirm the high-level intuitions outlined in introduction. Namely, gadgets with stronger security guarantees (e.g., PINI vs. O-PINI) or with lower latency (e.g., HPC3+ vs. HPC4) are more expensive (in gates and in randomness).

Table 1: High-level comparison of HPC AND gadgets.

Scheme	Security w/ glitches	Latency	Randomness	Field
HPC2 [CGLS21]	PINI	1 & 2	$\frac{d(d-1)}{2}$	$\mathbb{F}_2$
HPC3 [KM22]	PINI	1	$d(d-1)$	$\mathbb{F}_2$
HPC3+ [KM22]	O-PINI	2	d^2-1	$\mathbb{F}_2$
HPC4 [This]	O-PINI	1	$\frac{5d(d-1)}{2}$	Arbitrary

Table 2: Gate counts of HPC AND gadgets (instantiated on $\mathbb{F}_2$).

Gadget	$\otimes$	$\oplus$	NOT	REG
HPC2	$2d^2-d$	$3d^2-3d$	d	$4d^2-2d$
HPC3	$2d^2-d$	$4d^2-4d$	d	$3d^2-2d$
HPC3+	$2d^2-d$	$4d^2-3d$	d	$3d^2$
HPC4	$3d^2-2d$	$6d^2-6d$	0	$4d^2-3d$

We finally note that the HPC2 gadget of Algorithm 1 is insecure in larger fields than $\mathbb{F}_2$ [CGLS21] while the HPC3 gadget of Algorithm 2 is only proven secure in $\mathbb{F}_2$ [KM22], but it can be adapted to work in larger fields – see the recent [CGM+23].

4 Single-target implementations of Ascon

In this section, we move to the integration of the HPC4 (and other relevant) gadget(s) into the single-target implementations reported in the upper part of Figure 1. Before diving into the topic, we explain how we implemented the underlying permutation used in the modes. We also discuss how higher-level design choices impact the security requirements on the gadgets in use, and show that integrating the HPC3 gadget into a round-based implementation of the Ascon permutation leads to leakage. With these preliminaries in mind, we first consider a uniformly protected implementation targeting CIML2 for integrity with leakage and CCAmL2 for confidentiality with leakage in Section 4.2, we then consider a leveled implementation targeting CIML2 for integrity with leakage and CCAmL1 for confidentiality with leakage in Section 4.3. We finally evaluate performances for such implementations instantiated with the relevant gadgets in Section 4.4.

4.1 Implementations of the Ascon permutation

The Ascon AEAD is made of two permutations p^a and p^b that iteratively apply an SPN-based round transformation p that consists of three steps : p_C (a constant addition), p_S (a non-linear S-box layer) and p_L (a linear diffusion layer). The mentioned permutations p^a and p^b differ by the number of rounds and the initial round constant.

4.1.1 Unprotected implementations

We will use the following unprotected implementations:

- Round-based: a full transformation p is computed, then stored in a register to be fed back to the transformation. This implementation computes one round per cycle.
- Unrolled: instead of computing a single transformation, one can use more physical resources to implement u times the transformation p and perform u rounds per cycle.

Table 3: Area comparison of HPC AND gadgets. Post-synthesis results for the NanGate 45 Open Cell Library obtained with Cadence Design Vision design toolchain.

Scheme	d	Latency [cycle]	Randomness [bit]	Area [GE]
HPC2	2	2	1	86.6
	3	2	3	216
	4	2	6	402.6
HPC3	2	1	2	74.6
	3	1	6	177
	4	1	12	330.6
HPC3+	2	2	3	89
	3	2	8	200
	4	2	15	362.3
HPC4	2	1	5	102.3
	3	1	15	260
	4	1	30	487.3

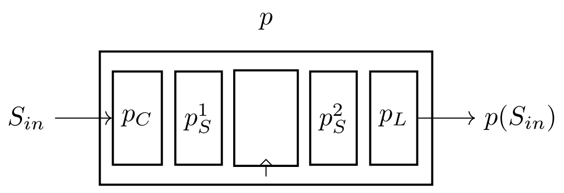


Figure 3: Masked Ascon primitive implementation: p_c denotes the round constant addition, p_s denotes the substitution layer that is split between the operations prior to the AND gate and the operations that use its output. For the operands that are not used in the AND gate, it require a synchronization register. The linear layer p_l is applied over the state and $p(S_{in})$ designates a round of the Ascon permutation made over the state S_{in}.

We will next denote a round-based implementation as rb and an unrolled implementation as unr-u where u is the number of round processed between two registers layers. These unprotected implementations will be used in the message processing parts of the mode.

4.1.2 Protected implementations

We will use the following protected implementations:

- Round-based: as depicted in Figure 3, a protected round-based implementation computes the same layers but take advantage of the register layer in the p_S step. Hence the substitution operation is split between the variables that are input to the AND operation (p_S^1) and the one that are computed with its output (p_S^2).
- Serialized: in order to reduce the high cost of masked gadget (shown in Table 3) we can leverage serialized implementations that trade latency for area by implementing only a fraction of the required logic gates and looping over them.

We will next denote a round-based implementation (which, as will be shown next, can only be implemented with HPC4) as rb. Our serialized implementation uses 16 S-boxes and is implemented with HPC2, looping on an 80 bit states before merging them for the p_L transformation. We denote this implementation as unr. These protected implementations will be used in the key derivation and tag generation parts of the mode.

4.1.3 Insecurity of HPC3 in a round-based Ascon permutation

In section Section 3, we raised the theoretical concern that transitions can harm the security of masked implementations. The HPC4 gadget offers strong security guarantees even in this case, whereas the HPC3 gadget does not. As discussed in [CS21], this is not always a problem and there are examples of (e.g., serial) architectures where transitions do not compromise security. In this section, we nevertheless show that in the specific case of single-cycle round-based architectures that is motivated by our low-latency goal, such transitions lead to concrete weaknesses. For this purpose, we study the integration of the HPC3 gadget in a round-based masked implementation of the Ascon permutation.

Informally, in such a round-based implementation of Ascon-p, the inputs and outputs of the p_S layer are not isolated by registers (see Figure 3). Therefore, when using gadgets that are not robust against transitions, the extended probes can recombine over the same gadget and lead to reduce the security order of the implementation. The HPC3 gadget, which is only PINI and not O-PINI, is typically vulnerable to such leakages. We confirmed this issue by using the PROLEAD tool [MM22] and analyzed the security of a toy implementation where a single Ascon S-box is looping on itself. This provides a subset of the leakage that a full Ascon implementation leads to and is already sufficient to identify critical leakage with the HPC3 gadget. (We note that running the tool on the full Ascon is not possible since the memory cost of the PROLEAD verification rapidly explodes in the presence of extended probes that "touch" many bits). We tested the same round-based architecture with HPC3 and HPC4 gadgets gadgets in the fixed vs. random setup with up to 2,500,000 simulations. Significant leakage was consistently flagged (during the second cycle of the S-box computation) when using HPC3 gadgets, while the detection threshold (of 5) was never reached when using HPC4 gadgets. We refer to [MMSS19, MKSM22] for in-depth discussions of how such robust probing security issues translate into concrete attacks.

4.2 CIML2+CCAmL2 with uniform masking

This first implementation achieves the highest security level (i.e., a combination of CIML2 and CCAmL2) at the cost of having all its operations being uniformly protected with masking. It is typically relevant when confidentility is required for a leaking decryption (e.g., which may be the case with software updates, movies or video games). This architecture is described in Figure 4. We keep the same color code as in Figure 2: the blue color, which reflects leaking operations that must be secure against DPA, is used everywhere. We then instantiate the masked Ascon permutation with either a round-based (parallel) HPC4-based implementation, since this is the architecture that best benefits from the gadget's low-latency features, or an 80-bit serialized HPC2-based implementation, since the higher latency of this gadget is amortized in this serial architecture.

The mode was implemented carefully in order to avoid the introduction of architectural-level leakages. In particular, the sensitive key additions are performed by muxing the key with a vector made of zeros of the same size. This mux is controlled by a flag that passes through a register in order to provide a clean edge and remove gliches. Another relevant mechanism that is included in our designs is the possibility to stall the design. Such a feature is usefull for applications where the data feed may not be synchronyzed with the cryptographic core. This stalling module is located just before the data processing, so that one can stall and directly unfreeze the design in order to process the data.

4.3 CIML2+CCAmL1 with leveling

This second implementation drops the CCAmL2 security requirement and only achieves confidentiality with leakage in encryption (CCAmL1), while preserving the highest integrity guarantees (CIML2). It corresponds to the leveled design of Figure 5 which, as reflected by

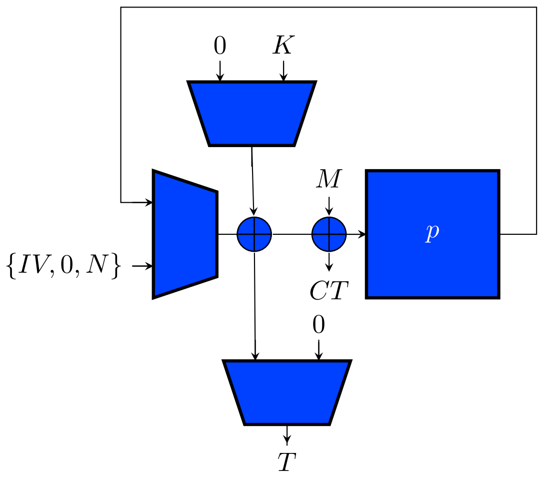


Figure 4: Uniformly protected implementation ensuring CIML2 and CCAmL2.

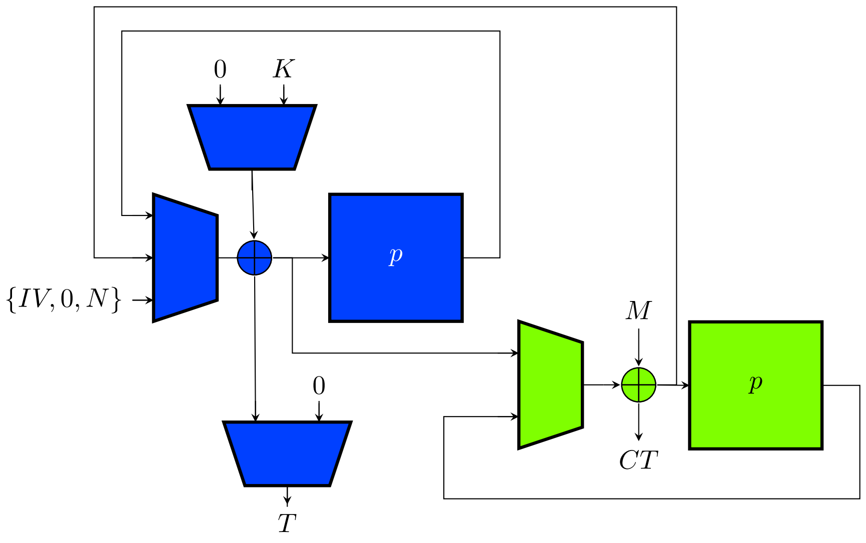


Figure 5: Leveled implementation ensuring CIML2 and CCAmL1.

the green color, leverages weaker (SPA) protections for the permutations used during the message processing (the key derivation and tag generation still require DPA protections).

Informally, the protected permutation processes the initial state to produce a fresh ephemeral key which is forwarded to the unprotected permutation. The unprotected permutation then loops over until the data is fully absorbed, and the final state is fed to the protected primitive again, to generate the tag. The masked permutation is the same as previously described (instantiated with HPC2 or HPC4 gadgets). The unprotected core achieves SPA security through a parrallel hardware implementation. As a result, this part of the implementation can be unrolled in order to achieve a lower latency.

4.4 Performance evaluation

We implemented the different single-target Ascon architectures in the NanGate 45 Open Cell Library, for designs synthetized using the Cadence Design Vision toolchain. Area

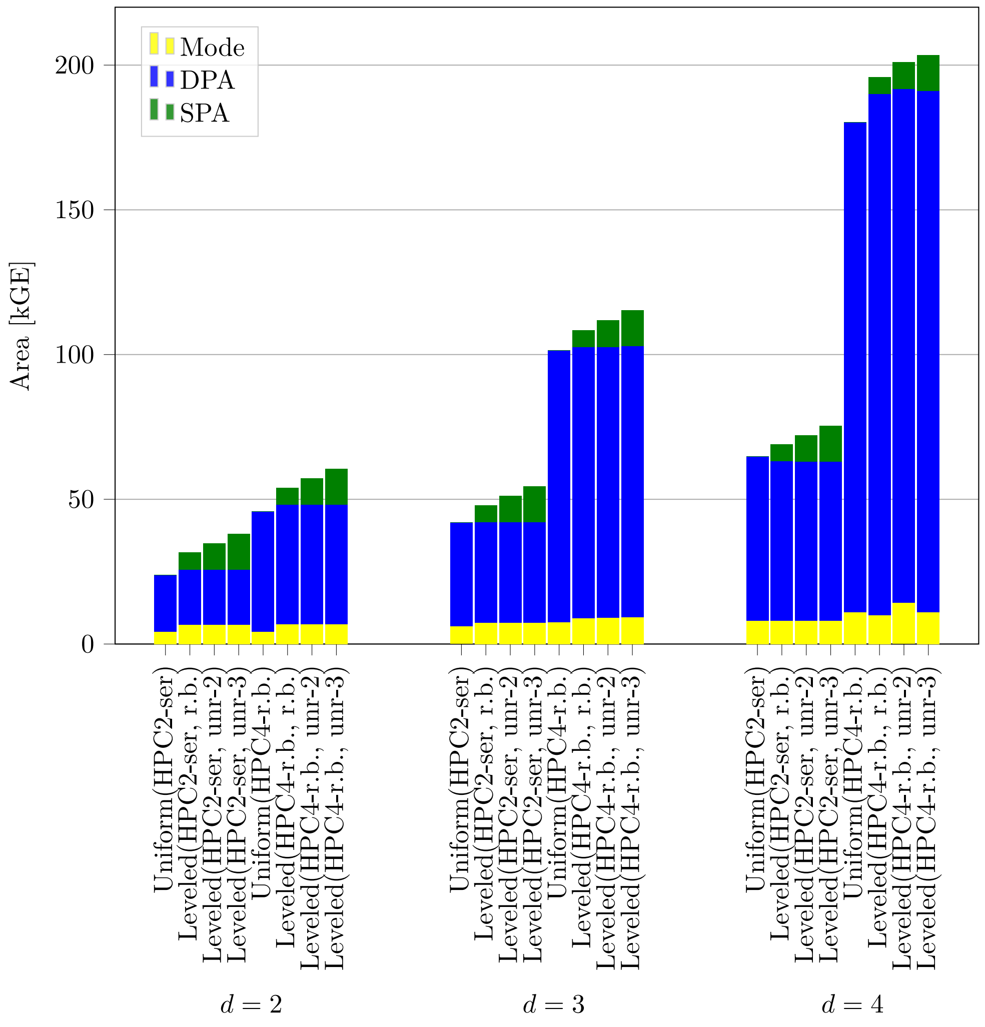


Figure 6: Single-target implementations: area comparisons.

comparisons are depicted in Figure 6. For completeness, we additionally report the exact gate counts (in kilo-gate equivalent) in Appendix A, Table 6. As for the latencies of the modes, they are depicted on Figure 7 and we also provide the explicit equations.

First, when using a uniformly protected implementation, we have:

$$L_u(m) = 24c + \left\lceil \frac{m+1}{64} \right\rceil * 6c, \tag{2}$$

which gives the latency of the mode of operation when encrypting an m-bit message, where $c = 6$ cycles if using HPC2-ser gadgets and $c = 1$ cycle if using HPC4-r.b. gadgets.

Next, when using a leveled implementation, we have:

$$L_l(m) = 24 * c + \left\lceil \frac{m+1}{64} \right\rceil * \frac{6}{u}, \tag{3}$$

where u is the number of rounds computed in one clock cycle for the unprotected implementation used in the message processing part of the mode. Similarly to the uniform case, $c = 6$ cycles (resp., $c = 1$ cycle) when using HPC2-ser (resp., HPC4-r.b.) gadgets for the key derivation and tag generation parts of the authenticated encryption mode.

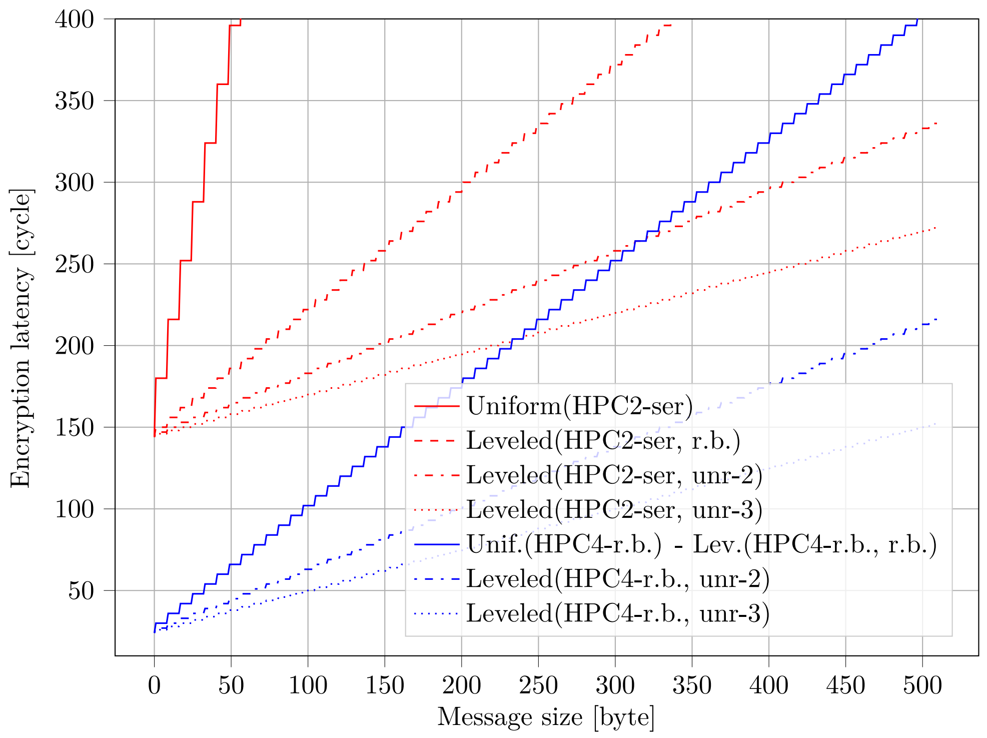


Figure 7: Single-target implementations: latency comparisons.

These evaluations confirm the intuitive recommendations outlined in introduction (Figure 1). In the uniformly protected implementation case, HPC2 is less expensive than HPC4 but implies significant latency overheads. In the leveled case, unrolling the unprotected implementation provides significant latency gains while the choice between HPC2 and HPC4 provides a cost vs. latency tradeoff. Overall, leveling comes at limited area cost and is therefore relevant in case only CCAmL1 security is needed.

Comparison with state of the art implementations. The work in [GIB18] reports an area of 42.75 kGE and consumes 2048 bits of randomness per cycle for a uniformly protected implementation. Our uniformly protected implementation with HPC4 requires and area of 46.66 kGE and consumes 1600 bits of randomness per cycle. In term of latency, they both achieve one round per cycle (see the continuous blue line on Figure 7). We therefore achieve similar performances while provably avoiding the flaws put forward in [MMSS19]. The work in [VCS22] proposes a leveled implementation where the key derivation and tag generation function are instantiated with 80-bit serialized implementations using HPC2 gadgets, while the message is processed by a round-based unprotected implementation. Their reported area is 43 kGE while we achieve 31.5 kGE for a similar architecture using similar latencies (see the dashed red line on Figure 7). We additionally offer the possibility to unrol the unprotected implementation. These comparisons are summarized in Table 4.

5 Multi-target implementations

In this section, we move the the integration of the HPC4 (and other relevant) gadget(s) into the multi-target implementations reported in the lower part of Figure 1. These implementations provide a run-time selection between two modes: either CIML2 and CCAmL2 security with uniform protection or CIML2 and CCAmL1 security with leveled protection.

Table 4: Comparison between the state of the art and our proposed solutions.

Reference	Architectures	Area (kGE)	Latency (cycle for specified blocks)		
			1	4	10
[VCS22]	(HPC2-ser, r.b.)	43	150	168	204
This		31.5	150	168	204
[GIB18]	GLM	42.75	30	48	84
This	HPC4-r.b.	46.66	30	48	84

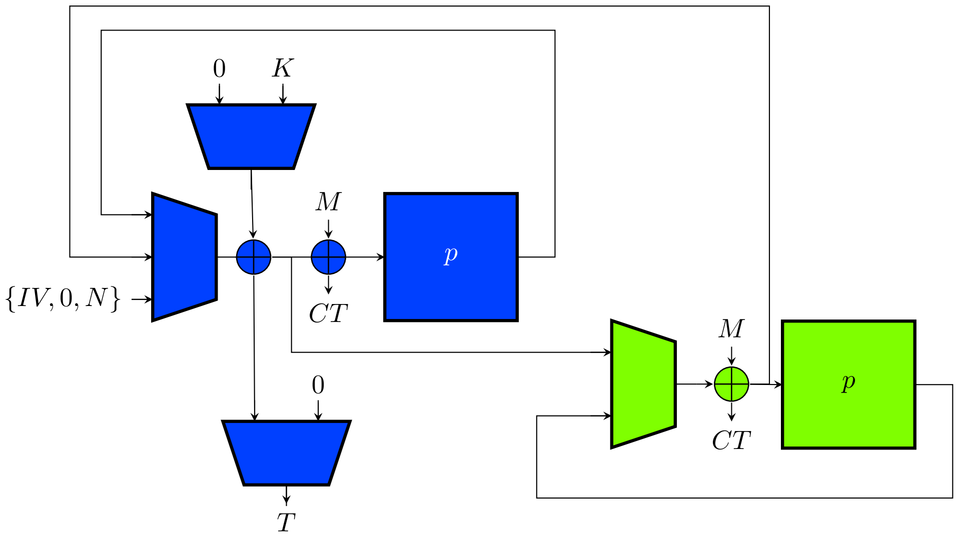


Figure 8: Multi-target implementation with partial resource sharing.

We propose two such implementations. The first one, described Section 5.1, instantiates one masked permutation (used for both modes) and a non-protected permutation which enables high throughput in the leveled mode. The other one, described in Section 5.2, instantiates a single masked permutation unit that can operate in a non-protected way, by clock gating most of its state, thereby saving energy in the CIML2+CCAmL1 mode. We finally evaluate performances for such implementations in Section 5.3.

5.1 Partial resource sharing with unrolling

We first design a multi-target implementation with an architecture similar to the leveled implementation of Section 4.3. It is represented in Figure 8 and mixes the two target security levels of the previous section with both a masked and an unprotected permutation. This implementation requires additional resources due to the control logic needed to interconnect the primitives. Its main advantage is to maintain the possibility to unroll the unprotected permutation, in order to improve the latency. The masked permutation also offers some flexibility, since it can be instantiated with HPC2 or HPC4 gadgets.

5.2 Full resource sharing without unrolling

We next push the resource sharing the previous implementation even further, by exploiting specific properties of the HPC4 gagdet. The main idea of this last architecture is to reuse the same (low-latency) masked permutation for both the DPA-protected and SPA-protected parts of the implementation, in order to reduce the area requirements while keeping the energy gains of a leveled implementation when only CCAmL1 is required for confidentiality

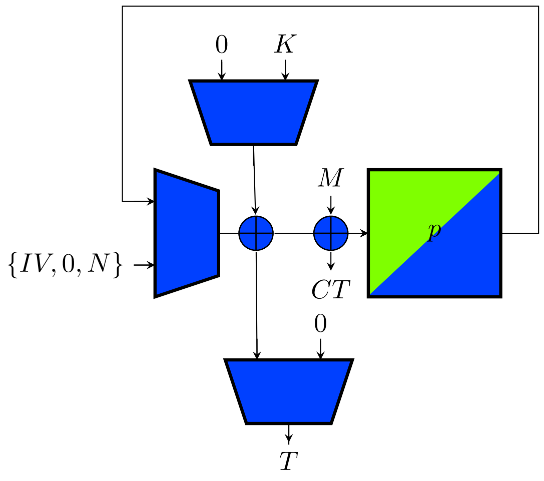


Figure 9: Multi-target implementation with full resource sharing.

with leakage (since the making's randomness can then be frozen during the message processing). This optimization comes at the cost of being limited to a single-cycle round architecture in this case as well (hence removing the possibility to unroll the unprotected permutation as in the partial sharing case). Such a design is depicted in Figure 9.

Concretely, this last design computes the SPA-protected permutation using the DPA-protected one by clock gating $d-1$ bits of each sharing and keeping the last bit running with the value on which to aim to compute. This is directly achievable for affine operations (i.e., it only require to coherently use the selected index), but requires a slight modification of HPC4 to perform the AND gate. Indeed, since the masked gadet mixes different shares, it is not possible to just ignore the other shares. To solve this issue, $d-1$ output shares of the HPC4 gadget are set to 0, while the last share forwards the already existing AND gate performed over $x_i y_i$. This modified AND gate design is represented on Figure 10.

5.3 Performance evaluation

We implemented different multi-target Ascon architectures in the NanGate 45 Open Cell Library, for designs synthetized using the Cadence Design Vision toolchain. Area comparisons are depicted in Figure 11. For completeness, we additionally report the exact gate counts (in kilo-gate equivalent) in Appendix A, Table 7. Latencies are the same as in Equation 2 and Equation 3. It should however be noted that the shared implementation cannot be unrolled and its latency is calculated by considering only $u = 1$.

The area figures essentially suggest that the overheads of a multi-target design (over a single-target one) are limited. For example, the full resource sharing option allows us to reduce the 2-share design area to 58 [kGE]. It additionally offers the possibility to perform an unprotected Ascon operation (e.g., for hashing). However we note that the mild overhead of a partially shared implementation comes with the capacity to unroll the unprotected implementation, leading to significant gains in latency.

This table depicts an interesting part of Ascon's design space. From a cost-performance point of view, two sweetspots clearly emerge, with an expensive but cycle-efficient Ascon-HPC4 and an area-efficient but more cycle-intensive Ascon-HPC2. Overall, we believe the mix of implementations in this and the previous section nicely reflect Ascon's good implementation features and can cover a wide range of applications.

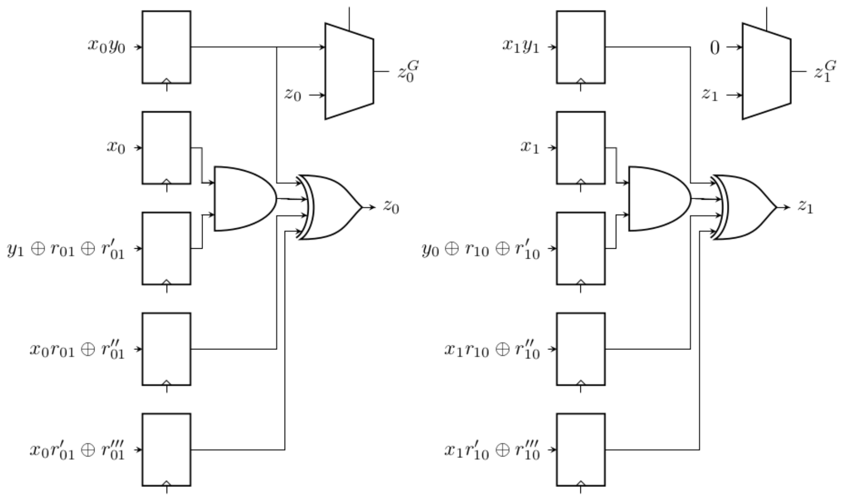

x_0y_0
x_0
$y_1 \oplus r_{01} \oplus r'_{01}$
$x_0r_{01} \oplus r''_{01}$
$x_0r'_{01} \oplus r'''_{01}$
z_0
z_0^G
z_0
x_1y_1
x_1
$y_0 \oplus r_{10} \oplus r'_{10}$
$x_1r_{10} \oplus r''_{10}$
$x_1r'_{10} \oplus r'''_{10}$
0
z_1
z_1^G
z_1

Figure 10: HPC4 gate adapted for sharing (examplary example for $d = 2$).

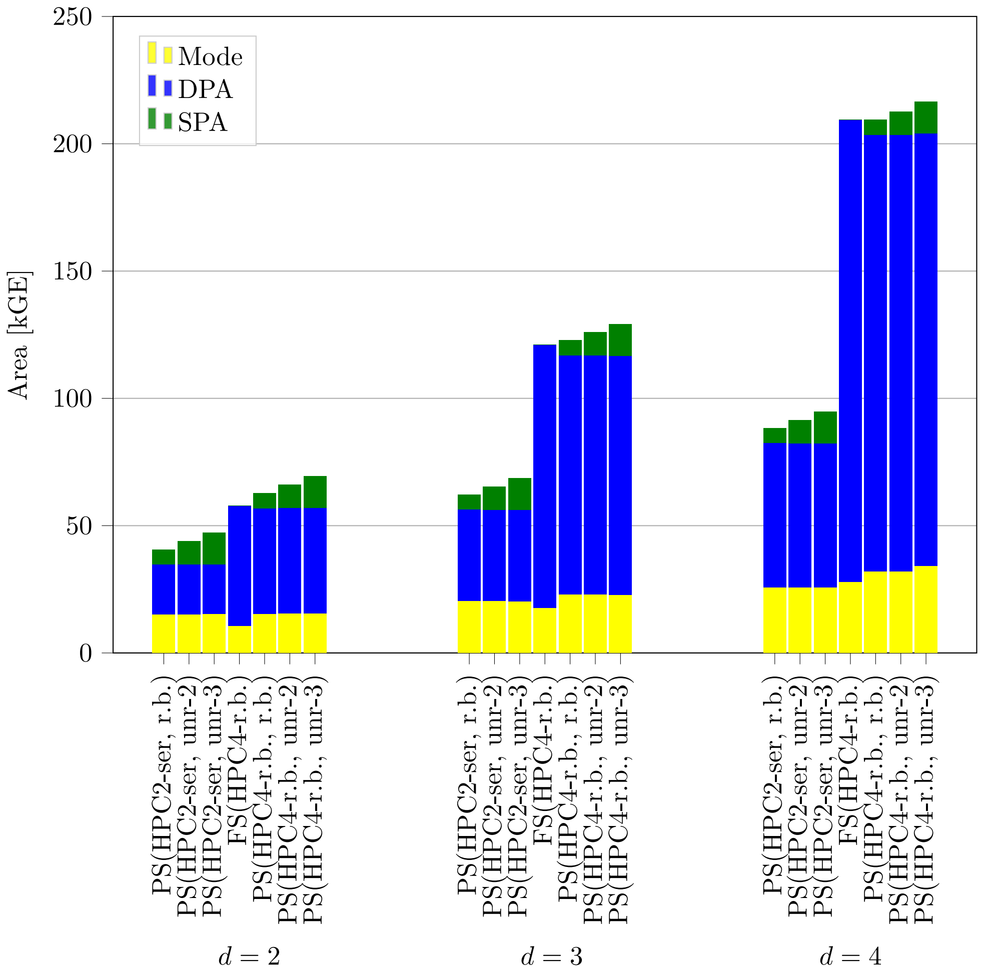


Figure 11: Multi-target implementations: area comparisons.

6 Randomness generation

Our evaluations so far excluded the cost of randomness generation. In this section, we therefore complete our investigations by analyzing this part of the implementation figures. For this purpose, we use the Trivium stream cipher which, as recently discussed in [CMM+23], provides a good tradeoff between strong masking randomness that should not weaken the security of the implementations and good performances.

Starting with generalities, we observe that since the cost of the PRNG is directly related to the amount of randomness required, the corresponding overheads will be the same for single- and multi-target designs, and depend on the underlying architecture of the protected primitive (serialized/round-based) and its masking scheme (HPC2/HPC4).

More concretely, for $d = 2$ (resp., $d = 3, 4$), the (serial) HPC2 implementation of the Ascon permutation requires 80 (resp., 240, 480) bits per cycle, for which a single (resp., two, four) Trivium instance(s), each one enabling to unroll up to 128 bits, is sufficient. By contrast, the (round-based, parallel) HPC4 one requires 1600 (resp., 4800, 9600) bits for $d = 2$ (resp., $d = 3, 4$), which requires 13 (resp., 38, 75) Trivium instances.

We implemented a selection of the previously presented designs using HPC2 and HPC4 gadgets in the NanGate 45 Open Cell Library, using the Cadence Design Vision toolchain. Area comparisons are depicted in Figure 12. For completeness, we additionally report the exact gate counts (in kilo-gate equivalent) in Appendix A, Table 8. Latencies are (again) the same as given by Equation 2 and Equation 3. Overall, these additional results essentially amplify the previous observations. HPC4-based implementations remain the most adequate when low-latency is the main design goal. Yet, their excellent performances come with significant area and, as shown in this section, randomness overheads – the optimization of which is an important open problem that we extend in conclusions.

7 Other applications

As argued in introduction, Ascon is a timely target for implementations with HPC4 gadgets: it is being standardized by the NIST and has good features for side-channel security, both in terms of latency and leveling opportunities. In this section, we briefly discuss additional primitives that can benefit from HPC4 gadgets. We start with other low-latency ciphers in Section 7.1 and follow with iterative AND computations that can be used by authenticated encryption schemes or even post-quantum (public-key) cryptographic algorithms.

7.1 Low-latency ciphers

Among the cipher using quadratic substitution layers, one illustration is Keccak and the SHA-3 standard, of which the permutation shares the low-latency features of Ascon. We give results for the (most standard) 1600-bit version SHA3-512. Another one is the authenticated encryption mode Ketje which is derived from Keccak (see `https://keccak.team/ketje.html`), which is more comparable to Ascon, but does not benefit from its strong model-level security guarantees. We give results for the 400-bit version Ketje Sr (which has a size close to Ascon). Our area (in kilo-gate equivalent) and latency results for uniformly protected implementations of these algorithms are in Table 5. Other examples can be found in the literature, like the Gimli permutation [BKL+17].

7.2 Iterative AND reduction

The computation of the conjuction of many bits, also known as the "AND reduction" of a vector of bits, appears in many cryptographic algorithms, for example when verifying

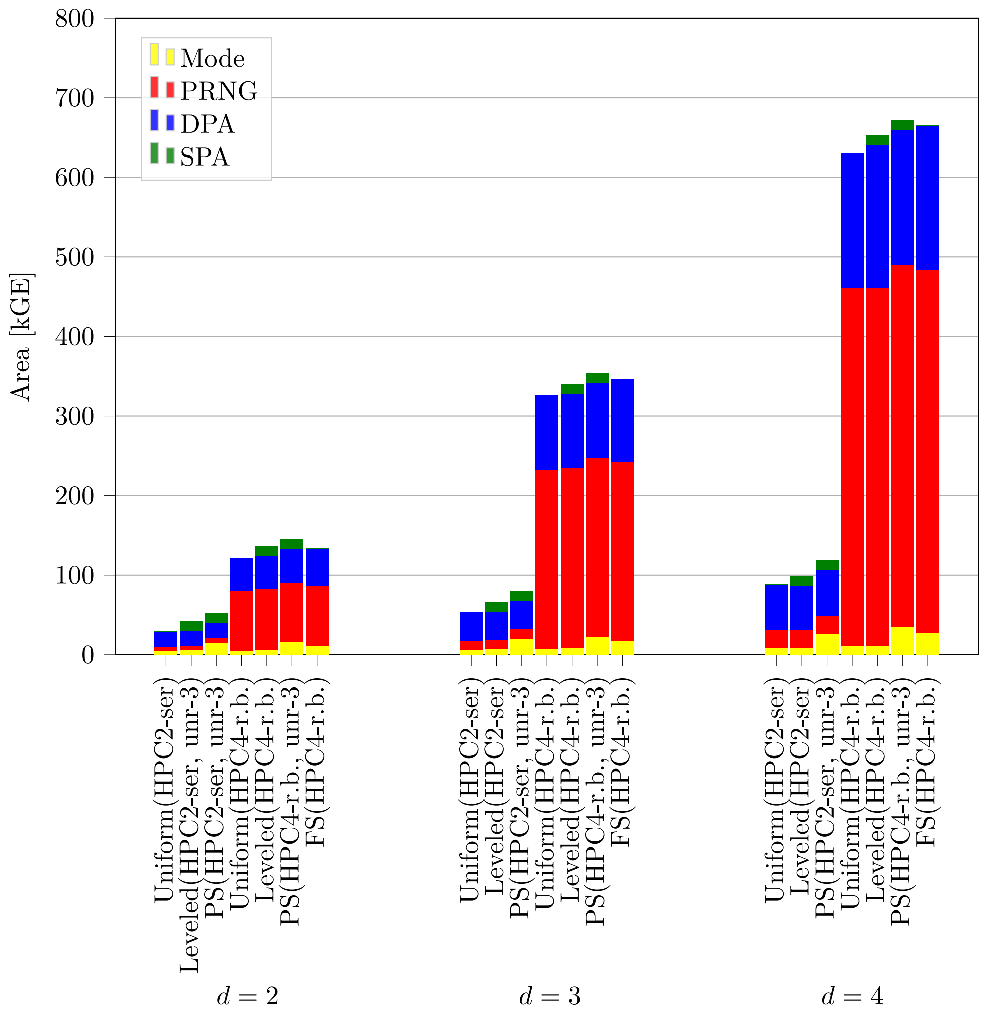


Figure 12: Examplary implementations with randomness generation: area comparisons.

Table 5: Performance characteristic for low-latency uniformly implementations of Ascon, SHA3-512 and Ketje Sr using HPC4 gadgets: area in [kGE] and latency results for 1, 4 and 10 blocks of 576 bits (the specified block size of SHA-512).

d	Primitive	Area (kGE)	Latency (in cycle to process the amount of bits)		
			576	2034	5760
2	SHA3-512	207.7	48	192	480
	Ketje-SR	50.3	39	85	201
	Ascon	45.8	78	216	564
3	SHA3-512	594	48	192	480
	Ketje-SR	118	39	85	201
	Ascon	100	78	216	564
4	SHA3-512	1072.5	48	192	480
	Ketje-SR	215.8	39	85	201
	Ascon	179	78	216	564

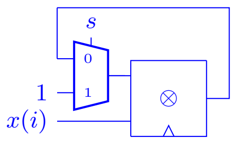


Figure 13: Iterative AND architecture.

a tag of an AE or when verifying component-wise properties of vectors in lattice-based cryptography (e.g., this happens for ciphertext equality checks in Kyber [ABD+19] or for norm checks in Dilithium [DKL+18]). In both cases, leakage on the AND reduction can lead to security issues requiring masking [BMPS21, UXT+22, ABC+23].[5] Therefore, these algorithms require verifying that all the bits of two η-bit vectors v and v' are identical, without leaking about these bits. For this purpose, a standard option is to compute the difference $x = \overline{v \oplus v'}$ and to output the conjunction of this difference to test equality.

Figure 13 depicts a simple architecture to perform this computation at a rate of one bit of x per cycle. The MUX selects the sharing of 1 to reset ($s = 1$) for one clock cycle, then activates the loop ($s = 0$) to iteratively process the AND with x's bits afterward. The throughput of this circuit can be increased by adding a pre-processing pipeline circuit connected to the input x that computes the AND of a chunk of the input bit vector. Due to its single-cycle interative architecture, this implementation may exhibit composition issues when the AND gadget is not O-PINI, leading to lower-order leakage on the value of the accumulator bit, that is, on the conjunction of a subset of the bits of x.

As a first step towards demonstrating that HPC4 gadgets fix concrete leakages that HPC3 gadgets can lead to in non-pipeline designs, we show in Appendix A that the design of Figure 13 leads to exploitable leakage when instantiated with HPC3 gadgets, that vanish if we make the design pipeline or replace the HPC3 gadgets by HPC4 ones.

8 Conclusion

In this paper, we investigated low-latency implementations of the Ascon cipher that ensure side-channel security in the presence of glitches and transitions. For this purpose, we introduced a new single-cycle gadget that can implement a secure AND gate in this setting, while existing solutions required at least two cycles. We then evaluated how to integrate this (and other relevant) gadgets in implementations of Ascon offering different tradeoffs between security and performance, which enable taking advantage of the interplay between mode-level and primitive-level countermeasures. We hope these investigations can better highlight the interesting features of lightweigth cryptographic algorithms like Ascon in order to provide high side-channel security levels at acceptable cost. We also expect these investigations to shed a light over the possibility offered by the Hardware Private Circuits to achieve low-latency with Ascon and other ciphers. As usual for such investigations, further improving performance figures remains an important scope for further research. In particular we believe that the cost of randomness can become a bottleneck for higher-order masked implementations. As a result, investigating how to boost these results is an important next step. Interestingly, and besides gains at the gadget level that could be obtained, design-specific optimizations enabling randomness re-use have already been introduced in the literature [FPS17, BDZ20, FKS+22], and are therefore promising candidates for combination with HPC4 or improvements thereof.

[5] In the AE case, leakage-resilient tag verification can serve as alternative [BPPS17, DM21].

Acknowledgments.

Gaëtan Cassiers and François-Xavier Standaert are respectively post-doctoral researcher and senior research associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by European Union through the Horizon project 1010706275 (acronym REWIRE) and the ERC Advanced Grant 101096871 (BRIDGE). Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[ABC$^+$23] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):58–79, 2023.

[ABD$^+$19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43, 2019.

[ABP$^+$17] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic keccak: SCA security and low latency in HW. *IACR Cryptol. ePrint Arch.*, page 1193, 2017.

[ADL17] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting authenticated encryption robustness with minimal modifications. In *CRYPTO (3)*, volume 10403 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2017.

[BBC$^+$19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskverif: Automated verification of higher-order masking in presence of physical defaults. In *ESORICS (1)*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.

[BBC$^+$20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

[BBD$^+$15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

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

[BBP+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

[BDMS22] Tim Beyne, Siemen Dhooghe, Amir Moradi, and Aein Rezaei Shahmirzadi. Cryptanalysis of efficient masked ciphers: Applications to low latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):679–721, 2022.

[BDZ20] Tim Beyne, Siemen Dhooghe, and Zhenda Zhang. Cryptanalysis of masked ciphers: A not so random idea. In *ASIACRYPT (1)*, volume 12491 of *Lecture Notes in Computer Science*, pages 817–850. Springer, 2020.

[BGI+18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal verification of masked hardware implementations in the presence of glitches. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.

[BKL+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A cross-platform permutation. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 299–320. Springer, 2017.

[BMD+20] Begül Bilgin, Lauren De Meyer, Sébastien Duval, Itamar Levi, and François-Xavier Standaert. Low AND depth and efficient inverses: a guide on s-boxes for low-latency masking. *IACR Trans. Symmetric Cryptol.*, 2020(1):144–184, 2020.

[BMPS21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):641–676, 2021.

[BPPS17] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Trans. Symmetric Cryptol.*, 2017(3):271–293, 2017.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[CGM+23] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. Compress: Generate small and fast masked pipelined circuits. Cryptology ePrint Archive, Paper 2023/1600, 2023. https://eprint.iacr.org/2023/1600.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CMM+23] Ga"etan Cassiers, Lo"ıc Masure, Charles Momin, Thorben Moos, Amir Moradi, and François-Xavier Standaert. Randomness generation for secure hardware masking - unrolled trivium to the rescue. *IACR Cryptol. ePrint Arch.*, page 1134, 2023.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

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

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[DKL$^+$18] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[DM21] Christoph Dobraunig and Bart Mennink. Leakage resilient value comparison with application to message authentication. In *EUROCRYPT (2)*, volume 12697 of *Lecture Notes in Computer Science*, pages 377–407. Springer, 2021.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[FKS$^+$22] Jakob Feldtkeller, David Knichel, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. Randomness optimization for gadget compositions in higher-order masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):188–227, 2022.

[FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In *ASIACRYPT (1)*, volume 10624 of *Lecture Notes in Computer Science*, pages 781–810. Springer, 2017.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for sidechannel resistance validation. *NIST non-invasive attack testing workshop*, 2011.

[GM17] Hannes Groß and Stefan Mangard. Reconciling d+1 masking in hardware and software. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 115–136. Springer, 2017.

[GPPS19] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Authenticated encryption with nonce misuse and physical leakage: Definitions, separation results and first construction - (extended abstract). In *LATINCRYPT*, volume 11774 of *Lecture Notes in Computer Science*, pages 150–172. Springer, 2019.

[GPPS20] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Towards low-energy leakage-resistant authenticated encryption from the duplex sponge construction. *IACR Trans. Symmetric Cryptol.*, 2020(1):6–42, 2020.

[GWDE15] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up! - made-to-measure hardware implementations of Ascon. In *DSD*, pages 645–652. IEEE Computer Society, 2015.

[GWDE17] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Ascon hardware implementations and side-channel evaluation. *Microprocess. Microsystems*, 52:470–479, 2017.

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

[KBG$^+$23] Aneesh Kandi, Anubhab Baksi, Tomas Gerlich, Sylvain Guilley, Peizhou Gan, Jakub Breier, Anupam Chattopadhyay, Ritu Ranjan Shrivastwa, Zdenek Martinasek, and Shivam Bhasin. Hardware implementation of Ascon. In *NIST Lightweight Cryptography Workshop 2023*, 2023.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. In *CCS*, pages 1799–1812. ACM, 2022.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In *ASIACRYPT (1)*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

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

[LMM23] Daniel Lammers, Nicolai Müller, and Amir Moradi. Glitch-free is not enough - revisiting glitch-extended probing model. *IACR Cryptol. ePrint Arch.*, page 35, 2023.

[MKSM22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional leakage in theory and practice unveiling security flaws in masked circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):266–288, 2022.

[MM22] Nicolai Müller and Amir Moradi. PROLEAD A probing-based hardware leakage detection tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):311–348, 2022.

[MMM21] Nicolai Müller, Thorben Moos, and Amir Moradi. Low-latency hardware masking of PRINCE. In *COSADE*, volume 12910 of *Lecture Notes in Computer Science*, pages 148–167. Springer, 2021.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.

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

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[NGPM22] Rishub Nagpal, Barbara Gigerl, Robert Primas, and Stefan Mangard. Riding the waves towards generic single-cycle masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):693–717, 2022.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptol.*, 24(2):292–321, 2011.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[SBB$^+$22] Mateus Simoes, Lilian Bossuet, Nicolas Bruneau, Vincent Grosso, Patrick Haddad, and Thomas Sarno. Self-timed masking: Implementing masked s-boxes without registers. In *CARDIS*, volume 13820 of *Lecture Notes in Computer Science*, pages 146–164. Springer, 2022.

[SBB$^+$23] Mateus Simões, Lilian Bossuet, Nicolas Bruneau, Vincent Grosso, Patrick Haddad, and Thomas Sarno. Low-latency masking with arbitrary protection order based on click elements. In *HOST*, pages 36–47. IEEE, 2023.

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

[USS$^+$20] Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. Retrofitting leakage resilient authenticated encryption to microcontrollers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):365–388, 2020.

[UXT$^+$22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/em analysis on post-quantum kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):296–322, 2022.

[VCS22] Corentin Verhamme, Gaëtan Cassiers, and François-Xavier Standaert. Analyzing the leakage resistance of the nist's lightweight crypto competition's finalists. In *CARDIS*, volume 13820 of *Lecture Notes in Computer Science*, pages 290–308. Springer, 2022.

[ZSS$^+$21] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. Low-latency keccak at any arbitrary order. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):388–411, 2021.

A Results displayed as table

Table 6: Performance characteristics of the proposed single-target implementations: area in [kGE] for three masking orders and the proposed implementations.

d	Architectures	Area (kGE)
2	Uniform(HPC2-ser)	23.7
	Leveled(HPC2-ser, r.b.)	31.6
	Leveled(HPC2-ser, unr-2)	34.8
	Uniform(HPC4-r.b.)	45.8
	Leveled(HPC4-r.b., r.b.)	54
	Leveled(HPC4-r.b., unr-2)	57.2
3	Uniform(HPC2-ser)	42
	Leveled(HPC2-ser, r.b.)	47.8
	Leveled(HPC2-ser, unr-2)	51.2
	Uniform(HPC4-r.b.)	100
	Leveled(HPC4-r.b., r.b.)	103
	Leveled(HPC4-r.b., unr-2)	107
4	Uniform(HPC2-ser)	64.8
	Leveled(HPC2-ser, r.b.)	69
	Leveled(HPC2-ser, unr-2)	72.2
	Uniform(HPC4-r.b.)	179.1
	Leveled(HPC4-r.b., r.b.)	180.6
	Leveled(HPC4-r.b., unr-2)	183.9

Table 7: Performance characteristics of the proposed multi-target implementations: area in [kGE] for three masking orders and the proposed implementations.

d	Architectures	Area (kGE)
2	Leveled(HPC2-ser, r.b.)	31.6
	Leveled(HPC2-ser, unr-2)	34.8
	Leveled(HPC4-r.b., r.b.)	54
	Leveled(HPC4-r.b., unr-2)	57.2
	Integrated Leveled(HPC4-r.b., r.b.)	57.9
3	Leveled(HPC2-ser, r.b.)	47.8
	Leveled(HPC2-ser, unr-2)	51.2
	Leveled(HPC4-r.b., r.b.)	103
	Leveled(HPC4-r.b., unr-2)	107
	Integrated Leveled(HPC4-r.b., r.b.)	120.9
4	Leveled(HPC2-ser, r.b.)	69
	Leveled(HPC2-ser, unr-2)	72.2
	Leveled(HPC4-r.b., r.b.)	180.6
	Leveled(HPC4-r.b., unr-2)	183.9
	Integrated Leveled(HPC4-r.b., r.b.)	211.1

Table 8: Performance characteristics of implementations considering randomness generation: area in [kGE] for three masking orders and the proposed implementations.

d	Architectures	Area (kGE)
2	Single Target Uniform(HPC2-ser)	29.3
	Single Target Leveled(HPC2-ser, unr-3)	42.7
	Single Target Uniform(HPC4-r.b.)	121.3
	Single Target Leveled(HPC4-r.b., unr-3)	136
	Multi Target Leveled(HPC2-ser, unr-3)	56.5
	Multi Target Leveled(HPC4-r.b., unr-3)	144.8
	Multi Target Integrated Leveled(HPC4-r.b., r.b.)	133.3
3	Single Target Uniform(HPC2-ser)	53.9
	Single Target Leveled(HPC2-ser, unr-3)	66
	Single Target Uniform(HPC4-r.b.)	326.5
	Single Target Leveled(HPC4-r.b., unr-3)	340.5
	Multi Target Leveled(HPC2-ser, unr-3)	80.4
	Multi Target Leveled(HPC4-r.b., unr-3)	354.2
	Multi Target Integrated Leveled(HPC4-r.b., r.b.)	346.2
4	Single Target Uniform(HPC2-ser)	88.4
	Single Target Leveled(HPC2-ser, unr-3)	98.4
	Single Target Uniform(HPC4-r.b.)	630.6
	Single Target Leveled(HPC4-r.b., unr-3)	653
	Multi Target Leveled(HPC2-ser, unr-3)	118.4
	Multi Target Leveled(HPC4-r.b., unr-3)	672.1
	Multi Target Integrated Leveled(HPC4-r.b., r.b.)	665

B Practical experimentations

In this additional section, we make a first step towards (i) empirically confirming the practical relevance of the glitch- and transition-robust probing model, and (ii) demonstrating that HPC4 gadgets indeed offer stronger side-channel security guarantees than HPC3 ones in the context of non-pipeline circuits. For this purpose, and similarly to [MKSM22], which also analyzed the glitch- and transition-robust probing model, we implement the circuit of Figure 13 with HPC3 and HPC4 AND gadgets at the first security order.

We performed a Fix-versus-Random Test Vector Leakage Assessment (TVLA) [GJJR11]. The evaluation was carried out on the Spartan-6 FPGA of a SAKURA-G board, measuring the supply current with a Tektronix CT1 probe, a Picoscope 6000E with a 500 MS/s sampling rate (12-bit resolution), and a target clock frequency of 6 MHz.

We considered three case studies. The first one is an implementation of the AND reduction using the iterative design of Figure 13 instantiated with an HPC3 gadget. The second one is the same design instantiated with an HPC4 gadget. The third one uses the same circuit as the first one, but keeps $s = 0$ so that the mux is controlled by a stable value and the output of the AND gadget is never fed back to its input. This makes the circuit effectively run as a pipeline (hence removing transitions between dependent variables).

The results are shown in Figure 14, leading to two main conclusions. First, the integration of HPC3 gadgets in the iterative design indeed leads to a first-order flaw. Second, this flaw vanishes if either the HPC3 gadget is replaced by a HPC4 gadget or if the design is run as a pipeline. The empirical evaluations are therefore in line with the theoretical results obtained within the glitch- and transition-extended robust probing model. Despite the general difficulty to interpret the origin of physical leakage with such preliminary tests, they at least do not suggest that the model is overly conservative from a qualitative viewpoint. Determining whether the lower-order leakages that we spot for

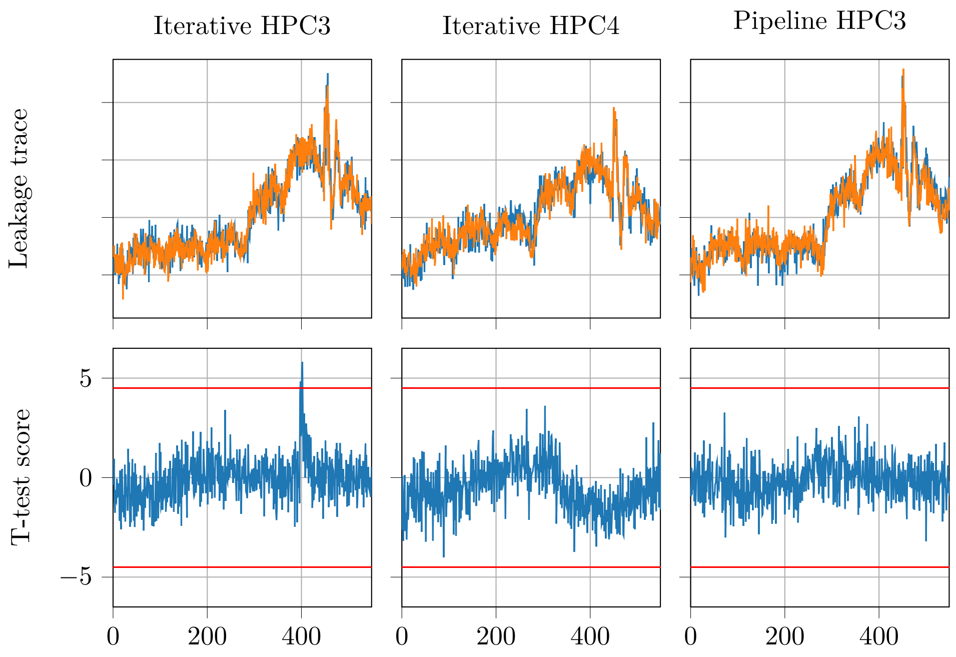


Figure 14: Acquired traces and first-order fixed vs. random t-test over time (10M traces). Left: iterative design of Figure 13 instantiated with HPC3 gadgets. Middle: same design instantiated with HPC4 gadgets. Right: same as left, but keeping $s = 0$.

the simple design of Figure 13 can be efficiently exploited in large designs as the ones presented in Sections 4 or 5 is an interesting scope for further investigations.