Full Key Recovery of Masked PRESENT on an Out-of-Order RISC-V Processor: A First Reported Case Study

Siddhartha Chowdhury, Nimish Mishra, Sarani Bhattacharya and Debdeep Mukhopadhyay

Indian Institute of Technology Kharagpur, India
{siddhartha.chowdhury92,neelam.nimish,bhattacharya.sarani.iitkgp,debdeep.mukhopadhyay}@gmail.com

Abstract. Masking-based countermeasures such as Threshold Implementations and Probe-Isolating Non-Interference (PINI) protect cryptographic software by maintaining separation between sensitive shares under a prescribed leakage model. Modern out-of-order (OoO) processors, however, introduce backend mechanisms such as register renaming, dynamic scheduling, forwarding, speculative execution, and physical-register reuse that can create additional observations not represented at the ISA level.

We develop a trace-driven backend analysis methodology that reconstructs physical-register histories and execution-time interactions from OoO RISC-V traces and relates these events to the semantics of masked computations. The analysis targets two classes of OoO-induced observations: Rename-Induced Transition Leakage (RIL), arising when distinct masked values successively occupy the same physical register, and IEW-Induced Dispatch Leakage, arising from transient overlap of share-processing instructions inside issue, execute, and writeback structures.

We evaluate the methodology on masked PRESENT and on controlled compositions of first-order PINI1 gadgets. For masked PRESENT, although Q12-style rotations protect selected nonlinear operations, the affine share pair (a_0, a_1) remains represented in distinct architectural registers. OoO physical-register reuse can nevertheless create transitions of the form

$$\mathrm{HW}_{\mathrm{bit}}(a_0[b] \oplus a_1[b]),$$

which reconstruct the affine intermediate at the leakage-model level and yield a key-dependent channel. Using an instrumented gem5 OoO RISC-V model, we recover the complete 64-bit first-round PRESENT subkey from these backend observations. For PINI, we extend the observation space with OoO-created physical-register and backend-execution interactions and test whether each modeled observation remains simulatable within the first-order PINI circuit-share budget. The isolated gadget already exhibits local simulator-bound violations, while extending the computation through a share-wise linear layer and a second PINI1 gadget introduces additional cross-composition violations. Under the stressed baseline configuration, the number of cross-composition violation observations progresses from 0 to 32 and then to 128, showing that PINI composability under its original probing model does not automatically extend to the considered OoO observation model.

Finally, we evaluate physical observability on a SiFive P550-class OoO RISC-V processor using Linux-accessible thermal telemetry. A profiled forced-reference methodology directly recovers 60 of the 80 PRESENT master-key bits. Since the positions of the remaining 20 bits are known, exhaustive search over the resulting 2^{20} candidate space completes recovery of the full 80-bit key.

Together, these results expose a cross-layer gap between software-level masking guarantees and OoO execution. Architecturally separated shares can acquire additional relationships through hidden backend state, producing observations that may exceed formal masking bounds and, in the PRESENT case, propagate into experimentally observable key-dependent behavior on real hardware.

Keywords: Out-of-Order · Masking · RISC-V

1 Introduction

Masking-based countermeasures such as Threshold Implementations (TIs) and Probe-Isolating Non-Interference (PINI) are widely used to protect cryptographic software against side-channel attacks [CJRR99, ISW03, FGMDP+18]. Their security relies on maintaining appropriate separation between randomized shares of sensitive variables under the assumed leakage model. Prior work has nevertheless shown that physical and microarchitectural effects such as glitches, pipeline forwarding, memory transitions, and register overwrites can introduce leakage even when a masked implementation is secure at the algorithmic level [RP10, LCGD18, GD23, ZMM23].

Modern out-of-order (OoO) processors further complicate this picture. Register renaming, speculative scheduling, dynamic issue, forwarding, writeback, and physical-register reuse allow architectural program order to diverge substantially from the internal execution history of the processor. Although these mechanisms preserve architectural correctness, they create transient backend state that is invisible at the instruction-set architecture (ISA) level. Consequently, masking shares that remain separated in the source code, assembly program, and architectural register state may acquire additional relationships through hidden physical-register and execution-time interactions.

In this work, we investigate whether masking guarantees established at the software or architectural level continue to hold when masked software executes on a realistic OoO RISC-V processor. We develop a trace-driven backend analysis methodology that reconstructs dynamic physical-register assignments, execution values, and backend timing relationships from fine-grained OoO traces. The analysis considers two principal classes of OoO-induced observations:

- **Rename-Induced Transition Leakage (RIL)**, arising when values associated with different masking domains successively occupy the same physical register, thereby creating a transition observation that is absent from the architectural execution; and
- **IEW-Induced Dispatch Leakage**, arising from transient overlap of share-processing instructions inside issue, execute, forwarding, or writeback structures.

We first apply this methodology to a masked PRESENT implementation derived from the software threshold implementation framework of Gaspoz and Dhooghe [GD23]. The implementation uses Q12-style rotated-share protection around selected nonlinear mixed-share computations and maintains the two masking domains in distinct architectural registers. However, the affine share pair (a_0, a_1) remains directly represented in the architectural state. The OoO backend can therefore reuse the same physical storage across these otherwise distinct architectural destinations and create transitions of the form

$$L[b] = \mathrm{HW}_{\mathrm{bit}}(a_0[b] \oplus a_1[b]), \tag{1}$$

which reconstruct the affine PRESENT intermediate because

$$a_0 \oplus a_1 = a.$$

Importantly, not every backend share transition is equally useful for key recovery. Frequent transitions involving randomized shares or cleared values may remain secret-independent.

The affine (a_0, a_1) transition instead provides a particularly clean key-dependent channel, making backend attribution essential for distinguishing exploitable events from generic switching activity.

Using an instrumented gem5 OoO RISC-V model, we reconstruct these physical-register histories and demonstrate recovery of the complete 64-bit first-round PRESENT subkey from masked execution traces. As a negative control, executing the same masked binary on an in-order processor model eliminates rename-induced physical-register transitions, localizing the identified mechanism to OoO backend behavior.

We additionally investigate whether OoO-created observations preserve the composability guarantees of PINI. Rather than treating every cross-domain backend interaction as a violation, we evaluate controlled first-order PINI1 compositions using a simulator-support test. For each modeled OoO observation, we determine the minimum set of circuit-share indices required to reproduce its distribution. An observation requiring both circuit-share domains exceeds the first-order PINI simulator budget under the extended OoO observation model. In the controlled sequence

$$G_1 \rightarrow L_{\text{lin}} \rightarrow G_2,$$

the isolated gadget already exhibits local OoO-induced simulator-bound violations, while extending the computation introduces additional observations spanning otherwise valid composition boundaries. Under the stressed baseline configuration, the number of cross-composition violation observations progresses from

$$0 \rightarrow 32 \rightarrow 128$$

for the isolated gadget, the gadget followed by a share-wise linear layer, and the complete two-gadget composition, respectively. These results do not constitute a counterexample to PINI under its original probing model; rather, they show that its composability guarantee does not automatically extend to additional observations introduced by an OoO backend.

Finally, we investigate whether key-dependent processor activity identified through simulation can become physically observable on a real OoO platform. We execute the masked PRESENT implementation on a SiFive P550-class OoO RISC-V processor and collect Linux-accessible thermal telemetry during repeated execution. Using a profiled forced-reference methodology with trimmed-median classification, we directly recover 60 of the 80 PRESENT master-key bits. Because the positions of the remaining 20 unresolved bits are known, the residual search space contains only

$$2^{20}$$

candidate keys. Exhaustive search over this reduced space completes recovery of the full 80-bit PRESENT master key.

Together, these experiments establish a cross-layer leakage path from architecturally separated masked computations to hidden OoO backend interactions, physical-register transitions, key-dependent switching activity, thermal accumulation, and complete key recovery on real hardware. At the same time, the PINI analysis demonstrates that OoO execution can introduce observations that exceed formal simulator-support bounds even when the software preserves its intended architectural sharing discipline. These results show that masking security cannot be evaluated solely at the algorithmic or ISA level once the implementation executes on a speculative OoO microarchitecture.

The main contributions of this work are summarized below:

- We develop a trace-driven methodology for reconstructing and attributing masking-relevant backend interactions in OoO RISC-V execution.
- We characterize Rename-Induced Transition Leakage (RIL) and IEW-Induced Dispatch Leakage as two classes of OoO-specific observations arising from physical-register reuse and dynamic backend execution.

- We identify physical-register reuse of the affine PRESENT share pair (a_0, a_1) as a clean exploitable backend channel and demonstrate complete 64-bit first-round PRESENT subkey recovery using instrumented gem5 traces.
- We extend the analysis to controlled first-order PINI1 compositions and show that OoO-created observations can require both circuit-share indices, exceeding the first-order simulator budget under the considered extended observation model. Cross-composition violations increase from 0 to 32 and then to 128 as the computation is extended from an isolated gadget to a complete two-gadget composition.
- We demonstrate physical observability on a real SiFive P550-class OoO RISC-V processor, directly recovering 60 of the 80 PRESENT master-key bits using software-accessible thermal telemetry and completing recovery of the full 80-bit key through exhaustive search of the remaining 2^{20} candidates.
- We establish a cross-layer connection between software-level masking, hidden OoO backend state, formal masking-model violations, key-dependent physical activity, and practical full key recovery, motivating hardware-aware verification of masked software.

The remainder of this paper is organized as follows. Section 2 introduces the required background on masking, PINI composability, and OoO execution. Section 3 presents the threat model, leakage taxonomy, and trace-driven backend analysis methodology. Section 4 presents the masked PRESENT backend attribution and key-recovery experiments, the controlled PINI composition analysis, and the SiFive P550 physical validation. Finally, Section 5 concludes the paper.

2 Background and Motivation

Masking schemes such as Threshold Implementations (TIs) and Probe-Isolating Non-Interference (PINI) protect cryptographic software by decomposing sensitive variables into randomized shares that are processed according to a prescribed leakage model [CJRR99, ISW03, FGMDP+18]. For a first-order Boolean masking scheme,

$$v = s_0 \oplus s_1, \tag{2}$$

neither share alone determines the sensitive value v. Security therefore depends not only on the algebraic sharing itself, but also on which combinations of shares can influence an observable hardware state.

These guarantees are typically reasoned about at the circuit, software, or architectural level. At the software level, complementary shares may occupy distinct architectural registers and be processed through carefully separated instruction sequences. Prior work has nevertheless shown that processor effects such as glitches, memory transitions, forwarding, and register overwrites can introduce leakage beyond the idealized software representation [RP10, LCGD18, GD23, ZMM23]. Modern out-of-order (OoO) processors extend this problem further because architectural execution is decoupled from the physical location and timing of intermediate values.

Prior simulation-driven work [CMBM25] demonstrated that masked cryptographic software executing on OoO RISC-V processors can experience leakage arising from hidden backend behavior, including physical-register reuse and execution overlap. Concurrent work has similarly shown that register renaming, forwarding structures, speculative scheduling, and shared execution resources can undermine software-level masking assumptions [DKV26]. These observations motivate a more precise distinction between *architectural share separation* and *backend physical separation*.

The present work extends this direction in three ways. First, we perform fine-grained backend attribution using instrumented gem5 traces, reconstructing dynamic physical-register histories, execution values, timing relationships, and masked-variable semantics. Second, for composable PINI gadgets, we do not treat every cross-domain backend interaction as a violation. Instead, each modeled OoO-created observation is evaluated against the first-order PINI simulator budget to determine the minimum circuit-share support required to reproduce its distribution. Third, we connect key-dependent backend activity to physical observability on a real Linux-capable SiFive P550-class OoO RISC-V processor and ultimately to complete recovery of the 80-bit PRESENT master key.

2.1 OoO Backend State and Masking

OoO processors improve performance using mechanisms such as register renaming, dynamic scheduling, issue queues, reorder buffers, forwarding paths, speculative execution, and out-of-order completion. Architectural correctness is preserved through in-order retirement, but values may temporarily occupy backend resources in an order that differs substantially from the sequential instruction stream.

The *rename stage* maps architectural destinations onto a larger physical-register file. Once an old physical-register mapping is released, that physical register can be allocated to another architectural destination. The *issue and execution logic* independently schedules instructions according to operand availability and resource availability. Consequently, the processor introduces two independent dimensions of hidden state: where values are physically stored and when instructions execute.

Figure 1 summarizes the two backend relationships considered in this work. The figure is intentionally organized around the masking-relevant observation path rather than around a conventional processor block diagram.

2.2 Rename-Induced Transition Leakage

The primary exploitable mechanism identified for masked PRESENT is backend physical-register reuse. Suppose a physical register initially contains V_{old} and is subsequently reused for a value V_{new}. The overwrite

$$V_{\text{old}} \rightarrow V_{\text{new}} \tag{3}$$

produces data-dependent switching activity commonly modeled using

$$L_{\text{RIL}} = \text{HD}(V_{\text{old}}, V_{\text{new}}) = \text{HW}(V_{\text{old}} \oplus V_{\text{new}}), \tag{4}$$

where $\text{HD}(\cdot, \cdot)$ denotes Hamming distance.

The important distinction is that this transition need not correspond to an architectural overwrite. Two different architectural destinations can successively occupy the same physical storage because of register renaming. The resulting physical-register history therefore introduces an observation that is absent from the software-level representation.

In the masked PRESENT implementation analyzed in this work, Q12-style rotated-share protection is applied around selected nonlinear mixed-share computations. However, the affine share pair (a_0, a_1) remains represented in separate architectural registers such that

$$a_0 \oplus a_1 = a.$$

If the OoO backend successively maps these destinations onto the same physical register, the corresponding bit-level transition follows

$$L_4[b] = \text{HW}_{\text{bit}}\left(a_0[b] \oplus a_1[b]\right). \tag{5}$$

Equation 5 reconstructs the affine PRESENT intermediate at the leakage-model level even though the architectural program never explicitly recombines the two shares. This

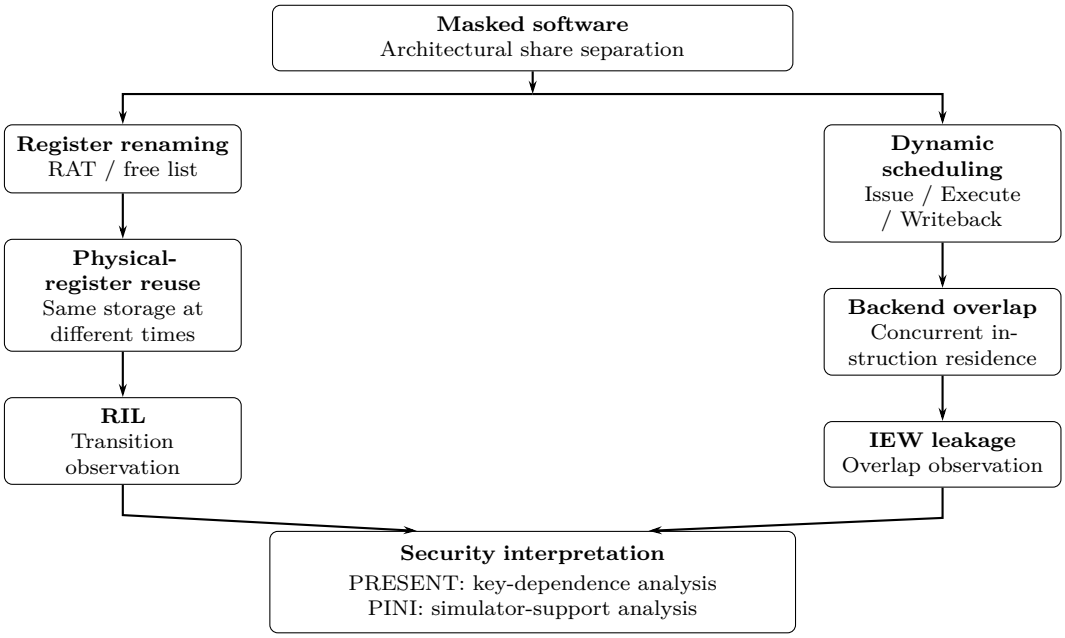


Figure 1: Masking-relevant OoO backend relationships considered in this work. Register renaming can create time-separated physical-register reuse, leading to Rename-Induced Transition Leakage (RIL), while dynamic scheduling can create transient execution overlap in backend structures. The reconstructed observations are subsequently evaluated according to PRESENT key dependence or the applicable PINI simulator-support criterion.

distinction is particularly important because not every share-related backend transition is exploitable. For example, transitions involving a randomized share and a cleared value may expose only the Hamming weight of one masked share and remain independent of the secret. Backend attribution is therefore required to distinguish generic switching activity from a key-dependent share-to-share transition.

2.3 IEW-Induced Dispatch Leakage and PINI

A second class of OoO-created observations arises from dynamic scheduling. Instructions belonging to different masking domains may simultaneously reside in issue queues, execution structures, forwarding paths, or writeback-related backend state. We refer to such candidate interactions as IEW-Induced Dispatch Leakage.

For conventional two-share masking, such overlap identifies a potential hidden microarchitectural interaction. For PINI, however, the presence of two domain labels in one backend event is not by itself sufficient to establish a formal violation. PINI provides a simulator-based composability guarantee, and the relevant question is whether an observation can be simulated using the permitted number of circuit-share indices.

For the first-order, two-share setting considered in this work,

$$d = 2, \qquad t = 1, \tag{6}$$

and one internal observation without an output-share probe must be simulatable from one of

$$\varnothing, \qquad \{D_0\}, \qquad \{D_1\}. \tag{7}$$

Let $S_{\min}(O)$ denote the minimum circuit-share support sufficient to reproduce the distribution of an OoO-created observation O. Under the extended observation model

considered here, the event exceeds the first-order PINI simulator budget when

$$S_{\min}(O) = \{D_0, D_1\}. \tag{8}$$

This distinction separates three concepts that must not be conflated: the existence of a backend interaction, violation of a masking condition under a particular observation model, and physical exploitability. In particular, an OoO-created PINI simulator-bound violation does not by itself imply key leakage or an observable analog side channel.

The controlled PINI experiments later in the paper therefore evaluate compositions of the form

$$G_1 \rightarrow L_{\text{lin}} \rightarrow G_2, \tag{9}$$

where G_1 and G_2 are independently structured PINI1 gadgets and L_{lin} is evaluated share-wise. The analysis determines both the component scope of each backend observation and its minimum simulator support. This allows local violations within an individual gadget to be distinguished from additional observations that cross otherwise valid composition boundaries.

2.4 From Backend Interaction to Physical Observability

The existence of an OoO-created observation in simulation does not imply a one-to-one physical leakage event on real hardware. Instrumented gem5 provides visibility into physical-register allocation, executed values, and backend timing and therefore allows the mechanism to be reconstructed and attributed. A real processor exposes none of this internal state.

The SiFive P550 experiment addresses a different question: whether repeated key-dependent processor activity can accumulate into a measurable physical signal. Thermal telemetry is a slow and integrated measurement, so it does not resolve individual physical-register transitions. Instead, repeated masked PRESENT execution amplifies small differences in switching activity into measurable temperature variation.

The complete interpretation considered in this work is therefore

$$\begin{aligned} \text{masked computation} &\rightarrow \text{OoO backend observation} \\ &\rightarrow \text{key-dependent switching} \\ &\rightarrow \text{thermal accumulation} \\ &\rightarrow \text{key classification.} \end{aligned} \tag{10}$$

Using this methodology, the real-hardware experiment directly determines 60 of the 80 PRESENT master-key bits. Since the locations of the remaining 20 unresolved bits are known, the residual key space contains only 2^{20} candidates, allowing exhaustive search to complete recovery of the full 80-bit master key.

These observations motivate backend-aware masking analysis. Software- and ISA-level reasoning remains necessary, but it is insufficient to characterize interactions created dynamically by register renaming and speculative OoO execution. The following sections therefore formalize the adversarial model, define the backend observation predicates, and separate structural backend interactions, masking-model violations, key dependence, and physical exploitability.

3 Threat Model and Leakage Taxonomy

This section formalizes the OoO-induced observations considered in this work and describes the trace-driven methodology used to reconstruct and interpret them. Unlike conventional masking analyses that reason primarily about source-level, assembly-level, or architectural

execution, our threat model considers transient backend relationships introduced by register renaming, physical-register reuse, speculative scheduling, forwarding, and overlapping execution.

The central observation is that *architectural share separation does not necessarily imply backend physical separation*. Values belonging to distinct masking domains may remain in separate architectural registers while successively occupying the same physical storage or transiently overlapping inside shared backend structures. These interactions are first treated as candidate microarchitectural observations. Whether they constitute a masking violation or an exploitable side channel is evaluated separately.

For masked PRESENT, the clearest exploitable channel arises from backend reuse of the affine share pair (a_0, a_1). Although Q12-style rotated-share protection reduces aligned interactions around selected nonlinear computations, the affine shares remain directly represented in distinct architectural registers. Physical-register reuse can therefore create a transition that depends on their recombination even though the software never explicitly performs that recombination.

For PINI, the interpretation is more stringent. A cross-domain or cross-composition backend interaction is not automatically classified as a PINI violation. Instead, the resulting observation is evaluated against the first-order PINI simulator budget to determine whether its distribution can be reproduced using the permitted circuit-share support.

3.1 Adversarial Model

We consider a masked cryptographic implementation executing on an OoO RISC-V processor. The implementation is assumed to be functionally correct and to preserve its intended masking representation at the architectural level. Complementary shares occupy distinct architectural destinations unless explicitly combined by the masking algorithm.

The analysis proceeds in two stages.

Offline attribution phase. The masked implementation executes unmodified on an instrumented OoO processor model. The analyst observes backend traces containing sufficient information to reconstruct:

- architectural-to-physical register mappings,
- physical-register allocation and reuse,
- execution values and writeback activity,
- instruction sequence numbers and runtime PCs,
- execution and completion timing,
- and transient overlap inside OoO backend stages.

The analyst does not force unsafe physical-register assignments, manipulate the free list, alter instruction scheduling, or modify the masked computation. The purpose of the offline phase is attribution: to identify which hidden backend relationships create observations involving sensitive masking domains.

For masked PRESENT, this analysis identifies physical-register reuse between complementary affine shares as a key-dependent transition channel. A representative architectural mapping is

$$(a_0, a_1) \Longleftrightarrow (s4, t4).$$

For PINI-tagged computations, the same backend reconstruction is augmented with semantic labels identifying the composing gadget, circuit-share domain, and intermediate value. Each resulting observation is then evaluated against the PINI simulator-support condition.

Online physical-observation phase. The real-hardware adversary has substantially less visibility. On the SiFive P550-class OoO RISC-V processor, the attacker can repeatedly execute the masked implementation and observe only software-accessible thermal telemetry.

The attacker does *not* require:

- invasive physical probing,
- direct access to physical-register identifiers,
- privileged backend instrumentation,
- control over physical-register allocation,
- manipulation of the rename free list,
- or fault injection.

The simulation and hardware experiments therefore answer different questions. The offline experiment determines *where and how* an OoO interaction arises, while the hardware experiment tests whether repeated key-dependent activity becomes physically observable.

3.2 PINI-Specific Observation Model

For the PINI experiments, we consider the first-order two-share setting

$$d = 2, \qquad t = 1. \tag{11}$$

For one internal observation and no output-share probe, the simulator may depend on at most one circuit-share index. The admissible supports are therefore

$$\varnothing, \qquad \{D_0\}, \qquad \{D_1\}. \tag{12}$$

Let $S_{\min}(O)$ denote the minimum circuit-share support sufficient to reproduce the distribution of an OoO-created observation O. We classify the observation as exceeding the first-order PINI simulator budget under the considered OoO observation model when

$$S_{\min}(O) = \{D_0, D_1\}. \tag{13}$$

The component scope of an observation is classified independently. For a controlled composition

$$G_1 \to L_{\text{lin}} \to G_2, \tag{14}$$

an event is labeled *local* when all participating values belong to one composing component and *cross* when the observation spans more than one component.

This distinction is important: a cross-composition interaction does not itself imply a PINI violation, and a PINI simulator-bound violation does not by itself imply secret-dependent physical leakage.

3.3 OoO-Induced Leakage Mechanisms

We focus on two classes of backend observations created naturally by OoO execution.

3.3.1 Rename-Induced Transition Leakage

Rename-Induced Transition Leakage (RIL) arises when two masking-related values successively occupy the same backend physical register. If a physical register changes from V_{old} to V_{new}, the resulting transition activity is modeled as

$$L_{\text{RIL}} = \text{HD}(V_{\text{old}}, V_{\text{new}}) = \text{HW}(V_{\text{old}} \oplus V_{\text{new}}). \tag{15}$$

The transition need not correspond to an architectural overwrite. Two distinct architectural destinations can produce consecutive values in the same physical register because of dynamic register renaming.

For masked PRESENT, the clearest exploitable RIL channel corresponds to the affine pair:

$$L_4[b] = \text{HW}_{\text{bit}}(a_0[b] \oplus a_1[b]). \tag{16}$$

Since

$$a_0 \oplus a_1 = a,$$

the physical-register transition reconstructs the affine PRESENT intermediate at the leakage-model level even though the software never explicitly recombines the shares.

Importantly, the largest number of structural transitions does not necessarily identify the most exploitable channel. Transitions involving one randomized share and a cleared value may reduce to the Hamming weight of an individual share and remain secret-independent. The affine share-to-share transition in Eq. 16 is therefore distinguished by its key dependence rather than merely by raw event frequency.

For the one-bit PINI analysis, an analogous physical-register transition is modeled as

$$O_{\text{PRF}} = V_{\text{old}} \oplus V_{\text{new}}, \tag{17}$$

and it is considered a formal first-order violation only if Eq. 13 holds.

3.3.2 IEW-Induced Dispatch Leakage

IEW-Induced Dispatch Leakage captures transient relationships created when share-processing instructions overlap inside Issue–Execute–Writeback-related backend structures.

Unlike RIL, which represents *time-separated reuse of one physical storage location*, IEW observations represent *simultaneous or overlapping residence of different dynamic instructions*. Relevant structures include issue queues, execution pipelines, forwarding paths, and writeback-related state.

For PINI, such an overlap is first treated as a candidate observation. The observation is subsequently subjected to the same minimum-support test used for physical-register transitions. Only observations whose distributions require both circuit-share indices exceed the first-order simulator budget.

Table 1 summarizes the two mechanisms.

3.4 Trace-Driven Backend Attribution

To analyze these effects systematically, we use a trace-driven framework that reconstructs dynamic backend relationships before applying any masking-security interpretation.

The principal trace records are abstracted as

$$\text{Rename}(t, \mathit{arch}, \mathit{preg}, \mathit{pc}), \tag{18}$$

$$\text{Write}(t, \mathit{preg}, v, \mathit{pc}), \tag{19}$$

$$\text{IEW}(t, \sigma, \text{SN}, \mathit{pc}), \tag{20}$$

Table 1: Summary of targeted OoO-induced leakage mechanisms.

Mechanism	**Micro-architectural origin**	**Security interpretation**
Rename-Induced Transition Leakage (RIL)	Successive use of the same physical register by different masking-related values.	Creates a transition observation based on Hamming distance. For PRESENT, selected share-to-share transitions can be key-dependent; for PINI, the simulator-support test determines whether the observation exceeds the permitted budget.
IEW-Induced Dispatch Leakage	Transient overlap of share-processing instructions inside issue, execute, forwarding, or writeback structures.	Creates an additional backend observation absent from sequential architectural execution. For PINI, formal violation is determined by the simulator-support requirement rather than overlap alone.

where t denotes simulator time, *arch* and *preg* denote architectural and physical registers, v is the executed value, *pc* is the runtime program counter, σ identifies a backend stage, and SN denotes a dynamic instruction sequence number.

The distinction between *pc* and SN is important. The program counter identifies the static architectural instruction, whereas the sequence number identifies its particular dynamic instance under OoO execution.

The analysis proceeds through four conceptual steps:

1. **Dynamic reconstruction:** rename allocations are associated with execution, writeback, and commit records, while physical-register histories and execution-time relationships are reconstructed.
2. **Semantic annotation:** backend events are mapped to masked variables. For PRESENT, architectural registers are mapped to share labels. For PINI, tagged program points additionally carry component and circuit-share-domain information.
3. **Observation construction:** consecutive physical-register histories instantiate RIL candidates, while overlapping backend activity instantiates IEW candidates.
4. **Security interpretation:** PRESENT candidates are evaluated for complementary-share structure and key dependence; PINI candidates are evaluated by determining $S_{\min}(O)$.

For a physical register p, let W_{old} and W_{new} denote consecutive usable writes associated with distinct dynamic rename allocations. The structural RIL estimate is

$$L_{\text{RIL}}(p) = \text{HD}\left(v(W_{\text{old}}), v(W_{\text{new}})\right). \tag{21}$$

The write pair is retained only when the corresponding allocations are consistent with the dynamic trace, the two writes are contiguous in the physical-register history, both execution values are available, and the temporal relationship satisfies the configured event window. This prevents unrelated reuse episodes of the same physical register from being merged into a single leakage event.

The output therefore preserves the physical resource, old and new architectural writers, runtime PCs, dynamic sequence numbers, execution values, timing relation, semantic share labels, observation type, and applicable security classification. This allows backend interaction, masking-model violation, and physical exploitability to remain distinct claims.

3.5 From Backend Observation to Physical Exploitability

A modeled backend observation is not itself a physical side-channel measurement. We therefore separate three levels of evidence:

1. existence of an OoO-created backend interaction,
2. violation of a masking condition or key dependence under the chosen leakage model,
3. and physical observability on real hardware.

gem5 attribution and key-dependence experiment. Using an instrumented gem5 OoO RISC-V model, we directly reconstruct physical-register transitions and associate them with masking semantics. For masked PRESENT, the affine share-to-share channel

$$a = a_0 \oplus a_1$$

provides a key-dependent Hamming-distance observation. The resulting traces enable recovery of the complete 64-bit first-round PRESENT subkey.

An in-order execution of the same masked binary serves as a negative control: without speculative register renaming and physical-register reuse, the rename-induced transition channel disappears.

PINI simulator-support experiment. For controlled PINI1 compositions, backend observations are evaluated independently of physical key leakage. Under the stressed baseline configuration, the number of cross-composition observations exceeding the first-order simulator budget evolves as

$$0 \to 32 \to 128 \tag{22}$$

for G_1, $G_1 \to L_{\text{lin}}$, and $G_1 \to L_{\text{lin}} \to G_2$, respectively.

Because the isolated gadget already exhibits local violations, this does not constitute a composition-only counterexample. Rather, it demonstrates that additional OoO-created observations can span otherwise valid PINI composition boundaries and exceed the original simulator-support budget under the extended observation model.

SiFive P550 thermal experiment. We finally evaluate physical observability on a real SiFive P550-class OoO RISC-V processor using Linux-accessible thermal telemetry. Repeated masked PRESENT execution integrates data-dependent processor activity into measurable temperature variation.

Using profiled forced-reference measurements and trimmed-median classification, 60 of the 80 PRESENT master-key bits are directly recovered. Since the positions of the remaining 20 bits are known, the residual search space is

$$2^{20}, \tag{23}$$

and exhaustive search completes recovery of the full 80-bit master key.

The PRESENT cross-layer interpretation is therefore

$$\begin{aligned} \text{masked shares} &\to \text{OoO backend observation} \\ &\to \text{key-dependent transition activity} \\ &\to \text{thermal accumulation} \\ &\to \text{direct bit classification} \\ &\to \text{full key recovery.} \end{aligned} \tag{24}$$

We do not claim a one-to-one correspondence between an individual simulated physical-register transition and one thermal sample. The simulator exposes event-level backend state, whereas the temperature sensor integrates activity over repeated execution. The two experiments therefore provide complementary evidence: fine-grained mechanism attribution in simulation and aggregate physical observability on real hardware.

4 Experimental Evaluation: Masked PRESENT as a Case Study

This section evaluates the trace-driven backend analysis methodology with masked PRESENT as the primary case study. The objective is to determine whether a masked implementation that preserves its intended sharing discipline at the architectural level can nevertheless acquire additional leakage through out-of-order (OoO) backend execution, and whether such key-dependent activity can ultimately become observable on real hardware.

A complementary PINI experiment is included to examine the broader masking-security implication of these backend interactions. While the PRESENT experiment establishes a concrete path from physical-register reuse to key recovery, the PINI experiment asks whether OoO-created observations remain within the simulator-support budget required by a composable masking construction.

The evaluation addresses four questions:

1. Can hidden backend interactions between architecturally separated masked values be reconstructed from an OoO execution trace?
2. Which reconstructed interactions produce key-dependent leakage in masked PRESENT?
3. Do OoO-created observations remain within the first-order simulator budget of PINI?
4. Can repeated key-dependent activity become physically observable on a real OoO RISC-V processor and lead to complete key recovery?

Figure 2 summarizes the experimental progression used throughout this section.

4.1 Target Implementation: Masked PRESENT

The primary workload is a masked implementation of PRESENT derived from the software threshold implementation framework of Gaspoz and Dhooghe [GD23]. PRESENT operates on 64-bit blocks and uses an 80-bit master key.

For first-round nibble i,

$$X_i = P_i \oplus K_i^{(1)}, \qquad i \in \{0, \ldots, 15\}, \tag{25}$$

and sensitive intermediate values are represented using two Boolean shares,

$$x = x_0 \oplus x_1. \tag{26}$$

The bitsliced implementation uses Q12-style rotated-share protection around selected nonlinear mixed-share operations. Let

$$(a, b, c, d) = A(X) \tag{27}$$

denote the affine preprocessing variables. In particular,

$$a = a_0 \oplus a_1. \tag{28}$$

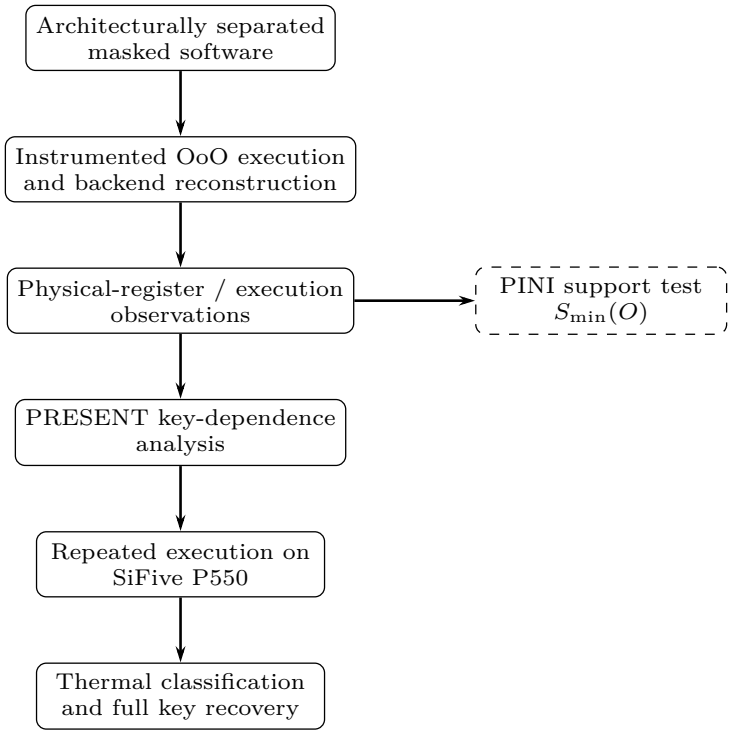


Figure 2: Experimental organization. Backend observations are reconstructed from instrumented OoO execution and interpreted according to the applicable security question. For PRESENT, key dependence and physical observability are evaluated; for PINI, the observations are subjected to a simulator-support test.

Table 2: Architectural-register interpretation used for masked PRESENT attribution.

Register pair	**Masked-variable interpretation**
`s1/t1`	b_0/b_1
`s2/t2`	c_0/c_1 or protected c'_0/c'_1
`s3/t3`	d_0/d_1 or protected d'_0/d'_1
`s4/t4`	a_0/a_1

The rotated-share protection constrains selected nonlinear interactions, but the affine pair (a_0, a_1) remains represented through distinct architectural registers. Consequently, architectural separation alone does not prevent these two values from occupying the same physical register at different times under OoO register renaming.

We monitor the four stable PRESENT share pairs listed in Table 2.

Architectural separation. The leakage investigated here is not introduced by explicitly writing complementary shares into one architectural register. A representative fragment illustrates the intended software-level separation:

```
# Domain A
and t0, a0, a1
xor a2, a2, t0
xor a2, a2, a0
xor a2, a2, a1

# Domain B
and t0, a3, a4
```

```
xor a5, a5, t0
xor a5, a5, a3
xor a5, a5, a4

# Later linear propagation
ror s1, a2, 1
xor t1, s1, a5
```

Thus, the two masking domains remain distinct at the architectural level. The interaction studied below is introduced by backend physical allocation and execution rather than by an intentional software recombination.

4.2 Experimental Platforms and Roadmap

The evaluation uses two complementary execution environments.

Instrumented gem5 platform. The simulation environment provides direct visibility into register renaming, physical-register allocation, execution values, backend timing, writeback, and commit activity. This visibility allows individual backend interactions to be reconstructed and attributed to masked variables.

SiFive P550 platform. The physical experiment uses a SiFive P550-class OoO RISC-V processor running Linux. Internal rename mappings, physical-register identifiers, and backend traces are unavailable on this platform. The physical observation is instead obtained through Linux-accessible thermal telemetry.

The two platforms therefore serve different purposes. The simulator establishes and attributes the backend mechanism, whereas the P550 experiment evaluates whether repeated key-dependent activity produces a measurable physical effect.

4.3 gem5 Backend Attribution

The simulator uses an RV32IM OoO processor model derived from `DerivO3CPU`. The backend includes register renaming, a physical-register file, dynamic issue, forwarding, writeback, reorder-buffer retirement, and speculative execution.

Instrumentation reconstructs dynamic mappings of the form

$$R_{\text{arch}} \mapsto R_{\text{phys}}, \tag{29}$$

and associates the resulting physical-register writes with the architectural writers, execution values, program counters, and dynamic instruction instances that produced them.

For a physical-register transition event e, the structural switching estimate is

$$\ell_e = \mathrm{HW}\left(V_{\text{prev},e} \oplus V_{\text{curr},e}\right). \tag{30}$$

A strict transition candidate is retained only when the two values correspond to consecutive usable writes to the same physical register, are associated with valid dynamic rename allocations, have available execution values, and satisfy the configured temporal relationship.

The principal simulator configuration is summarized in Table 3.

Backend tracing records rename, execution, writeback, IEW-related timing, and commit information. The analysis then joins these records using runtime program counters and dynamic instruction identifiers before attaching PRESENT share semantics.

Table 3: gem5 OoO RISC-V configuration used for masked PRESENT attribution.

Parameter	**Value**
gem5 version	25.1.0.0
ISA	RV32IM
CPU model	DerivO3CPU-style OoO core
Integer physical registers	192
ROB entries	256
Issue queue entries	64
L1 instruction cache	32 kB
L1 data cache	32 kB
Shared L2 cache	512 kB
Core frequency	2.5 GHz

4.4 Strict Candidate Detection

Candidate generation is deliberately separated from semantic filtering. Across the masked PRESENT trace, the strict detector reconstructs

$$51{,}707 \tag{31}$$

physical-register transition candidates. Restricting these events to the four monitored complementary-share relationships of Table 2 leaves

$$344 \tag{32}$$

strict share-mixing candidates.

A representative affine-share event occurs on physical register P_{74}. The first write is associated with architectural destination `x29`, labeled a_1, and the subsequent write is associated with `x20`, labeled a_0:

$$V_{\text{old}} = \texttt{0x0000c144}, \tag{33}$$

$$V_{\text{new}} = \texttt{0x00007fb9}. \tag{34}$$

The transition has

$$\mathrm{HD}\,(V_{\text{old}}, V_{\text{new}}) = 13. \tag{35}$$

Figure 3 presents this event using a representation distinct from the submission-time figure.

The important property is not simply reuse of P_{74}, but the complete dynamic relationship between the two writes: both values are available, the writes are consecutive in the usable physical-register history, and their architectural destinations correspond to complementary masked values.

4.5 Validation of the Transition Reconstruction

We perform two controlled checks before interpreting the PRESENT key-recovery channel.

Strict versus relaxed matching. Relaxing the contiguity requirement increases the overall candidate count from

$$51{,}707$$

to approximately

$$52{,}002,$$

an increase of only about 295 events. We therefore retain strict consecutive-write matching to avoid combining unrelated physical-register reuse episodes.

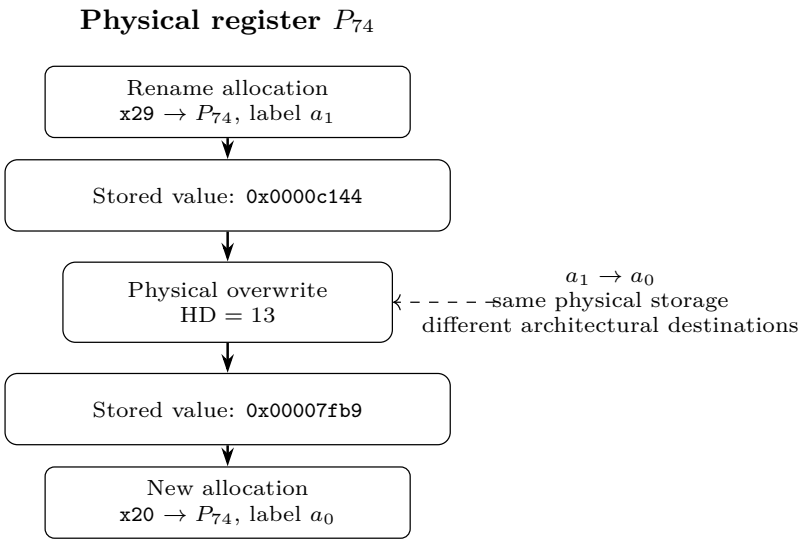


Figure 3: Representative strict RIL event reconstructed from the masked PRESENT execution. Two distinct architectural destinations carrying complementary affine shares successively occupy P_{74}, producing a Hamming-distance transition of 13.

Table 4: Known-answer validation of physical-register transition reconstruction.

Share pair	**Count**	**Hamming distance**
a_{s0}/a_{s1}	1999	32 (exact)
b_{s0}/b_{s1}	1999	32 (exact)
c_{s0}/c_{s1}	3998	mean 14, max 16

Known-answer masked Toffoli validation. We independently validate the rename-to-value association and Hamming-distance calculation using a two-share masked Toffoli computation with fixed all-ones inputs. For the input-share pairs, the two values are exact bitwise complements. A genuine 32-bit share-to-share overwrite must therefore have Hamming distance 32.

Table 4 reports the result.

The exact 32-bit transitions for both input-share pairs provide a controlled check that the reconstructed rename allocations, execution values, and physical-register write histories are being associated correctly.

Phase-aware attribution. The PRESENT implementation is additionally instrumented with execution-region markers separating affine preprocessing, Q12-protected nonlinear computation, post-Q12 propagation, and later masked-share processing.

These markers are relevant because backend lifetime does not necessarily coincide with source-level lifetime. A share-derived value can remain present in the physical-register file, rename state, forwarding logic, or other backend structures after the source-level operation that created it has completed.

4.6 Which PRESENT Channel Is Exploitable?

Table 5 reports the strict conflict-filtered candidates associated with the four monitored PRESENT relationships.

Table 5: Strict conflict-filtered PRESENT share-mixing candidates reconstructed in gem5.

Channel	Variable pair	Count	$\mathbf{HD_{max}}$	$\mathbf{HD_{mean}}$
L_3	`s3/t3` $= d_0/d_1$ or d_0'/d_1'	153	23	9.59
L_2	`s2/t2` $= c_0/c_1$ or c_0'/c_1'	124	14	7.91
L_4	`s4/t4` $= a_0/a_1$	66	13	7.77
L_1	`s1/t1` $= b_0/b_1$	1	18	18.00

The largest candidate population belongs to L_3, followed by L_2. Raw event count, however, is not equivalent to exploitability.

A substantial fraction of the high-frequency events correspond to share-to-zero or related transitions. For such an event,

$$\mathrm{HD}(S, 0) = \mathrm{HW}(S),$$

so the observation may depend only on one randomized share and therefore need not expose the secret.

The cleaner key-dependent observation is the affine share-to-share channel

$$L_4[b] = \mathrm{HW}_{\mathrm{bit}}\left(a_0[b] \oplus a_1[b]\right). \tag{36}$$

Since

$$a_0 \oplus a_1 = a, \tag{37}$$

Eq. 36 reconstructs a bit of the affine PRESENT intermediate at the leakage-model level.

Thus, the L_4 channel is the *clearest exploitable channel*, even though it does not have the largest raw candidate count. This distinction is important: backend attribution is needed not merely to count transitions, but to determine which transitions carry useful key dependence.

In-order negative control. The identical masked binary is also executed on `TimingSimpleCPU`. This processor model produces no rename stream and therefore no RIL candidates.

Hence, with the same software, masking scheme, inputs, and key,

$$N_{\mathrm{RIL}}^{\text{in-order}} = 0. \tag{38}$$

Removing speculative register renaming and physical-register reuse therefore eliminates the identified transition mechanism, localizing it to the OoO backend.

4.7 gem5 Key-Recovery Experiment

For nibble i, the first-round PRESENT input is

$$X_i = P_i \oplus K_i^{(1)}.$$

Let $A_a(\cdot)$ denote the affine component associated with the exploitable share pair. The corresponding leakage model is

$$L_4 = \mathrm{HW}(a_0 \oplus a_1) = \mathrm{HW}\left(A_a(P_i \oplus K_i^{(1)})\right). \tag{39}$$

For each four-bit key hypothesis

$$g \in \{0, \ldots, 15\},$$

we construct the predicted leakage signature and compare it with the observed secret signature using Euclidean distance,

$$D_i(g) = \sqrt{\sum_v \left\|\mathcal{S}_{i,\mathrm{sec}}(v) - \mathcal{S}_i(g; v)\right\|^2}. \tag{40}$$

The nibble estimate is

$$\widehat{g}_i = \arg\min_g D_i(g). \tag{41}$$

All 16 first-round S-box nibbles are recovered correctly. Consequently,

$$16 \times 4 = 64 \tag{42}$$

bits of first-round key material are recovered, yielding the complete 64-bit first-round PRESENT subkey.

This experiment establishes a complete simulator-side chain:

$$\begin{aligned} \text{architecturally separated shares} &\to \text{physical-register reuse} \\ &\to \mathrm{HD}(a_0, a_1) \\ &\to \text{key-dependent affine signature} \\ &\to \text{64-bit first-round subkey recovery.} \end{aligned} \tag{43}$$

4.8 Complementary PINI Composition Experiment

Masked PRESENT establishes exploitability of one concrete backend transition. We next use a controlled PINI1 composition to evaluate a different question: whether observations introduced by OoO execution remain within the simulator-support budget assumed by first-order PINI.

The experiment is intentionally separated from the PRESENT key-recovery claim. A PINI simulator-bound violation is a formal result under the extended observation model; it is not by itself evidence of an analog side channel or key leakage.

Controlled programs. We evaluate three progressively extended computations:

$$\begin{aligned} &\mathcal{M}_1 : G_1, \\ &\mathcal{M}_2 : G_1 \to L_{\mathrm{lin}}, \\ &\mathcal{M}_3 : G_1 \to L_{\mathrm{lin}} \to G_2, \end{aligned} \tag{44}$$

where G_1 and G_2 are first-order PINI1 multiplication gadgets and L_{lin} is evaluated share-wise.

The baseline configuration uses:

- 36 integer physical registers,
- a 128-entry ROB,
- four-wide execution,
- a 2.5 GHz clock,
- 64 dynamic iterations,
- one discarded warm-up iteration, and
- 32 iterations subjected to the formal support analysis.

The issue queue is left at the default provided by the instantiated simulator build.

Table 6: Controlled first-order PINI analysis at 36 integer physical registers. Counts denote dynamic observations whose minimum support exceeds the first-order simulator budget.

Mode	Tagged	Formal	Local	Cross	PRF	Complete
G_1	448	96	96	0	32	64
$G_1 \to L_{\text{lin}}$	512	128	96	32	32	96
$G_1 \to L_{\text{lin}} \to G_2$	960	256	128	128	128	128

One physical-register observation. Consider two tagged Boolean intermediates from different circuit-share domains:

$$Q_0 = A_0 B_0 \tag{45}$$

and

$$P_{10}^0 = (A_1 \oplus 1) R_0. \tag{46}$$

If the backend creates the physical history

$$P_k \leftarrow Q_0, \qquad P_k \leftarrow P_{10}^0, \tag{47}$$

the one-bit transition observation is

$$O_{\text{PRF}} = A_0 B_0 \oplus (A_1 \oplus 1) R_0. \tag{48}$$

Neither a D_0-only nor a D_1-only simulator suffices to reproduce the full distribution of this observation. Hence,

$$S_{\min}(O_{\text{PRF}}) = \{D_0, D_1\}, \tag{49}$$

which exceeds the first-order simulator budget under the considered OoO observation model.

Composition-boundary observations. The analysis also distinguishes local observations from those spanning components in

$$G_1 \to L_{\text{lin}} \to G_2.$$

A cross-composition event is not automatically a violation. The event is classified as such only when its modeled distribution requires both circuit-share indices.

Two observation models are evaluated:

- `PRF_HD1`, representing the one-bit transition $V_{\text{old}} \oplus V_{\text{new}}$ for consecutive physical-register writes; and
- `COMPLETE_SUM1`, an explicit model-level aggregate over tagged values completing at the same backend time.

The latter is an analysis abstraction and is not interpreted as proof of a literal shared analog writeback bus.

Baseline results. Table 6 summarizes the 36-register experiment.

The cross-composition violation count therefore progresses as

$$0 \to 32 \to 128. \tag{50}$$

The same three runs contain 1, 2, and 13 recurring physical-register transition types, respectively. These recurring types are structurally distinct relationships and must not be confused with the dynamic violation-row counts in Table 6.

Table 7: Sensitivity of the complete PINI composition to integer physical-register capacity. Values denote simulator-violation rows over the 32 analyzed iterations.

PRFs	$\mathcal{M}_3$ Cross	$\mathcal{M}_3$ PRF_HD1
36	128	128
40	192	128
44	128	64
48	96	32
52	96	32
56	64	0
64	64	0
72	64	0
80	64	0
96	64	0
128	64	0
192	64	0

The isolated gadget already contains local OoO-created observations requiring both circuit-share indices. Consequently, these results do *not* establish a composition-only counterexample to PINI. Rather, composition introduces additional observations spanning otherwise valid PINI boundaries under an observation model containing OoO backend state.

Physical-register sensitivity. We repeat the controlled experiment while varying the integer physical-register capacity between 36 and 192 entries.

Table 7 reports the key resource-pressure result for the complete composition $\mathcal{M}_3$.

Direct physical-register-transition violations decrease with additional register capacity:

$$128 \to 64 \to 32 \to 0,$$

and disappear from 56 integer physical registers onward.

However, cross-composition simulator violations remain at 64 rows from 56 through 192 registers. Thus, increasing the physical-register capacity removes the direct reuse-driven violation class but does not eliminate every OoO-created observation that exceeds the first-order PINI simulator budget.

The aggregate cross count is not globally monotonic. We therefore interpret this sweep as evidence that the observations are *backend-resource sensitive*, rather than claiming a simple monotonic relationship between physical-register capacity and masking security.

4.9 SiFive P550 Thermal Validation

We next evaluate whether repeated key-dependent processor activity becomes observable on a real SiFive P550-class OoO RISC-V processor.

The physical board exposes no backend rename map, physical-register identifiers, or execution-event trace. Instead, the experiment uses temperature telemetry provided through the Linux hardware-monitor interface:

```
/sys/devices/platform/soc/50b00000.pvt/hwmon/hwmon1/temp1_input
/sys/devices/platform/soc/52360000.pvt/hwmon/hwmon2/temp1_input
```

Each sensor reports temperature in millidegrees Celsius. When both are available, the experiment uses

$$T = \frac{T_1 + T_2}{2}, \tag{51}$$

and measures

$$\Delta T = T_{\text{after}} - T_{\text{before}}. \tag{52}$$

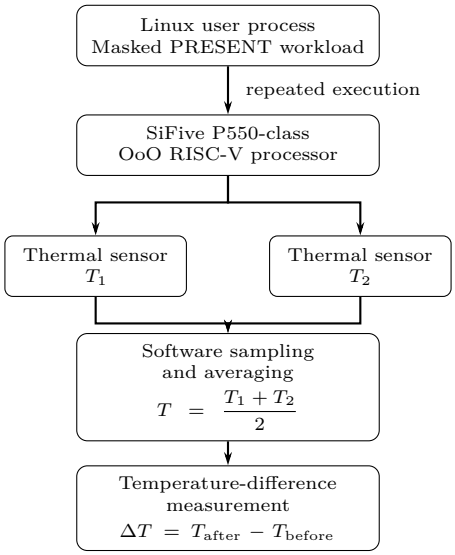


Figure 4: Schematic of the SiFive P550 thermal experiment. Masked PRESENT executes repeatedly on the OoO RISC-V processor. Two software-accessible thermal sensors are sampled and averaged, and the temperature difference before and after repeated execution is used as the physical observation.

Table 8: Measurement constants used in the SiFive P550 experiment.

Parameter	**Value**
Inner repetitions per invocation	24
Outer repetitions per measurement	6000
Total encryptions per measurement	144,000
Cooling interval before measurement	8 s
Settling interval after execution	2 s
Temperature samples per reading	9
Gap between temperature samples	200000 μs

Temperature cannot resolve a single physical-register transition. It is instead a slow, integrated signal that reflects accumulated processor activity. Small key-dependent switching differences are therefore amplified through repeated masked-PRESENT execution.

Figure 4 gives an independently drawn representation of the physical setup. It captures the relevant information from the experimental platform without reproducing the submission-time photograph.

The victim executes with the 80-bit PRESENT master key

$$\texttt{A73C910F55C826B4E17D}. \tag{53}$$

Each measurement contains

$$24 \times 6000 = 144{,}000$$

masked PRESENT encryptions.

Table 8 summarizes the acquisition constants.

4.10 Forced-Reference Thermal Recovery

The physical experiment uses a *profiled forced-reference* methodology and targets one master-key bit at a time.

For bit j, three distributions are measured:

$$\Delta T_0^{(j)}, \qquad \Delta T_1^{(j)}, \qquad \Delta T_s^{(j)}, \tag{54}$$

where the first two correspond to executions in which the target bit is forced to 0 or 1, and the third corresponds to the secret-key execution.

For each distribution, we compute a trimmed median,

$$m_0^{(j)}, \qquad m_1^{(j)}, \qquad m_s^{(j)}. \tag{55}$$

Trimming reduces sensitivity to scheduling variation, thermal drift, sensor quantization, and unrelated background activity.

The secret measurement is compared with the two references using

$$d_0^{(j)} = \left| m_s^{(j)} - m_0^{(j)} \right|, \qquad d_1^{(j)} = \left| m_s^{(j)} - m_1^{(j)} \right|, \tag{56}$$

and the bit is classified as

$$\widehat{k}_j = \begin{cases} 1, & d_1^{(j)} < d_0^{(j)}, \\ 0, & d_0^{(j)} \le d_1^{(j)}. \end{cases} \tag{57}$$

Positions exhibiting weak reference separation are retargeted for additional measurements.

Figure 5 shows the recovery procedure in a layout different from that used in the submission.

The forced-reference experiment is a controlled profiling procedure intended to establish key-dependent physical observability. It is not presented as a blind single-trace attack.

4.11 P550 Recovery Result

The thermal classifier directly recovers

$$60 \tag{58}$$

of the 80 PRESENT master-key bits, corresponding to a

$$75\% \tag{59}$$

direct bit-recovery rate.

The 20 unsuccessful positions are known:

$$\begin{aligned} \{&79, 78, 76, 73, 51, 48, 40, 39, 36, 33, \\ &30, 28, 22, 18, 17, 14, 9, 8, 5, 2\}. \end{aligned} \tag{60}$$

Hence the physical experiment does not leave an unconstrained 80-bit search. The unresolved key space is reduced to

$$2^{20} \approx 1.05 \times 10^6 \tag{61}$$

candidate master keys.

Exhaustive search over this residual space completes recovery of the full 80-bit PRESENT master key,

$$\boxed{\texttt{A73C910F55C826B4E17D}}. \tag{62}$$

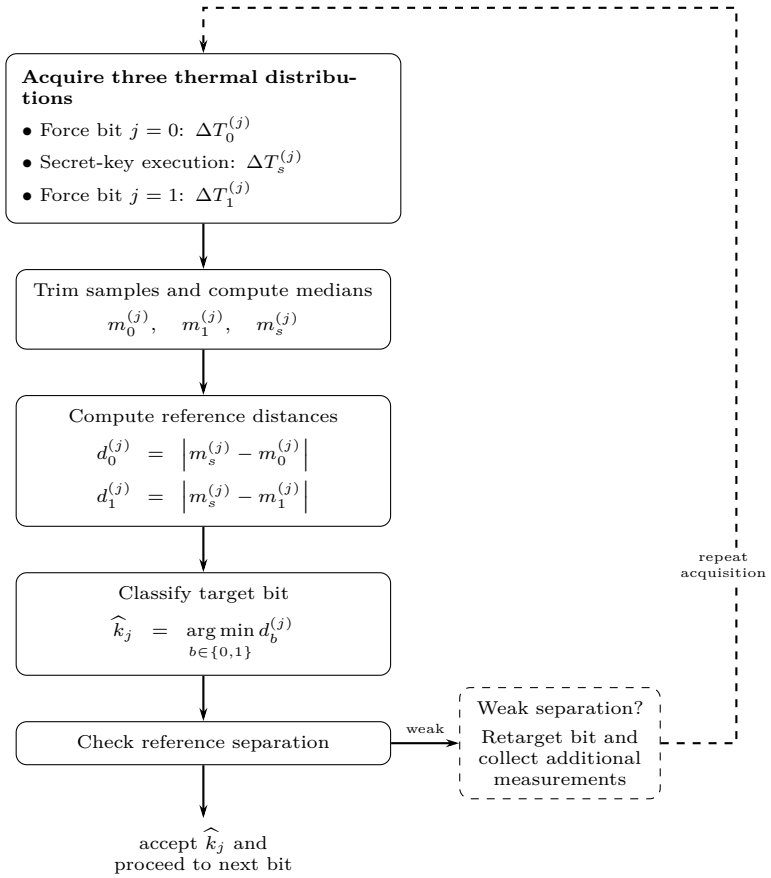


Figure 5: Profiled forced-reference thermal recovery procedure. For each target bit, forced-0, forced-1, and secret-key executions produce three thermal distributions. Trimmed medians are compared to classify the secret bit. Positions with weak reference separation are retargeted and the acquisition stage is repeated.

Table 9 summarizes the distinction between direct thermal classification and final key recovery.

For completeness, the bit-wise classification stage before residual search produces the raw classification string

$$\texttt{753C9106545A76F2A259}, \tag{63}$$

which agrees with the secret at 60 positions and differs at the 20 positions listed in Eq. 60. We therefore refer to this value as the *direct classification output*, rather than the final recovered key.

Figure 6 visualizes the direct classification outcome using an 8×10 layout that differs from the submission-time representation.

4.12 Connecting Backend Attribution and P550 Observability

The gem5 and P550 experiments observe different levels of the same cross-layer process.

The simulator directly exposes physical-register history and identifies the affine pair

$$(a_0, a_1)$$

as the clearest exploitable PRESENT recombination channel. The physical processor

Table 9: Summary of the SiFive P550 masked-PRESENT recovery.

Quantity	**Value**
Target master-key size	80 bits
Bits directly classified correctly	60
Unresolved bit positions	20
Direct bit-recovery rate	75%
Residual candidate space	2^{20}
Final key after residual search	`A73C910F55C826B4E17D`

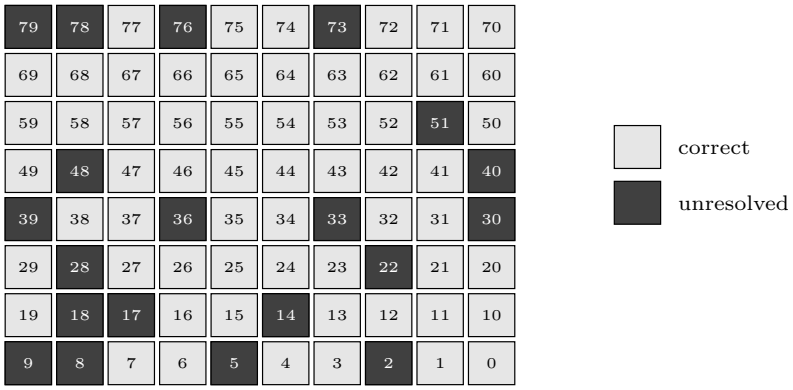


Figure 6: Direct P550 bit-classification result arranged as eight groups of ten key positions. Sixty positions are classified correctly; the twenty unresolved locations define the subsequent 2^{20} residual search.

exposes no such internal state; its thermal sensors instead integrate processor activity over many repeated encryptions.

The experimentally supported interpretation is therefore

$$\begin{aligned} \text{masked PRESENT shares} &\rightarrow \text{OoO physical-register reuse} \\ &\rightarrow \text{key-dependent HD activity} \\ &\rightarrow \text{repeated processor switching} \\ &\rightarrow \text{thermal accumulation} \\ &\rightarrow \text{60-bit direct classification} \\ &\rightarrow 2^{20} \text{ residual search} \\ &\rightarrow \text{80-bit master-key recovery.} \end{aligned} \tag{64}$$

We do not claim that one individual simulated RIL event corresponds directly to one P550 temperature measurement. The strict simulator detector isolates contiguous share-to-share physical-register transitions, whereas the thermal sensor integrates the aggregate activity of the processor over 144,000 encryptions per measurement.

Accordingly, the two experiments support complementary claims. The simulator identifies and attributes a concrete key-dependent backend mechanism; the P550 experiment demonstrates that repeated masked execution contains sufficient key-dependent physical information to recover most master-key bits directly and reduce the remaining uncertainty to a tractable exhaustive-search space.

The controlled PINI experiment provides an orthogonal result. It shows that OoO-created observations can also exceed a formal simulator-support budget even when no physical key-recovery claim is made. Together, the experiments demonstrate why masked-

software security must distinguish among architectural correctness, backend observation, formal masking-model violation, key dependence, and physical exploitability.

5 Conclusion

This work studies how masked cryptographic software behaves when executed on modern out-of-order (OoO) RISC-V processors. Using a trace-driven backend analysis methodology, we analyzed masked PRESENT together with controlled PINI compositions and identified additional observations created by hidden OoO mechanisms, including physical-register reuse and transient execution overlap.

In the masked PRESENT case study, the clearest exploitable leakage channel originates from backend reuse of the affine share pair (a_0, a_1), producing transitions of the form

$$\mathrm{HW}_{\mathrm{bit}}(a_0[b] \oplus a_1[b]).$$

Although the implementation employs Q12-style rotated-share protection around selected nonlinear computations, the affine shares remain separately represented at the architectural level and can successively occupy the same physical register. Since

$$a_0 \oplus a_1 = a,$$

such physical-register transitions reconstruct the affine PRESENT intermediate at the leakage-model level even though the software never explicitly recombines the shares.

Using an instrumented gem5 OoO RISC-V model, we attribute this leakage to physical-register reuse and recover the complete 64-bit first-round PRESENT subkey from masked execution traces. An in-order execution of the same masked binary produces no rename-induced transition candidates, further localizing the identified mechanism to the OoO backend.

The controlled PINI experiments provide a complementary formal result. OoO-created observations are evaluated according to the minimum circuit-share support required to reproduce their distributions. Under the considered first-order observation model, the isolated gadget already exhibits local simulator-bound violations, while extending the computation through

$$G_1 \to L_{\mathrm{lin}} \to G_2$$

introduces additional cross-composition violations. Under the stressed baseline configuration, the cross-composition violation count progresses from 0 to 32 and then to 128. These results do not contradict PINI under its original probing model; rather, they show that its composability guarantee does not automatically extend to additional observations introduced by an OoO backend.

Finally, we evaluate physical observability on a real SiFive P550-class OoO RISC-V processor using Linux-accessible thermal telemetry. A profiled forced-reference methodology with trimmed-median classification directly recovers 60 of the 80 PRESENT master-key bits. Since the locations of the remaining 20 unresolved bits are known, the residual key space contains only

$$2^{20}$$

candidates. Exhaustive search over this reduced space completes recovery of the full 80-bit PRESENT master key.

Overall, the results demonstrate that masking shares that remain separated in software and architectural state can acquire additional relationships inside hidden OoO backend structures. For PRESENT, these relationships create key-dependent physical-register transitions that propagate into measurable thermal behavior and enable full key recovery. For PINI, they can create observations that exceed the simulator-support budget assumed

by the original masking model. These findings motivate hardware-aware verification of masked software that explicitly accounts for register renaming, physical-register reuse, dynamic scheduling, and other backend execution effects.

References

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 398–412, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[CMBM25] Siddhartha Chowdhury, Nimish Mishra, Sarani Bhattacharya, and Debdeep Mukhopadhyay. Architectural leakage analysis of masked cryptographic software on RISC-v cores. Cryptology ePrint Archive, Paper 2025/1944, 2025.

[DKV26] Eden Desmet, Suparna Kundu, and Ingrid Verbauwhede. Masking out of order: Side-channel leaks from software-masked cryptography on out-of-order processors. Cryptology ePrint Archive, Paper 2026/123, 2026.

[FGMDP+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 Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, Aug. 2018.

[GD23] John Gaspoz and Siemen Dhooghe. Threshold implementations in software: Micro-architectural leakages in algorithms. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):155–179, Mar. 2023.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[LCGD18] Yann LE CORRE, Johann GROSZSCHÄDL, and Dumitru-Daniel DINU. Micro-architectural power simulator for leakage assessment of cryptographic software on arm cortex-m3 processors. In *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*. Springer, April 2018.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 413–427, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[ZMM23] Jannik Zeitschner, Nicolai Müller, and Amir Moradi. Prolead_sw: Probing-based software leakage detection for arm binaries. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):391–421, Jun. 2023.