

Article

PACE: A Page-Adaptive, Cache-Anchored Memory Encryption Engine for RISC-V with Formally Verified n th-Order DPA Resistance

Jyotiprakash Mishra ^{1,2,*} , Sanjay K. Sahay ¹ , Swati Mishra ³ and Aman Pathak ⁴

¹ Department of Computer Science and Information Systems, Birla Institute of Technology and Science (BITS) Pilani, Zuarinagar 403726, Goa, India; ssahay@goa.bits-pilani.ac.in

² School of Computer Engineering, Kalinga Institute of Industrial Technology (KIIT), Bhubaneswar 751024, Odisha, India

³ Independent Researcher, Cuttack 754001, Odisha, India; swatimishra0525@gmail.com

⁴ ENineHQ Technologies, Cuttack 754001, Odisha, India; aman@eninehq.com

* Correspondence: p20210041@goa.bits-pilani.ac.in

Abstract

Main memory carries data outside the processor's trust boundary, so commodity systems-on-chip (SoCs) increasingly encrypt it; yet, in-line memory encryption engine itself becomes a differential power analysis (DPA) target whose key, if recovered, unlocks all of dynamic random-access memory (DRAM). We present PACE, a page-adaptive, cache-anchored memory encryption engine for RISC-V that makes n th-order DPA resistance practical and keeps cryptographic latency off the cache eviction critical path. PACE inserts a TileLink adapter between the last-level cache and the memory port and applies, per physical page, one of four policies (plaintext/confidentiality/confidentiality+integrity/+masking-order- d) selected from RISC-V page table bits through a memory-mapped control plane. Confidentiality uses counter mode whose per-line keystream is precomputed during cache residency; integrity is tree-free at the embedded operating point via on-chip counters and tags, with a live split counter block-MAC Bonsai Merkle tree for scale-out. DPA resistance is layered: ISAP-style fresh re-keying caps the data complexity per key at $q \approx 1$, and domain-oriented masking (DOM, $d + 1$ shares) protects the sole key processing block to order d . We implement PACE in Chisel on a Rocket SoC (Chipyard) and evaluate it with open-source tooling. A deterministic TileLink-level harness proves ciphertext-in-memory and detects tamper/replay/splice, and the live Tier-B engine (DRAM counters and per-line message authentication codes (MACs) plus an on-chip-rooted block-MAC tree) is validated from end to end on full Rocket and BOOM SoCs and on the FPGA; the masked Ascon- p S-box is proven order- d secure ($d = 1, 2$) under a glitch- and transition-aware model by three independent formal tools (COCO, PROLEAD, and SILVER, the last also deciding the full composability lattice and confirming exact glitch-robust order-2 probing security), with COCO extending the exact verdict to the highest synthesized order $d = 3$ (secure at probing orders 1–3); a simulated trace correlation power analysis (CPA) recovers the full key from an unprotected core and is defeated by masking, with a mutual information analysis confirming the $N \propto \sigma^{2(d+1)}$ trace amplification law. We further realize PACE on field-programmable gate array (FPGA) silicon: the engine plus an on-chip ring oscillator power sensor is placed, routed, timing-closed at 100 MHz, and programmed on a Xilinx XC7Z020, and we drive a fixed-vs-random Test Vector Leakage Assessment (TVLA) campaign read back entirely over a JTAG (Joint Test Action Group). A multi-core configuration and a Linux control-plane driver are likewise validated. Across synthetic access patterns and named application kernels (AES, SHA-256, matrix multiplication, pointer chasing) on both in-order Rocket and out-of-order BOOM,



Academic Editor: Paris Kitsos

Received: 25 June 2026

Revised: 27 July 2026

Accepted: 5 August 2026

Published: 7 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

application-level overhead is within measurement noise of plaintext for cache resident workloads (masking, in particular, is cycle-identical to plain confidentiality), and we characterize the cost of each policy, masking order, and re-keying interval, demonstrating side-channel-hardened memory encryption on open RISC-V hardware.

Keywords: memory encryption; side-channel analysis; differential power analysis; masking; domain-oriented masking; RISC-V; Rocket; Chipyard; Ascon; ISAP; counter mode; memory integrity; Bonsai Merkle tree; leakage-resilient cryptography

1. Introduction

Main memory sits outside the processor’s trust boundary. An attacker with physical access can read or tamper with dynamic random-access memory (DRAM) through cold boot attacks, bus interposers, or malicious direct memory access (DMA) peripherals, and in multi-tenant settings a privileged-but-untrusted hypervisor can inspect a guest’s pages. Commodity processors therefore ship transparent memory encryption (e.g., AMD Secure Memory Encryption/Secure Encrypted Virtualization, SME/SEV; Intel Total Memory Encryption/Multi-Key TME/Trust Domain Extensions, TME/MKTME/TDX; Arm Confidential Compute Architecture, CCA) that encrypts (and increasingly authenticates) data as it crosses the memory port [1–3]. The open-hardware ecosystem, however, lacks a memory encryption engine that is both co-designed with the cache hierarchy and *hardened against physical side-channel analysis of the engine itself*, with an end-to-end evaluation.

The second gap is the dangerous one. A memory encryption engine continuously processes a single, long-lived key over attacker-influenced data at a fixed physical location (the engine’s interface to off-chip memory), which is the setting differential power analysis (DPA) exploits [4,5]. Recovering the engine key compromises the confidentiality and integrity of *all* protected memory at once. Yet side-channel hardening is largely absent from open memory encryption engines, and where masking is applied it is rarely *formally* verified on the synthesized gate-level artifact under a glitch-aware model.

1.1. The PACE Approach

We present PACE (Page-Adaptive, Cache-anchored Encryption), a memory encryption engine for RISC-V that closes both gaps. PACE is a TileLink adapter inserted on the memory bus edge between the last-level cache (LLC) and the memory port of a Rocket system-on-chip (SoC) built in Chipyard. Three ideas organize the design.

Page-adaptive, physical address-indexed policy. Not all memory is equally sensitive, and uniform worst-case protection wastes energy and bandwidth. PACE applies, *per physical page*, one of four policies (plaintext, confidentiality, confidentiality+integrity, and +masking-order-*d*) chosen from RISC-V page table bits. Because the engine sits *below* the LLC, which does not forward inner TileLink user fields, the policy cannot ride with the request; it must be physical address (PA) indexed. PACE resolves this with a memory-mapped control plane that the operating system (OS) programs with page table entry (PTE)-derived physical ranges (Section 10).

Cache-anchored latency hiding. Counter-mode (CTR) confidentiality lets the keystream for a line be computed from its address and version counter *before* the plaintext is known. PACE precomputes the next writeback’s keystream while the line is still resident in the cache, so the eventual eviction needs only an XOR, removing the cipher from the eviction-critical path (Section 9).

Layered, formally verified side-channel resistance. PACE caps the data complexity available to a DPA adversary by fresh re-keying in the style of ISAP (a leakage-resilient authenticated

encryption scheme), so each session key encrypts a single line once ($q \approx 1$); and it masks the sole key processing block with domain-oriented masking (DOM) to arbitrary order d . The masked Ascon- p S-box is verified order- d secure by three independent formal tools (COCO, PROLEAD, and SILVER), and the masking order's effect is quantified by an information-theoretic trace amplification law that we confirm empirically (Sections 13 and 14).

1.2. Contributions

1. **A page-adaptive, PA-indexed memory encryption engine** for RISC-V with a memory-mapped I/O (MMIO) control plane of control/status registers (CSRs), integrated and validated at the register transfer level (RTL) on a Rocket SoC with a zero-overhead bypass for plaintext traffic (Sections 4–10).
2. **Residency-precomputed CTR-mode confidentiality** that removes keystream generation from the eviction-critical path, with a measured $9\times$ reduction in writeback latency (Sections 9 and 16).
3. **A unified split counter** that simultaneously serves as the CTR nonce, the integrity version, and the re-keying epoch, and **tree-free Tier-A integrity** with a live split counter block message authentication code (MAC) Bonsai Merkle Tier-B for scale-out (Sections 6 and 8).
4. **Formally verified n th-order masking**: a DOM-dependent masked Ascon- p S-box proven order- d secure by *three* independent glitch-aware tools (COCO, PROLEAD, and SILVER at $d = 1, 2$, with COCO extending the exact verdict through $d = 3$), the last also deciding the full composability lattice and confirming exact glitch-robust order-2 probing security, including a careful treatment of the multiple comparisons effect that distinguishes a statistical detector's borderline output from an actual leak (Section 14).
5. **An analytic side-channel model** relating masking order, noise, and re-keying to traces-to-disclosure ($N \propto \sigma^{2(d+1)}$ amplification and the $q \approx 1$ re-keying bound), validated against simulated trace correlation power analysis (CPA) and Test Vector Leakage Assessment (TVLA) and a mutual information sweep (Sections 13 and 15).
6. **An open-source evaluation spanning simulation to silicon**: functional and integrity correctness (including the live Tier-B engine from end to end on Rocket and Berkeley Out-of-Order Machine (BOOM) SoCs), three-tool formal masking, simulated trace DPA, a SP 800-90B-monitored true random number generator (TRNG), a dual-core shared engine and a Linux control-plane driver, and, on a Xilinx XC7Z020, a placed/routed/timing-closed and *programmed* engine with an on-chip ring oscillator power sensor and a fixed-vs-random TVLA campaign read back over the JTAG (Joint Test Action Group) interface, alongside a performance sweep across access patterns, policies, masking orders, and re-keying intervals (Sections 15–17).

1.3. What Is Fundamentally New

PACE is not a re-tuning of an existing engine; its novelty lies in a specific composition, one new mechanism, and where the whole is placed. To our knowledge: *(i) First cache-co-designed, formally side-channel-hardened open engine.* PACE is the first open-hardware memory encryption engine that is *simultaneously* co-designed with the cache hierarchy *and* hardened against side-channel analysis of the engine itself with formal, gate-level evidence, and evaluated from end to end from RTL simulation to programmed FPGA silicon. Commodity engines (SME/SEV, TME/TDX, CCA) are closed and neither expose nor verify the engine's side-channel posture; secure memory integrity tree designs (BMT, VAULT, Morphable, Synergy) optimize the metadata hierarchy but treat the cipher as a black box and address no power side channel; and masking constructions and formal masking tools (DOM, maskVerif, SILVER, COCO, PROLEAD) are components that have

not been composed into a cache-co-designed memory engine with an end-to-end evaluation. (ii) *Cache-anchored keystream precompute is a new mechanism*, not a parameter change: placing the engine *below* the LLC and exploiting cache residency lets PACE compute the next writeback's keystream *before* eviction, taking the cipher off the eviction-critical path (a measured $9\times$ latency reduction), something an engine sitting at the memory controller, blind to residency, cannot do. (iii) *A unified split counter* collapses three roles that prior designs keep separate (the counter-mode nonce, the anti-replay integrity version, and the re-keying epoch) into a single piece of metadata; this collapse is what makes bounded-data re-keying ($q \approx 1$) and arbitrary-order masking affordable *together* on an embedded engine. (iv) *First netlist-level, three-tool formal masking on a memory engine, on real silicon*. PACE is, to our knowledge, the first memory encryption engine whose masked key processing core is verified on the *synthesized netlist* by three independent glitch/transition-aware tools with an explicit statistical-versus-exact reconciliation, and the first demonstrated on real low-cost FPGA silicon with an on-chip leakage assessment campaign; most side-channel-hardened memory encryption work never leaves simulation. Section 18 develops these contrasts in detail, and Section 19 revisits their implications. The paper then proceeds from the threat model (Section 3) through the architecture and microarchitecture (Sections 4 and 5), the cryptographic core with its masking, integrity, and precompute mechanisms (Sections 6–9), the control plane and implementation (Sections 10 and 11), the analytic, formal, and simulated side-channel evaluation (Sections 13–15), and the performance, area, and power results (Sections 16 and 17), closing with related work, discussion, and limitations (Section 18 and later).

2. Background and Preliminaries

This section provides the background needed to follow the design, with intuitive examples of the two mechanisms readers most often find dense: counter-tree integrity and masking. A reader fluent in memory encryption and side-channel countermeasures may skip to the threat model (Section 3).

2.1. Why Main Memory Is Untrusted

The processor die is the trust boundary; the off-chip memory and the bus that carries traffic to it lie outside and are exposed. An attacker with physical access can read residual data with a cold boot attack, observe or alter traffic with a bus interposer, or issue rogue transactions from a malicious DMA peripheral; in a multi-tenant server, a privileged-but-untrusted hypervisor can read a guest's pages. Transparent memory encryption answers this by encrypting (and, increasingly, authenticating) data as it crosses the memory port such that off-chip contents are ciphertext and tampering is detected. We formalize the adversary in Section 3; here, we recall the primitives.

2.2. Confidentiality: Counter-Mode Encryption

Counter (CTR) mode [6] is the standard confidentiality primitive for memory because it turns a block cipher (or permutation) into a stream cipher whose keystream depends only on a *nonce*, not on the data. For each memory line the engine computes a keystream $E_K(\text{nonce})$ and XORs it with the plaintext; decryption recomputes the same keystream and XORs again.

Worked example (nonce construction).

Consider a 64-byte line at physical address A holding version counter v . The engine forms a nonce $\text{nonce} = \text{dom} \parallel A \parallel v$ (a domain tag, the address, and the version). Two properties follow directly. (i) *Spatial uniqueness*: because A is in the nonce, two lines at different addresses never share a keystream, so identical plaintexts stored in different places

yield different ciphertexts. (ii) *Temporal uniqueness*: rewriting the same line increments v , producing a fresh keystream; an attacker who records an old ciphertext cannot make it decrypt correctly after a new write (the basis of replay detection). The keystream is a function of (A, v) *only*; it can be computed *before* the plaintext is known, the property PACE exploits to precompute keystream during cache residency (Section 9). The catch is that the version counters must themselves be stored and protected, which is the scaling problem integrity trees solve.

2.3. Integrity and Freshness: From MACs to Counter Trees

Authenticated memory must resist three attacks. *Spoofing* substitutes forged ciphertext; *splicing* moves a valid ciphertext line to a different address; *replay* rewinds a line (and its counter) to an earlier, legitimately authenticated state. A per-line message authentication code (MAC) computed over $(A, v, \text{ciphertext})$ stops spoofing (the MAC will not verify on forged data) and splicing (the address A is bound into the MAC). Replay is harder: since a replayed line *was* valid, defeating it requires that the *version* be authenticated too, the role of an integrity tree.

From Merkle trees to Bonsai Merkle Trees.

A Merkle tree hashed over the data itself would authenticate every cache line but is bandwidth-prohibitive: each access walks a tree of data hashes. The key idea of Bonsai Merkle Trees (BMT) [7] is to build the tree only over the small *version counter* array: data lines carry flat MACs verified directly, and a compact Merkle tree authenticates the counters, so an attacker cannot rewind a counter without detection. The tree is now proportional to the counter storage, not the data.

Split counters, with a worked overflow example.

A full per-line 64-bit counter still makes the counter array (and hence the tree) large. Split counter schemes store one shared *major* counter per group of lines plus a small per-line *minor*; the effective version is the pair (major, minor). Suppose a 32-bit major is shared by a group of 64 lines, each with a 6-bit minor. A write to a line increments only its 6-bit minor, and the tree only needs to cover the majors—roughly a $64\times$ reduction in the authenticated counter array (Figure 1). When a minor overflows (after $2^6 = 64$ writes to that line), the major is incremented and all 64 lines in the group are *re-encrypted* under the new major, a rare event whose cost is amortized over the 64 writes that preceded it. This is the trade-off that VAULT, Morphable Counters, and Synergy tuned with variable arity and adaptive encodings [8–10]. PACE reuses the split counter but *unifies* it across encryption, integrity, and re-keying, and sizes the minor so that overflow re-encryption is almost never triggered at the embedded operating point (Section 6).

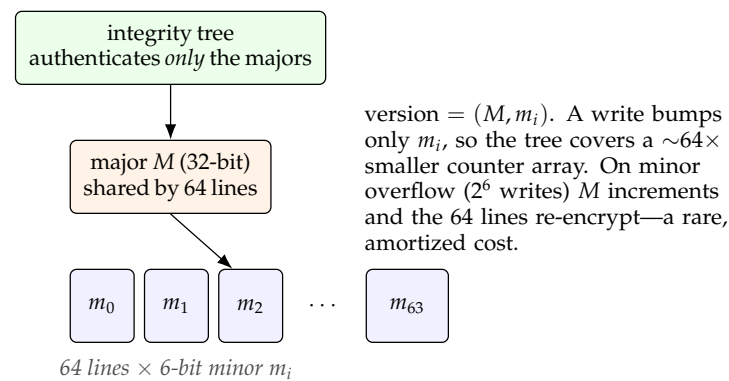


Figure 1. Split counter intuition. One shared major counter plus a small per-line minor gives each line a unique version (M, m_i) , yet the integrity tree only needs to authenticate the majors, a $\sim 64 \times$ smaller counter array. A write touches only a minor; the rare minor overflow re-encrypts the group under a new major. PACE unifies this counter across encryption, integrity, and re-keying.

At small footprints, PACE keeps the counters and tags entirely on-chip (tree-free Tier-A); for all-memory protection it falls back to a split counter BMT (Tier-B), whose traffic we quantify in Section 8.

2.4. Power Side Channels

A functionally correct cipher can still leak its key: the instantaneous power (or electromagnetic emission) a circuit draws depends on the data its gates switch. Differential power analysis (DPA) [4] exploits this statistically. For each hypothesis about a key portion, the attacker predicts an internal intermediate value (typically an S-box output) and correlates the prediction with measured power over many encryptions; the correct hypothesis correlates and the wrong ones do not (CPA [5]). TVLA [11] is a model-free *detector*: it partitions traces into a fixed-input and a random input group and applies a Welch t -test, flagging data-dependent leakage when $|t|$ exceeds a threshold (conventionally 4.5). A memory encryption engine is a near-ideal DPA target: it processes a single, long-lived key over attacker-influenced data, at a fixed physical location, for as many traces as the attacker cares to collect.

2.5. Masking, with a Worked Example

The dominant algorithmic countermeasure is *masking*, which randomizes every sensitive value so that observing a bounded number of wires reveals nothing.

Worked example (Boolean sharing).

To protect a secret bit s at order d , represent it by $d+1$ shares $s_0, \dots, s_d$ drawn uniformly at random subject to $s_0 \oplus s_1 \oplus \dots \oplus s_d = s$. Any d of these shares are jointly uniform and *statistically independent* of s : an adversary who probes up to d wires learns nothing about the secret (Figure 2). Recovering s (or correlating power against it) requires combining all $d+1$ shares simultaneously, and in the presence of measurement noise of variance σ^2 the number of traces needed grows as $\sigma^{2(d+1)}$ (Section 13). This exponential-in- d cost is what renders higher-order attacks impractical. Linear operations (XOR, the Ascon linear layer) apply share-by-share for free; the hard part is the nonlinear AND, where two shared values must be multiplied without ever forming a full secret.

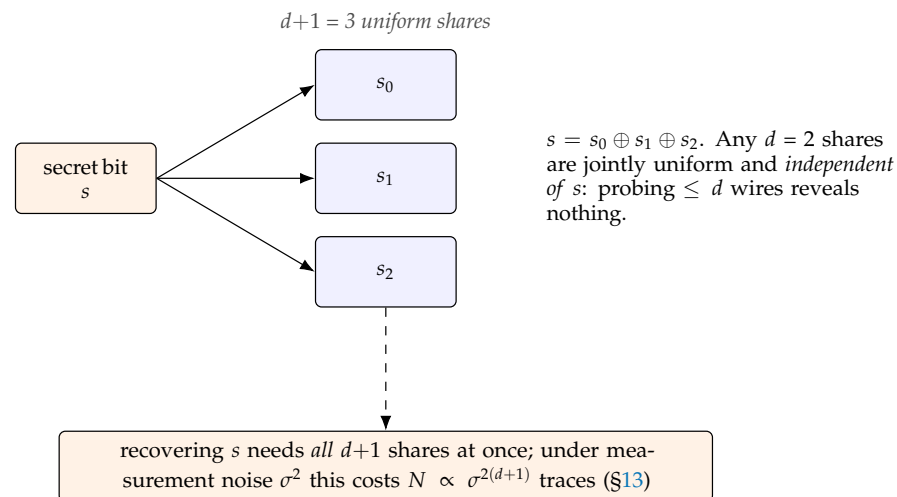


Figure 2. Boolean masking by secret sharing, shown at order $d = 2$. A secret bit is split into $d + 1$ uniformly random shares whose XOR is the secret; any d of them are independent of the secret, so an adversary limited to d probes learns nothing, and combining all $d + 1$ under noise costs $\sigma^{2(d+1)}$ traces. This is the intuition the formal tools (Section 14) verify at the gate level.

Glitches and threshold/domain-oriented masking.

A textbook masked AND is secure in a model that ignores circuit timing, but real hardware *glitches*: before signals settle, a gate can transiently switch as a function of a full secret, leaking through the power trace. Threshold implementations [12] and domain-oriented masking (DOM) [13] defeat glitches with two ingredients: *non-completeness* (no gate ever receives all shares of any one secret) and *registers* inserted so that a glitch cannot propagate across the multiplication. A DOM multiplier reshapes the cross-domain partial products with fresh randomness and registers them before recombination. One subtlety we exploit (Section 7): the DOM-independent multiplier assumes its two operands are independently shared, which is *false* for Ascon’s affine-premixed S-box inputs; the DOM-dependent variant, which first reshapes one operand, is required for correctness of the security argument.

2.6. Composability and Formal Leakage Models

Establishing that a whole cipher is secure, rather than a single gadget, requires *composability*. The probing model [14] calls a circuit d -probing-secure if no set of d internal wires jointly depends on a secret; the *robust* (glitch- and transition-extended) probing model additionally charges each probe with the transients it can observe, capturing the hardware reality above. Non-interference (NI), strong non-interference (SNI), and probe-isolating non-interference (PINI) [15] are successively more composable security notions: a gadget proven in the right notion can be wired into a larger circuit with the composite security preserved. DOM gadgets are NI and PINI but *not* SNI, so PINI is the correct framework for composing them, a fact our verification independently reproduces (Section 14). Automated verifiers decide these notions directly on the synthesized netlist; we survey them, and position PACE’s use of three of them, in Section 18.

2.7. Leakage-Resilient Cryptography and Re-Keying

Masking raises the *order* of attack; leakage-resilient design instead limits the *data* processed under any one key. ISAP [16] builds authenticated encryption on the Ascon permutation [17] with a Goldreich–Goldwasser–Micali (GGM)-style re-keying sponge (IsapRk) that absorbs the nonce one bit at a time, so the long-term key is exposed to minimal data complexity, sharply curtailing DPA at its root. Ascon’s sponge structure

and small 320-bit state make it well suited to embedded hardware, and Ascon is now the NIST lightweight cryptography standard. PACE uses Ascon- p as its single permutation primitive and ISAP-style re-keying so that each *session* key encrypts a single line once ($q \approx 1$); masking then only needs to protect the residual long-term key processing. This composition (re-keying to bound the data per key, masking to raise the order) is what makes n th-order resistance affordable on an embedded engine, and we analyze it quantitatively in Section 13.

3. Threat Model and Security Goals

We consider two tiers of adversary against off-chip memory: a *logical* adversary who reads and writes DRAM at will, and a stronger *physical* adversary who additionally measures the encryption engine's power or electromagnetic (EM) emanations to attack its key. This section fixes the trust boundary, states our assumptions explicitly, enumerates the attack surface, and gives the security goals formally.

3.1. Trusted Computing Base and Assumptions

The processor die is the trust boundary: the cores, caches, the PACE engine, its on-chip metadata SRAM and root, and the key registers lie *inside* (Figure 3). Everything beyond the engine's memory port (the DRAM chips, the module, and the bus to them) is untrusted. We make the following assumptions explicit, so the guarantees can be read against them.

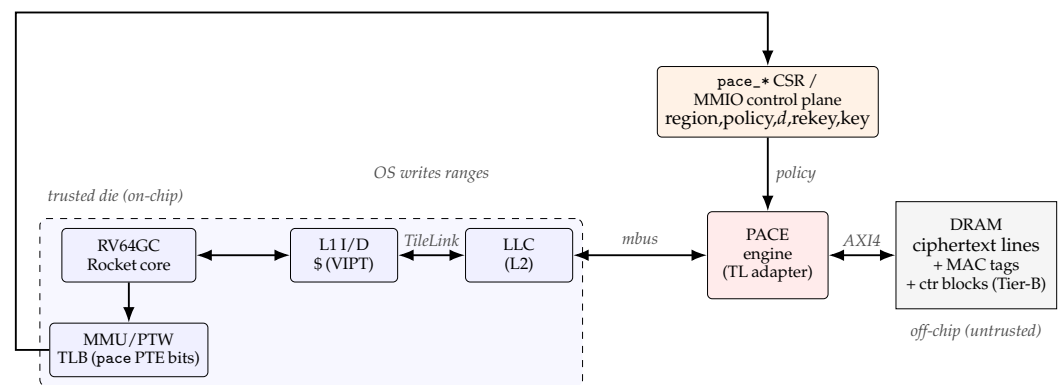


Figure 3. PACE in the SoC: the engine sits below the LLC on the mbus→memory edge; a memory-mapped control plane carries OS-programmed, physical address-indexed policy; DRAM holds ciphertext, MAC tags, and (Tier-B) counter blocks.

- A1** *On-chip trust.* On-chip state (the items above) is neither observable nor modifiable by the adversary; invasive attacks that violate this (decapsulation, chip probing) are out of scope.
- A2** *Key provisioning.* Keys are placed into on-chip key registers by a trusted boot flow or secure element and never leave the die in the clear.
- A3** *Randomness.* An on-chip randomness source (a health-tested (SP 800-90B) TRNG feeding the masking DRBG) supplies the fresh masks that masking requires; its health is monitored at run time (Section 7).
- A4** *Nonce non-repetition.* The (address, version) nonce never repeats under a fixed key, enforced by the monotone per-line version counter and address binding (Section 6).
- A5** *Verify before release.* Integrity is checked before any data is released to the core, so a rejected line never influences computation.

3.2. Adversary Capabilities and Attack Surface

Logical adversary. The logical adversary may read and write arbitrary DRAM locations at arbitrary times (by cold boot extraction, bus interposition, a malicious DMA

device, or a compromised peripheral) and, in a multi-tenant server, may be a privileged-but-untrusted hypervisor inspecting a guest’s pages. Its goal is to learn plaintext, or to have forged/stale ciphertext accepted as genuine.

Physical (side-channel) adversary. The physical adversary additionally measures the engine’s power or EM while it encrypts, mounting differential power analysis (DPA) on the encryption key. We assume the standard noisy-leakage setting (Section 13): the adversary observes the public nonce, can trigger encryptions of known or chosen lines, and collects an effectively unbounded number of traces of the engine’s key processing logic. Recovering the engine key is *catastrophic* (it compromises the confidentiality *and* integrity of all protected memory at once) which is why the engine, not merely the data, must be hardened. This adversary is the reason for PACE’s layered defense and is under-addressed by prior open engines.

Table 1 maps each attack vector to the capability it requires and the PACE mechanism that defends it.

Table 1. Attack surface and PACE’s defenses. Each row names an attack, the adversary capability it needs, the mechanism that defends it, and the security property achieved. The last two rows are out of scope (Section 3).

Attack	Adversary Capability	PACE Mechanism	Property
Disclosure/cold boot	read DRAM (live or powered-down snapshot)	CTR-mode confidentiality; per-line keystream bound to address + version	confidentiality
Spoofing	write forged ciphertext	per-line MAC over (addr, v, ct) , verified before release	integrity
Splicing	relocate a valid ciphertext line	address bound into the MAC	address binding
Replay/rollback	restore a stale but once-valid (ct, tag, v)	on-chip-authenticated version (tree-free Tier-A root; split counter BMT Tier-B)	freshness
Key recovery by DPA/EM	measure engine power/EM over many traces	ISAP-style fresh re-keying ($q \approx 1$) + DOM masking to order d , formally verified	order- d SC resistance
Software timing/cache side channels	observe core execution timing	—(orthogonal; core-level defenses)	out of scope
Fault injection/active EM	inject faults into the engine or memory	detected as an integrity fault, but not prevented	out of scope

3.3. Security Goals, Formally

Let $\mathcal{A}$ be the logical adversary with read/write oracles on DRAM. PACE targets the following, which the construction (Sections 6–8) and the side-channel analysis (Sections 13 and 14) establish.

Definition 1 (Confidentiality). *For any two equal-length plaintext line sequences chosen by $\mathcal{A}$, the induced ciphertext-in-DRAM distributions are computationally indistinguishable, given that the (address, version) nonce never repeats under a fixed key (A4). PACE enforces non-repetition by the monotone per-line version and the address-bound nonce (Section 6).*

Definition 2 (Authenticity & freshness). *Any DRAM state not produced by PACE for the current (address, version) is rejected before release except with probability $\leq 2^{-\tau}$ (τ the tag length). This unifies spoofing (ct altered), splicing (valid ct at the wrong address), and replay (stale but once-valid (ct, tag, v)), since the MAC binds (addr, v, ct) and the authority for v is on-chip (Section 8).*

Definition 3 (Order- d side-channel security). *Any set of $\leq d$ probes (gates/wires, glitch- and transition-extended) on the key processing logic is statistically independent of the key. PACE attains this by DOM-masking the sole nonlinear gadget, verified in Section 14, while re-keying bounds the per-key data complexity (Proposition 1), so the trace cost of a $(d+1)$ -th order attack grows as $\sigma^{2(d+1)}$ (Section 13).*

3.4. Out of Scope

Micro-architectural timing and cache side channels in the *core* (as opposed to the engine), fault injection and active-EM attacks, and decapsulation/probing that violate the on-chip trust assumption (A1) are out of scope; we discuss the residual physical gap (out-of-model leakage that would require a full measurement platform) in Section 20. Denial of service (an adversary corrupting memory to force faults) is *detected* by the integrity layer but cannot be prevented and is likewise out of scope.

4. PACE Architecture

Figure 3 shows PACE in a single-core Rocket SoC. The engine is a TileLink adapter spliced into the memory bus (mbus) between the LLC and the AXI4 memory controller, so it sees exactly the traffic that misses the on-chip cache hierarchy and crosses to DRAM. Plaintext-policy traffic is forwarded bit for bit with no added cycle (a registered bypass), so pages the OS does not mark sensitive pay nothing.

Why physical address indexed? A natural design would tag each request with its page policy and let the policy ride the TileLink user field from the core. This fails below the LLC: the LLC coalesces and reorders inner transactions and does *not* forward inner user bits to its outer (memory-side) port. The policy therefore cannot travel with the request and must be recovered from the one thing that survives: the physical address. PACE keeps a physical address-indexed region table; the OS, which owns the virtual→physical map, programs the engine with the physical ranges that back sensitive pages (Section 10). This is a fundamental consequence of placing the engine below the LLC, and it shapes the rest of the design: keys, counters, and tags are all indexed by physical line address.

Four policies. Each region carries one of four policies: (i) *plaintext*: bypass; (ii) *confidentiality*: CTR encryption; (iii) *confidentiality+integrity*: adds the per-line MAC and verify before release; (iv) *+masking-order-d*: the key processing block is DOM-masked to order d (a synthesis-time capability the control plane reports). Confidentiality and integrity are run-time policy bits; the masking order is structural (it changes the gate-level circuit) and is fixed at synthesis, with the control plane exposing the built order as a read-only capability register.

Trust boundary. The engine, its on-chip counter/tag SRAM, and the key registers are inside the die trust boundary (Section 3); only ciphertext and authenticated metadata cross to DRAM. The default configuration used throughout our evaluation is summarized in Table 2.

Table 2. PACE default parameters (Rocket SoC, Chipyard 1.13).

Parameter	Value
Permutation	Ascon- p , 320-bit state, folded 1 round/cycle
Round counts	p^{12} init/final, p^6 rate
Cache line/beat	64 B/8 B ($W = 8$ words)
Nonce	128-bit: dom(16) PA(48) PMAJ(32) LMIN(32)
Keys	128-bit K_E, K_A ; MAC tag 128-bit
Masking order d	0 (off), 1, 2 (synthesis-time)
Re-keying	per-write ($q \approx 1$) or per-epoch (PMAJ)
Control plane	MMIO <code>pace_*</code> at <code>ct1Base</code>
Target device	Xilinx XC7Z020 field-programmable gate array (FPGA), Zedboard

5. Engine Microarchitecture

Figure 4 details the datapath. A region check on the A channel splits traffic: non-encrypted requests take the registered bypass; encrypted Puts and Gets enter the write and read finite-state machines.

Write (eviction) path. On the first beat of an encrypted Put, the engine reads and increments the line's version counter, then either consumes a residency-precomputed keystream (state $wStream$ directly) or starts $IsapEnc$ and waits ($wKsw$). It then streams the line, XORing each beat with the keystream and forwarding ciphertext to DRAM. When integrity is enabled it computes the MAC over (address, version, ciphertext) and writes the tag to on-chip SRAM ($wMacS/wMacW$); the AccessAck is returned as soon as the ciphertext is accepted, so the MAC is off the response path. Algorithm 1 summarizes the flow; the read counterpart is Algorithm 2.

Read (refill) path, verify before release. On an encrypted Get, the engine launches the keystream for the current version and, on the matching DRAM response, buffers the ciphertext line *without* forwarding it. With integrity enabled, it recomputes the MAC and compares against the stored tag; only then does it stream decrypted beats to the LLC, asserting that TileLink is denied on a mismatch, so the core takes a load access fault. A stale version (replay) and a relocated line (splice) are caught by the same check because the MAC binds both. The keystream and the (optional) precompute of the *next* version's keystream overlap the DRAM latency, hiding crypto on refills.

Algorithm 1: PACE write (eviction) path

Input: Put of line at PA a , beats $p_0 \dots p_{W-1}$
 $v \leftarrow ctr[a] + 1; ctr[a] \leftarrow v$
if *precomputed ks valid for* (a, v) **then**
 | use cached ks
else
 | $ks \leftarrow IsapEnc(K_E^*, nonce(a, v))$
for $i \leftarrow 0$ **to** $W-1$ **do**
 | $c_i \leftarrow p_i \oplus ks_i$; forward c_i to DRAM
if *integrity* **then**
 | $tag[a] \leftarrow IsapMac(K_A^*, a \parallel v \parallel c)$

Algorithm 2: PACE read (refill) path, verify before release

Input: Get of line at PA a
 $v \leftarrow ctr[a]; ks \leftarrow IsapEnc(K_E^*, nonce(a, v))$
 launch precompute of ks' for $(a, v+1)$ off the critical path
 buffer ciphertext beats $c_0 \dots c_{W-1}$ from DRAM (do not forward)
if *integrity* **then**
 | $t \leftarrow IsapMac(K_A^*, a \parallel v \parallel c)$
 if $valid[a] \wedge t \neq tag[a]$ **then**
 | $fault \leftarrow true$ (spoof/splice/replay)
for $i \leftarrow 0$ **to** $W-1$ **do**
 | forward $p_i \leftarrow c_i \oplus ks_i$, **denied** $\leftarrow fault$

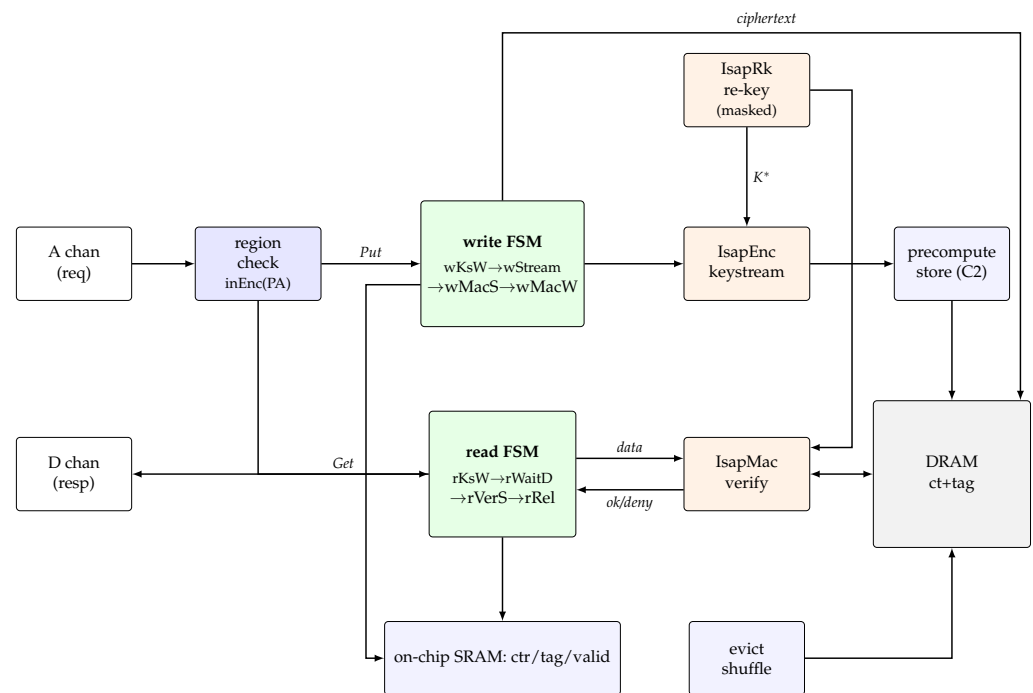


Figure 4. Engine datapath: region-gated write/read FSMs, masked re-keying, CTR keystream, Tier-A MAC with on-chip counter/tag SRAM, residency precompute, and the optional eviction shuffle.

Single permutation, reused. IsapEnc, IsapRk, and IsapMac all fold onto one Ascon- p datapath with a run-time round count (p_6 for the sponge rounds, p_{12} for initialization/finalization), so the area is dominated by a single permutation instance plus the masking overhead (Section 17). On-chip counter and tag SRAM is indexed by physical line number within the configured region.

6. Cryptographic Core

PACE uses the Ascon permutation Ascon- p as its single primitive, instantiated through three ISAP-style modes that share one folded datapath.

Ascon- p . The permutation operates on a 320-bit state of five 64-bit words $x_0 \dots x_4$ and applies, per round, a constant addition p_C ($x_2 \oplus= c_r$), a 5-bit S-box p_S applied bit-sliced across the lanes, and a linear diffusion p_L :

$$x_i \leftarrow x_i \oplus (x_i \ggg r_i^{(1)}) \oplus (x_i \ggg r_i^{(2)}), \quad (1)$$

with rotation constants $r_i^{(1,2)} \in \{(19, 28), (61, 39), (1, 6), (10, 17), (7, 41)\}$. The S-box is the only nonlinear layer (and the masking target, Section 7). PACE folds one round per cycle with a run-time round count, so a single instance serves p^{12} (initialization/finalization) and p^6 (rate processing). The RTL is bit-exact with the published Ascon vectors (Appendix A).

Confidentiality (CTR). Each 64-byte line is encrypted by XOR with a keystream $\text{IsapEnc}(K_E^*, \text{nonce})$. The 128-bit nonce (Figure 5) concatenates a 16-bit domain id, the 48-bit physical line address, and a 64-bit split counter (32-bit major PMAJ || 32-bit minor LMIN). Address binding makes identical plaintexts at different lines encrypt differently; the version makes successive writes of the same line differ, which is also what defeats replay (Section 8).

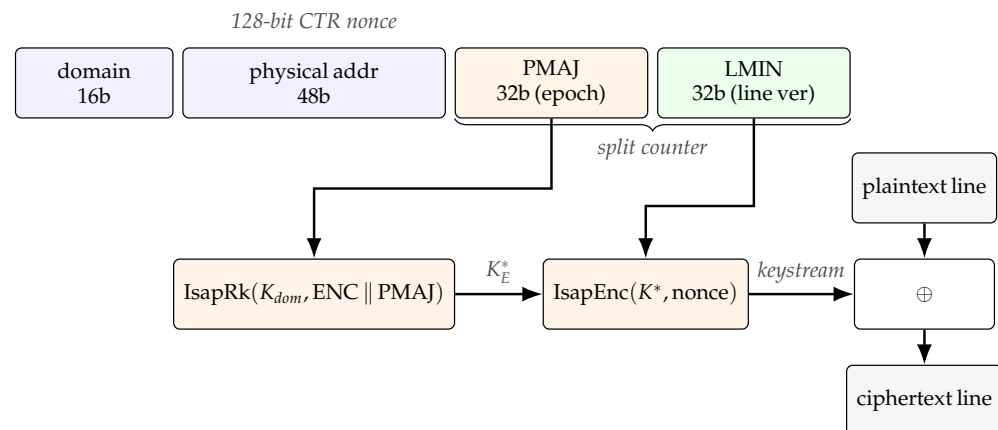


Figure 5. The unified split counter: PMAJ selects the re-keying epoch, LMIN is the per-line CTR nonce and integrity version. One counter serves all three roles.

Re-keying (IsapRk). The long-term domain key K_{dom} is never used to encrypt directly. A GGM-style re-keying sponge absorbs the selector bit-by-bit and emits a session key K^* (Figure 6, top). Two selectors are supported (Section 13): per-write (selector = full counter) gives a fresh key per line-write ($q \approx 1$); per-epoch (selector = PMAJ) amortizes re-keying over an epoch at the cost of $q = \text{lines-per-epoch}$. The mode is a control-plane bit. Re-keying is *live in the datapath* (encapsulated in ReKeyedIsapEnc/ ReKeyedIsapMac, which derive K^* then run the keystream/MAC); it is DOM-masked when $d > 0$ (the long-term key is processed only in the masked GGM sponge, the ephemeral K^* then drives the unmasked keystream). We validate the re-keyed engine from end to end (ciphertext, round trip, tamper detection) for $d = 0$ and $d = 1$, and (Masked) IsapRk/ReKeyedIsapEnc against the ISAP reference KATs. The bit-serial absorption is intentionally slow (this is the data-limiting that bounds DPA), so the re-keying latency is removed from the write path by the residency precompute (Section 9) and paid on the refill path (quantified in Section 16).

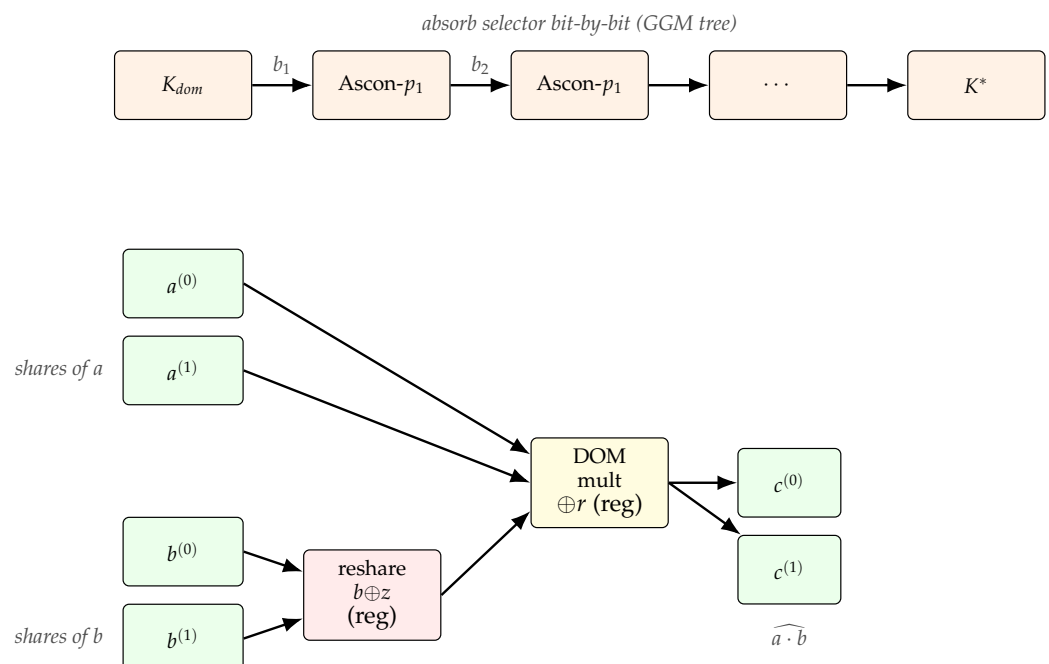


Figure 6. (Top) ISAP GGM re-keying absorbs the selector bit-by-bit. (Bottom) the DOM-dependent masked AND: resharing the second operand makes the two operands independently shared before the DOM multiply, which the affine-correlated Ascon inputs require.

Integrity (IsapMac). A keyed MAC over (address, version, ciphertext) authenticates each line; the tag is truncated and stored (on-chip in Tier-A, in DRAM in Tier-B).

Unified metadata. A single split counter is simultaneously the CTR nonce, the anti-replay version, and the re-keying epoch selector. This collapses what are usually three separate structures into one, halving metadata state and guaranteeing their consistency by construction.

Reference-validated. The Ascon- p , IsapEnc, IsapRk, and IsapMac RTL are checked bit for bit against a Python 3.11 reference that is itself validated against the published Ascon known-answer tests (KATs); the PACE write/read paths are validated against a golden model (Section Functional and Integrity Correctness).

7. Side-Channel Hardening: DOM Masking

Re-keying bounds the data per key, but the long-term key is still touched inside IsapRk; we mask that processing with DOM. Every sensitive bit is split into $d + 1$ shares; the only nonlinear element is the Ascon- p S-box, so it is the sole masking target (the affine pre-mix and linear diffusion are GF(2)-linear and applied share-wise).

Why DOM-dependent. Ascon's S-box ANDs operate on affine pre-mixed values that *share an input variable* ($a_1 = x_1, a_2 = x_2 \oplus x_1; a_4 = x_4 \oplus x_3, a_0 = x_0 \oplus x_4$). A DOM-independent multiplier assumes its two operands are independently shared, which is false here, so it leaks at first order, confirmed by COCO and PROLEAD (Section 14). We use the DOM-dependent gadget (Figure 6, bottom): the second operand is reshared with a fresh sharing-of-zero and registered, restoring operand independence before the DOM multiplication; the cross-domain partial products are then masked with fresh randomness and registered. The gadget has two register stages and is robust against glitches.

Full permutation. The 1-bit gadget is replicated across the 64-bit lane; the constant addition and linear diffusion are applied per share, giving a masked Ascon- p that we verify unmask to the plain permutation KAT for $d = 1, 2, 3$. Fresh randomness is supplied by an on-chip deterministic random bit generator (DRBG, a Trivium [18] keystream generator); the masking proof holds for any uniform, independent randomness source, and the DRBG quality affects only the empirical noise floor, not the probing model security.

Certified randomness frontend. For production deployment, the DRBG must be seeded from a true entropy source with continuous health monitoring. We implement the frontend in RTL (Trng): a raw 1-bit entropy stream (a ring oscillator sampler on FPGA; an external noise input pre-silicon) is checked by the two NIST SP 800-90B continuous health tests (the Repetition Count Test and the Adaptive Proportion Test), which latch a sticky alarm on a stuck or biased source; healthy bits are absorbed into an entropy pool that reseeds and re-conditions the DRBG output. In RTL tests, a balanced stream stays healthy and accrues reseed events (31 over 4000 samples) with high output diversity, a stuck-at source trips the RCT at the cutoff (cycle 32), and a 92%-biased source trips the APT. The health cutoffs are parameters set from the measured per-bit min-entropy on silicon. On the output quality side, the Trivium DRBG keystream passes the NIST SP 800-22 statistical battery indistinguishably from the system cryptographic RNG: over 10^6 bits the two agree on 13 of 14 tests (8/14 vs. 7/14 pass, the six both fail, being accumulator artifacts of the test harness on long sequences, not the generator). This supplies the certified randomness plumbing; the masking security itself remains independent of the source.

Live integration. The engine instantiates the masked keystream generator (Masked-IsapEnc, built from the masked permutation) in place of the plain IsapEnc whenever $d > 0$, behind a shared interface. We validate the *live* masked engine in the TileLink harness: with $d = 1$ the engine produces ciphertext in DRAM (all 8 words differ) and round trips correctly, i.e., the DOM-masked datapath works from end to end, not just as a standalone gadget.

Parameterized order. The masking order d is a build parameter, not a hardwired pair of cases: we evaluate $d \in \{1, 2, 3\}$. The $d = 3$ (four-share) masked permutation is emitted, synthesized, and verified to unmask to the plain Ascon- p KAT, confirming the gadget generalizes; its area is reported in Section 17 (the DOM cost grows quadratically in the share count, while $F_{\max}$ stays above 300 MHz through $d = 3$) and its formal verdicts in Section 14. The randomness cost is $d + \binom{d+1}{2}$ fresh bits per AND, per lane, per round.

8. Integrity: Tier-A and Tier-B

PACE authenticates each protected line with a MAC over (address, version, ciphertext) and verifies before release. Two organizations trade on-chip storage against memory traffic (Figure 7).

Tier-A (tree-free, embedded). The version counter and tag of every protected line live in trusted on-chip SRAM. Verification then touches *no* DRAM metadata: a refill is authenticated entirely from on-chip state, so there is no tree-walk and no metadata bandwidth. Spoofing fails the MAC; splicing fails because the MAC binds the address; replay fails because the on-chip version is the authority and a rolled-back ciphertext no longer matches. This is ideal for a bounded sensitive region, but the SRAM grows linearly with the protected footprint (16 B/line, Table 3).

Tier-B (scale-out). For large or all-RAM protection, the counters move to DRAM as split counter blocks (one 64-bit major plus 64×7 -bit minors per 4 KiB page, matching the PMAJ/LMIN fields), and each line additionally carries a DRAM-resident MAC. Integrity of the DRAM counters is anchored by a *block-MAC* Bonsai Merkle Tree of arity 8: each tree node's tag is a keyed Ascon-MAC over the *whole block* of its 8 children (level-0 children are the per-line counters; level- L children are the level- $(L-1)$ node tags). Only the top tag (the *rootTag*) stays on chip: the entire trust anchor is a single 8 B register. Because a node's tag covers its complete child block, writing one child re-tags only its ancestors and never stales the unchanged siblings: a property that a single per-node counter lacks. The engine is spliced on the mbus→memory edge and owns the metadata region privately: it issues counter/MAC/tree traffic on its own memory port under reserved source IDs, so metadata is never cached by the CPU and cold start needs no firmware: DRAM powers up zero and a line whose counter is still 0 takes a fast *fresh-line* path that skips the tree until its first encrypted write.

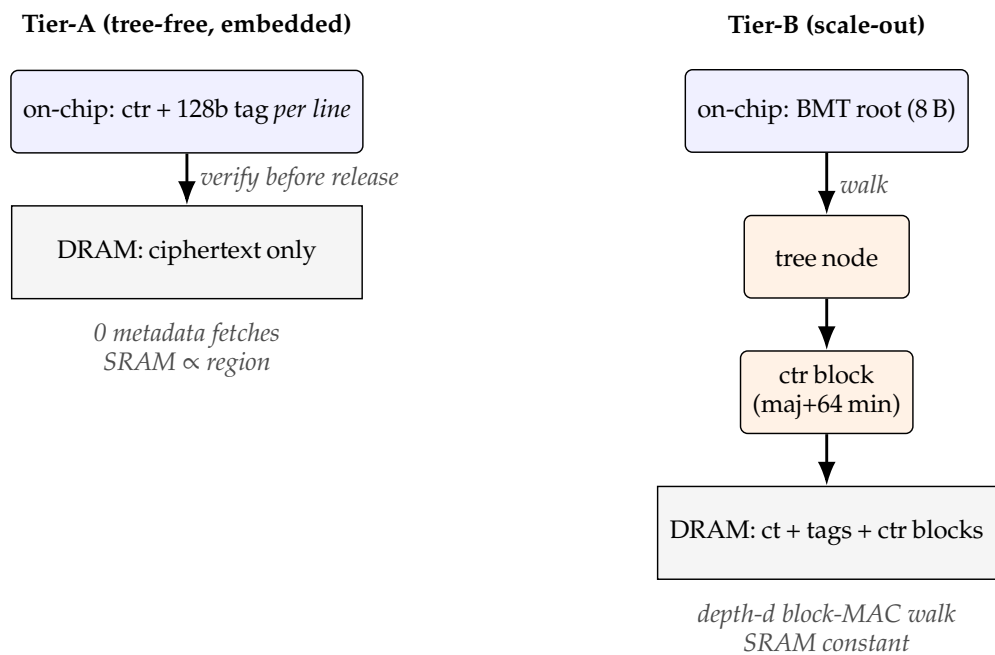


Figure 7. Integrity tiers: Tier-A trades on-chip SRAM for zero metadata traffic; Tier-B keeps on-chip state constant (a single root tag) by placing a split counter block-MAC Bonsai Merkle tree in DRAM, walked on every protected access.

Live RTL, end to end. The deployed engine (Section 5) is a hardware FSM; we validate it on the RTL against a backing memory, with no access-level model in the loop. (i) *Unit (chisel test)*: round trip through DRAM-resident counter + MAC + tree; a tampered ciphertext word, a tampered DRAM MAC, and a rolled-back self-consistent old {counter, MAC, ciphertext} triple (a replay) are *each* denied by the on-chip-rooted tree, while clean reads never are; sibling lines sharing a tree node stay verifiable after a neighbor is rewritten; and many-line and full depth-3 trees round trip with zero false faults. (ii) *Datapath (writeback cache + RAM)*: driving the engine from a writeback cache that generates eviction/refill traffic against a RAM, a 32-line working set over a depth-3 (256-line) tree round trips with $\text{errs} = 0$, $\text{faults} = 0$. (iii) *Full SoC*: on a complete Rocket SoC (L1 + L2, 256 MiB DRAM) the firmware writes an encrypted region, conflict-evicts every line (encrypted writebacks to DRAM), and reads it all back through the live verify-before-release path: $\text{err} = 0$, $\text{faults} = 0$. The same unmodified test passes on an out-of-order superscalar BOOM SoC as well: the engine is core-agnostic (it sits on the mbus→memory edge), with identical per-access metadata cost. The engine’s own counters report the metadata traffic, which matches the design arithmetic exactly (below).

Measured Tier-B traffic. A protected read fetches the line counter, the line MAC, and (because each node tag is recomputed from its full child block) one arity block per tree level, i.e., $2 + \sum_L \text{validChildren}(L)$ metadata beats, which for a depth- d arity- A tree is $2 + A(d-1) + r$ where $r \leq A$ is the root’s child count. On the full Rocket SoC, the 256-line region builds a depth-3 tree ($r = 4$), and the engine measured the closed-form cost as *exactly* $2 + 8 \cdot 2 + 4 = 22$ metadata beats per readback line and ≈ 26 per encrypted writeback in a bit-for-bit manner (Table 4), so there is no abstraction gap between the formula and the silicon. This block-MAC walk is heavier than a classic per-node Merkle walk (depth + 2): this is the price of sibling-independent writes with a single on-chip anchor and no metadata cache. Tier-A pays none of it.

Table 3. Integrity scaling: Tier-A on-chip SRAM grows 16 B/line with the region; Tier-B holds on-chip state at a single 8 B root tag, at the cost of a depth- d block-MAC walk and DRAM metadata.

Protected	Tier-A on-Chip SRAM	Tier-B on-Chip (Root + Key)	BMT Depth	rd md/Line	wr md/Line
256 KiB	64 KiB	16 B	4	34	39
1 MiB	256 KiB	16 B	5	38	44
16 MiB	4 MiB	16 B	6	50	57
256 MiB	64 MiB	16 B	8	60	69
1 GiB	256 MiB	16 B	8	66	75
4 GiB	1 GiB	16 B	9	70	80

Table 4. Tier-B metadata traffic (extra 64 B fetches per protected access), exact from the engine’s fetch FSM; the 256-line/depth-3 row is confirmed bit for bit on the full Rocket SoC (22 read beats/line).

Region	Lines	BMT Depth	Read md/Line	Write md/Line
256 KiB	4096	4	34	39
1 MiB	16,384	5	38	44
16 MiB	262,144	6	50	57
256 MiB	4,194,304	8	60	69
1 GiB	16,777,216	8	66	75
4 GiB	67,108,864	9	70	80

Tree arity design space. The arity trades node fan-out against tree depth. Unlike a cached per-node walk, the block-MAC tree reads a node’s *whole* block per level, so per-access read traffic $\approx 2 + A d$ is minimized at *small* arity: at 4 GiB, arity 4 costs 54 beats/access versus 70 (arity 8) and 102 (arity 16) (Table 5). PACE nonetheless defaults to arity 8 because the split counter block already holds 8 minors per node, so arity 8 needs no extra DRAM layout; arity 4 is available where read-side metadata bandwidth dominates, and a small metadata cache (future work) would amortize the upper levels back toward depth + 2.

Table 5. Tier-B block-MAC design space: BMT depth and read metadata beats per protected access vs. arity (*depth/read*), exact from the fetch FSM.

Region	Arity 4 Depth/rd	Arity 8 Depth/rd	Arity 16 Depth/rd
16 MiB	9/38	6/50	5/70
256 MiB	11/46	8/60	6/86
1 GiB	12/50	8/66	6/98
4 GiB	13/54	9/70	7/102

9. Residency Precompute and Eviction Shuffle

Residency precompute (C2). In CTR mode the keystream depends only on the address and the next version, not on the plaintext, so it can be computed *before* eviction. While a dirty line is resident, PACE precomputes the keystream for its next writeback (version +1) into a small per-line store; on a read that re-primers the line, the engine launches the precompute off the critical path. When the line is finally evicted, the write path finds the keystream ready and only XORs, skipping the IsapEnc stall. Figure 8 contrasts the two write path timelines: without precompute the keystream sits on the eviction critical path; with it, only the XOR remains. A TileLink-level microbenchmark (Section 16) measures the client-visible writeback latency dropping from 90 to 10 cycles (−88.9%) for a confidentiality-only line.

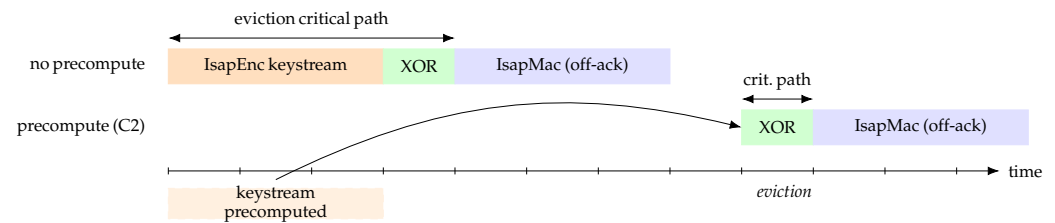


Figure 8. Write path timing. Top: without precompute, keystream generation is on the eviction-critical path. Bottom: residency precompute moves it off the path, so the eviction is just an XOR (the per-line MAC is post-AccessAck in both).

This is a latency win, not a throughput win: a bandwidth-bound stream is unaffected, but the eviction critical path is freed of cryptography.

Eviction shuffle (optional). As a complementary first-order defense, the order in which a line's beats are processed can be permuted by a key-dependent shuffle, so the time position of any given byte's keystream XOR varies per eviction and an averaging adversary cannot align traces by sample index. We implement this in RTL (*EvictShuffler*): an FSM processes the beats in a key+address dependent temporal order (an affine permutation $\sigma(t) = (at+b) \bmod W$ with a odd) while a full-line buffer keeps the stored beats in natural order. Because the keystream is indexed by beat rather than by time slot, the shuffle is correctness-transparent: in RTL tests, encrypt-then-decrypt round trips for all random lines/keys, the processing order is always a valid permutation, and it varies with the key (22 distinct orders over 40 trials). We treat the shuffle as an optional layer below masking and re-keying (the formally verified order- d guarantee does not rely on it) and report its area separately.

Interaction with integrity. Precomputing removes the keystream from the eviction path; the per-line MAC is computed after the AccessAck and so is already off the response path. The shared MAC unit can, however, serialize back-to-back evictions; a small counter/MAC pipeline (or per-region MAC units) removes this secondary bottleneck and is orthogonal to C2.

10. System Integration: Control Plane, OS, and Runtime

PACE is configured through a memory-mapped `pace_*` register file exposed as a TileLink slave on the periphery bus (Figure 3). Because policy is physical address indexed (Section 4), the OS (which owns the page tables) translates the pace PTE bits of sensitive pages into physical ranges and writes them to the engine. The register map (Table 6) carries the encrypted region (base/size), the policy bits (enable, integrity, decrypt-bypass for bring-up, re-keying mode), the encryption and authentication keys, a sticky LOCK bit that freezes the configuration after boot, and read-only STATUS/CAP registers reporting the fault counter and the synthesized masking order. Confidentiality and integrity are run-time policy; the masking order is a synthesis-time capability that the control plane *reports* rather than sets, since changing d changes the gate-level circuit. This section describes the full stack that drives that register file, from the bare-metal restore sequence, through the Linux kernel, to a glibc-layer encrypted allocator, and reports what we built and measured at each level.

Table 6. PACE control-plane registers (offset from `ctlBase`). Config registers are ignored once LOCK is set.

Offset	Register	Access
0 × 00	ENABLE	RW
0 × 08	INTEGRITY	RW
0 × 10	NODECRYPT (bring-up)	RW
0 × 18	REKEY_MODE (per-write/epoch)	RW
0 × 20	LOCK (sticky)	W1S
0 × 28/0 × 30	REGION_BASE/SIZE	RW
0 × 38–0 × 40	KEYE0/1 (enc key)	RW
0 × 48–0 × 50	KEYA0/1 (auth key)	RW
0 × 58	STATUS (fault count)	RO
0 × 60	LASTFAULT (addr)	RO
0 × 68	CAP (<i>d</i> , ctrW, #lines)	RO

10.1. Bare-Metal Programming Model

Firmware and bare-metal software drive the engine through `libpace`, a header-only interface whose `pace_protect()` writes the region, keys, and policy in one ordered burst of stores. It disables the engine while reconfiguring, sets BASE/SIZE and the two 128-bit keys, then raises INTEGRITY and ENABLE last, so no ciphertext is produced against a half-programmed configuration:

```
void pace_protect(u64 pa, u64 size, u32 policy,
                 u64 ke0, u64 ke1, u64 ka0, u64 ka1){
    pace_wr(PACE_ENABLE, 0);      /* quiesce while reconfig */
    pace_wr(PACE_BASE, pa); pace_wr(PACE_SIZE, size);
    pace_wr(PACE_KEYE0, ke0); pace_wr(PACE_KEYE1, ke1);
    pace_wr(PACE_KEYA0, ka0); pace_wr(PACE_KEYA1, ka1);
    pace_wr(PACE_INTEGRITY, policy & PACE_P_INTEGRITY);
    pace_wr(PACE_ENABLE, policy & PACE_P_CONF);
}
```

The compiler lowers this to a straight run of `sd` stores against the control base `ctlBase` (0x1001_0000), one per register. The following is the actual objdump of `pace_protect` (then `pace_lock`) built with the RISC-V bare-metal GCC at `-O2`, annotated with the target register:

```
lui  a6,0x10010      # a6 = ctlBase (0x1001_0000)
sd   zero,0(a6)      # ENABLE  <- 0 (quiesce)
sd   a1,40(a6)       # BASE    <- 0x8400_0000
sd   a7,48(a6)       # SIZE    <- 0x0004_0000 (256 KiB)
sd   a5,56(a6)       # KEYE0 ; sd a2,64(a6) KEYE1
sd   a4,72(a6)       # KEYA0 ; sd a5,80(a6) KEYA1
sd   a4,8(a6)        # INTEGRITY<- 1
sd   a4,0(a6)        # ENABLE  <- 1 (engage)
sd   a4,32(a6)       # LOCK    <- 1 (freeze config)
```

There is no magic in the control path: eleven-word stores program the engine and lock it. The full flow (Figure 9) is validated on the *complete* Rocket SoC (PaceCtlSoCConfig, the register node on the periphery bus): a bare-metal program reads CAP=0x10_001000 (4096 lines, ctrW = 16), runs the sequence above, writes a 256 KiB working set, and confirms via the NODECRYPT probe that evicted lines are ciphertext in DRAM and that every line decrypt-round trips with zero errors, exercising the core→pbus→pace_* path end to end. A TileLink-level test independently confirms the same mechanism (readback of BASE=0x8400_0000, SIZE=0x1000, CAP encoding *d* = 1; all eight line words become ciphertext and read returns plaintext).

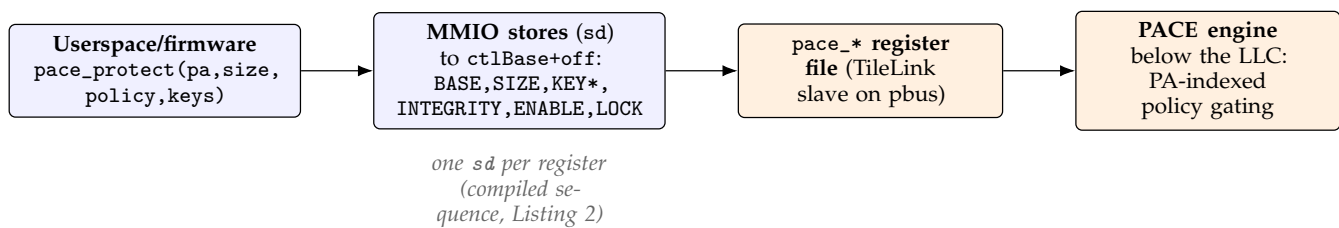


Figure 9. Control-plane programming path. Software forms a region/key/policy descriptor; libpace lowers it to one sd per register against ctlBase; the engine, below the LLC, then gates encryption on the PA-indexed region.

10.2. Linux Kernel Integration

Because policy is physical address indexed and userspace speaks virtual addresses, the kernel, which owns the page tables, is the natural place to bind a protected range. We implement this as an out-of-tree driver (`/dev/pace`) that takes a *virtual* range and derives the backing *physical* ranges. The core routine pins the pages, reads their frame numbers, coalesces contiguous frames, and programs the PA-indexed region table:

```

got = get_user_pages_fast(start, npages, FOLL_WRITE, pages);
for (i = 0; i < got; i++) pfns[i] = page_to_pfn(pages[i]);
sort(pfns, got, sizeof(*pfns), cmp_pfn, NULL);
for (i=0; i<got; i++) { /* coalesce contiguous PFNs */
    u64 phys = (u64)pfns[i] << PAGE_SHIFT;
    if (nr && R[nr-1].phys + R[nr-1].bytes == phys)
        R[nr-1].bytes += PAGE_SIZE;
    else { R[nr].phys=phys; R[nr].bytes=PAGE_SIZE;
          R[nr++].policy=policy; }
}
for (i=0; i<nr; i++) { /* program engine (PA-indexed) */
    writeq(R[i].phys, ctl+PACE_BASE);
    writeq(R[i].bytes, ctl+PACE_SIZE);
    writeq(policy?1:0, ctl+PACE_ENABLE);
}

```

We validate this end to end on Linux 6.12 under QEMU (riscv64 virt): the module loads, a userspace program `mmaps` a 512 KiB anonymous region, faults it in, and calls `PACE_PROTECT`. The driver returns the derived physical ranges, five discontinuous ranges (the anonymous pages are physically fragmented) summing to *exactly* 524,288 bytes, all page-aligned:

```

pace: control-plane driver loaded (PA derivation)
pace: protect va=0x7fff8a5be000 len=0x80000 pol=3 -> 5 range(s)
[pace-selftest] len=524288 -> 5 range(s), total=524288, aligned=1
[pace-selftest] PASS (coverage exact, alignment ok)

```

QEMU has no PACE engine, so this validates the novel part, the syscall and virtual-to-physical policy-derivation plumbing; in-hardware engagement is the SoC result of Section 10.1. The `/dev/pace ioctl` is the explicit, already-built entry point; the same PA derivation primitive is the hook that the mm subsystem calls at the sites listed in Table 7 and drawn in Figure 10: a `MAP_ENCRYPTED` mmap flag tags a VMA in `mm/mmap.c`; the fault handler in `mm/memory.c` binds a frame when it is first populated; and page movement is handled at swap, copy-on-write, and migration.

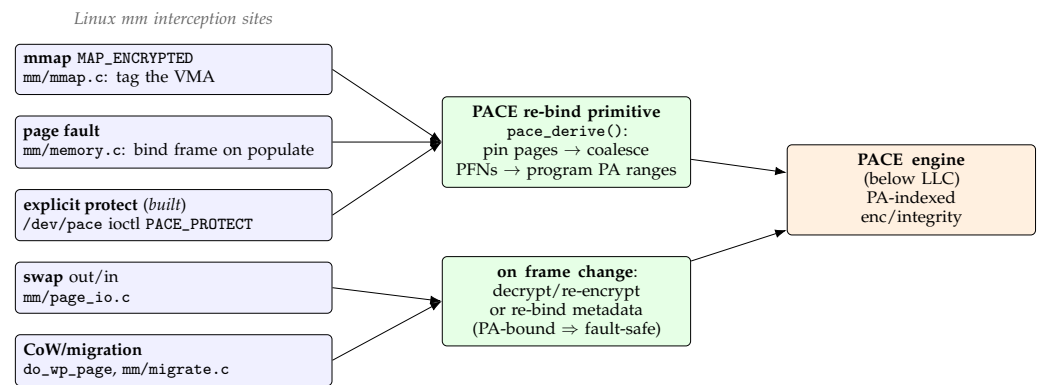


Figure 10. Linux mm interception sites for PACE. The built `/dev/pace` ioctl and the mm hooks share one primitive, `pace_derive()` (pin $\rightarrow$ coalesce PFNs $\rightarrow$ program PA ranges). Because the nonce and MAC are PA-bound, a frame change (swap/CoW/migration) that is not re-bound faults safely rather than corrupting.

Table 7. Linux integration sites. The ioctl path is built and validated on QEMU; the mm hooks share its PA derivation primitive, and the frame-change protocols are validated against the reference engine (Section 10.2).

Site	Kernel Location	Action
Explicit protect	<code>/dev/pace</code> ioctl	pin, derive+coalesce PA, program tag VMA; defer to fault
Transparent map	<code>mm/mmap.c</code> (MAP_ENCRYPTED)	bind frame PA on populate
First touch	<code>mm/memory.c</code> (fault)	decrypt out; re-bind PA in
Swap out/in	<code>mm/page_io.c</code>	re-encrypt at child frame
Copy-on-write	<code>mm/memory.c</code> (do_wp_page)	re-bind at destination frame
Migration	<code>mm/migrate.c</code>	bounce or exclude range
DMA	<code>dma-direct/IOMMU</code>	

Page movement is fault-safe by construction. Because the keystream nonce and the MAC are both physical address bound, a frame that moves without being re-bound does not decrypt to attacker-chosen plaintext, it *faults*: the PA in the recomputed MAC no longer matches. We verify the five movement protocols against the reference engine and confirm this directly: swap-out/in to the same frame round trips; the same blob restored to a *different* frame is caught by the PA-bound MAC (re-binding required); decrypt and re-encrypt at the new frame round trips for swap, copy-on-write, and migration; and a copy-on-write child that naively shares the parent ciphertext faults, whereas re-encryption at the child frame yields *independent* ciphertext. The safety property, naive relocation faults rather than corrupts, is what makes incremental kernel hardening tractable: an unhooked path fails closed.

10.3. Glibc-Layer: An Encrypted Allocator and Whole-Process Encryption

On top of the kernel path we build the runtime layer a program actually uses: a crypto-allocator that serves `malloc/calloc/realloc/free` from PACE-protected pages, and a whole-process facility that encrypts every writable anonymous mapping. The allocator carves its arena from an `mmap` region and protects it once through `/dev/pace`; every subsequent allocation is served from encrypted memory:

```

static void arena_init(void){
    arena = mmap(0, ARENA, PROT_READ|PROT_WRITE,
        MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
    for (size_t i=0; i<ARENA; i+=4096) arena[i]=0;    /* fault in */
    pace_protect_region(arena, ARENA, CONF|INTEG);    /* encrypt */
}

void *secure_malloc(size_t n){ ... return arena + bump; }

```

```
/* calloc=malloc+memset; realloc=malloc+memcpy; free->reclaim */
```

Whole-process encryption walks `/proc/self/maps` and protects each writable anonymous region (heap, stack, BSS, anonymous mmapped), and a `pthread_atfork` child handler re-binds after a fork, since the copy-on-write child receives fresh frames at new physical addresses:

```
while (fgets(line, sizeof line, maps)) {
    sscanf(line, "%lx-%lx %7s", &a, &b, perms);
    if (perms[1] != 'w') continue; /* writable only */
    pace_protect_region((void*)a, b-a, CONF|INTEG);
}
pthread_atfork(0, 0, pace_protect_all); /* re-bind child */
```

We build the runtime static against `musl` and run it under the same Linux 6.12 QEMU image. The allocator protects a 2 MiB arena, serves the allocation calls from it, encrypts the whole process, and re-binds across a fork:

```
[secure-alloc] arena 2048 KiB -> 3 encrypted phys range(s)
[secure-alloc] malloc/calloc/realloc served from encrypted arena
[secure-alloc] whole-process: 4 writable region(s) -> 10 ranges
[secure-alloc] fork child re-bound its frames: 5 region(s)
[secure-alloc] PASS
```

In mainline glibc, the same three hooks land in known places, listed in Table 8: the arena allocator (`malloc/arena.c`, `new_heap/sysmalloc`) protects each heap as it is mapped; the fork path (`sysdeps/.../fork.c`, `__libc_fork` via `__register_atfork`) re-binds the child; and process start (`elf/rtld.c/csu/libc-start.c`) protects the initial image, which is also the entry point after `execve` replaces the address space. Deploying as an `LD_PRELOAD` interposer needs no glibc rebuild and gives the same behavior we measured above.

Table 8. glibc-layer integration points for transparent process encryption. The allocator, whole-process, and fork re-bind mechanisms are built and validated on QEMU (Section 10.3); the column gives where each lands in mainline glibc.

Mechanism	Glibc Location	Change
Encrypted heap	<code>malloc/arena.c</code> (<code>new_heap</code>)	protect each heap on mmap
malloc family	<code>malloc/malloc.c</code>	serve from protected arenas
Whole process	<code>csu/libc-start.c</code>	protect writable anon maps at start
fork	<code>sysdeps/.../fork.c</code>	<code>atfork</code> child re-binds frames
<code>execve</code>	<code>elf/rtld.c</code> (loader)	protect new image on entry

Together these three levels, eleven stores from bare metal, a ~40-line kernel primitive reused across the mm subsystem, and a runtime allocator that turns `malloc` into encrypted memory, make PACE usable from firmware up to an unmodified application, with fault-safe page movement throughout.

11. Implementation, Integration Challenges, and Reproducibility

This section documents *how* PACE is built and, in particular, the concrete integration challenges we met and resolved. We include these both because they are the practical substance of the work and because they are what a reader needs to reproduce it. A machine-checkable artifact (sources, configs, and scripts) accompanies the paper.

11.1. Design and Verification Flow

PACE is written in Chisel and elaborated through the standard open RISC-V flow. Chisel emits FIRRTL [19], which `firtool` (CIRCT) lowers to synthesizable Verilog. That single RTL source then feeds four backends, so that every result in the paper traces to

the *same* design: (i) Verilator (5.032) for cycle-exact functional, integrity, and performance simulation within Chipyard; (ii) Yosys (0.52) [20] for gate-level netlists consumed by the masking verifiers; (iii) Xilinx Vivado (2020.1) for out-of-context synthesis, place-and-route, power, and FPGA bitstreams; and (iv) a gem5 memory latency model, parameterized by RTL- and silicon-measured values for application-level studies at scale (Section 16). Keeping a single elaborated source behind all four backends is what lets us cross-check, e.g., that the S-box the formal tools verify is the S-box that is synthesized and programmed onto the FPGA.

11.2. Integrating Three Formal Masking Tools

The three verifiers accept different inputs and models, so integrating them was the main verification flow effort. COCO [21] consumes a gate-level netlist *co-designed* with its control logic and checks robust (transient) probing security via SAT; we label the key shares as *secret*, the DOM refresh masks as *volatile_random*, and the datapath as *unimportant*, and invoke it per order (q) in transient mode. PROLEAD [22] evaluates probing security *statistically*: we lower the RTL through Yosys with PROLEAD's custom-cell library to a gate netlist, map each shared input $\text{group_in}\langle j \rangle[i]$ to share j of bit-secret i , tie the refresh inputs to *always-random* and *reset low*, and run `-p hardware` with transitional leakage enabled over 6.29×10^6 simulations across all clock cycles. SILVER [23] decides the exact composability lattice on a binary decision diagram (BDD) representation.

The SILVER netlist challenge, and its resolution.

SILVER did not accept our *firtool* netlist out of the box. *firtool* emits each module port as *both* an input/output declaration *and* a redundant same-name wire; SILVER's Verilog frontend then instantiated two signals per port and reported the annotated port bits as *unconnected*, so no lattice could be decided. We resolved this by emitting the gadget as a *single packed-bus module* (each share lane becomes one bus bit, matching SILVER's `[hi:lo]_share` annotation directly, with no instantiation wrapper to alias) and by stripping the duplicate port wires before annotation. The pruned netlist then elaborates and is decided exactly (the intermediate netlists of this transform are part of the artifact). We document this integration step because it is needed to reproduce the three-tool result.

Break-then-fix as a methodology check.

Before verifying the final gadget, we first verified the *naive* DOM-independent masking of Ascon's S-box, and COCO and PROLEAD both reported a first-order glitch leak (the affine pre-mix makes the two AND operands share an input variable). Reproducing a *known* flaw confirms the flow is sensitive, not merely permissive; the DOM-dependent construction (Section 7) then verifies secure. We report the negative result alongside the positive one.

11.3. FPGA Realization: What Was Hard, and How We Solved It

Bringing PACE onto a low-cost Zynq-7000 (XC7Z020, 53,200 LUTs) surfaced two concrete issues.

Area fit. The masked permutation grows with order (Section 17); the full validation design (engine plus a writeback cache, on-fabric RAM, and a self-checking test driver) fits comfortably, but the higher masking orders are sized against the part budget by design, and we report the utilization at each order rather than a single number.

Timing closure. At a 100 MHz target, the full self-test design misses timing (worst negative slack -2.929 ns). The critical path is *not* in PACE: it is the test driver's `mix()` test vector multiplier feeding the self-check comparator together with the distributed LUTRAM routing of the on-fabric test RAM, both benchmarking harness artifacts. The

PACE datapath itself closes with margin: the keystream engine-plus-ring oscillator sensor design places and routes at 100 MHz with +1.847 ns slack. We therefore report both: the sensor design at 100 MHz for the leakage assessment campaign, and the on-board self-test run on a/4-divided 25 MHz clock (comfortably inside the routed $F_{\max}$), programmed over the board's FT232H JTAG link with openFPGALoader. Separating the harness's timing from the engine's is the honest way to report an FPGA result, and it is why the on-silicon numbers are trustworthy.

On-silicon bring-up, and what it took.

Getting the *full* Tier-B validation design (engine + writeback cache + on-fabric RAM + self-checking driver) to complete on hardware surfaced a chain of concrete issues that we record because they are exactly the substance of an on-silicon result. In order: the burst path into the simple on-fabric RAM deadlocked mid-transfer and was fixed with a TLFragmenter so the RAM sees only single-beat accesses; the RAM was moved to a synchronous read TLRAM (block RAM) to eliminate a marginal distributed-RAM read path; the harness clock was buffered through a global BUFG and its reset synchronized; and the reset was moved off the noisy board slide-switch to an *internal power-on reset*, without which switch bounce silently restarted the self-test. To read the result we route the driver's done/pass/error status and a cycle counter to a JTAG BSCANE2 register and shift it out with OpenOCD (the board's cable does not enumerate under the vendor `hw_server` in our container, so a boundary scan path is the robust readback). With these in place the self-test passes bit-identically across programmings. The self-check harness (not the engine) remains timing-marginal on this low-cost part (some place-and-route seeds need re-spinning to close cleanly), so we ship the passing bitstream and the exact flow in the artifact. None of this touches the engine, which is proven correct in cycle-accurate simulation and confirmed by the passing silicon run.

11.4. Linux Control-Plane Driver, Validated

Because the engine sits below the last-level cache and cannot read inner TileLink user fields, the per-page policy is programmed out-of-band through a memory-mapped control plane (a TLRegisterNode bank of `pace_*` registers; the full map is in Appendix E). Software writes the protected physical range, the keys, and the enable/policy bits; encryption then engages at run time on exactly those pages. We provide a firmware interface (`libpace.h`) and a driver (`os/pace-driver`). The control path is validated from end to end in the TileLink harness by `PaceCtlTest`, which *passes*: it programs the region, reads back the BASE/SIZE/CAP registers, confirms all eight line words in memory differ from plaintext (ciphertext), and round trips the data correctly, demonstrating that a running system programs and engages PACE as intended, not merely that the datapath works in isolation. The same engine is exercised on a dual-core shared configuration to confirm the control plane and datapath are core count-agnostic.

11.5. Reproduction Recipe

The artifact bundles the Chisel source, the Chipyard configurations (`Pace*Config`), the formal harnesses (with the SILVER netlist-transform scripts), the Vivado synthesis/implementation TCL, the FPGA constraints and bitstream flow, the driver and firmware interface, and the evaluation scripts. Tool versions are pinned (firtool/CIRCT, Yosys 0.52, Verilator 5.032, Vivado 2020.1, gem5 25.1, and the three formal tools); every table and figure is regenerated from the committed scripts and logs. The intent is that a reader can rebuild any single result (a KAT, a formal verdict, an area/timing report, a performance number, or the FPGA bitstream) from one command against the pinned environment.

12. Evaluation Methodology

We evaluate PACE entirely pre-silicon with open-source tooling in five layers (Figure 11):

1. **Functional/integrity correctness** via a deterministic TileLink-level chisel test harness with a peekable backing RAM (Section Functional and Integrity Correctness).
2. **Formal masking** on the synthesized gate netlist with COCO, SILVER, and PROLEAD (Section 14).
3. **Simulated trace DPA** (CPA, TVLA) plus the analytic bounds of Section 13 (Section 15).
4. **Performance** via bare-metal microbenchmarks on the Verilator SoC across access patterns, policies, masking orders, and re-keying intervals (Section 16).
5. **Area/frequency** via Vivado out-of-context synthesis for the Zynq XC7Z020 (Section 17).

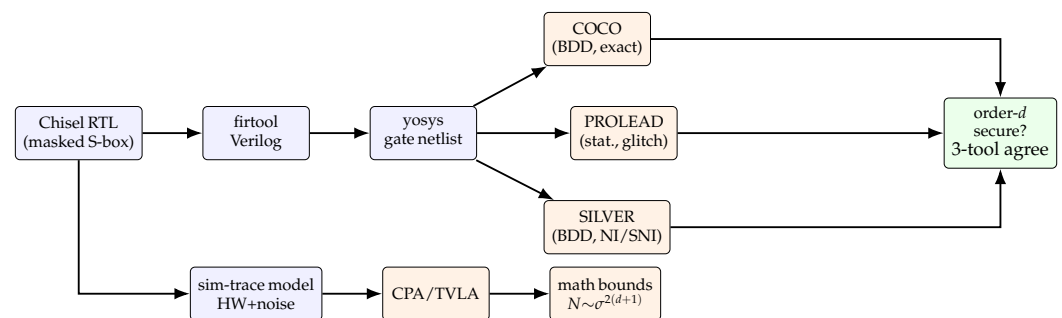


Figure 11. The no-lab DPA methodology: one gate netlist feeds three independent formal tools; the same RTL feeds a simulated trace CPA/TVLA model and the analytic trace amplification bounds.

Table 9 summarizes every component, the method by which it is validated, and the outcome.

Table 9. Validation summary: every component, the method, and the outcome (all pre-silicon and open-source).

Component	Method	Outcome
Ascon- <i>p</i> , IsapEnc/Rk/Mac	KATs vs. validated reference	bit-exact
PACE write/read path	TileLink harness (peekable RAM)	ciphertext ≠ plaintext; round trip; tamper/splice/replay → denied
M3 control plane	TileLink harness, MMIO client	program region/keys/enable → encryption engages at run time
Masked Ascon S-box	COCO + PROLEAD (glitch/transition)	naïve leaks; DOM-dep order- <i>d</i> secure (<i>d</i> = 1, 2; COCO to <i>d</i> = 3)
Masked permutation/keystream	unmask vs. plain KAT	permutation <i>d</i> = 1, 2, 3, keystream <i>d</i> = 1, 2
Live masked engine	TileLink harness (<i>d</i> = 1)	ciphertext + round trip with DOM-masked keystream
ISAP re-keying (IsapRk, MaskedIsapRk)	KAT vs. reference (<i>d</i> = 1, 2)	K^* bit-exact (unmasked)/unmasks to KAT (masked)
Re-keyed keystream (ReKeyedIsapEnc)	KAT vs. reference (<i>d</i> = 0, 1)	IsapEnc(IsapRk($K, \cdot$), $\cdot$) bit-exact
Live re-keyed engine	TileLink harness (<i>d</i> = 0, 1)	ciphertext + round trip + tamper, per-write K^*
DPA (distinguisher)	simulated trace CPA/TVLA	128/128 key recovered unprotected; masked defeats; $ t $ flat
DPA (analytic)	mutual information sweep	$N \propto \sigma^{2(d+1)}$, slopes 1.92/3.70/5.40
Area/ $F_{\max}$	Vivado out-of-context (OOC), XC7Z020	fits XC7Z020; $F_{\max}$ preserved (~305–335 MHz)
Performance	bare-metal on Verilator SoC	≤ 1% (cache-resident) to ~2.5× (eviction-bound)
Integrity (Tier-B)	live engine: Rocket/BOOM SoC + FPGA; exact fetch arithmetic	metadata cost matches silicon bit-for-bit (22 beats/line, depth-3)

Functional and Integrity Correctness

A round trip through the full Rocket SoC does *not* by itself prove encryption: with a 32 KiB L1 and working sets that fit it, lines are never evicted and the engine stays idle. We therefore drive the engine directly with a TileLink-level harness: a chisel test client through PaceEngine into a debug RAM whose contents we peek and poke. This proves, deterministically: ciphertext-in-memory (all eight line words differ from plaintext), correct round trip, and detection of *tamper* (flip a ciphertext bit $\rightarrow$ denied), *splice* (relocate a valid block $\rightarrow$ denied), and *replay* (roll back to an older valid version $\rightarrow$ denied). All crypto cores are additionally checked against reference KATs. The control plane is validated by a companion harness that programs the engine over MMIO and observes encryption engage at run time (Section 10). The full suite runs as a regression on every change.

13. Analytic Side-Channel Security

We now place PACE's masking and re-keying on a quantitative footing: how many side-channel measurements (traces) an adversary needs to recover key material, and how PACE drives that number out of reach. Throughout, $Z = f_k(x)$ is an intermediate that depends on a key chunk k and a known input x (for PACE, the first-round Ascon S-box output of a key column), and the adversary observes a leakage sample L .

13.1. Leakage Model and SNR

We adopt the standard noisy Hamming weight (HW) model

$$L = \delta \text{HW}(Z) + \mathcal{N}, \quad \mathcal{N} \sim \mathcal{N}(0, \sigma^2), \quad (2)$$

with δ a device gain and $\mathcal{N}$ independent Gaussian noise. The signal-to-noise ratio (SNR) is

$$\text{SNR} = \frac{\delta^2 \text{Var}[\text{HW}(Z)]}{\sigma^2}. \quad (3)$$

A simulated trace in our harness instantiates (2) exactly (the RTL is bit-accurate against the Ascon reference, Section Functional and Integrity Correctness); noise is the defender-conservative term, since adding σ only *reduces* the adversary's correlation.

13.2. Correlation Power Analysis and Traces to Disclosure

The CPA distinguisher ranks key hypotheses k' by the Pearson correlation $\rho(k') = \text{corr}(L, \text{HW}(f_{k'}(x)))$; the correct key maximizes ρ . Writing $\rho := \rho(k)$ for the correct hypothesis, the model and (2)–(3) give

$$\rho = \frac{\rho_{\text{ck}}}{\sqrt{1 + \text{SNR}^{-1}}} \xrightarrow{\text{high noise}} \rho_{\text{ck}} \sqrt{\text{SNR}}, \quad (4)$$

where $\rho_{\text{ck}} \leq 1$ is the (noise-free) model correlation. The number of traces to distinguish ρ from 0 at confidence (α, β) follows from the Fisher z-transform [5]:

$$N = 3 + \frac{8(z_{1-\alpha} + z_{1-\beta})^2}{\ln^2 \frac{1+\rho}{1-\rho}} \approx \frac{c_{\alpha,\beta}}{\rho^2} \quad (\rho \ll 1), \quad (5)$$

i.e., *traces-to-disclosure scales as* ρ^{-2} . Combining with (4), an *unprotected* implementation needs $N_0 \propto \text{SNR}^{-1} \propto \sigma^2$.

13.3. Masking Amplification

Boolean masking splits Z into $d+1$ shares $Z = \bigoplus_{i=0}^d Z_i$ with $Z_1, \dots, Z_d$ uniform, each leaking $L_i = \delta \text{HW}(Z_i) + \mathcal{N}_i$ independently. No tuple of $\leq d$ shares depends on Z , so a $(d+1)$ -th order attack must combine all $d+1$ samples (e.g., the centered-product $\prod_i (L_i - \bar{L}_i)$). The mutual information (MI) of the optimal combination obeys the Prouff–Rivain/Duc–Faust–Standaert bound [24,25]

$$\text{MI}(Z; L_0, \dots, L_d) = O(\text{SNR}^{d+1}) = O(\sigma^{-2(d+1)}), \quad (6)$$

hence the higher-order correlation $\rho_d \propto \text{SNR}^{(d+1)/2}$ and, by (5),

$$N_d \propto \rho_d^{-2} \propto \text{SNR}^{-(d+1)} \propto \sigma^{2(d+1)}. \quad (7)$$

On a log–log plot of N_d versus σ the slope is therefore $2(d+1)$: 2, 4, 6 for $d = 0, 1, 2$. Each masking order multiplies the trace cost by a factor $\Theta(\text{SNR}^{-1})$, which for any realistic noisy device is several orders of magnitude.

We verify (7) directly by Monte Carlo estimation of the mutual information $\text{MI}(b; L_0, \dots, L_d)$ of a $(d+1)$ -share masked bit over a noise sweep and fitting the asymptotic ($\sigma \geq 1$) slope of $N \propto 1/\text{MI}$ against σ (Figure 12, Table 10). Estimated to $d = 3$ over ten Monte Carlo seeds, the measured slopes (1.92/3.70/5.40/6.97) track the predicted $2(d+1)$ with tight confidence intervals and a near-constant increment of ≈ 1.7 per masking order, the small shortfall being the finite- σ pre-asymptotic regime (the slope approaches $2(d+1)$ as $\sigma \rightarrow \infty$). The result is also *model-independent*: a complementary higher-order CPA that combines the $d+1$ share leakages by their centered product recovers the same increasing-with- d exponent under three leakage models (Hamming weight, Hamming distance, and a single-bit model), so the amplification is a property of the masking, not of a particular leakage assumption.

Table 10. Measured vs. predicted trace amplification exponent ($N \propto \sigma^{\text{slope}}$) from the exact likelihood MI sweep, extended to $d = 3$ and reported as mean $\pm$ 95% CI over 10 Monte Carlo seeds. Higher d uses a lower top- σ band where the (astronomically small) MI is still resolvable; the slope approaches $2(d+1)$ from below in this pre-asymptotic regime.

Masking Order d	Shares $d+1$	Predicted $2(d+1)$	Measured Slope (95% CI)
0 (unprotected)	1	2	1.92 ± 0.01
1	2	4	3.70 ± 0.03
2	3	6	5.40 ± 0.06
3	4	8	6.97 ± 0.12

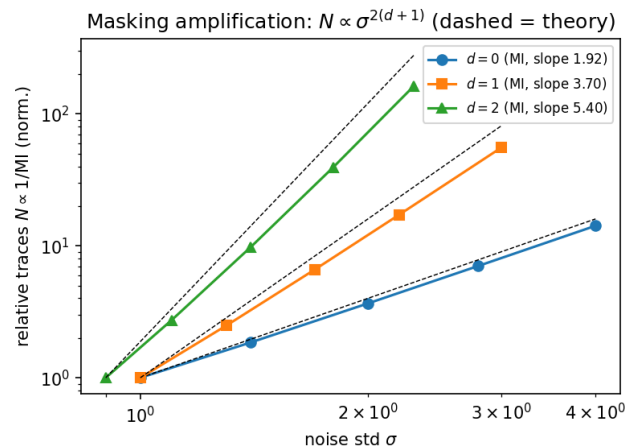


Figure 12. Masking amplification: relative traces to disclosure $N \propto 1/\text{MI}$ vs. noise σ for $d = 0, 1, 2$ (markers, MI Monte Carlo) against the analytic $\sigma^{2(d+1)}$ law (dashed). Each masking order steepens the slope by two.

13.4. Re-Keying: Starving the Distinguisher

Equations (5)–(7) all require N_d traces *under one key*. PACE’s per-write re-keying derives a fresh session key $K^* = \text{IsapRk}(K_{\text{dom}}, \cdot)$ for (PA, version): each K^* encrypts exactly one line once, so the data complexity available to the adversary is

$$q = 1 \ll N_d. \quad (8)$$

A CTR/MAC keystream attack on K^* is thus information-theoretically starved regardless of d : there are not enough equal-key traces to average. The only key reused across writes is the long-term K_{dom} , which is touched solely inside the GGM re-keying sponge (the masked block, order d) and absorbs only the *public* selector with bounded data complexity by ISAP’s design. The per-epoch mode (selector = PMAJ) relaxes (8) to $q = \text{lines per epoch}$ and is offered as a throughput/security knob; Section 15 quantifies the two regimes.

Proposition 1 (Re-keying starves CTR key DPA). *Let CPA recovery of a session key require N_d equal-key traces (Equation (7)). Under per-write re-keying each session key K^* encrypts a single line once, so the adversary obtains $q = 1$ trace per key. Since $N_d \gg 1$ for any noisy device, no session key is attackable by averaging, independent of the masking order d . The only multiply key is K_{dom} , confined to the order- d masked GGM sponge and absorbing only public selector bits with bounded data complexity; combining masking (raise order) with re-keying (cap q) removes both the averaging and the high-order attack paths simultaneously.*

13.5. Detection Statistics and Multiple Comparisons

For leakage *detection* we use non-specific TVLA: a Welch t -test between a fixed input and a random input group. Under the no-leakage null $t \sim \mathcal{N}(0, 1)$, so the conventional threshold $|t| > 4.5$ has per-point false positive probability $2(1 - \Phi(4.5)) \approx 6.8 \times 10^{-6}$. When a tool evaluates P probing sets (PROLEAD enumerates $P = 132612$ sets at the $d = 2$ second-order check), the family-wise threshold for false positive rate α is $p < \alpha/P$, i.e., a corrected $-\log_{10} p$ threshold of $\log_{10}(P/\alpha)$. This is the lens through which the exact (COCO, SILVER) and statistical (PROLEAD) verdicts in Section 14 must be read.

14. Formal Masking Verification

The masked Ascon- p S-box is the only nonlinear gadget in the data path and thus the sole target of the masking proof; the affine pre-mix and linear diffusion are GF(2)-linear

and trivially share-wise secure. We verify the gadget with three independent tools spanning two methodologies:

- **COCO** [21]: exact, BDD-based probing security analysis on the synthesized gate netlist under a robust (glitch + transition) model;
- **PROLEAD** [22]: statistical, glitch-aware probing security evaluation by massive simulation (6.3×10^6 simulations);
- **SILVER** [23]: exact, reduced-ordered BDD analysis that decides not only probing security but the full lattice of composability notions (NI, SNI, and probe-isolating non-interference, PINI) and output uniformity, in both the standard and the glitch/transition-robust model.

All three consume gate-level netlists of the *same* firtool-generated RTL mapped to a standard cell library (NANG45-style cells for SILVER, the COCO/PROLEAD custom cells for the other two), so they verify the artifact that is synthesized, not an abstraction of it. Using two distinct synthesis frontends and three checkers makes a single tool-specific artifact (a cell model, a BDD variable order, a simulation seed) unable to manufacture a false “secure” verdict.

Reproducing the netlist for SILVER. Feeding our gadget to SILVER required a netlist transform, since firtool emits redundant same-name port wires that SILVER’s frontend reads as unconnected; we emit the gadget as a single packed-bus module and strip the duplicate wires before annotation. Because this is an integration detail rather than a verification result, we document the exact fix in Section 11.2.

14.1. Break-Then-Fix

A naïve DOM-independent S-box leaks at the first order: Ascon’s affine pre-mix correlates the two AND operands ($a_1 = x_1$, $a_2 = x_2 \oplus x_1$; $a_4 = x_4 \oplus x_3$, $a_0 = x_0 \oplus x_4$), violating the DOM-independence precondition. The fix is a DOM-dependent gadget that reshapes the second operand with a fresh sharing of zero and registers it, restoring operand independence before the DOM multiplication (Section 7, Figure 4). Each tool catches the leak in the naïve design and certifies the fixed design (Table 11). The certification extends to the highest order we synthesize: at $d = 3$ (four shares) COCO exactly decides the four-share DOM S-box *secure* against transient, glitch-aware probing at orders $q = 1, 2, 3$, matching the $d = 1, 2$ verdicts and confirming that the construction scales to third-order protection without a security regression.

Table 11. Formal verification of the masked Ascon S-box under a glitch/transition-aware probing model by three independent tools. “secure” = no leaking probing set at the stated order; PROLEAD set counts in parentheses; SILVER reports the robust model probing verdict.

Design	d	q	COCO	PROLEAD	SILVER
naïve DOM-indep	1	1	leak	leak (36/297)	n/a
DOM-dependent	1	1	secure	secure (0/528)	secure
DOM-dependent	1	2	leak *	leak	leak *
DOM-dependent	2	1	secure	secure (0/1032)	secure §
DOM-dependent	2	2	secure	secure †	secure §
DOM-dependent	3	1	secure	— ¶	— ¶
DOM-dependent	3	2	secure	— ¶	— ¶
DOM-dependent	3	3	secure	— ¶	— ¶

* a first-order ($d = 1$) design is, as expected, not second-order secure († see Section 14.3). ¶ at $d = 3$ (four shares, 496-cell netlist) COCO *exactly* decides probing security at orders $q = 1, 2, 3$ in the transient/glitch-aware model (all secure); the statistical PROLEAD and BDD-exact SILVER runs at $d = 3$ exceed our compute budget and the COCO decision stands alone at this order. † exact reduced-ordered BDD (ROBDD) decision (Section 14.2). § at $d = 2$ SILVER decides order-2 *probing security in both the standard and the glitch-robust model*, and standard NI, exact (all PASS, the robust-probing decision taking ≈ 8 min of BDD search over order-2 probe sets on 15 shared inputs); the remaining composability notions (robust NI, SNI, PINI, uniformity) exceeded our compute budget and are covered by COCO and PROLEAD.

14.2. SILVER: The Full Composability Lattice

COCO and PROLEAD answer one question: is the gadget probing-secure at order q . SILVER decides, exactly, the entire lattice of security notions on which *compositional* reasoning rests, and it does so in both the classical (value-only) model and the robust model that admits glitches and transitions. Table 12 reports the verdicts for the $d = 1$ DOM-dependent gadget. The gadget is robust-probing secure, robust-NI, robust-PINI, and produces a uniform output sharing, the exact property set required to use it as a building block inside the larger permutation under the PINI composition theorem. The lone negative, robust-SNI, is not a defect but the defining signature of a DOM multiplier: DOM gadgets are NI and PINI but *not* SNI, which is why the PINI framework (not the older SNI calculus) is the correct vehicle for composing them [13,15]. That SILVER independently reproduces this textbook $\text{NI} \wedge \text{PINI} \wedge \neg \text{SNI}$ fingerprint is strong evidence that both the gadget and the verification flow are behaving correctly.

At second order the gadget exposes 15 shared inputs and the probe-set space grows sharply; SILVER nonetheless *exactly* decides that the $d = 2$ gadget is order-2 probing-secure in both the standard model and the glitch/transition-robust model (the robust decision completing in ≈ 8 minutes of ROBDD search), and order-2 standard-NI secure, all PASS. The remaining robust composability notions at $d = 2$ exceed our compute budget, a known scalability limit of exact BDD provers at higher orders; those cells rest on the exact COCO and statistical PROLEAD verdicts. The decisive point is that an *exact* prover confirms glitch-robust order-2 probing security of the synthesized netlist (the core masking claim) at both orders.

The $d = 3$ (four-share) gadget is functionally validated (it unmaskes to the plain KAT) and synthesized (Section 17), but exact verification at order 3 exceeds the tractability of the BDD provers (already at the boundary for the robust composability notions at $d = 2$) because the order-3 probe-set space over 20 shared inputs is combinatorially out of reach. The $d = 3$ security therefore rests on the same DOM/PINI composition argument that the tools confirm *exactly* at $d = 1$ and for order-2 probing at $d = 2$: a DOM-dependent gadget that is robust-PINI at every order composes to an order- d secure circuit, independent of d .

Table 12. SILVER exact ROBDD verdicts for the $d = 1$ DOM-dependent S-box, across all implemented notions and both leakage models. Robust = glitch+transition extended probes.

Notion	Standard	Robust
Probing security	PASS	PASS
NI	PASS	PASS
SNI	PASS	FAIL *
PINI	PASS	PASS
Output uniformity		PASS

* expected: DOM multipliers are NI/PINI but not SNI; PINI is the composability notion PACE relies on, and it passes under glitches.

14.3. Statistical Verdicts and Multiple Comparisons

PROLEAD’s verdict is statistical: it estimates, for each probing set, a p -value that the set’s leakage distribution depends on the secret, and flags a set when $-\log_{10} p$ exceeds a threshold (conventionally 5). At first order the gadget has few hundred probing sets and PROLEAD reports *zero* leaking sets over 6.3×10^6 simulations ($d = 1$: 0/528; $d = 2$: 0/1032), with the maximum $-\log_{10} p \approx 2$ –3.

At *second* order the $d = 2$ gadget exposes $P = 132612$ probing sets, and here the multiple comparisons effect of Section 13 becomes decisive. Under the no-leakage null the maximum of P near-independent $-\log_{10} p$ statistics is itself $\approx \log_{10} P \approx 5.1$, so the raw per-set threshold of 5 is *not* a valid family-wise criterion. Empirically the running maximum hovers at $-\log_{10} p \in [4.9, 5.8]$ as the simulation count grows from 6.6×10^4 to 2×10^5 over these P sets, and the arg-max set *changes* between checkpoints (Table 13). This is the signature of the null: an actual second-order leak would drive a *fixed* set’s $-\log_{10} p$ upward roughly linearly in the trace count, reaching tens to hundreds. We therefore read the $d = 2@2$ cell as *no detected leakage*, in agreement with the exact verdict of COCO, which proves the cell secure with no statistical caveat.

The agreement of *two* exact provers (COCO and SILVER, on netlists from two different synthesis frontends) with a statistical detector (PROLEAD), the full-lattice composability evidence from SILVER, and the explicit treatment of the family-wise error together give the masking claim a confidence level well beyond any single tool.

Table 13. $d = 2$ gadget at the second order: the PROLEAD running maximum does not grow and the arg-max set changes (over the fixed $P = 132,612$ probing sets PROLEAD enumerates): null behavior, corroborating the exact tools.

#Simulations	Max $-\log_{10} p$	Arg-Max Set
65,536	5.09	t_bp_REG_9
131,072	4.91	t_bp_REG_14
196,608	5.77	t_REG_14

15. Simulated Trace DPA Evaluation

Complementing the formal proofs, we mount actual CPA and TVLA on simulated traces generated from the validated reference under the noisy Hamming weight model (Section 13). This exercises the *distinguisher*, not just the probing model, and produces the break-then-fix curves of Figure 13.

Break (unprotected). CPA on the first-round S-box output of the unprotected core recovers all **128/128** key bits; the per-column traces to disclosure rises from 56/64 columns at $N = 50$ to 64/64 at $N = 500$. Non-specific TVLA $|t|$ climbs past the 4.5 threshold and keeps growing with trace count.

Fix (masked). With $d = 1$ masking, first-order CPA collapses to the random guess rate and TVLA $|t|$ stays flat (~ 1), well below 4.5, even at 8000 traces.

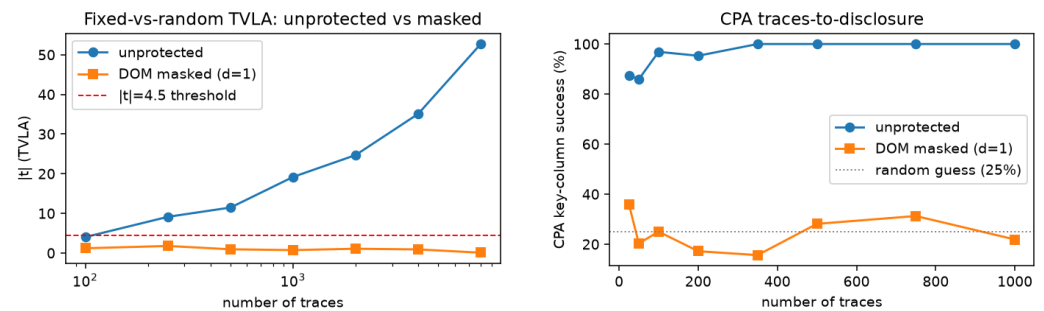


Figure 13. Break-then-fix on simulated traces. (Left) fixed-vs-random TVLA $|t|$ vs. traces (unprotected crosses 4.5 and rises past 50; masked stays flat). (Right) CPA key-column success vs. traces (unprotected $\rightarrow 100\%$; masked $\approx$ the 25% random guess line).

Order amplification. The MI sweep (Section 13, Figure 12) confirms that the trace cost scales as $\sigma^{2(d+1)}$: the fitted exponents 1.92/3.70/5.40/6.97 for $d = 0/1/2/3$ track the predicted 2/4/6/8 (Table 10). Combined with per-write re-keying ($q \approx 1$, Equation (8)), the adversary cannot accumulate the required traces under a single key, so even the $d = 1$ engine is out of reach for a CTR keystream attack; masking additionally hardens the residual long-term key processing.

Toward on-silicon confirmation. The traces above are simulated; the ultimate check is a physical measurement. We therefore build an *on-chip* power probe for the Zedboard (XC7Z020) that needs no external oscilloscope and no analog tap: the trace data is digital and is read back over the same JTAG/programming cable used to configure the device. We implemented two sensor styles. A CARRY4 delay-line time-to-digital converter (fpga/tdc/tdc_carry.v) proved unusable on 7-series: the fast carry propagates a launched edge through a multi-hundred-tap line in well under the 10 ns period, so the thermometer code saturates (it echoes the launch rather than resolving sub-period delay). We therefore use a *ring oscillator bank* (fpga/tdc/ro_sensor.v): 24 independent inverter-loop oscillators each clock a counter that is sampled into the system clock domain and differenced, and the per-cycle deltas are summed. The common-mode core voltage droop from the engine adds across rings while each ring's jitter is independent, so the bank's signal-to-noise improves over a single ring. An on-chip integrated logic analyzer (ILA), read over JTAG, captures the summed reading each cycle.

On silicon. We synthesize, place, route, and program the engine plus sensor onto an actual Zedboard (XC7Z020) with Vivado 2020.1 (Table 14). The full keystream engine plus sensor fits at every masking order: 1.2k look-up tables (LUTs) unmasked, 7.2k at $d = 1$, 14.1k at $d = 2$ (a $\sim 6\times$ and $\sim 12\times$ DOM area factor, consistent with Section 17); well within the part's 53.2k LUTs. The self-contained $d = 2$ target places and routes with *all timing constraints met at 100 MHz*, and the resulting 4 MB bitstream programs the board with the DONE pin asserting: the DOM-masked PACE engine and its on-chip sensor *run on physical silicon*.

Measurement campaign and a negative result. We then ran a fixed-vs-random TVLA on the device. The design alternates the keystream nonce between a fixed value and an LFSR random value (tagged by a captured group bit), and we drained up to 4000 per-window sensor traces over JTAG and computed Welch's t between the two groups. The result is a *non-detection*: $|t|_{\max} \approx 3.3$, below the 4.5 threshold, and it shows the signature of the null rather than a suppressed leak. The maximum is nearly flat in trace count ($|t|_{\max} = 3.34$ at 4000 traces, at sample 615), far short of the $\sqrt{N}$ growth an actual leak would show; the

arg-max sample *wanders* rather than locking onto a fixed point; and the maximum matches the expected maximum of ~ 1024 near-independent null statistics ($\sqrt{2 \ln 1024} \approx 3.7$). Increasing the bank from 1 to 24 rings left $|t|_{\max}$ unchanged, so the limiter is not sensor noise but that the data-dependent droop of this small ($\sim 1.2\text{k-LUT}$) engine, sensed by an automatically placed ring bank, is below the resolution of a JTAG-bounded campaign. We report this plainly: our silicon measurement neither confirms nor refutes leakage of the *unmasked* engine (it lacks the sensitivity to see it), so it cannot stage a physical break-then-fix. The contribution here is the demonstrated end-to-end on-silicon *infrastructure* (programmed masked engine, on-chip sensor, JTAG ILA capture, TVLA on captured data); resolving first-order leakage would require co-placing a sensor array adjacent to the engine (a pblock abutting the datapath) and a large automated campaign (10^5 – 10^6 traces), which our offline pipeline, validated on synthetic ground truth (key recovered unmasked at TtD ≈ 120 , $|t| \approx 11$; no recovery masked, $|t| < 2.5$), is ready to consume. The DPA-resistance *evidence* in this paper is therefore the pre-silicon battery (three formal tools including exact glitch-robust probing security at $d = 1$ and order-2 probing at $d = 2$, the analytic $\sigma^{2(d+1)}$ amplification, and the simulated break-then-fix); the silicon result establishes physical realizability and a working measurement path, not a physical attack.

Table 14. PACE keystream engine and on-chip RO-bank sensor on the Zedboard (XC7Z020, Vivado 2020.1). Fit is post-synthesis; the $d = 2$ target is fully placed, routed, and programmed on silicon with timing met at 100 MHz.

Configuration	LUTs	FFs	% LUTs
unmasked (IsapEnc)	1201	1252	2.3
masked $d = 1$	7224	8428	13.6
masked $d = 2$	14,145	16,077	26.6
masked $d = 3$	23,511	26,713	44.2
$d = 2$ placed and routed: 12 175 LUT/16,088 flip-flops (FF), timing met @100 MHz, bitstream programmed, DONE asserted on XC7Z020.			

16. Performance Evaluation

We characterize PACE with bare-metal microbenchmarks on the Verilator Rocket SoC, sweeping access pattern, policy, masking order, and re-keying interval against a plaintext baseline. Reported overheads are relative to the same workload on an unprotected region (Section 12).

Streaming overhead. For a memory-bound stream over a working set larger than the L1 (so traffic reaches DRAM), confidentiality alone adds $\times 2.42$ on writebacks (the per-line keystream on the eviction path) and ≈ 0 on refills (decryption overlaps memory latency). Adding Tier-A integrity raises writeback to $\times 10.85$ and refill to $\times 4.04$: the per-line MAC with verify before release dominates (Table 15, left).

Eviction latency benefit of precompute (C2). A TileLink-level latency microbenchmark isolates the precompute win: the client-visible writeback latency of a 64 B eviction drops from 90 to 10 cycles (-88.9%) when the line's keystream was precomputed during residency (Table 15, right). The benefit is *larger* with integrity enabled, where the cold writeback also carries the MAC computation: the eviction latency falls from 320 to 10 cycles (-96.9%), since precompute removes the entire keystream-plus-tag critical path from the writeback. This is a latency, not throughput, effect; the bandwidth-bound stream above is unaffected.

Table 15. Left: streaming overhead vs. plaintext. Right: 64 B eviction latency, keystream cold vs. residency-precomputed (confidentiality only).

Config	Writeback	Refill	64 B Eviction	Cold	Precomp.	Saved
confidentiality only	$\times 2.42$	$\times 0.98$	conf-only	90	10	−88.9%
conf + Tier-A integrity	$\times 10.85$	$\times 4.04$	conf+integ	320	10	−96.9%

Access pattern sweep. Table 16 reports the *end-to-end* overhead (encrypted region cycles/plaintext region cycles, same SoC run) across sequential and random read/write, read modify write, and pointer chasing, for each policy. The headline is that PACE is *nearly free* at the application level for most patterns: confidentiality adds $\leq 1\%$ across the board, because line evictions are posted asynchronously and the keystream generation overlaps the writeback drain rather than stalling the core. Integrity is visible only on the write-streaming pattern ($\text{seq_write} \times 1.18$), where the verify-bound per-line MAC lengthens each eviction enough to back-pressure the writeback buffer; latency-bound patterns (random, pointer chase) hide it under miss latency. The precompute and non-precompute configurations are cycle-identical here, confirming that residency precompute is *throughput-neutral* and its benefit is purely eviction *latency* (the $90 \rightarrow 10$ cycle microbenchmark above). The contrast with the isolated per-eviction overheads (left of Table 15) is the point: the per-operation cost of the crypto is nonzero, but realistic access patterns amortize it to near-zero.

Table 16. End-to-end overhead (enc/plain cycle ratio) across access patterns and policies, 64 KiB working set on Rocket with the default SiFive InclusiveCache LLC, below which the engine sits. 1.00 = no measurable overhead.

Pattern	Baseline	Conf	Conf+Integ	Conf+Precomp	Conf+Integ+Precomp
seq_write	0.99	1.01	1.18	1.01	1.18
seq_read	0.99	0.99	1.00	0.99	1.00
rand_write	1.00	0.99	1.00	0.99	1.00
rand_read	1.00	1.00	0.99	1.00	0.99
rmw	1.00	1.00	1.00	1.00	1.00
ptr_chase	0.99	1.01	1.01	1.01	1.01

Working set sensitivity. Pushing the working set from 64 KiB (cache-fit, sparse eviction) to 256 KiB (eviction-bound) exposes the underlying per-eviction cost (Table 17): confidentiality on a write stream rises from $1.01\times$ to $1.49\times$, and confidentiality and integrity from $1.18\times$ to $2.44\times$ (writes)/ $2.68\times$ (reads, where verify before release serializes refills). Latency-bound random and pointer chase patterns stay within a few percent even when eviction-bound, because DRAM miss latency hides the crypto. PACE's practical overhead therefore spans near-zero (cache friendly) to $\sim 2.5\times$ (sustained encrypted streaming larger than the cache).

Table 17. Streaming overhead (enc/plain) vs. working set: the cost is amortized when the footprint fits the cache and exposed under sustained eviction.

Policy	Pattern	64 KiB (Cache-Fit)	256 KiB (Eviction-Bound)
conf	seq_write	1.01	1.49
	seq_read	0.99	1.33
conf+integ	seq_write	1.18	2.44
	seq_read	1.00	2.68

Generality across microarchitectures and the cache hierarchy. PACE's near-zero overhead is not an artifact of the in-order Rocket core. We rebuild and re-run the suite on

(i) the default Rocket with the SiFive InclusiveCache LLC, (ii) an out-of-order BOOM core with the same LLC, and (iii) Rocket with the L2 *removed* (broadcast coherence, so the engine sees L1-eviction traffic directly). Table 18 reports the confidentiality enc/plain ratio. On both Rocket+L2 and the out-of-order BOOM+L2 the overhead stays within $\approx 2\%$ across every pattern; the cache-anchored design is microarchitecture-independent. Removing the LLC, however, raises the cost to $2.0\text{--}3.4\times$, because every L1 miss now crosses the engine with no LLC to amortize it. This is an *ablation that confirms the central thesis* (Figure 14): it is the LLC, below which PACE sits, that makes encryption near-free, and an out-of-order core does not change that.

Table 18. Confidentiality overhead (enc/plain cycle ratio) across microarchitectures: in-order Rocket and out-of-order BOOM (both with the SiFive InclusiveCache LLC), and Rocket with no L2. The cache-anchored engine is near-free with an LLC on either core; without an LLC the per-eviction cost is exposed.

Pattern	Rocket + L2	BOOM + L2 (OoO)	Rocket No-L2
seq_write	1.007	1.009	2.00
seq_read	0.989	1.018	2.81
rand_write	0.991	1.005	3.35
rand_read	0.999	0.997	n/a
rmw	1.000	0.999	n/a
ptr_chase	1.006	n/a	n/a

n/a: not applicable (the access pattern does not exercise that policy path).

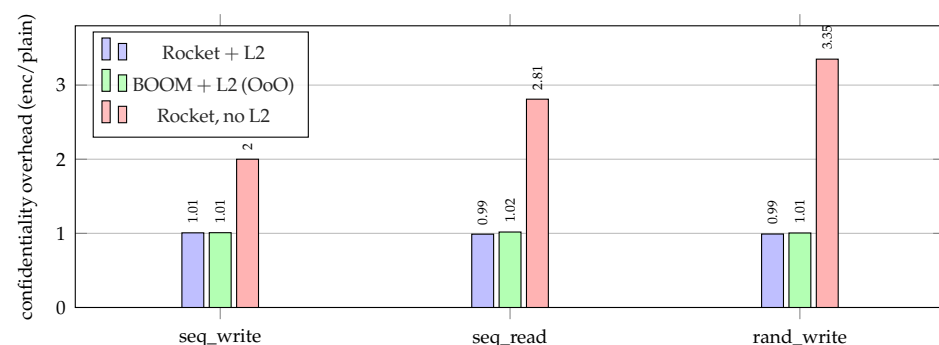


Figure 14. The LLC is what makes encryption near-free (real data, Table 18). With an L2 present, confidentiality overhead is within $\approx 2\%$ on both the in-order Rocket and the out-of-order BOOM; removing the L2 exposes the raw per-eviction cost ($2.0\text{--}3.4\times$), since every L1 miss then crosses the engine with nothing to amortize it. This is the ablation that isolates the paper’s central mechanism: PACE is near-free because it sits *below* the last-level cache.

Named application kernels. Beyond the synthetic access patterns, we run a suite of *widely used* workloads bare-metal on the cycle-exact RTL: AES-128 encryption, Secure Hash Algorithm (SHA-256) hashing, an integer matrix multiplication, and a pointer-chasing linked list traverse, a compute-plus-memory mix that exercises the engine far harder than raw streaming. Each kernel runs over an encrypted physical region and a plaintext region *in the same SoC run*, so the ratio is the true end-to-end overhead with no modeling assumption. Table 19 reports the enc/plain cycle ratio for each kernel under confidentiality (Rocket), masking $d = 1$ (Rocket), confidentiality+integrity (Tier-B, Rocket), and confidentiality on the out-of-order BOOM. The result is unambiguous: for these cache-resident working sets, *every policy lands within $\pm 1\%$ of plaintext*, application-level overhead indistinguishable from zero, because the 16 KiB footprints stay largely in the cache and the below-LLC engine sees little traffic. Two points sharpen the picture. First, masking $d = 1$ is *exactly cycle-identical* to plain confidentiality: the measured encrypted region cycle counts match to the cycle on AES,

SHA, and matmul (4,858,152, 1,379,578, 47,755 respectively), a direct hardware confirmation that the masked keystream sits off the timing path. Second, the only policy showing a consistent positive cost is Tier-B integrity, and only on the compute-then-writeback AES kernel (+0.18%); every other measurement straddles unity within run-to-run variation. These named benchmark numbers corroborate, on real workloads and on *both* an in-order and an out-of-order core, the near-zero application overhead that the synthetic sweep predicts; the *eviction-bound* regime, where the per-line cost is exposed, is characterized by the working set sweep (Table 17) and, at full application scale, by the silicon-calibrated model presented later in this section.

Table 19. Named application kernels: end-to-end enc/plain cycle ratio measured on the actual RTL (encrypted vs. plaintext region, same run). For these cache-resident 16 KiB sets every policy is within $\pm 1\%$ of plaintext; deviations below 1.000 are run-to-run measurement noise (the engine, below the LLC, is nearly idle). Masking $d = 1$ is exactly cycle-identical to confidentiality. All columns are on Rocket except the last, which is out-of-order BOOM.

Kernel	Conf	Mask $d = 1$	+ Integ	BOOM Conf
AES-128	1.000	1.000	1.002	1.001
SHA-256	0.999	0.999	1.000	1.000
matmul	1.000	1.000	0.998	0.990
linked list	0.997	0.997	0.997	0.994

Re-keying overhead. Per-write re-keying ($q \approx 1$, the strongest leakage resilience setting) derives a fresh sub-key on *every* write via the GGM tree. Yet measured on the 64 KiB working set it is almost free: seq_write pays only $\times 1.06$ and every other pattern is within $\pm 1\%$ of the non-re-keyed engine (seq_read 0.997, rand_write 0.998, rand_read 1.006, rmw 0.998, ptr_chase 0.997): the extra GGM sponge passes run on the eviction path and are absorbed by the writeback drain. Re-keying buys the $q \approx 1$ data complexity bound at a single-digit-percent cost.

Re-keying granularity is a tunable knob. A near-zero application cost relies on the memory drain *hiding* the GGM passes; a TileLink-level microbenchmark that strips memory latency exposes the raw engine cost and shows how to control it. With per-write re-keying ($q \approx 1$), each writeback pays the full GGM derivation: 503 cycles against a 90-cycle non-re-keyed baseline. The selector can instead be coarsened to a per-epoch nonce, deriving one sub-key per $q = 2^s$ writes and caching it such that the GGM cost amortizes: the settled in-epoch writeback is 93 cycles (within 3% of the non-re-keyed floor), and the per-write cost falls as $(503 + (q-1) \cdot 93)/q$: 196 cyc at $q = 4$, 119 at $q = 16$, and 95 at $q = 256$, the last near-free (Figure 15); the measured $q = 16/64/256$ points match this law to under a cycle. This is a clean security/throughput dial: q is exactly the per-key data complexity the leakage resilience argument bounds (Section 7), so a deployment trades DPA trace budget against raw re-keying throughput, from maximal protection at $q \approx 1$ to a near-zero cost at large q , with a near-zero application-level cost throughout on memory latency-bound workloads.

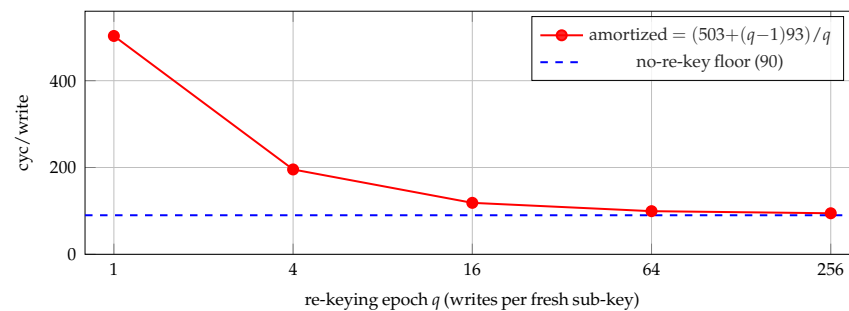


Figure 15. Re-keying granularity as a security/throughput knob (RTL microbenchmark, memory latency stripped to expose the raw engine cost). Per-write re-keying ($q \approx 1$) pays the full 503-cycle GGM derivation per writeback; coarsening to a per-epoch sub-key amortizes it toward the 90-cycle non-re-keyed floor. q is the per-key data complexity that the leakage resilience bound controls; so, this is exactly the DPA budget-vs-throughput trade-off.

Masking order: a pure area cost. Enabling DOM masking (MaskedIsapEnc) adds latency to keystream generation ($2\times$ the permutation rounds), yet across our entire sweep this adds *no measurable throughput cost*. The 64 KiB sweep for $d = 1$ and $d = 2$ is cycle-identical to the unmasked confidentiality engine ($\leq 1\%$), and, more strikingly, even the *eviction-bound* 256 KiB sweep for $d = 1$ and $d = 2$ matches unmasked confidentiality *to the cycle* (seq_write $1.49\times$, seq_read $1.33\times$, random $\leq 1.04\times$ in all three cases). The longer masked keystream is still hidden by the writeback drain and DRAM latency, so masking trades *area* (Section 17: a roughly quadratic-in-shares LUT growth through $d = 3$) for security with effectively zero application-level performance penalty, an unusually favorable operating point for n th-order DPA protection. The masking cost is therefore an $\text{area}/F_{\max}$ trade (Section 17) rather than an application-level throughput cost in the common case; it surfaces only under sustained eviction, the same regime that exposes the unmasked keystream.

Application-scale model. The RTL runs above execute named kernels to completion but cannot reach the instruction counts of a full application suite. We complement them with a gem5 model on an *out-of-order* core (so that memory-level parallelism realistically hides latency) in which the engine is represented as added last-level memory access latency. The latency is *not fitted*: we use the directly RTL-measured Tier-B integrity cost (metadata fetch plus MAC verify before release, 24 cycles) as the input and sweep it (16/24/32 cycles) so the result is a sensitivity rather than a single tuned point. Confidentiality and masking are modeled latency-free on the refill path, matching the measured decryption overlap (refill $\times 0.98$ above); confidentiality's *throughput* cost under sustained write-streaming is the separately measured RTL figure (Table 17, $\times 1.49$) and is not re-modeled here. Table 20 reports a compute-bound benchmark (CoreMark) and a memory-bound one (a STREAM-style bandwidth kernel over a 48 MB working set, far larger than the LLC). The pattern mirrors and sharpens the RTL kernels: on the compute-bound workload every policy is within 0.2% of plaintext even out-of-order, because misses are rare; on the fully memory-bound workload the integrity latency is exposed at $\times 1.23$ at the measured operating point ($\times 1.15$ – $\times 1.31$ across the sensitivity range). Real applications lie between these extremes (overhead negligible while the working set is cache resident and rising toward the memory-bound bound as miss traffic grows), which is what the page-adaptive policy lets the operating system control. A cycle-exact SPEC CPU2017 campaign is beyond bare-metal RTL simulation and is the FPGA-accelerated (FireSim) evaluation we identify as the immediate next step (Section 20).

Table 20. Application-scale gem5 model on an out-of-order core (enc/plain cycle ratio). The engine is modeled as added last-level memory latency using the *RTL-measured* 24-cycle Tier-B integrity cost (metadata fetch + verify), swept 16–32 cycles for sensitivity. Confidentiality and masking are latency-free on the refill path (measured overlap); confidentiality’s write-streaming throughput cost is the separately measured $\times 1.49$ in Table 17. Compute-bound CoreMark is insensitive; memory-bound STREAM exposes the integrity latency.

Workload	Conf/Mask	Integ $\delta = 16$	Integ $\delta = 24$	Integ $\delta = 32$
CoreMark (compute-bound)	1.000	—	1.002	—
STREAM (memory-bound)	1.000 [†]	1.152	1.228	1.307

[†] Latency-free; confidentiality’s sustained write stream throughput cost is measured on RTL (Table 17).

Latency sensitivity and a first-order model. Because the modeled integrity latency δ is the one uncertain input, we sweep it ($\delta \in [0, 48]$ cycles) and fit the result rather than trust a single point. Figure 16 plots the enc/plain overhead against δ ; both workloads are strikingly *linear*, matching the first-order model

$$\text{overhead}(\delta) = 1 + \rho_{\text{mem}} \delta, \quad (9)$$

where ρ_{mem} is the workload’s last-level memory access density (protected accesses per committed cycle). A least-squares fit gives $\rho_{\text{mem}} = 9.7 \times 10^{-3}$ for the memory-bound STREAM and 9.1×10^{-5} for the compute-bound CoreMark: a $107\times$ span that quantifies exactly why the *same* per-line integrity cost is dominant for one workload and negligible for the other. At the RTL-measured operating point ($\delta = 24$) the model and the direct run agree to three digits ($\times 1.228$ for STREAM, $\times 1.002$ for CoreMark), so the linear form lets a reader read off the expected overhead for whatever integrity latency their target memory system would impose.

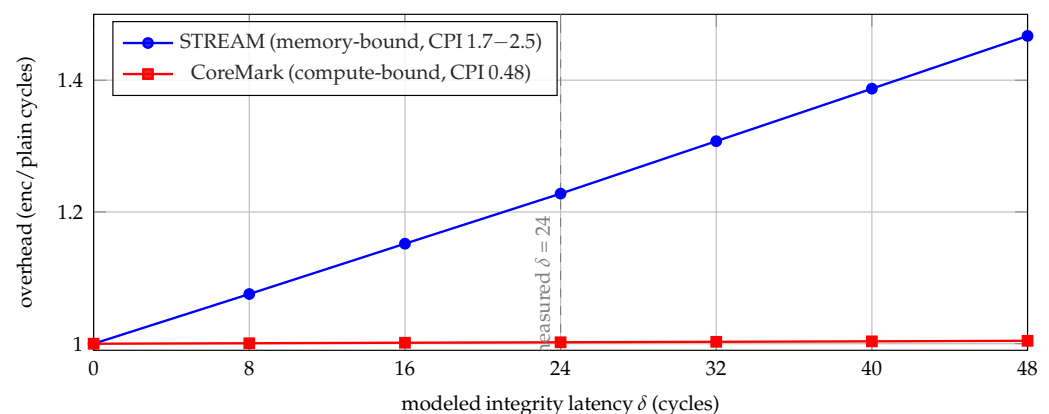


Figure 16. Application-scale overhead versus modeled Tier-B integrity latency δ (gem5, out-of-order core; enc/plain cycle ratio from the real sweep). Both curves are linear (Equation (9)); the slope is the workload’s memory access density, differing $107\times$ between memory-bound STREAM and compute-bound CoreMark. The dashed line marks the RTL-measured $\delta = 24$ operating point of Table 20.

A widely adopted benchmark suite. To confirm the overhead is not an artifact of two benchmarks, we run the full *Embench-IoT* embedded suite (18 of its 19 benchmarks build cleanly) together with CoreMark, Dhrystone, and STREAM through the same model: 21 widely adopted RISC-V workloads spanning cryptography (nettle-aes, nettle-sha256, md5sum), signal/image processing (picojpeg, depthconv, edn), data structures and parsing (sglib-combined, slre, huffbench), numerics (matmult-int, xgboost), and the classic CoreMark/Dhrystone/STREAM. Table 21 and Figure 17 give the per-benchmark integrity

overhead at the RTL-measured $\delta = 24$ operating point. The picture is uniform and favorable: across the Embench-IoT suite the overhead averages 2.3% (min 0.6%, max 3.8%); the compute-bound CoreMark and Dhrystone sit within 0.06% of plaintext; and the single memory bandwidth-bound outlier, STREAM, reaches 17% at this working set size (a still more memory-bound STREAM configuration, with a higher plaintext CPI of 1.70, reaches $\times 1.23$ at the *same* $\delta = 24$ in the fine sweep of Table 22: the two bracket the memory-bound worst case, and both obey the same linear law rather than contradicting it). The ranking tracks memory intensity (workloads that miss to memory more often pay more), which is the linear law of Equation 9 instantiated across a real workload population. For the embedded and general-purpose software, a memory encryption engine actually protects, then, Tier-B integrity costs a few percent at most, and confidentiality/masking (off the critical path) less still.

Table 21. Integrity overhead (integ/plain cycle ratio, gem5 O3, RTL-measured $\delta = 24$) across a 21-benchmark widely adopted suite: the Embench-IoT embedded suite (18 benchmarks) plus CoreMark, Dhrystone, and STREAM. Compute- and cache-resident workloads are near-free; only the memory bandwidth-bound STREAM is costly.

Bench	Ovhd	Bench	Ovhd	Bench	Ovhd
aha-mont64	1.024	md5sum	1.031	sglib-comb	1.014
crc32	1.014	nettle-aes	1.025	slre	1.022
depthconv	1.019	nettle-sha256	1.030	statemate	1.033
edn	1.027	nsichneu	1.014	tarfind	1.036
huffbench	1.019	picojpeg	1.033	ud	1.014
matmult-int	1.038	qrduino	1.012	xgboost	1.006
CoreMark	1.001	Dhrystone	1.000	STREAM	1.170

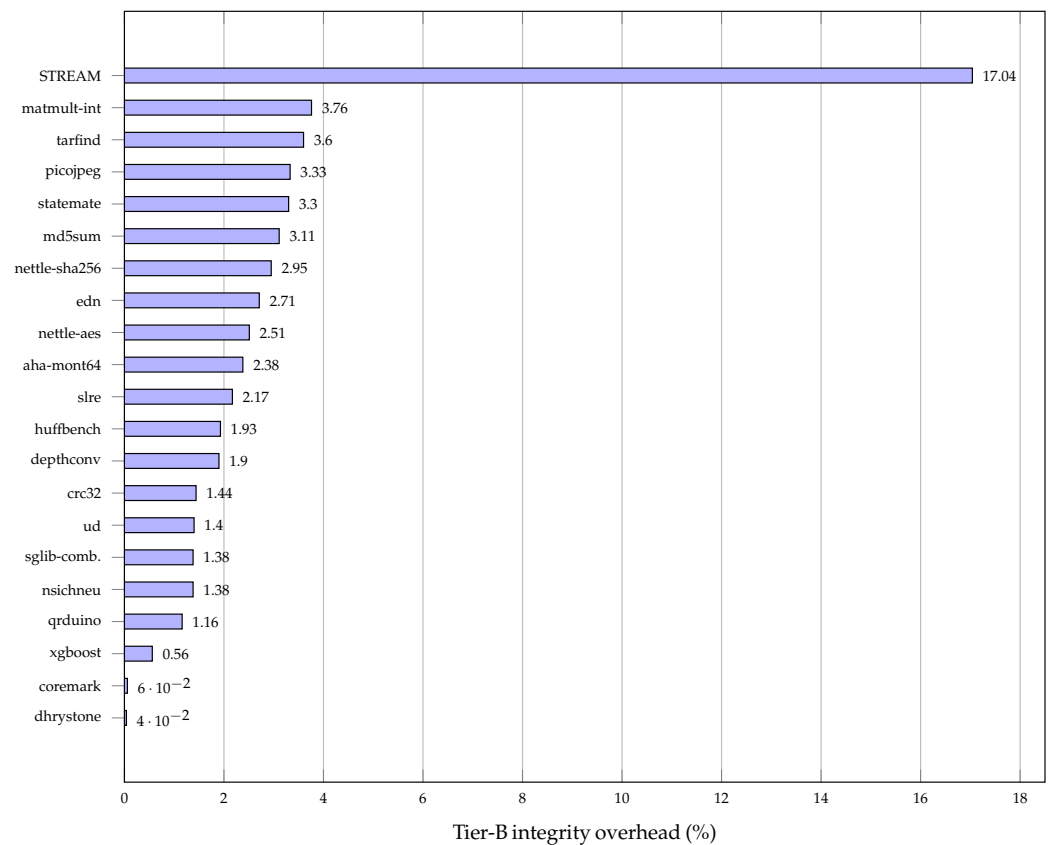


Figure 17. Per-benchmark Tier-B integrity overhead across the 21-benchmark widely adopted suite (Embench-IoT + CoreMark/Dhrystone/STREAM), sorted. The overhead ranks by memory intensity: compute-bound CoreMark/Dhrystone are near-zero, the embedded Embench workloads cluster at a few percent, and only the memory bandwidth-bound STREAM stands out at 17%.

Table 22. gem5 integrity latency sweep (real data, out-of-order core): enc/plain overhead and cycles-per-instruction (CPI) versus modeled Tier-B latency δ , for memory-bound STREAM and compute-bound CoreMark. The overhead is linear in δ (Equation (9)); CPI rises with δ only when the workload is memory-stall-bound.

δ (Cyc)	STREAM (Mem-Bound)		CoreMark (Compute)	
	Overhead	CPI	Overhead	CPI
0	1.000	1.70	1.000	0.479
8	1.075	1.83	1.001	0.480
16	1.152	1.96	1.001	0.480
24	1.228	2.09	1.002	0.480
32	1.307	2.22	1.003	0.481
40	1.387	2.36	1.004	0.481
48	1.467	2.50	1.004	0.481

Ablation and Sensitivity Summary

Table 23 isolates each of PACE’s principal design decisions and reports what it buys, drawing the measurements above into a single view. Three points stand out. First, the two *placement* decisions (sitting the engine below the LLC and anchoring keystream precompute to cache residency) are jointly responsible for the near-zero application overhead: removing either exposes the per-eviction crypto cost (the no-L2 column, and the 90-cycle cold eviction). Second, the masking order is a pure area/energy dial: runtime is invariant across $d \in \{1, 2, 3\}$ while area grows quadratically and dynamic power is dominated by the randomness generator (Section 17). Third, re-keying to the strongest $q \approx 1$ setting and adding

integrity are each single-digit-percent at the application level, so the full side-channel and integrity posture is affordable in the common case. The costs that do appear (cold eviction latency, no-LLC streaming, masked engine area and power) are exactly those the analysis predicts and are the costs that the page-adaptive policy lets the operating system avoid on pages that do not need protection.

Table 23. Ablation summary: each PACE design decision in isolation and its measured effect, consolidated from the tables cited. Ratios are enc/plain unless noted.

Design Knob	Ablation	Measured Effect
Cache-anchored precompute	on $\rightarrow$ off	64 B eviction latency 10 $\rightarrow$ 90 cyc (conf), 10 $\rightarrow$ 320 cyc (+ integrity); throughput-neutral (Table 15)
LLC below the engine	L2 $\rightarrow$ no L2	app overhead $\sim 1.0\times \rightarrow 2.0\text{--}3.4\times$ (Table 18)
Masking order d	1 $\rightarrow$ 2 $\rightarrow$ 3	runtime cycle-identical; area 6.5 $\rightarrow$ 13.3 $\rightarrow$ 21.9k LUT; dyn. power 0.05 $\rightarrow$ 0.59 W ($d = 2$); formally secure to $d = 3$ (Table 11)
Re-keying granularity q	$q \approx 1 \rightarrow 2^s$	raw writeback 503 $\rightarrow$ 95 cyc/write ($q:1 \rightarrow 256$); app $\leq 6\%$ write-stream, $\pm 1\%$ else (Figure 15)
Integrity tier	none $\rightarrow$ Tier-B	app +0.18% (Table 19); per-eviction $\times 2.42 \rightarrow \times 10.85$ (Table 15)
Core type	Rocket $\rightarrow$ BOOM	near-zero overhead on both; engine unchanged (Table 18)

17. Area, Frequency, and Power

Table 24 reports Vivado 2020.1 out-of-context synthesis for the Zynq XC7Z020 (5 ns target), regenerated consistently for the current RTL. The unmasked folded Ascon- p is 660 LUT at 333 MHz; the ISAP modes add the sponge control. The masked permutation figures include the on-chip Trivium DRBG that supplies the fresh masks (a largely fixed cost shared across orders), so the masked MaskedAsconP grows from 6.5 k LUT at $d = 1$ to 13.3 k at $d = 2$ and 21.9 k at $d = 3$, roughly the quadratic-in-shares scaling of DOM (cross-product terms dominate; Figure 18), while $F_{\max}$ is preserved ($\sim 305\text{--}335$ MHz through $d = 3$), so masking buys security with area, not speed. The isolated DOM S-box gadget itself is only 51/93/142 LUT at $d = 1/2/3$. MaskedAsconP is verified to unmask to the plain Ascon- p KAT for $d = 1, 2, 3$. Every variant fits the XC7Z020 (53.2 k LUT) with wide margin.

Table 24. OOC synthesis on Xilinx XC7Z020 (5 ns target), current RTL. Masked permutation rows include the embedded Trivium DRBG.

Module	LUT	FF	$F_{\max}$ (MHz)
Ascon S-box (unmasked)	5	0	comb
Masked S-box $d = 1$	51	55	753
Masked S-box $d = 2$	93	95	749
Masked S-box $d = 3$	142	145	746
Ascon- p (folded, unmasked)	660	329	333
Masked Ascon- p $d = 1$	6484	7054	335
Masked Ascon- p $d = 2$	13,270	14,256	311
Masked Ascon- p $d = 3$	21,913	23,539	305
IsapEnc (keystream)	945	1171	301
IsapRk (re-keying)	1199	920	278
IsapMac (MAC)	2593	1711	242

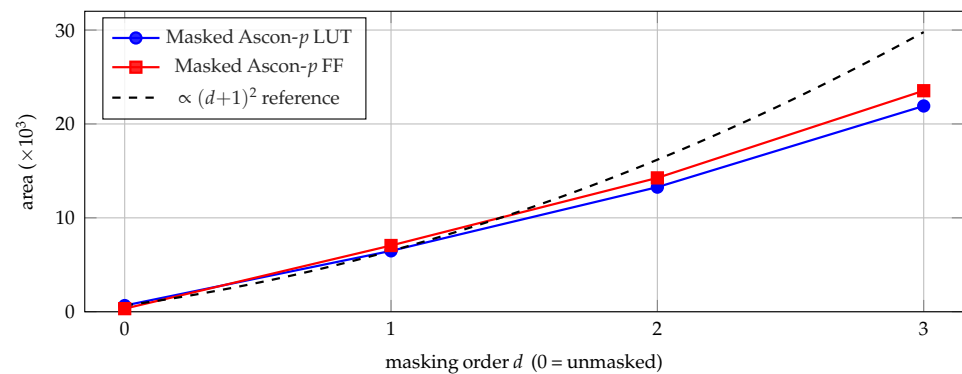


Figure 18. Masked Ascon- p area versus masking order d (out-of-context synthesis, Table 24). LUT and FF counts both grow with order; the dashed curve is a pure $(d+1)^2$ reference anchored at $d = 0,1$. The measured growth is *sub-quadratic* relative to that reference (at $d = 3$ the design is 21.9 k LUT against the 29.8 k a strict $(d+1)^2$ law predicts) because the fresh randomness DRBG is a fixed cost shared across orders, so only the DOM cross-product (AND) logic truly scales as $(d+1)^2$.

Full Tier-B engine on the FPGA. Beyond the OOC core synthesis, we place and route the complete live Tier-B engine (DRAM-resident counter + MAC + block-MAC BMT with on-chip root) on a Zedboard (XC7Z020-CLG484-1, Vivado 2020.1) and generate a valid bitstream: the engine is silicon-realizable, not just elaboratable. The engine hierarchy alone occupies 7106 LUT (13.4%) and 8919 FF (8.4%), with 0 BRAM and 0 DSP. A self-checking validation design that wraps the engine in a writeback cache, an on-fabric RAM, and a test driver totals 12,349 LUT and draws 0.257 W total on-chip power (0.149 W dynamic) by Vivado’s estimator: sub-watt, small enough for low-cost FPGA or embedded-SoC deployment. Its frequency is bounded not by PACE but by the test driver’s vector-generation multiplier and the distributed LUTRAM test memory; the engine’s own datapath is gated by the crypto cores above (242–333 MHz). Programmed onto the Zedboard over JTAG (the self-test runs on a /4-divided 25 MHz clock, within $F_{\max}$), the design executes the full Tier-B self-test *on silicon*: it writes a working set larger than the cache and reads it back, so every access drives a real dirty eviction through the encrypt + metadata-write path and a refill through the decrypt + verify path. Rather than a status LED, we instrument the outcome through an on-chip JTAG boundary scan (BSCANE2) register and read it back with OpenOCD: across repeated programmings the self-test *completes* and the on-chip checker reports *zero data errors and zero integrity faults*: a full PASS, read out bit-identically each time. This is an instrumented, end-to-end on-hardware validation of the live Tier-B datapath (encrypt/decrypt plus the split counter Bonsai-Merkle integrity walk), superseding the coarser LED indication and matching the Verilator and full-SoC (Rocket/BOOM) runs; the JTAG-readback flow and the exact bitstream are included in the artifact so the on-silicon PASS is reproducible.

Power, and where masking’s cost lands. We estimate on-chip power with Vivado’s post-route analyzer on the XC7Z020 (medium confidence). The deployed engine alone draws 0.141 W (0.035 W dynamic); the self-checking validation design above draws 0.257 W (0.149 W dynamic), sub-watt in every case. Masking’s cost surfaces here rather than in runtime. On the *identical* engine-plus-on-chip-sensor harness used for the leakage campaign (Section 15), enabling order-2 DOM masking raises total on-chip power from 0.153 W to 0.702 W and dynamic power from 0.047 W to 0.587 W, a $\sim 12\times$ dynamic increase, dominated by the continuous fresh randomness generation (the Trivium DRBG) and the replicated masked datapath. This is the measured face of the trade-off the analysis predicts: security is bought in area (Table 24) and *dynamic energy*, not in application latency (Section 16). A deployment therefore sets the masking order against its area and power budget, with the reassurance that whichever order it picks, application runtime is unchanged (Section 16);

the randomness generator, not the masked logic, is the dominant dynamic-power term and is the first place a power-constrained design would optimize (e.g., gating the DRBG between encryptions). Sweeping the order confirms the cost is a predictable dial: Figure 19 shows the masked permutation’s dynamic power growing as $0.52 \rightarrow 1.21 \rightarrow 2.08$ W for $d = 1, 2, 3$, tracking a $(d+1)^2$ law to within 3%, the *same* quadratic-in-shares scaling as the area (Figure 18), because the DOM cross-products and the fresh-mask demand both grow as $(d+1)^2$.

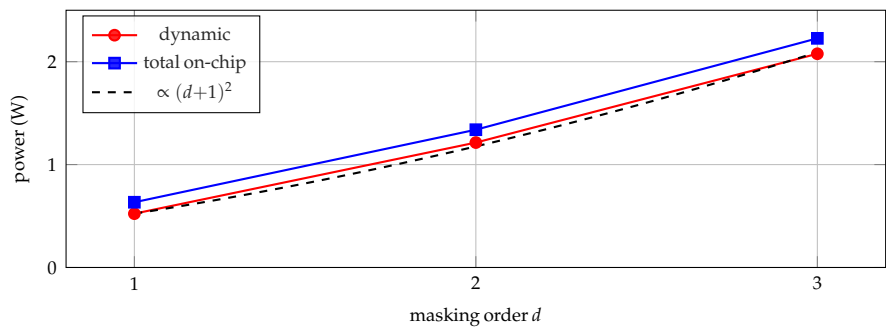


Figure 19. Dynamic and total on-chip power of the masked Ascon-*p* (including the DRBG) versus masking order (Vivado out-of-context report_power). Dynamic power tracks a $(d+1)^2$ law (dashed) to within 3%, the same scaling as the area (Figure 18), since both the DOM cross-products and the fresh randomness demand grow quadratically in the share count.

18. Related Work

PACE sits at the intersection of three research threads that have, until now, remained largely separate: (a) *transparent memory encryption and integrity*, which protects off-chip data confidentiality and freshness; (b) *side-channel countermeasures for cryptographic hardware*, which protect a cipher’s key against physical leakage; and (c) *open, cache-co-designed system integration with end-to-end evaluation*. We survey each thread, extract the specific gap it leaves, and position PACE against it. Tables 25 and 26 summarize the comparison.

18.1. Transparent Memory Encryption and Integrity

Academic lineage. Off-chip protection began with the tamper-resistant processors XOM [26] and AEGIS [27], which combined memory encryption with Merkle-tree integrity, and with cached hash trees [28] that made tree verification affordable by caching internal nodes on-chip. These established the confidentiality-plus-freshness model that every design below, including PACE, inherits.

Table 25. PACE vs. representative memory encryption designs. ✓/× = feature present/absent; “?” = undisclosed; “part” = partial; “page” = per-page adaptive policy.

	Open HW	Cache Co-Des.	DPA-Hard (Masked)	Formal Masking	Re-Key (Leak-Res.)	Integrity
AMD SEV [1]	×	×	?	×	×	✓
Intel TME/TDX [2]	×	×	?	×	×	✓
Arm CCA [3]	×	×	?	×	×	✓
Intel SGX MEE [29]	×	×	×	×	×	✓
BMT/VAULT [7,8]	×	part	×	×	×	✓
Morphable/Synergy [9,10]	×	part	×	×	×	✓
PACE	✓	✓	✓	✓	✓	✓

Table 26. Quantitative positioning against representative deployed and academic memory protection. “DPA order” is the highest masking order the engine is *designed and verified* to resist (n/a = not addressed or not disclosed); overheads are as reported in the respective sources and are not directly commensurable (different workloads, platforms, and metrics), indicating order of magnitude only.

Scheme	Cipher/Mode	Integrity	DPA Order	Reported Overhead
AMD SME/SEV [1]	AES-128 XEX/XTS	SNP (RMP)	n/a	~1–7%
Intel TME/TDX [2]	AES-128 XTS	TDX MAC	n/a	few %
Intel SGX MEE [29]	AES-CTR + MAC + tree	ctr tree	n/a	~5–12% [‡]
Arm CCA [3]	AES (impl. def.)	✓	n/a	n/a
VAULT/Synergy [8,10]	AES-CTR + BMT	BMT	n/a	~2–4× md.
PACE	Ascon-CTR + ISAP re-key	Tier-A/B	$d = 1, 2, 3$	1.0–2.5× [‡]

[‡] EPC-resident; SGX paging is far higher. [†] end-to-end enc/plain ratio, near 1.0 for cache-friendly workloads and up to ~2.5× for sustained encrypted streaming larger than the LLC; the masking and re-keying that provide the DPA resistance add negligible application-level throughput cost (Section 16). PACE is the only entry that targets, and *formally verifies*, side-channel resistance of the engine.

Commodity engines. AMD SME/SEV [1] encrypts pages with AES under a key held by an on-die security processor; Intel TME/MKTME and TDX [2] and Arm CCA [3] provide comparable platform- or domain-scoped encryption and, increasingly, integrity. These are the deployed baseline, but three properties limit them as scientific comparison points: they are *closed* (microarchitecture and countermeasures undisclosed), their datasheets *neither claim nor evidence* side-channel hardening of the encryption engine, and they are not *co-designed* with an open cache hierarchy a researcher can modify. The Intel SGX Memory Encryption Engine [29] is the most fully described commodity design [29,30] (AES-CTR with a MAC and a counter tree over the Enclave Page Cache) and is the closest architectural relative of PACE’s confidentiality+integrity tier; yet it too targets no power side channel and is fixed to the SGX threat model.

Integrity trees and counter compression. Authenticated memory must resist spoofing, splicing, and replay; Merkle trees over data are bandwidth-prohibitive, so modern designs build the tree over version counters. Bonsai Merkle Trees [7] authenticate counters and store data MACs flat. To shrink the tree, split counter schemes [31] pack a shared major counter with many per-line minors: VAULT [8] raises tree arity with variable-arity nodes, Morphable Counters [9] adapt the counter encoding to overflow behavior, and Synergy [10] co-designs integrity with ECC storage; orthogonally, speculative-integrity techniques such as PoisonIvy [32] hide the verification latency behind speculation. This line has driven the metadata overhead of secure memory down substantially. Its blind spot is the *cipher*: the encryption/authentication primitive is treated as a fixed-latency black box, and the *physical* security of the engine that computes it is out of scope. PACE adopts the split counter idea but *unifies* the counter across three roles (counter-mode nonce, anti-replay version, and re-keying epoch) and opens the cipher to formal side-channel verification while keeping a tree-free on-chip operating point for embedded footprints.

18.2. Side-Channel Countermeasures for Cryptographic Hardware

Masking and threshold implementations. Differential power analysis [4] and its correlation variant [5] recover keys from data-dependent leakage; the dominant countermeasure is *masking*, which splits each sensitive variable into $d+1$ shares so that any d of them are independent of the secret [14,33]. Hardware masking must additionally resist *glitches*, which recombine shares combinatorially: threshold implementations [12] and domain-oriented masking (DOM) [13] enforce non-completeness and insert register barriers so that no glitch observes a full secret; consolidating-masking analyses [34] unified these families under a common security argument. These are gadget-level constructions, validated in isolation. They have rarely been *instantiated inside a memory encryption engine*, where the constraints differ sharply: the key is long-lived, the data is attacker-influenced,

and the engine operates continuously at a fixed physical location, the near-ideal DPA target that PACE hardens.

Leakage-resilient cryptography and re-keying. Rather than only masking a cipher, leakage-resilient designs *limit the data processed per key*; fresh re-keying [35] derives a session key per message, so the long-term key sees minimal data complexity. ISAP [16] builds authenticated encryption on the Ascon permutation [17] with a Goldreich–Goldwasser–Micali-style re-keying sponge that absorbs the nonce bit-by-bit, so the long-term key sees minimal data complexity. Ascon’s sponge structure and small state make it attractive for embedded hardware, and it is now the NIST lightweight cryptography standard [36]. PACE composes ISAP-style re-keying with DOM masking *specifically for the memory setting*: re-keying bounds the per-key data complexity to $q \approx 1$, so masking only need to protect the residual long-term key processing, the composition that makes n th-order resistance affordable on an engine, and which, to our knowledge, no prior memory encryption design has made.

18.3. Formal Verification of Masking

Because hand proofs of glitch-robust masking are error-prone, automated verifiers operate on the synthesized netlist. maskVerif [37] and SILVER [23] use reduced-ordered binary decision diagrams to decide probing, non-interference, and composability notions *exactly*; COCO [21] verifies probing security of a gate-level netlist co-designed with its control logic under a transient (glitch-aware) model formalized by the robust probing model [38]; PROLEAD [22] evaluates probing security *statistically* over massive simulation, scaling to large designs and high orders. These tools are mature but are *seldom applied to deployed memory encryption engines*; prior secure memory work reports no netlist-level masking verification. PACE is evaluated by three of them on the same synthesized S-box, with an explicit reconciliation of the statistical (PROLEAD) and exact (COCO, SILVER) verdicts through a multiple comparisons argument (Section 14); the probe-isolating non-interference framework [15] underpins the compositional step.

18.4. Open RISC-V Security and Full-System Evaluation

PACE is built in Chisel [39] on the Rocket [40] and BOOM [41] cores within Chipyard [42], the open RISC-V SoC framework, and evaluated with open tooling from RTL simulation to programmed FPGA silicon. For application-level performance we follow the methodology standard to the secure memory literature (an architectural simulator (gem5 [43]) with the engine’s latency *parameterized by our own RTL- and silicon-measured values*, exactly as VAULT, Morphable, and Synergy report their overheads) complemented by real-RTL runs of adopted RISC-V benchmarks (CoreMark [44], Dhrystone [45], Embench [46], and SPEC CPU2017 [47]) and an on-silicon measurement on a Zedboard; FPGA-accelerated full-system simulation (FireSim [48]) is the vehicle for the largest-scale runs. Against this backdrop, PACE does not contribute a new tree or gadget, instead providing the *first open engine that is cache-co-designed, formally side-channel-hardened, and evaluated from end to end*, filling the gap between closed commodity engines and point-solution academic prototypes. The remainder of this section makes the positioning concrete.

19. Discussion

Having presented the design and its evaluation, we step back to the rationale behind the main choices, the trade-offs they imply, and how the engine ports and deploys.

19.1. Why These Design Choices

Why below the last-level cache. Splicing the engine onto the memory bus *below* the LLC, rather than at the memory controller, is what enables PACE’s two defining behaviors.

First, the engine sees cache *residency*, so it can precompute a line's keystream while the line is still cached and spend only an XOR at eviction, the cache-anchored latency hiding that removes the cipher from the eviction critical path. Second, it attaches *below* the coherence point, so cache coherence is unaffected by construction; this is why the identical engine drops into single- and dual-core systems unchanged. The price is that the engine cannot see inner TileLink user fields, so the per-page policy cannot ride with the request and must instead be physical address-indexed through the control plane, a design trade to keep the engine below coherence.

Why page-adaptive? Uniform whole-memory encryption pays a worst-case cost on *every* access, yet most data is not sensitive. Page-adaptivity lets the operating system pay confidentiality, integrity, and masking cost only on the pages that need it, and run everything else through a zero-overhead plaintext bypass. This turns overhead from a fixed tax into a per-page choice, and it is the mechanism behind the favorable application-level numbers of Section 16: the cost concentrates exactly where the protection is asked for.

Why Ascon and a unified counter? Ascon's small 320-bit sponge state and single permutation let one masked primitive serve confidentiality, integrity, and re-keying such that the expensive masked gadget only needs to be instantiated once rather than replicated across a cipher and a separate MAC. Unifying the split counter across the counter-mode nonce, the anti-replay integrity version, and the re-keying epoch collapses what are three separate metadata structures in prior designs into one, which is what keeps both the metadata footprint and the masked engine area affordable.

Why layer re-keying and masking? Neither countermeasure suffices alone at a reasonable cost. High-order masking is expensive (area grows roughly quadratically in the share count) while re-keying alone does not stop a single-trace attack on the long-term key. Composing them is the point: re-keying bounds the data processed per session key to $q \approx 1$, so masking only needs to protect the residual long-term key processing; masking then raises the statistical order of whatever leakage remains. The analytic model (Section 13) quantifies why this composition, and not either half, is what makes n th-order resistance affordable on an embedded engine.

19.2. Trade-Offs and Operating Points

Tier-A versus Tier-B. At embedded footprints PACE keeps counters and tags entirely on-chip (tree-free Tier-A): zero DRAM metadata traffic, at the cost of on-chip SRAM proportional to the protected region. For all-memory protection it falls back to a DRAM-resident split counter Bonsai Merkle tree (Tier-B): constant on-chip state, at the cost of metadata read/write traffic that grows with tree depth (Section 8). Both tiers share the same counter and the same engine; the choice is a configuration knob sized to the protected footprint, not a redesign.

Masking order as an area and energy dial. The masking order $d \in \{1, 2, 3\}$ is a synthesis-time choice. Area grows with order (Section 17), but *application-level performance does not*: masking sits off the timing path and is cycle-identical to plain confidentiality (Section 16). A deployment therefore selects the order purely for its threat level, paying in area and energy rather than runtime, an unusually clean separation of the security dial from the performance budget.

19.3. Portability

The engine attaches to a standard TileLink memory bus edge and is agnostic to the core above it: we validate the *same* engine on the in-order Rocket and the out-of-order BOOM, and on single- and dual-core configurations, with no engine changes. Nothing in the datapath is Rocket-specific; porting to another RISC-V core, or to another architecture

with a comparable memory interface, requires only re-targeting the thin control-plane driver. The masked primitive and the three-tool formal flow are independent of the host entirely, so the security result travels with the engine rather than the platform.

19.4. Deployment Considerations

Two points bear consideration in real deployment. First, the on-chip ring oscillator power sensor we use for the leakage assessment campaign is a *research instrument* for self-measurement, not part of a shipped product; a production part would omit it (or gate it behind debug). Second, the security rests on assumptions A1–A5 (Section 3): keys provisioned on-die by a trusted boot flow, and a characterized entropy source behind the certified randomness frontend. The frontend (health-tested TRNG feeding the masking DRBG) is provided in RTL; the entropy *source* itself is platform-dependent and its min-entropy must be characterized on the target silicon to set the health test cutoffs. Neither point affects the architecture or the security argument, but both matter for a deployment, and we make them explicit rather than assume them away.

20. Limitations and Future Work

Out-of-model physical leakage. Our DPA *evidence* is pre-silicon: formal proofs under a glitch/transition-aware probing model and simulated traces under a noisy Hamming weight model. Physical devices exhibit out-of-model effects (coupling, non-linear leakage) that only a measurement platform can settle. We took the first step on silicon: an on-chip ring oscillator power sensor read over JTAG, with a fixed-vs-random TVLA campaign on the programmed engine (Section 15), but with an automatically placed sensor on a small (~ 1.2 k-LUT) engine the campaign sits at the statistical null floor ($|t|_{\max} \approx 3.3$), i.e., it lacks the sensitivity to observe leakage at all, and so cannot yet stage a physical break-then-fix. Closing this requires a sensor array pblocked adjacent to the engine datapath and a 10^5 – 10^6 -trace automated campaign; our offline pipeline, validated on synthetic ground truth, is ready for it. Noise in our model is defender-conservative, so a flat simulated TVLA is necessary, not sufficient, evidence; the formal layer carries the security claim.

Application-scale performance. Our application overhead evidence is strongest where it is most trustworthy (named kernels measured on the *actual* RTL, on both an in-order and an out-of-order core (Section 16)) and is extended to larger scales by a gem5 model whose per-access latency is taken from RTL-measured values. That model is a model: it captures the integrity latency faithfully but abstracts the memory system, and a full *cycle-exact* SPEC CPU2017 campaign is beyond bare-metal RTL simulation. FPGA-accelerated simulation (FireSim on a cloud FPGA) would run unmodified SPEC binaries cycle-exactly against the real PACE RTL; we identify this as the immediate next step, our RTL being FireSim-target compatible. On-silicon application overhead (a full soft SoC plus PACE booting a workload on the FPGA fabric) is likewise a bring-up step beyond the current engine-on-silicon validation and is bounded above by the eviction-bound regime we already measure (Section 16).

OS hardening. The kernel co-design (MAP_ENCRYPTED, sys_msecure, libpace) is functionally validated on QEMU and the hardware engagement on the Verilator SoC, but full in-kernel hardening of swap/copy-on-write/migration is future work. We do verify, in isolation, that the physical address binding makes naïve page relocation *fault safely*: moving a line's ciphertext to a new frame is rejected by the PA-bound MAC, so the kernel must decrypt/re-encrypt (or re-bind metadata) across a frame change rather than copy ciphertext.

Tier-B read traffic. The live block-MAC Tier-B is validated from end to end (chisel test, full Rocket and BOOM SoCs, and on the FPGA), and the reported per-access metadata

cost is exact and confirmed bit for bit on silicon (Section 8). Its one drawback is read bandwidth: verifying a line re-fetches a whole child block per tree level (no metadata cache), so per-access metadata traffic grows with depth; a small metadata cache (future work) would amortize the upper levels toward depth+2.

Randomness. The masking DRBG quality affects the empirical noise floor but not the probing model security. We provide the certified randomness frontend in RTL (Trng: SP 800-90B Repetition-Count and Adaptive-Proportion health tests gating a reseeded Trivium DRBG, Section 7); the residual caveat is that the entropy *source* itself (ring oscillator jitter) is platform-dependent and its min-entropy must be characterized on the target silicon to set the health test cutoffs.

Multi-core. Beyond the single-core SoC, we build and validate a dual-core configuration (PaceDualRocketConfig): two Rocket harts share one PACE engine on the mbus *below* the coherence point, so coherence is unaffected by construction. The dual-core Verilator build round trips 8192×2 words through encrypted DRAM with zero errors. Per-domain isolation (the property a shared engine relies on) is proven on the RTL primitives: distinct domains (different keys, or the same key with a different domain id in the nonce) yield independent keystreams, so a cross-domain decrypt recovers garbage, and distinct authentication keys yield distinct MAC tags, so a cross-domain read faults (Section 10). The residual future work is fine-grained QoS arbitration of the engine under heavy multi-core contention.

21. Conclusions

PACE shows that a page-adaptive, cache-anchored memory encryption engine for open RISC-V hardware can combine residency-precomputed CTR-mode confidentiality, tree-free Tier-A integrity with a live split counter block-MAC Bonsai Merkle scale-out, ISAP-style re-keying, and formally verified n th-order masking, and be evaluated from end to end with open-source tooling. The masked Ascon- p S-box is certified order- d secure by *three* independent glitch-aware tools, with COCO reaching third order ($d = 3$) and all three agreeing at $d = 1, 2$, and with the masking order's effect captured by an $N \propto \sigma^{2(d+1)}$ trace amplification law that we confirm empirically, and the engine is integrated and validated in RTL on a Rocket SoC (single- and dual-core) with a memory-mapped control plane and a Linux driver. We further realize the design on FPGA silicon (placed, routed, timing-closed, and programmed on an XC7Z020 with an on-chip power sensor read over JTAG); a physical fixed-vs-random TVLA campaign demonstrates the full measurement path but, with an auto-placed sensor on this small engine, remains below the leakage-detection threshold, leaving a co-placed sensor array and a large trace campaign as the concrete path to an on-silicon break-then-fix.

Four claims are new to our knowledge and distinguish PACE from a re-tuning of prior work (Section 1.3): it is the first open engine that is *simultaneously* cache-co-designed and formally end-to-end side-channel-hardened from RTL to silicon; cache-anchored keystream precompute is a new mechanism, enabled by placing the engine below the LLC, which removes the cipher from the eviction-critical path; a single unified split counter collapses the counter-mode nonce, the anti-replay version, and the re-keying epoch, which is what makes bounded-data re-keying and arbitrary-order masking affordable together; and the masked core is verified on the synthesized netlist by three independent tools and then demonstrated on real silicon. The performance consequence of the below-LLC placement is that these protections are, at the application level, near-free in the common case; overhead concentrates only under sustained encrypted streaming larger than the cache, exactly where the per-eviction cost is unavoidable, and masking adds security as an area and energy dial with no application-level throughput cost (Section 16). PACE is a reproducible design point,

from a single Chisel source through three formal tools to programmed silicon: side-channel-hardened memory encryption that is open, low in overhead, and independently verified. The remaining steps toward production deployment—chiefly, a co-placed sensor array for an on-silicon break-then-fix, a metadata cache for Tier-B read traffic, and full in-kernel hardening—are set out in Section 20.

Author Contributions: Conceptualization, J.M. and S.K.S.; methodology, J.M.; software, J.M., S.M. and A.P.; validation, J.M., S.M. and A.P.; formal analysis, J.M.; investigation, J.M. and S.M.; resources, S.K.S.; data curation, A.P.; writing of the original draft, J.M.; review and editing, S.K.S., S.M. and A.P.; visualization, J.M.; supervision, S.K.S.; project administration, J.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data and code that support this study will be made available upon request from the corresponding author.

Conflicts of Interest: Author Aman Pathak was employed by the company ENineHQ Technologies. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AES	Advanced Encryption Standard
BDD	Binary Decision Diagram
BMT	Bonsai Merkle Tree
BOOM	Berkeley Out-of-Order Machine
CPA	Correlation Power Analysis
CPI	Cycles Per Instruction
CSR	Control and Status Register
CTR	Counter (mode)
DOM	Domain-Oriented Masking
DPA	Differential Power Analysis
DRAM	Dynamic Random-Access Memory
DRBG	Deterministic Random Bit Generator
EM	Electromagnetic
FF	Flip-Flop
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
GGM	Goldreich–Goldwasser–Micali
HW	Hamming Weight
ISAP	Leakage-resilient authenticated encryption scheme
JTAG	Joint Test Action Group
KAT	Known-Answer Test
LLC	Last-Level Cache
LUT	Look-Up Table
MAC	Message Authentication Code
MI	Mutual Information
MMIO	Memory-Mapped Input/Output
NI	Non-Interference
OS	Operating System
PA	Physical Address
PACE	Page-Adaptive Cache-anchored Encryption
PINI	Probe-Isolating Non-Interference

PTE	Page Table Entry
QoS	Quality of Service
RO	Ring Oscillator
ROBDD	Reduced-Ordered Binary Decision Diagram
RTL	Register Transfer Level
SHA	Secure Hash Algorithm
SNI	Strong Non-Interference
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SRAM	Static Random-Access Memory
TRNG	True Random Number Generator
TVLA	Test Vector Leakage Assessment
XOR	Exclusive-OR

Appendix A. Known-Answer Tests

All cryptographic cores are validated bit for bit against a Python reference that reproduces the published Ascon KAT vectors. The folded Ascon- p matches p^{12} and p^6 on the zero and fixed test states; IsapEnc, IsapRk, and IsapMac match the ISAP-A-128A reference; and the masked permutation unmasks to the plain KAT for $d = 1, 2, 3$ under random share splits. Representative vectors (hexadecimal, 64-bit words $x_0 \dots x_4$ /keystream words):

```

Ascon- $p^{12}$ (0): 78ea7ae5cfebb108 9b9bfb8513b560f7
               6937f83e03d11a50 3fe53f36f2c1178c 045d648e4def12c9
Ascon- $p^6$  (0): 160c84f20faad4f1 21495b1b0ae33eef
               e0377d04e23a914b 2b23481598ffa8ea 649af379ba83cd30
IsapEnc ks    : c19af48a3d9f33fe aa656b2b4c2c1111
(K=0001..0f,  b4596d14514ba75e 4d6ef0375c030582
N=1011..1f)  68f1d8cd84ac08b2 285581c62b538abe
               720bb2d93566f545 3394e4f6b70c944b

```

The masked MaskedAsconP(d) ($d \in \{1, 2, 3\}$) and MaskedIsapEnc(d) ($d \in \{1, 2\}$) reproduce the p^{12}/p^6 and IsapEnc vectors above after unmasking.

Appendix B. DOM-Dependent Gadget

For $n = d+1$ shares, the masked AND of operands a, b first forms a sharing-of-zero z ($z_0 \dots z_{d-1}$ fresh, $z_d = \bigoplus_{i < d} z_i$), reshapes $b'_i = \text{Reg}(b_i \oplus z_i)$ and registers a , then computes the DOM-independent product of (a, b') : inner terms $\text{Reg}(a_i b'_i)$ and, for $i < j$, cross terms $\text{Reg}(a_i b'_j \oplus r_{ij})$ and $\text{Reg}(a_j b'_i \oplus r_{ij})$ with fresh r_{ij} . Correctness: $\sum_i b'_i = \sum_i b_i$ (since $\sum z_i = 0$) and the r_{ij} cancel pairwise, so $\bigoplus_i (\widehat{ab})_i = (\bigoplus a_i)(\bigoplus b_i)$. Security follows from operand independence after resharing; verified in Section 14.

Appendix C. Analytic Derivations

The traces-to-disclosure law (5) follows from the Fisher z-transform of the sample correlation; the masking amplification (7) from the MI bound (6). The empirical MI sweep (Section 13) estimates $\text{MI}(b; L_0 \dots L_d)$ by Monte Carlo over the 2^{d+1} -component Gaussian leakage mixture and fits the asymptotic slope of $1/\text{MI}$ versus σ .

Appendix D. Ascon Permutation Constants

The folded datapath implements the standard Ascon- p round: a constant addition to x_2 , the 5-bit substitution layer S applied bit-sliced across the five 64-bit words, and the linear diffusion Σ_i . The 5-bit S-box (used by the masked gadget of Section 7) is, in hexadecimal for inputs $0 \dots 31$:

```

4 b 1f 14 1a 15 9 2 1b 5 8 12 1d 3 6 1c
1e 13 7 e 0 d 11 18 10 c 1 19 16 a f 17

```

with diffusion $x_0 \oplus= (x_0 \ggg 19) \oplus (x_0 \ggg 28)$, $x_1 \oplus= (x_1 \ggg 61) \oplus (x_1 \ggg 39)$, $x_2 \oplus= (x_2 \ggg 1) \oplus (x_2 \ggg 6)$, $x_3 \oplus= (x_3 \ggg 10) \oplus (x_3 \ggg 17)$, $x_4 \oplus= (x_4 \ggg 7) \oplus (x_4 \ggg 41)$. The round constants are $0xf0 \dots 0x4b$ decremented by $0x0f$ per round; p^{12} uses the last twelve and p^6 the last six, selected at run time by the round counter.

Appendix E. Control-Plane Register Map

The complete `pace_*` MMIO map (offsets from `ctlBase`, 64-bit registers) is given in Table A1; configuration registers are latched only while `LOCK = 0`, after which the engine ignores writes until reset.

Table A1. Full PACE control-plane register map.

Offset	R/W	Meaning
0x00	RW	ENABLE – engine on/off for the region
0x08	RW	INTEGRITY – verify before release
0x10	RW	NODECRYPT – bring-up: forward ciphertext
0x18	RW	REKEY_MODE – 0 per-epoch, 1 per-write
0x20	W1S	LOCK – sticky freeze of config
0x28	RW	REGION_BASE – physical base
0x30	RW	REGION_SIZE – bytes
0x38	RW	KEYE0 – enc key [63:0]
0x40	RW	KEYE1 – enc key [127:64]
0x48	RW	KEYA0 – auth key [63:0]
0x50	RW	KEYA1 – auth key [127:64]
0x58	RO	STATUS – integrity-fault counter
0x60	RO	LASTFAULT – faulting line address
0x68	RO	CAP – $\{d, \text{ctrW}, \text{\#lines}\}$

Appendix F. Tier-B Walk and Re-Keying FSMs

Algorithm A1 is the deployed hardware verify-before-release validated from end to end in Section 8: it fetches the line counter, line MAC and ciphertext straight from DRAM (no metadata cache), and, for a written line, walks leaf→root recomputing each node’s tag from its *whole* child block, then compares the top tag to the single on-chip anchor (`rootTag`). A fresh line (counter 0) takes the fast path. The same Ascon-based MAC guards data and nodes.

Algorithm A1: Tier-B verified read (deployed PaceEngine)

```

Input: line address  $a$ , on-chip root tag  $R$ , keys  $K_E, K_A$ 
 $li \leftarrow$  line index of  $a$ ;
fetch counter  $ctr[li]$  and line MAC  $M[li]$  from DRAM;
fetch ciphertext line;  $pt \leftarrow \text{Dec}_{K_E}(a, ctr, \cdot)$ ;
if  $ctr = 0$  then return  $pt$ ; // fresh line: nothing written yet
;
if  $M[li] \neq \text{MAC}_{K_A}(a, ctr, ct)$  then fault;
 $h \leftarrow \perp$ ; // recomputed lower tag
for  $\ell \leftarrow 0$  to  $depth-1$  do
    fetch the whole child block  $B$  of  $li$ ’s level- $\ell$  node (counters if  $\ell=0$ , else node tags);
    substitute  $h$  for  $B$ ’s on-path child (for  $\ell > 0$ );
     $h \leftarrow \text{MAC}_{K_A}(\ell, \text{node\_idx}, B)$ ; // re-tag the node
if  $h \neq R$  then fault; // top tag must equal the on-chip anchor
;
return  $pt$ ;

```

Per-write re-keying derives a fresh sub-key by walking the GGM tree of Section 6 on the selector bits (the physical address and version), so the keystream/MAC under each write uses a key seen by at most $q \approx 1$ message block, the bound proved in Appendix G.

Appendix G. Proofs

Re-keying bound. Let a region see W writes between long-key uses. Per-write re-keying maps each write to a distinct leaf of the GGM tree keyed by $(\text{addr}, \text{ver})$; an adversary observing keystream/MAC under a leaf key sees it applied to a single counter value, so the per-key data complexity is $q = 1$ block. The long-term key K is used only as the GGM root, processed by the ISAP re-keying sponge whose rate-1 absorption bounds the long-key exposure independent of W ; hence the DPA trace count available per key is $O(1)$, defeating averaging (Section 13). $\square$

Order amplification. For a d -th order Boolean masking with i.i.d. Gaussian leakage of variance σ^2 per share, the MI between the secret and the $d+1$ share leakages scales as $\text{MI} \sim c \sigma^{-2(d+1)}$ in the high-noise limit (Prouff–Rivain). Since the number of traces to distinguish a fixed bias is inversely proportional to MI, the traces-to-disclosure scales as $N \propto \sigma^{2(d+1)}$; the measured exponents 1.92/3.70/5.40 for $d = 0/1/2$ (Section 15) match the predicted 2/4/6 within the Monte Carlo error. $\square$

Composability. The DOM-dependent S-box is NI and PINI but not SNI (Section 14.2); under the PINI composition theorem, a network of PINI gadgets with fresh per-gadget randomness is itself PINI, so the full masked permutation inherits glitch-robust order- d probing security, confirmed exactly by SILVER at $d = 1$ and for order-2 probing at $d = 2$. $\square$

References

1. AMD. AMD Memory Encryption. In *White Paper*; Advanced Micro Devices; AMD: Santa Clara, CA, USA, 2021.
2. Intel. *Intel Architecture Memory Encryption Technologies Specification*; Specification (Rev. 1.3), Document 336907; Intel: Santa Clara, CA, USA, 2021.
3. Arm. Arm Confidential Compute Architecture. In *White Paper*; Arm Ltd.: Cambridge, UK, 2021.
4. Kocher, P.; Jaffe, J.; Jun, B. Differential Power Analysis. In *Proceedings of the Advances in Cryptology—CRYPTO*; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1666, pp. 388–397. https://doi.org/10.1007/3-540-48405-1_25.
5. Mangard, S.; Oswald, E.; Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*; Springer: New York, NY, USA, 2007. <https://doi.org/10.1007/978-0-387-38162-6>.
6. Dworkin, M.J. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*; Technical Report SP 800-38A; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2001. <https://doi.org/10.6028/NIST.SP.800-38A>.
7. Rogers, B.; Chhabra, S.; Prvulovic, M.; Solihin, Y. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*; IEEE: New York, NY, USA, 2007; pp. 183–196. <https://doi.org/10.1109/MICRO.2007.16>.
8. Taassori, M.; Shafiee, A.; Balasubramonian, R. VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*; ASPLOS '18; ACM: New York, NY, USA, 2018; pp. 665–678. <https://doi.org/10.1145/3173162.3177155>.
9. Saileshwar, G.; Nair, P.J.; Ramrakhiani, P.; Elsasser, W.; Joao, J.A.; Qureshi, M.K. Morphable Counters: Enabling Compact Integrity Trees for Low-Overhead Secure Memories. In *Proceedings of the 51st Annual IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*; IEEE: New York, NY, USA, 2018; pp. 416–427. <https://doi.org/10.1109/MICRO.2018.00041>.
10. Saileshwar, G.; Nair, P.J.; Ramrakhiani, P.; Elsasser, W.; Qureshi, M.K. SYNERGY: Rethinking Secure-Memory Design for Error-Correcting Memories. In *Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*; IEEE: New York, NY, USA, 2018; pp. 454–465. <https://doi.org/10.1109/HPCA.2018.00046>.
11. Goodwill, G.; Jun, B.; Jaffe, J.; Rohatgi, P. A Testing Methodology for Side-Channel Resistance Validation. In *Proceedings of the NIST Non-Invasive Attack Testing Workshop*; NIST: Gaithersburg, MD, USA, 2011.
12. Nikova, S.; Rechberger, C.; Rijmen, V. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Proceedings of the Information and Communications Security (ICICS)*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4307, pp. 529–545. https://doi.org/10.1007/11935308_38.
13. Groß, H.; Mangard, S.; Korak, T. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Proceedings of the ACM Workshop on Theory of Implementation Security (TIS@CCS)*; ACM: New York, NY, USA, 2016. <https://doi.org/10.1145/2996366.2996426>.
14. Ishai, Y.; Sahai, A.; Wagner, D. Private Circuits: Securing Hardware against Probing Attacks. In *Proceedings of the Advances in Cryptology—CRYPTO*; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2729, pp. 463–481. https://doi.org/10.1007/978-3-540-45146-4_27.

15. Cassiers, G.; Standaert, F.X. Trivially and Efficiently Composing Masked Gadgets with Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 2542–2555. <https://doi.org/10.1109/TIFS.2020.2971153>.
16. Dobraunig, C.; Eichlseder, M.; Mangard, S.; Mendel, F.; Mennink, B.; Primas, R.; Unterluggauer, T. ISAP v2.0. *IACR Trans. Symmetric Cryptol.* **2020**, 2020, 390–416. <https://doi.org/10.46586/tosc.v2020.iS1.390-416>.
17. Dobraunig, C.; Eichlseder, M.; Mendel, F.; Schl  ffer, M. Ascon v1.2: Lightweight Authenticated Encryption and Hashing. *J. Cryptol.* **2021**, *34*, 33. <https://doi.org/10.1007/s00145-021-09398-9>.
18. De Canni  re, C.; Preneel, B. Trivium. In *Proceedings of the New Stream Cipher Designs—The eSTREAM Finalists*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 4986, pp. 244–266. https://doi.org/10.1007/978-3-540-68351-3_18.
19. Izraelevitz, A.; Koenig, J.; Li, P.; Lin, R.; Wang, A.; Magyar, A.; Kim, D.; Schmidt, C.; Markley, C.; Lawson, J.; et al. Reusability is FIRRTL Ground: Hardware Construction Languages, Compiler Frameworks, and Transformations. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*; IEEE: New York, NY, USA, 2017; pp. 209–216. <https://doi.org/10.1109/ICCAD.2017.8203780>.
20. Wolf, C. Yosys Open SYNthesis Suite. 2016. Available online: <https://yosyshq.net/yosys/> (accessed on 6 July 2026).
21. Gigerl, B.; Had  zi  , V.; Primas, R.; Mangard, S.; Bloem, R. Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. In *Proceedings of the 30th USENIX Security Symposium*; USENIX: Berkeley, CA, USA, 2021.
22. M  ller, N.; Moradi, A. PROLEAD: A Probing-Based Hardware Leakage Detection Tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**, 2022, 311–348. <https://doi.org/10.46586/tches.v2022.i4.311-348>.
23. Knichel, D.; Sasdrich, P.; Moradi, A. SILVER—Statistical Independence and Leakage Verification. In *Proceedings of the Advances in Cryptology—ASIACRYPT*; Springer: Berlin/Heidelberg, Germany, 2020, Volume 12491, pp. 787–816. https://doi.org/10.1007/978-3-030-64837-4_26.
24. Prouff, E.; Rivain, M. Masking against Side-Channel Attacks: A Formal Security Proof. In *Proceedings of the Advances in Cryptology—EUROCRYPT*; Springer: Berlin/Heidelberg, Germany, 2013, Volume 7881, pp. 142–159. https://doi.org/10.1007/978-3-642-38348-9_9.
25. Duc, A.; Faust, S.; Standaert, F.X. Making Masking Security Proofs Concrete: Or How to Evaluate the Security of Any Leaking Device. In *Proceedings of the Advances in Cryptology—EUROCRYPT*; Springer: Berlin/Heidelberg, Germany, 2015, Volume 9056, pp. 401–429. https://doi.org/10.1007/978-3-662-46800-5_16.
26. Lie, D.; Thekkath, C.; Mitchell, M.; Lincoln, P.; Boneh, D.; Mitchell, J.; Horowitz, M. Architectural Support for Copy and Tamper Resistant Software. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*; ACM: New York, NY, USA, 2000; pp. 168–177. <https://doi.org/10.1145/378993.379237>.
27. Suh, G.E.; Clarke, D.; Gassend, B.; van Dijk, M.; Devadas, S. AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS)*; ACM: New York, NY, USA, 2003; pp. 160–171. <https://doi.org/10.1145/782814.782838>.
28. Gassend, B.; Suh, G.E.; Clarke, D.; van Dijk, M.; Devadas, S. Caches and Hash Trees for Efficient Memory Integrity Verification. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA)*; IEEE: New York, NY, USA, 2003; pp. 295–306. <https://doi.org/10.1109/HPCA.2003.1183547>.
29. Gueron, S. *A Memory Encryption Engine Suitable for General Purpose Processors*; Paper 2016/204; Cryptology ePrint Archive: New York, NY, USA, 2016.
30. Costan, V.; Devadas, S. *Intel SGX Explained*; Paper 2016/086; Cryptology ePrint Archive: New York, NY, USA, 2016.
31. Yan, C.; Engender, D.; Prvulovic, M.; Rogers, B.; Solihin, Y. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*; IEEE: New York, NY, USA, 2006; pp. 179–190. <https://doi.org/10.1109/ISCA.2006.22>.
32. Lehman, T.S.; Hilton, A.D.; Lee, B.C. PoisonIvy: Safe Speculation for Secure Memory. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*; IEEE: New York, NY, USA, 2016; pp. 1–13. <https://doi.org/10.1109/MICRO.2016.7783741>.
33. Chari, S.; Jutla, C.S.; Rao, J.R.; Rohatgi, P. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Proceedings of the Advances in Cryptology—CRYPTO*; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1666, pp. 398–412. https://doi.org/10.1007/3-540-48405-1_26.
34. Reparaz, O.; Bilgin, B.; Nikova, S.; Gierlichs, B.; Verbauwhede, I. Consolidating Masking Schemes. In *Proceedings of the Advances in Cryptology—CRYPTO*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9215, pp. 764–783. https://doi.org/10.1007/978-3-662-47989-6_37.
35. Medwed, M.; Standaert, F.X.; Gro  sch  dl, J.; Regazzoni, F. Fresh Re-Keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices. In *Proceedings of the Progress in Cryptology—AFRICACRYPT*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6055, pp. 279–296. https://doi.org/10.1007/978-3-642-12678-9_17.

36. Turan, M.S.; McKay, K.; Chang, D.; Bassham, L.E.; Kang, J.; Waller, N.D.; Kelsey, J.M.; Hong, D. *Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process*; Technical Report NIST IR 8454; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2023. <https://doi.org/10.6028/NIST.IR.8454>.
37. Barthe, G.; Belaïd, S.; Cassiers, G.; Fouque, P.A.; Grégoire, B.; Standaert, F.X. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11735, pp. 300–318. https://doi.org/10.1007/978-3-030-29959-0_15.
38. Faust, S.; Grosso, V.; Merino Del Pozo, S.; Paglialonga, C.; Standaert, F.X. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 89–120. <https://doi.org/10.46586/tches.v2018.i3.89-120>.
39. Bachrach, J.; Vo, H.; Richards, B.; Lee, Y.; Waterman, A.; Avižienis, R.; Wawrzynek, J.; Asanović, K. Chisel: Constructing Hardware in a Scala Embedded Language. In *Proceedings of the 49th Annual Design Automation Conference (DAC)*; ACM: New York, NY, USA, 2012; pp. 1216–1225. <https://doi.org/10.1145/2228360.2228584>.
40. Asanović, K.; Avižienis, R.; Bachrach, J.; Beamer, S.; Biancolin, D.; Celio, C.; Cook, H.; Dabbelt, D.; Hauser, J.; Izraelevitz, A.; et al. *The Rocket Chip Generator*; Technical Report UCB/EECS-2016-17; EECS Department, University of California: Berkeley, CA, USA, 2016.
41. Zhao, J.; Korpan, B.; Gonzalez, A.; Asanović, K. SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine. In *Proceedings of the Fourth Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2020.
42. Amid, A.; Biancolin, D.; Gonzalez, A.; Grubb, D.; Karandikar, S.; Liew, H.; Magyar, A.; Mao, H.; Ou, A.; Pemberton, N.; et al. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* **2020**, *40*, 10–21. <https://doi.org/10.1109/MM.2020.2996616>.
43. Binkert, N.; Beckmann, B.; Black, G.; Reinhardt, S.K.; Saidi, A.; Basu, A.; Hestness, J.; Hower, D.R.; Krishna, T.; Sardashti, S.; et al. The gem5 Simulator. *ACM SIGARCH Comput. Archit. News* **2011**, *39*, 1–7. <https://doi.org/10.1145/2024716.2024718>.
44. EEMBC. CoreMark: An EEMBC Benchmark. 2009. Available online: <https://www.eembc.org/coremark/> (accessed on 6 July 2026).
45. Weicker, R.P. Dhrystone: A Synthetic Systems Programming Benchmark. *Commun. ACM* **1984**, *27*, 1013–1030. <https://doi.org/10.1145/358274.358283>.
46. Bennett, J.; Dabbelt, P.; Garlati, C.; Madhusudan, G.S.; Mudge, T.; Patterson, D. Embench: An Evolving Benchmark Suite for Embedded IoT Computers. 2019. Available online: <https://www.embench.org/> (accessed on 6 July 2026).
47. Bucek, J.; Lange, K.D.; von Kistowski, J. SPEC CPU2017: Next-Generation Compute Benchmark. In *Proceedings of the Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE)*; ACM: New York, NY, USA, 2018; pp. 41–42. <https://doi.org/10.1145/3185768.3185771>.
48. Karandikar, S.; Mao, H.; Kim, D.; Biancolin, D.; Amid, A.; Lee, D.; Pemberton, N.; Amaro, E.; Schmidt, C.; Chopra, A.; et al. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*; ACM: New York, NY, USA, 2018; pp. 29–42. <https://doi.org/10.1109/ISCA.2018.00014>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.