ACSE RESEARCH SERIES · WP-02

PME — Engineering the Polymorphic Mutation Engine

Implementation architecture of the ACSE reference engine: five subsystems, five production TEE adapters, and the 3-line integration API

Author: Arul Raj · Patent IN202641070690 · June 2026

Primary Audience: Security Engineers · Implementation Teams · Deployment Architects · Classification: Public Technical Paper

ABSTRACT

The Polymorphic Mutation Engine (PME) is the production reference implementation of the ACSE architecture. This paper describes the engineering decisions behind its five core subsystems, the 3-line integration API that exposes all mutation capabilities to any Rust application, the complete TEE adapter stack — five adapters across all major hardware-rooted security environments, all production-ready — and the testing and benchmarking methodology that validates the Kali Invariant across 683 automated tests and six red-team scenarios.

PME is implemented in Rust. The choice is not incidental: Rust's ownership model eliminates the entire class of memory safety vulnerabilities — buffer overflows, use-after-free, dangling pointers — that have been the root cause of the majority of critical security vulnerabilities in C/C++ security implementations. Zero unsafe blocks exist in the engine core.

1. SYSTEM ARCHITECTURE

1.1 The five subsystem model

PME comprises five co-operating subsystems, orchestrated by MutationEngineCore (KaliCore). Each subsystem has a single responsibility and a clean interface. No subsystem bypasses another. The architecture is designed so that any subsystem can be independently replaced or upgraded without touching the others.

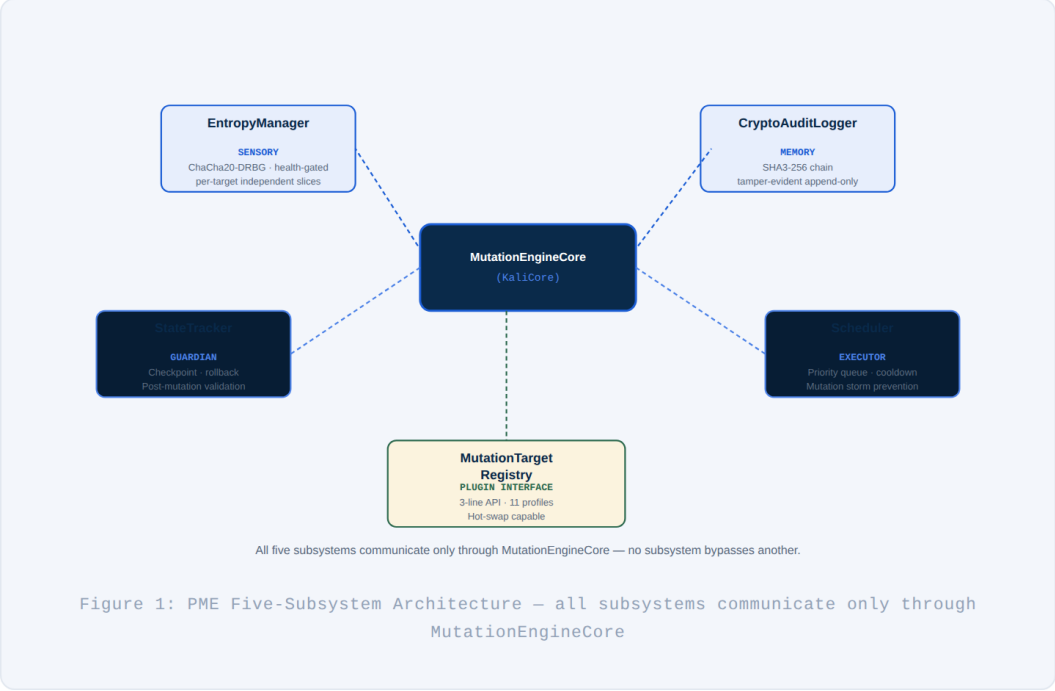

Figure 1: PME Five-Subsystem Architecture — all subsystems communicate only through MutationEngineCore

Figure 1: PME Five-Subsystem Architecture — all communication through MutationEngineCore

Subsystem	Responsibility	Key Guarantee
EntropyManager	Supplies high-quality entropy to each mutation cycle	Statistical independence across all registered targets
CryptoAuditLogger	Maintains SHA3-256 chained mutation history	Tamper-evidence: any modification invalidates all subsequent records
StateTracker	Cryptographic checkpointing and atomic rollback	Partial mutations cannot occur — either complete or roll back fully
Scheduler	Trigger management, priority queuing, cooldown enforcement	Kali Invariant cannot be starved or bypassed by scheduling
MutationTarget Registry	Plugin interface for all 11 mutation profiles	Profile swap requires changing one word — no application code changes

1.2 MutationEngineCore — KaliCore

MutationEngineCore is the governing intelligence that wires the five subsystems together and drives the autonomous state transitions between Sachs (steady-state), Hunter (elevated threat), and Main Organ (full response) modes. It owns the EWMA anomaly scorer that computes threat level from incoming signals and decides when to escalate mutation aggression.

KaliCore is not a scheduler. It is a decision system. The distinction matters: a scheduler executes instructions; KaliCore evaluates threat context and adapts mutation behaviour accordingly. An authenticated anomaly signal from a SIEM, a peer node via ASMP, or a physical sensor changes KaliCore's cognitive state — not just its timing.

2. ENTROPYMANAGER

2.1 ChaCha20-DRBG entropy pool

The EntropyManager maintains a ChaCha20-DRBG entropy pool seeded from the OS CSPRNG at initialisation and periodically reseeded. ChaCha20-DRBG is chosen over CTR-DRBG for its cache-timing side-channel resistance — a property that matters when PME is deployed on hardware that may be partially observable by an attacker.

The entropy pool is health-gated: if the pool fails its health tests (NIST SP 800-90B continuous health tests), entropy supply is suspended and the scheduler receives a degraded-mode signal. Mutation continues at reduced rate using a fallback entropy source rather than failing open.

2.2 Per-target entropy slicing

Multiple registered mutation targets share the same entropy pool but receive statistically independent entropy slices. Each slice is derived as:

```
entropy_for_target = SHA3-256(entropy_bundle || target_id || cycle_seq)
```

SHA3-256 preimage resistance guarantees that the entropy slices for any two targets are computationally independent even if the attacker observes the slices for one target. Compromising one target's mutation inputs reveals nothing about any other target's inputs.

3. CRYPTOAUDITLOGGER

3.1 SHA3-256 chain construction

The CryptoAuditLogger maintains an append-only cryptographically chained log of every mutation event. Each record contains:

- Record identifier (UUID v4)
- Timestamp (Unix nanoseconds)
- Profile identifier and target identifier
- Fingerprint before mutation (SHA3-256 of pre-mutation surface state)
- Fingerprint after mutation (SHA3-256 of post-mutation surface state)
- Entropy hash (SHA3-256 of the entropy input — not the entropy itself)
- Trigger reason (PerInvocation / Timer / Anomaly / PeerSignal / Manual)
- TEE attestation quote (from the active TEE adapter)
- Chain hash: SHA3-256(prev_chain_hash || all record fields)

The chain hash of each record incorporates all preceding records. Chain verification is O(N) — a single forward pass recomputes all chain hashes and compares to stored values. Any modification to any record changes its chain hash, invalidating all subsequent records. This property is the audit tamper-evidence guarantee.

3.2 TEE-bound records

In production deployment with a hardware TEE adapter, each audit record carries a TEE attestation quote that binds the record to the hardware platform that produced it. A log record that cannot be verified against the expected TEE attestation is treated as suspect, regardless of chain hash validity. This prevents an adversary from substituting a complete fake audit log — even one with a valid internal chain — generated on different hardware.

4. STATETRACKER

4.1 Cryptographic checkpointing

Before every mutation, the StateTracker creates a cryptographic checkpoint of the current surface state. The checkpoint comprises the current fingerprint, the current chain hash, the cycle sequence number, and a TEE-attested timestamp. The checkpoint is the rollback target if the mutation fails validation.

4.2 Atomic rollback

If Phase 3 (VALIDATE) of the atomic mutation cycle fails — because the new fingerprint is too close to the previous (Hamming distance < 128 bits) or because profile-specific validation logic rejects the result — the StateTracker rolls back to the checkpoint atomically. The surface returns to its pre-mutation state. The audit log receives a ROLLBACK record documenting the failure reason. The Kali Invariant is not violated: a failed mutation does not produce a new surface state.

4.3 Post-mutation invariant validation

After every successful mutation, the StateTracker performs three invariant checks:

(a) Fingerprint change: assert fp_after != fp_before

(b) Hamming distance: assert hamming(fp_after, fp_before) >= 128 bits

(c) Profile-specific validity: assert target.validate(fp_after) == true

All three must pass. A single failure triggers rollback. The invariant validation is the enforcement point of the Kali Invariant — it is where "no surface survives an access cycle unchanged" is computationally verified on every cycle.

5. SCHEDULER

5.1 Trigger taxonomy

The Scheduler processes five trigger types, each with different priority and cooldown semantics:

Trigger	Source	Priority	Cooldown
PerInvocation	Application API call	Normal	None — every invocation mutates
Timer	Internal interval scheduler	Normal	Configurable per profile
Anomaly	EWMA score threshold breach	High	Short — allows rapid escalation

PeerSignal	ASMP-MSG-003 from peer node	High	Short — estate coordination
Manual	Direct API / management console	Highest	None — operator override

5.2 Mutation storm prevention

If multiple high-priority triggers arrive simultaneously — for example, anomaly signals from multiple ASMP peers during a coordinated attack — the Scheduler prevents mutation storms through burst coalescing: multiple triggers within a configurable window are coalesced into a single mutation cycle that processes all signals. The Kali Invariant is still satisfied (one complete mutation occurs), but the engine does not thrash.

6. MUTATIONTARGET REGISTRY AND 3-LINE API

6.1 Plugin architecture

Every mutation profile — all 11 of them — is registered as a plugin through the MutationTarget Registry. The registry provides a single trait interface that all profiles implement. Adding a new profile to a running PME instance, or replacing one profile with another, requires changing a single registration call with no modifications to application code.

6.2 The 3-line integration API

Any Rust application gains full ACSE protection — entropy management, audit logging, checkpointing, scheduling, and hardware TEE attestation — with three lines:

```
let mut engine = MutationEngineCore::new(tee_adapter);
// Line 1: engine + TEE
engine.register(Box::new(SquidShieldTarget::new("payments")), "fin");  // Line 2: register profile
engine.trigger_mutation(TriggerReason::PerInvocation, None);            // Line 3: mutate
```

Swapping profiles or adding multiple profiles requires changing only the registration call:

```
// Finance + healthcare simultaneously
engine.register(Box::new(SquidShieldTarget::new("txn")), "fin");
engine.register(Box::new(GlassFrogTarget::new("phi")), "hipaa");
// KaliCoreTarget: one trigger fires all registered profiles
engine.register(Box::new(KaliCoreTarget::new()), "all");
```

7. TEE ADAPTER STACK — ALL FIVE PRODUCTION-READY

PME's TEE adapter layer abstracts hardware attestation behind a single trait, allowing the engine to produce hardware-bound audit records and management attestation tokens on any TEE-capable platform. All five adapters are production-ready:

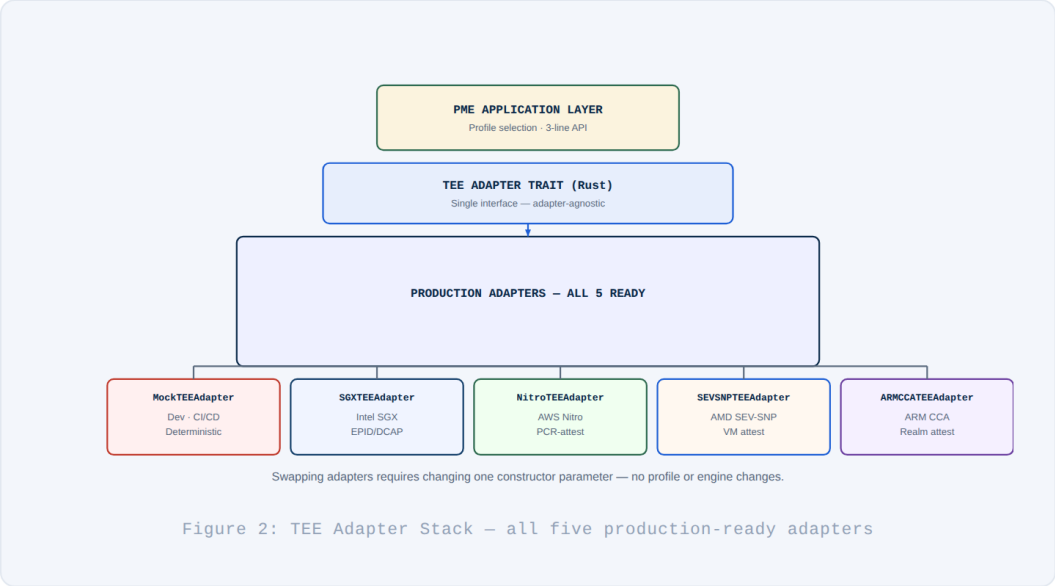

Figure 2: TEE Adapter Stack — all five production-ready adapters (Mock, SGX, Nitro, SEV-SNP, ARM CCA)

Adapter	Platform	Attestation Mechanism	Use Case
MockTEEAdapter	Any platform	Deterministic mock quotes (fixed pattern)	Development, testing, CI/CD pipelines
SGXTEEAdapter	Intel SGX (x86_64)	EPID/DCAP remote attestation, enclave measurement	On-premise enterprise, classified environments
NitroTEEAdapter	AWS Nitro Enclaves	PCR-based attestation, enclave document	AWS cloud deployments, managed services
SEVSNPTEEAdapter	AMD SEV-SNP	VM-level hardware attestation, VCEK certificate chain	AMD-based cloud, on-premise AMD infrastructure
ARMCCATEEAdapter	ARM CCA (Realm)	Realm attestation token, platform token	ARM-based cloud instances, mobile-adjacent edge

The adapter interface is a single Rust trait. Switching from MockTEEAdapter to SGXTEEAdapter in a production deployment is a single parameter change at engine construction time. The mutation logic, audit chain, and all profiles are identical across adapters — only the attestation quote format differs.

PRODUCTION TEE COVERAGE

The five-adapter stack covers every major enterprise TEE environment: Intel SGX for on-premise x86 deployments, AWS Nitro for cloud-native workloads, AMD SEV-SNP for AMD infrastructure, ARM CCA for ARM-based edge and cloud instances, and MockTEEAdapter for development and CI/CD pipelines. An

organisation deploying PME on any of these platforms has a production-ready, hardware-attested audit chain out of the box.

8. TESTING AND BENCHMARKING METHODOLOGY

8.1 Test suite

The PME test suite comprises 683 automated tests across all 11 profiles and the management console. Every test runs in CI with zero tolerance for failure or warning. The test categories are:

- **Unit tests:** Isolated subsystem tests covering every code path in EntropyManager, CryptoAuditLogger, StateTracker, Scheduler, and MutationTarget Registry
- **Profile tests:** Per-profile tests verifying Kali Invariant enforcement, fingerprint uniqueness, rollback correctness, and domain-specific invariants
- **Integration tests:** End-to-end mutation cycles verifying the complete 4-phase cycle across all profiles and all TEE adapters
- **Chain verification tests:** Audit chain integrity verification — insert, tamper, verify-failure, rollback, re-verify
- **Red team simulation:** 6 automated red team scenarios including linkability attack simulation and Forced Twitch detection validation

8.2 Criterion benchmarking

All performance numbers in PME documentation are Criterion p50 medians: 100 samples per measurement, 3-second warmup, --release build, on commodity x86_64 hardware. Criterion provides statistically rigorous benchmarks with outlier detection and regression tracking. Numbers are reproducible:

```
cargo bench --bench mutation_benchmark
```

9. PME-CONSOLE MANAGEMENT INTERFACE

The pme-console is a full-featured Actix-Web management interface (default port 8888) that provides real-time visibility into all PME subsystems. It is deployed as a standalone binary alongside the engine and communicates through the PME API.

The console provides six operational tabs:

- **Profile Dashboard:** Real-time mutation rate, fingerprint history, and Kali Invariant confirmation for each registered profile
- **Mutation History:** Scrollable audit chain viewer with before/after fingerprints, entropy hashes, and TEE attestation status per record
- **Anomaly Telemetry:** EWMA score visualisation, state machine status (Sachs/Hunter/Main Organ), and trigger event log
- **ASMP Peer Network:** Connected peer status, Defensive Leap events, and anomaly signal propagation log
- **KaliCoreTarget Control:** One-click estate-wide rotation trigger with real-time confirmation across all registered profiles

- **Demo Harness (Tab 6):** 11-step dual-panel attack simulation for CERT-In and enterprise evaluation — demonstrates attacker's view vs. defender's view in real time

10. CONCLUSION

PME demonstrates that continuous surface mutation at cryptographic granularity is engineerable at production scale. Sub-millisecond mutation latency, atomic rollback, hardware-rooted attestation across five TEE environments, and a 3-line integration API collectively deliver the Kali Invariant as a deployable property rather than a theoretical one.

The 683-test suite and formal Criterion benchmarks provide the evidence base for the claims in this paper. All numbers are reproducible from the public repository at commit reference d538fa6 (benchmark baseline); the test suite reflects current HEAD.