Coprime-Factor Security Architecture — Blueprint v0.1

Purpose. Define a defense-in-depth model where each protective layer is engineered to be *mathematically independent* from the others—analogous to coprime moduli. The goal is to minimize common-mode failures and make successful compromise require simultaneous defeat of multiple, pairwise-independent layers.

---

1) Executive Summary

- **Problem:** Conventional defense-in-depth often hides correlated risks (same crypto family, same supply chain, same control plane). One bug or key leak can pierce multiple layers.
- **Idea:** Engineer layers as **coprime factors**—each using disjoint primitives, trust roots, vendors, failure modes, and ops paths—so that compromise probabilities multiply instead of correlate.
- **Outcome:** Lower breach probability, provable separation of concerns, auditable independence, graceful degradation.

---

2) Principles & Independence Criteria

Design layers so that for any pair (Li, Lj): - **Cryptographic independence:** different algorithm *families* (e.g., AES-GCM ↔ ChaCha20-Poly1305; Ed25519 ↔ P-256; RSA ↔ ECDSA) and distinct RNG sources. - **Trust root independence:** separate CAs/KMS/HSMs (ideally different vendors + firmware trees). - **Codebase independence:** different libraries, compilers, languages; no shared critical deps (e.g., not both OpenSSL). - **Runtime independence:** different isolation tech (VM ↔ container ↔ enclave/microVM), different kernels when feasible. - **Ops path independence:** separate teams, approval flows, credentials, monitoring stacks. - **Supply-chain independence:** distinct artifact pipelines, signing roots, registries. - **Control-plane independence:** policy engines and orchestration stacks that cannot impersonate each other.

> **Independence Rule:** If any two layers share a single catastrophic common dependency, they are *not* coprime.

---

3) Formal Model (Concise)

Let (L_1, ..., L_k) be layers; (C_i) is event “Layer i compromised” with probability (p_i). We target **approximate independence**: (P(C_i C_j) P(C_i)P(C_j)) for all (ij). Then overall protection event (C = _i C_i) yields breach probability (P(C) - _i (1- p_i)). Independence is enforced via the criteria above; audits measure residual correlation.

Independence Score (IS): For each dimension d ∈ {crypto, trust-root, codebase, runtime, ops, supply, control}, assign (w_d). For layer pair (i,j), set (s_{d}^{(i,j)}) (independent or not). Then [IS = _{i<j}*d w_d, s*{d}^{(i,j)}.] Target **IS ≥ 0.85** before production.

---

4) Reference Architecture (High Level)

User ↔ Edge ↔ App Plane ↔ Data Plane ↔ Control Plane ↔ Audit Plane

Layer A — Identity & Access (Coprime AuthN): - FIDO2/WebAuthn (ECDSA P-256) on vendor HSM-A - PLUS independent TOTP/HOTP (HMAC-SHA1/256) or passkey on HSM-B - Optional biometric check locally; never elevates trust by itself

Layer B — Transport & Session: - mTLS (TLS 1.3, library L1) *and* secondary WireGuard tunnel (library L2) for admin paths - Different cipher families (AES-GCM vs ChaCha20-Poly1305)

Layer C — Authorization (Dual Policy Consensus): - OPA/Rego engine and AWS Cedar (or equivalent) run in separate control planes - High-risk actions require **AND-consensus**; low-risk allow OR with rate-cap

Layer D — Data At Rest (Dual Encryption): - Envelope encryption with AES-GCM (KMS-A) - Nested layer with ChaCha20-Poly1305 (KMS-B) - Keys held in HSMs from different vendors with distinct firmware

Layer E — Integrity & Build Trust: - Provenance via Sigstore (Fulcio/Rekor) + separate in-house CA - Reproducible builds verified by two independent verifiers

Layer F — Observability & Audit: - Append-only Merkle log (e.g., Trillian) inside org - Daily external anchoring (public transparency log or blockchain) with separate signer

Layer G — Runtime Isolation: - Service set S1 on microVM/Firecracker; S2 on containers with SELinux/AppArmor; disjoint kernels when possible

---

5) Pattern Catalogue (Implementable)

P1. Dual-Primitive Encryption (DPE): - Apply two independent AEAD schemes with unrelated keys from independent KMS/HSM - Decrypt path requires both layers; failure leaves data safe

P2. Dual Policy Consensus (DPC): - Two policy engines evaluate requests; decision = AND for privileged ops - Engines have separate repos, CI, and deploy channels

P3. Split-Vendor Key Ceremony (SVK): - Root keys generated in two vendor HSMs, different RNGs; quorum requires both

P4. Divergent Transport (DT): - Administrative channels traverse a second, distinct VPN/overlay using disjoint crypto

P5. Twin Attestation (TA): - Workload must present attestation from *two* independent roots (e.g., TPM quote + SEV-SNP/TEE report)

---

6) Example Config (Pseudo-YAML)

```yaml
authn:
  factors:
    - type: fido2
      curve: p256
      hsm: vendorA
    - type: totp
      algo: hmac-sha1
      hsm: vendorB
  require: AND

transport:
  primary_tls:
    lib: rustls
    aead: aes-256-gcm
  admin_overlay:
    type: wireguard
    aead: chacha20-poly1305

authorization:
  engines:
    - name: opa
      source_repo: git://corp/opa-policies
    - name: cedar
      source_repo: git://corp/cedar-policies
  decision: AND

data_at_rest:
  layer1:
    aead: aes-256-gcm
    kms: KMS-A
  layer2:
    aead: chacha20-poly1305
    kms: KMS-B

attestation:
  require:
    - sigstore-fulcio
    - org-ca
```

---

7) Threat Model & Common-Mode Kill Switches

Adversaries: APT with supply-chain access; insider with privileged creds; cryptographic downgrade attacks; cloud control-plane compromise.

Common-Mode Risks & Mitigations: - *Shared library vuln (e.g., OpenSSL)* → Use different TLS stacks (BoringSSL vs rustls), different AEADs. - *Single CA/KMS breach* → Split vendors/roots; rotate quorum. - *Cloud provider control-plane bug* → Cross-account + cross-region + shadow control plane. - *Policy engine bug* → Dual consensus with canary deny on anomaly.

Kill Switches: - Per-layer hard failure defaults to *deny* for privileged ops - Emergency policy: force AND→OR only via out-of-band M-of-N key ceremony with board-level approval

---

8) Verification & Testing

- **Tabletop & Red Team:** Simulate single-layer failures; verify others hold.
- **Fault Injection:** Disable one HSM or policy engine; ensure system continues with reduced capability.
- **Synthetic Compromise Runs:** Introduce signed but invalid artifact to test twin attestation.
- **Metrics:** see §9.

---

9) Metrics & SLOs

- **Independence Score (IS):** ≥ 0.85 pre-prod; ≥ 0.9 for Tier-1 systems.
- **Common-Mode Risk Index (CMRI):** measured via shared-dep graph; target ≤ 0.1.
- **Attack Path Reduction (APR):** ≥ 70% fewer valid single-path exploits vs baseline.
- **Dual Decision Coverage:** ≥ 99% of privileged actions gated by DPC.
- **Key Diversity Index:** ≥ 2 vendors, ≥ 2 RNG classes, ≥ 2 crypto families in use.

---

10) Rollout Plan

Phase 0 (2–3 weeks): Dependency graphing, IS/CMRI baseline, choose dual primitives, pick vendors.

Phase 1 (4–6 weeks): Implement P1–P3 for a critical service; wire metrics; run failure drills.

Phase 2 (6–10 weeks): Extend to transport (P4) and twin attestation (P5); introduce automated audits.

Phase 3 (Quarterly): Expand to all Tier-1 services; enforce org policy: any new high-risk control must be coprime with existing stack.

---

11) Governance & Audit

- Quarterly **Independence Review** with sign-off on IS/CMRI.
- SBOM and provenance checks from *two* independent verifiers.
- External anchoring of logs; third-party attest review.

---

12) Limitations & Tradeoffs

- Complexity and latency overhead; dual encryption and consensus add cost.
- Operational burden: two stacks to maintain; require clear runbooks.
- Some domains can't easily achieve runtime independence (e.g., single cloud kernel).

Mitigate via scoping (Tiered application), automation, and clear exemption process with compensating controls.

---

13) Visuals (Placeholders)

- **Layered Stack Diagram:** user→edge→app→data with dual paths highlighted.
- **Independence Matrix:** rows=layers, cols=dimensions (crypto/trust/code/runtime/ops/supply/control), heatmap of independence.

---

14) Appendix: Coprime Analogy

- In number theory, if factors are **coprime**, divisibility properties compose cleanly: to defeat the product you must defeat each prime factor separately.
- In security, we emulate this by engineering **pairwise-independent failure modes** so that a single exploit cannot transitively defeat all layers.

Blueprint Outcome: A repeatable method to design, measure, and audit independence across security layers, turning "defense-in-depth" from a slogan into a quantifiable, provable architecture.