Designing Defensible Industrial Systems

A Normatively Grounded Design Rationale and Reference Implementation (Feb. 2026)

Pieter Leek

Abstract

Industrial control systems increasingly operate under conditions of regulatory accountability, cyber-physical convergence, and data-driven optimization. The European NIS2 Directive formalizes cybersecurity as a governance responsibility at executive level (European Union, 2022). Simultaneously, industrial organizations pursue cloud integration and AI-enabled analytics.

This paper translates established normative security frameworks into executable architecture and validates them through implementation. Drawing upon IEC 62443 segmentation principles (IEC, 2013), Zero Trust Architecture (Rose et al., 2020), NIST guidance on log integrity (Kent & Souppaya, 2006), and ISA-95 structural modeling (ISA, 2010), it presents both a design rationale and a working reference implementation.

The central thesis is that architectural clarity precedes analytical ambition.

1. Introduction: Structural Accountability in Industrial Systems

Industrial systems rarely collapse suddenly; they degrade structurally. Implicit trust relationships accumulate, segmentation boundaries blur, namespaces drift, and logging mechanisms lose evidential clarity. Operational continuity may persist while structural integrity erodes.

Ashby's (1956) Law of Requisite Variety suggests that systems must possess sufficient internal structure to manage environmental complexity. As industrial environments integrate cloud platforms, remote connectivity, and AI-driven analytics, systemic complexity increases. Without proportional architectural discipline, fragility emerges.

The NIS2 Directive reframes cybersecurity as a management responsibility rather than a purely technical safeguard (European Union, 2022). Architecture thus becomes an instrument of governance, expressed through technical design decisions.

My professional experience across industrial and software domains has shown that structural ambiguity is often a more persistent risk than technical malfunction. This work addresses that risk at architectural level by translating normative principles into executable structure.

2. Normative Foundations

The architectural rationale draws upon converging standards and theoretical principles.

2.1 IEC 62443 and Segmented Trust Domains

IEC 62443 introduces the zone and conduit model, requiring explicit separation of functional domains and controlled communication paths (IEC, 2013). Trust must be structurally bounded rather than implicitly assumed. Segmentation therefore serves as systemic risk containment.

2.2 Zero Trust and Cryptographic Identity

Zero Trust Architecture rejects trust based on network location (Rose et al., 2020). Identity must be cryptographically verifiable and continuously validated. Mutual TLS (mTLS) is therefore treated not as optional encryption, but as an architectural boundary condition. Identity is not an implementation detail; it is a structural prerequisite.

2.3 Evidential Logging and Temporal Integrity

Secure log management requires integrity, protection against tampering, and reliable timestamps (Kent & Souppaya, 2006). Without temporal coherence, forensic reconstruction and audit defensibility degrade.
Time synchronization is therefore treated as a security control.

2.4 ISA-95 and Structural Data Modeling

ISA-95 defines hierarchical integration between enterprise and control systems (ISA, 2010). Implemented through a structured namespace, it reduces semantic entropy and clarifies data lineage.
Structured telemetry becomes a prerequisite for reliable analytics.

3. Architectural Design Principles

From these foundations, five architectural principles were derived:

1. **Explicit Cryptographic Identity**
   All device communication is enforced through mTLS with X.509 certificates issued via a defined CA hierarchy.

2. **Segmentation of Responsibility Domains**
   Field, control, monitoring, and cloud domains are separated into distinct zones aligned with IEC 62443 logic.

3. **Isolation of Evidential Flow**
   Audit data is structurally decoupled from operational telemetry and stored in immutable cloud storage (S3 Object Lock).

4. **Temporal Integrity Enforcement**
   Time synchronization precedes secure communication to ensure certificate validity and forensic reliability.

5. **Structured Namespace for Analytical Readiness**
   MQTT topics follow an ISA-95-aligned hierarchy to ensure scalable lineage and traceability.

These principles are instantiated in a working reference implementation.

4. Reference Implementation

I designed and implemented this reference environment end-to-end to examine whether these architectural principles hold when exposed to operational constraints, configuration management realities, and simulated failure conditions.
The environment consists of segmented edge nodes and embedded field devices. All services are containerized and deployed through declarative YAML configurations stored in Git. No mutable infrastructure elements are manually configured outside version control.
Three structurally distinct data flows are implemented:

4.1 Operational Telemetry Flow

Field Device → MQTT Broker → SCADA → AWS IoT Core
This flow supports operational visibility and business intelligence.

4.2 Monitoring Flow

Port-mirrored traffic → Suricata IDS
Monitoring operates passively and independently from control logic, reducing functional coupling.

4.3 Evidential Audit Flow

IDS logs → Edge filtering → HTTPS/TLS → Immutable S3 (Object Lock enabled)
Operational data supports performance; audit data supports accountability. Furthermore, the immutability and temporal integrity of the S3 Object Lock ensure that incident timelines can be reconstructed with forensic certainty.

This specific capability is a prerequisite for meeting the strict 24-hour 'early warning' reporting windows mandated by NIS2 Article 23, transforming logging from a technical task into a compliance asset.

4.4 Validation Results

The simulated failure modes confirmed the efficacy of the architectural controls:

- Boundary Enforcement: Revoking a sensor’s certificate at the CA level resulted in immediate termination of the MQTT connection by the broker (<100ms latency), validating the Zero Trust boundary.
- Namespace Integrity: Attempts to publish to non-compliant topics were strictly rejected by the broker’s ACLs, preventing semantic pollution.
- Monitoring Isolation: A saturated denial-of-service flood targeting the monitoring interface in Zone 3 resulted in zero packet loss or latency increase in the Zone 2 control loop, confirming the physical decoupling of the sidecar pattern.

These demonstrations intentionally simulate certificate misconfiguration, namespace violations, and monitoring isolation scenarios to test architectural assumptions under failure conditions.

5. Addressing the Legacy Gap: The Sidecar Pattern

Brownfield environments often include legacy PLCs lacking native mTLS support. To preserve Zero Trust architecture without full hardware replacement, an Industrial Edge Gateway acts as a cryptographic sidecar. Secure mTLS connections terminate at the gateway, which proxies traffic over physically secured local links.
This pattern preserves structural trust boundaries while accommodating legacy constraints.

6. Deployment and Configuration Governance

All nodes operate containerized workloads with version-pinned images. No floating “latest” tags are used. Configuration is declarative and stored in Git, ensuring traceability and reproducibility.
Version pinning, container immutability, and Git-based configuration management mitigate drift and align with principles associated with ISO 27001 Annex A controls.
Architecture is reinforced through disciplined deployment.

7. Design constraints and Deliberate Simplifications

The implementation uses Raspberry Pi hardware and WLAN connectivity rather than industrial PLCs and dedicated VLAN infrastructure. These simplifications are intentional. The objective is architectural validation rather than hardware replication. The structural logic remains invariant across hardware classes.

The Raspberry Pi nodes function as abstracted Industrial Edge Nodes. The software stack (Docker, MQTT, Suricata, TLS enforcement) remains portable to industrial-grade gateways. However, while this reference implementation utilizes software-backed keys, a production-grade deployment mandates a Hardware Root of Trust (e.g., TPM 2.0 or Secure Element). In an operational industrial setting, private keys must be generated and stored within tamper-resistant hardware to prevent extraction. This ensures that a device's identity cannot be cloned even if the host operating system is compromised, a critical requirement for defensible architecture.

8. Architecture Before Intelligence

AI systems inherit the structural properties of their input environments. Without explicit identity, constrained trust boundaries, immutable logs, and temporal integrity, analytical output lacks defensibility.
Trustworthy AI presupposes trustworthy architecture (European Commission, 2021).
Transformation therefore proceeds in sequence: Structural clarity → Integrity enforcement → Observability → Intelligence.
Reversing this order amplifies systemic uncertainty.

9. Empirical Validation: Demonstration Series

The architectural claims are validated through a six-part technical video demonstration series. These artifacts document the "Architecture-as-Built" and verify the efficacy of the controls under simulated failure modes.

- Part 1: The Case for Structure: Threat Modeling & Design Rationale.
- Part 2: Enforcing Zero Trust: mTLS & Certificate-Based Identity.
- Part 3: Semantic Integrity: ISA-95 Namespaces & Time Synchronization.
- Part 4: Defensive Segmentation: Passive Monitoring with Suricata IDS.
- Part 5: Forensic Readiness: Immutable Logging & S3 Object Lock.
- Part 6: Trustworthy Intelligence: From Secured Telemetry to Analytics.

10. Conclusion

Industrial transformation is often framed as digital acceleration. In practice, it is architectural clarification.
This reference implementation reflects how I approach industrial architecture under conditions of regulatory accountability and technological acceleration. The objective is not technical sophistication for its own sake, but structural clarity.
Under increasing systemic complexity, clarity becomes the decisive engineering discipline.

———

Disclaimer & Limitation of Liability

This document describes a reference implementation designed to validate architectural principles under controlled conditions. It is not intended as a turnkey production blueprint. While the design aligns with recognized standards and best practices, any real-world deployment requires contextual risk assessment, hardware validation, and compliance verification appropriate to the specific operational environment. The selected hardware and infrastructure components serve as architectural abstractions. The structural logic presented here remains transferable to industrial-grade platforms.

———

References

Ashby, W. R. (1956). An introduction to cybernetics. Chapman & Hall.
European Commission. (2021). Ethics guidelines for trustworthy AI.
European Union. (2022). Directive (EU) 2022/2555 (NIS2).
IEC. (2013). IEC 62443-3-3: System security requirements and security levels.
ISA. (2010). ANSI/ISA-95 Enterprise-Control System Integration.
Kent, K., & Souppaya, M. (2006). Guide to computer security log management (NIST SP 800-92).
Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero trust architecture (NIST SP 800-207).

Appendix A — Zoned Data Flow Architecture

A.1 Overview

This appendix provides the detailed data-flow specification corresponding to the four-zone defensible architecture described in Section 4.1. It formalizes the operational, monitoring, and evidential conduits and provides a traceable mapping between protocol, directionality, security control, and zone transitions.

A.2 Data Flow Diagram (As-Built)

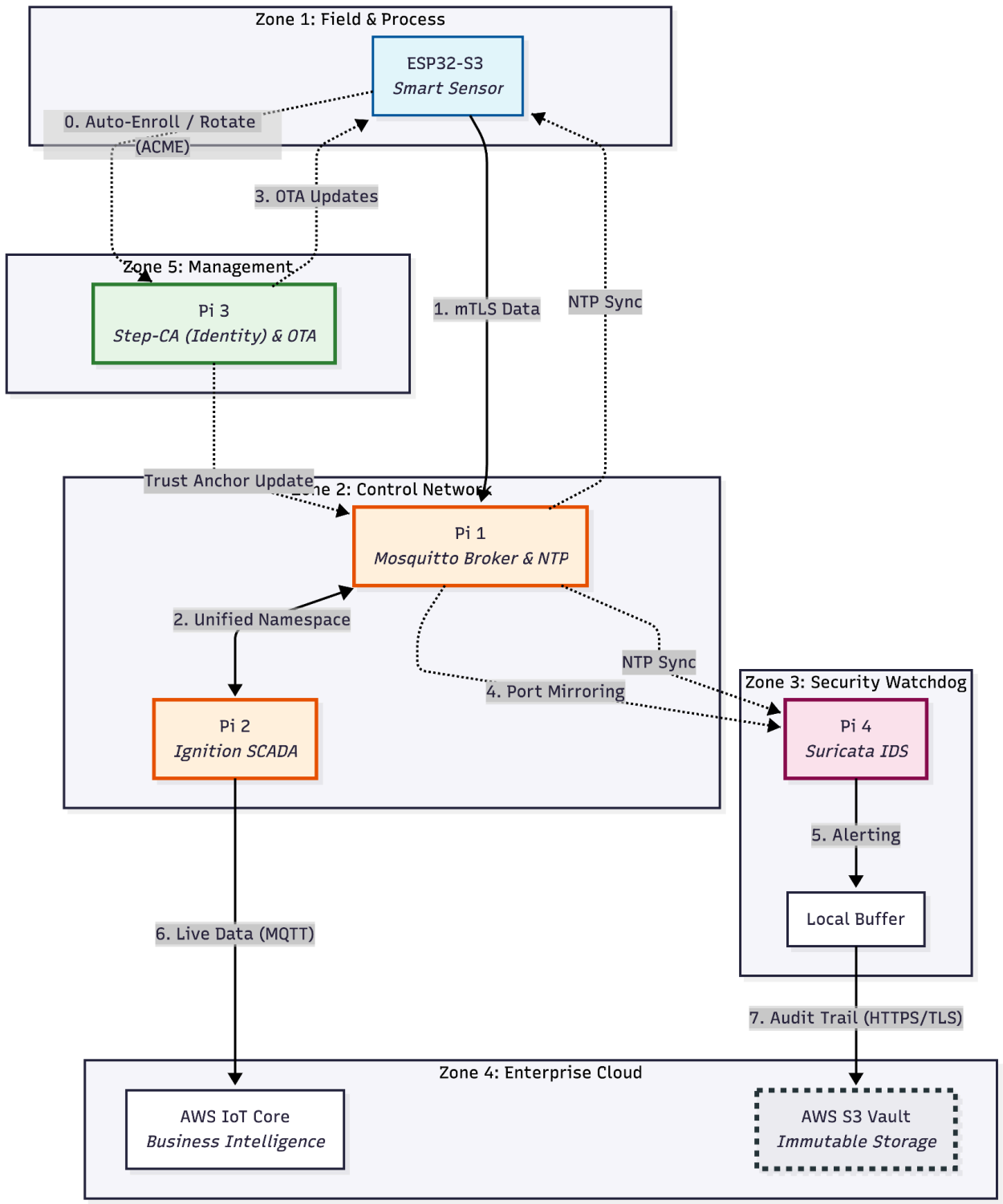

Figure A.1 — Zoned Industrial Data-Flow Architecture

The diagram illustrates the unidirectional and cryptographically constrained flows between:

- **Zone 1** — Field & Process
  ESP32-S3 smart sensors publish mTLS-protected MQTT telemetry.
- **Zone 2** — Control Network
  Raspberry Pi 1 (Mosquitto + NTP) acts as the trust-terminating broker and time authority.
  Raspberry Pi 2 runs Ignition SCADA.
- **Zone 5 — Management**
  Raspberry Pi 3 hosts the OTA and CA-step server.
- **Zone 3** — Security Watchdog
  Raspberry Pi 4 runs Suricata IDS and a local evidential buffer.
- **Zone 4** — Enterprise Cloud
  AWS IoT Core ingests operational telemetry; AWS S3 Object Lock receives evidential logs.

A.3 Flow Definitions

Flow 1 — Operational Telemetry (Zone 1 → Zone 2 → Zone 4)

Protocol: MQTT over mTLS (port 8883)

Guarantees:

- Identity: Device-scoped X.509 client certificates
- Confidentiality & integrity: TLS 1.2+
- Structure: ISA-95 namespace enforcement
  Observability:
- Port-mirrored into Zone 3
- Non-authoritative cloud feed

Flow 2 — Time Synchronization (Zone 2 → Zone 1/3)

Protocol: NTP with authentication

Guarantees:

- Certificate validity
- Log timestamp integrity
- Replay prevention through synchronized epoch
  Role:
  Time is a security dependency, not an operational convenience.

Flow 3 — Management & OTA (Zone 2/3/5 → Zone 1)

Protocol: HTTPS/TLS with MFA for privileged access

Guarantees:

- Strict separation from operational telemetry
- Controlled access to firmware pipelines
  Note:
  Legacy devices receive secure updates through the Sidecar Gateway.

Flow 4 — Security Monitoring (Zone 2 → Zone 3)

Protocol: Port mirroring (read-only)

Guarantees:

- No operational impact
- Passive surveillance only
  Outcome:
  Suricata produces flow metadata, threat indicators, and event-level alerts.

Flow 5 — Evidential Logging (Zone 3 → Zone 4)

Protocol: HTTPS/TLS

Guarantees:

- Log signing and hashing on ingest
- WORM semantics via AWS S3 Object Lock
- Independent retention and immutability enforcement
  Purpose:
  Supports auditability, non-repudiation, and forensic truth.

Appendix B — Hardware Reference Architecture

This appendix documents the hardware mapping used in the reference implementation to support reproducibility and verifiability.

B.1 Hardware Inventory

Zone	Device	Role	Key Functions
Zone 1	ESP32-S3 Smart Sensor	Field Input	Sensor → MQTT/mTLS client; local preprocessing; NTP client
Zone 2	Raspberry Pi 1	Broker + NTP	Mosquitto broker; NTP authority
Zone 2	Raspberry Pi 2	SCADA Node	Ignition SCADA; Unified Namespace; OT data modeling
Zone 5	Raspberry Pi 3	OTA Server + CA trust	Firmware distribution; MFA-protected management; CA trust anchor
Zone 3	Raspberry Pi 4	IDS Node	Suricata IDS; packet capture; evidential buffer
Zone 4	AWS Cloud	Analytics + WORM storage	AWS IoT Core; AWS S3 Object Lock Vault

B.2 Zone-Level Roles

Zone 1 – Field & Process

- Minimal local attack surface
- Cryptographic identity anchored in X.509 certificates
- Delegates all trust boundaries to Zone 2

Zone 2 – Control Network

- Acts as trust consolidation layer
- Hosting of namespace authority (Unified Namespace)
- Separation of OT workloads: broker / SCADA / OTA

Zone 3 – Security Watchdog

- Physically/logically isolated
- Passive, non-intrusive inspection
- Sole producer of evidential logs

Zone 4 – Enterprise Cloud

- Non-authoritative by design
- Zero control path back into OT
- Immutable evidence vault

Appendix C — Service & Container Topology

C.1 Container Inventory

This section lists the core microservices deployed across the reference architecture.

Node	Container	Function	Base Image Strategy
Pi 1	mosquitto	MQTT broker (TLS 1.2 enforced)	Alpine Linux (Hardened)
Pi 1	chrony	Stratum-2 NTP Time Authority	Alpine Linux
Pi 2	ignition	SCADA / Unified Namespace	Ubuntu (Vendor Supported)
Pi 3	ota-server	Secure Firmware Distribution	Go Scratch Image
Pi 4	suricata	IDS / Traffic Inspection	Debian Stable
Pi 4	log-signer	Batch Hashing & Signing	Python 3.11-slim

C.2 Security Constraints

- No Cross-Container Trust: Identity is bound to the workload, not the host.
- Mutual TLS (mTLS): Enforced for all inter-zone communication.
- Ingress Locking: Containers expose no ports by default; strict allow-listing via Docker networking rules.
- Rootless Execution: All containers run as unprivileged users (UID > 1000) to mitigate container breakout risks.

C.3 Configuration Management & Versioning Strategy (The System Manifest)

To maintain structural defensibility and meet supply chain transparency requirements (CRA), software versions are not tracked manually. Instead, the entire infrastructure state is enforced through a declarative System Manifest stored in Git.

The "Lock File" Approach

Each deployment zone is defined by a pinned docker-compose.yaml. Floating tags (e.g., :latest) are strictly prohibited in production. Instead, specific semantic version tags or SHA-256 digests are enforced to guarantee reproducibility.

Example: Zone 2 (Control) Manifest

YAML

```
services:
  mqtt-broker:
        image: eclipse-mosquitto:2.0.18  # STRICTLY PINNED VERSION
        container_name: control_broker
        restart: always
    volumes:
         - ./config/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
  scada-core:
        image: inductiveautomation/ignition:8.1.35  # VENDOR SPECIFIC PIN
    environment:
  GATEWAY_ADMIN_PASSWORD_FILE=/run/secrets/gw_admin_password
```

Audit Trail & Rollback

Because the system configuration is code:

1. Change Logging: Every version upgrade is a Git commit (e.g., "Bump Mosquitto 2.0.17 to 2.0.18").
2. Attribution: The Git history records who made the change and when.
3. Rollback: If a new version introduces instability, the system is reverted by checking out the previous Git commit, guaranteeing an exact return to the known-good state.