International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Software Update Architectures for Embedded Automotive and Consumer Systems: A Policy-Driven Architectural Recommendation Framework

Vignesh Alagappan

Senior Member, IEEE
Rheem Manufacturing, Roswell, Georgia, USA,
Vignesh.alagappan@rheem.com

Abstract:
Software update systems in embedded and automotive domains have evolved from simple firmware delivery mechanisms into distributed governance and execution platforms. Modern systems must operate under intermittent connectivity, power instability, heterogeneous hardware, and in automotive systems, safety-critical operating conditions constrained by functional safety standards such as ISO 26262 and in-vehicle networks governed by protocols including Controller Area Network (CAN) and Automotive Ethernet. This paper synthesizes two decades of cross-industry experience spanning automotive infotainment platforms, manufacturing IoT systems, and connected consumer products to present a comprehensive policy-driven architectural framework for software update systems. The framework treats Over-the-Air (OTA) updates as a distributed control problem requiring coordinated governance across cloud and device domains, introducing dual governance layers, cryptographically signed deployment intent aligned with ISO/SAE 21434 cybersecurity requirements, local rule enforcement mechanisms, transactional update execution patterns, and two-hop OTA distribution architectures for vehicle systems compliant with AUTOSAR specifications. An architectural recommendation model guides system designer in selecting appropriate update patterns and governance depth based on operational constraints, safety criticality, and regulatory compliance requirements. This work bridges theoretical safety and security frameworks with practical implementation patterns validated across multiple production systems managing millions of connected devices.

Keywords: Software updates, over-the-air (OTA) updates, automotive systems, embedded systems, IoT security, policy-driven architecture, ISO 26262, ISO/SAE 21434, AUTOSAR, firmware updates, distributed systems.

I. INTRODUCTION

Software updates represent a foundational capability in the lifecycle management of embedded and automotive products. These systems must address security vulnerabilities, deliver feature enhancements, optimize operational performance, and ensure ongoing regulatory compliance [2], [15]. However, contemporary understanding of update failures reveals that the majority of incidents arise not from file transfer mechanisms, but from unsafe application of updates during prohibited operational states, violations of component dependency constraints, or inadequate recovery mechanisms when power loss or system faults occur during critical update phases.

Automotive systems amplify these engineering challenges through multi-ECU architectures where updates must propagate across heterogeneous computing platforms, safety-critical operational states defined under functional safety standards such as ISO 26262 [1], and constrained in-vehicle networks governed by

protocols including Controller Area Network (CAN) [11] and Automotive Ethernet [10], [12]. The complexity escalates further when considering that modern vehicles integrate dozens of electronic control units, each potentially operating different software versions, update schedules, and dependency relationships.

This paper emerges from synthesis of over two decades of cross-industry experience designing, implementing, and operating software update architectures across diverse domains including automotive infotainment platforms managing tens of thousands of vehicles, manufacturing IoT systems controlling industrial equipment, and consumer connected products spanning HVAC systems and water heating equipment at millions-of-devices scale. This breadth of implementation experience across automotive, industrial, and consumer domains provides unique insight into common architectural patterns, failure modes, and governance requirements that transcend individual industry verticals.

Rather than treating OTA as fundamentally a transport or file delivery problem, this work positions software updates as a distributed governance and state transition system where cloud infrastructure defines deployment intent and authorization, while device-resident control planes enforce execution safety, dependency satisfaction, and recovery guarantees. This architectural perspective aligns with emerging standards in automotive cybersecurity [2] and secure boot architectures [5], while extending these frameworks with practical governance patterns validated in production deployments.

The framework presented in this paper provides system architects and embedded systems engineers with structured guidance for designing update architectures appropriate to their specific operational constraints, safety requirements, and compliance obligations. By grounding architectural recommendations in established standards including ISO 26262 for functional safety [1], ISO/SAE 21434 for automotive cybersecurity [2], AUTOSAR software update specifications [3], and NIST firmware resiliency guidelines [4], this work bridges theoretical frameworks with implementation patterns that have demonstrated reliability across production systems managing device populations exceeding ten million units.

II. DESIGN PHILOSOPHY

A. Design Priority Hierarchy

When architectural decisions present conflicting requirements, the update system must adhere to the following non-negotiable priority ordering:

Safety → Security → System Integrity → Compliance → User Experience → Speed

This hierarchy reflects fundamental principles established in safety engineering standards [1] and cybersecurity engineering philosophy [2]. Safety requirements take absolute precedence because unsafe system behavior can result in physical harm or loss of life, particularly in automotive and industrial control contexts. Security follows immediately because compromised systems can be weaponized to cause safety incidents or breach user privacy. System integrity ensures the device can perform its intended function and recover from failures. Compliance enables legal operation and market access. User experience and deployment speed, while commercially important, must never compromise higher-priority requirements.

B. Alignment with Safety and Security Standards

The architectural framework presented in this paper explicitly aligns with established industry standards. ISO 26262 [1] defines functional safety requirements for automotive systems, establishing safety integrity levels (ASIL) and requiring systematic approaches to hazard analysis and risk management. Updates to safety-critical ECUs must respect ASIL classifications and undergo appropriate validation before deployment.

ISO/SAE 21434 [2] establishes cybersecurity engineering requirements for road vehicles, mandating threat analysis, risk assessment, and secure development processes throughout the vehicle lifecycle. The

framework's emphasis on cryptographic verification, anti-rollback controls, and secure communication channels directly implements requirements from this standard.

AUTOSAR (AUTomotive Open System ARchitecture) [3] specifications define software architectures for automotive ECUs, including standardized interfaces for software update and diagnostic protocols. The two-hop distribution model and ECU dependency management patterns presented in Section VIII align with AUTOSAR software update specifications.

NIST SP 800-193 [4] provides guidelines for platform firmware resiliency, establishing requirements for protection, detection, and recovery mechanisms. The framework's recovery architecture and anti-rollback controls implement these resiliency principles.

III. THREAT AND SAFETY MODEL

A. Threat Model

The architecture assumes adversarial conditions consistent with firmware supply-chain threat models [4] and automotive cybersecurity frameworks [2]. Specifically, the system must maintain security properties under the following threat scenarios:

- Man-in-the-middle attacks: Adversaries with network access can intercept, modify, or inject update traffic between cloud infrastructure and devices.
- Replay attacks: Previously captured legitimate update artifacts can be retransmitted to force installation of older software versions.
- Build pipeline compromise: Adversaries may compromise build systems or signing infrastructure to inject malicious code into otherwise legitimate update artifacts.
- Unauthorized campaign triggers: Attackers with cloud infrastructure access attempt to initiate update campaigns without proper authorization.
- Downgrade attacks: Forced installation of older firmware versions containing known vulnerabilities that have been patched in current versions.

These threat scenarios align with secure update architectures such as SUIT (Software Updates for Internet of Things) [6] and Uptane [5], which provide cryptographic frameworks for verifying update authenticity and integrity.

B. Security Implications and Controls

The identified threat model mandates specific architectural controls:

- Update artifacts and deployment intent envelopes must be cryptographically signed using asymmetric key cryptography [9], with public keys provisioned to devices during manufacturing or secure commissioning flows.
- Devices must verify authenticity, integrity, and compatibility of all received artifacts before staging to storage or initiating update transactions.

Anti-rollback controls must exist at device or ECU level to prevent installation of firmware versions older than currently executing software, with monotonic counters stored in tamper-resistant storage [5].

C. Safety Model

Updates must not execute when system operational state increases risk of unsafe behavior, particularly in vehicles subject to functional safety constraints defined in ISO 26262 [1]. Prohibited operational states include but are not limited to:

- Vehicle in motion (velocity > 0 km/h)
- Ignition state active or engine operating
- Critical diagnostic trouble codes (DTCs) active indicating system malfunction
- Battery state of charge below threshold for safe update completion

Systems must additionally tolerate fault conditions including power loss during flashing operations and unexpected system reboots mid-installation. Recovery behavior must ensure the platform returns to a verified software state, consistent with firmware resiliency guidelines [5]. This typically requires dual-bank flash architectures, golden image protection, or other mechanisms enabling rollback to last-known-good configuration.

IV. OTA AS A DISTRIBUTED GOVERNANCE SYSTEM

OTA update systems function as distributed control architectures comprising two coordinated operational planes, each with distinct responsibilities and authority boundaries. This separation mirrors secure automotive update frameworks such as Uptane [6], which explicitly partitions authority between director and image repositories.

Operational Plane	Primary Responsibilities
Cloud Plane	Fleet-level governance, campaign authorization and orchestration, rollout pacing and risk control, artifact signing and distribution, policy definition and enforcement
Device/ Vehicle Plan	Local precondition evaluation, rule-based execution gating, transactional update processing, recovery management, compliance telemetry

TABLE I. RESPONSIBILITIES MATRIX

The cloud plane exercises governance authority over fleet-wide update operations but cannot directly observe or control device operational state. Conversely, the device plane possesses complete operational state visibility but lacks fleet-wide context and long-term policy management capabilities. Effective update architecture requires these planes to cooperate through well-defined interfaces and trust boundaries.

This distributed model addresses fundamental constraints in connected systems: intermittent connectivity prevents real-time cloud control of device behavior, while safety requirements demand that devices make autonomous decisions about update application based on local operational state. The architecture therefore provides cloud systems with intent expression capabilities while reserving execution decisions to device-resident control logic.

V. CLOUD PLANE ARCHITECTURE

The cloud plane implements fleet-level governance through coordinated subsystems:

A. Campaign Orchestration Engine

Manages phased rollout execution, implementing gradual deployment patterns that limit blast radius of update failures. Controls rollout velocity based on observed failure rates, device telemetry, and risk thresholds. Provides pause, resume, and emergency rollback capabilities

B. Campaign Manager

Handles device targeting based on attributes including current software version, hardware variant, geographical region, and fleet segmentation. Defines campaign lifecycle states and transitions. Implements maker-checker workflows requiring independent authorization for high-risk campaigns.

C. Artifact Distribution Manager

Ensures scalable artifact delivery through content delivery networks (CDN) or peer-to-peer distribution when appropriate. Manages artifact versioning, delta update generation, and compatibility matrices. Implements bandwidth optimization for constrained networks.

D. Signing and Security Manager

Provides cryptographic signing services for update artifacts and deployment intent envelopes using CMS (Cryptographic Message Syntax) [9] or equivalent standards. Manages key lifecycle including generation, rotation, and revocation. Implements hardware security module (HSM) integration for key protection

E. Policy and Rules Manager

Defines update eligibility policies including time windows, operational prerequisites, and dependency constraints. Generates executable rule predicates that devices evaluate locally. Maintains policy versioning and audit trails for compliance documentation.

F. Governance and Compliance Manager

Enforces authorization workflows requiring multiple approvals for production campaigns. Maintains compliance records demonstrating adherence to regulatory requirements such as IEC 62443 [15] for industrial systems. Provides audit logging and forensic capabilities

G. Signed Deployment Intent

The cloud produces a cryptographically signed deployment intent envelope containing target software versions, execution constraints, dependency requirements, rollout strategy parameters, and authorization proofs. This intent document serves as both authorization credential and execution specification, similar to intent-based control frameworks in secure OTA architectures [6]. Devices validate signature authenticity and envelope integrity before processing contained directives.

VI. DEVICE AND VEHICLE PLANE ARCHITECTURE

Device-resident subsystems enforce execution safety and manage update lifecycle through state machine implementation. The device plane operates autonomously from cloud connectivity making local decisions based on operational state and policy constraints.

A. Update Lifecycle Manager

The lifecycle manager governs state transitions through: IDLE → DOWNLOAD → STAGED → VERIFIED → READY → INSTALLING → COMMIT → ACTIVE. Each transition requires validation. Failed validations return to previous stable state or trigger recovery. State machine implementation uses persistent state storage (typically dedicated flash sector at fixed address) maintaining state across power cycles.

DOWNLOAD state fetches artifacts over HTTPS with resume capability per RFC 7233 HTTP Range requests. Downloads chunk artifacts into 64KB-256KB blocks accommodating connection interruptions common in cellular networks. Each block includes CRC32 or SHA-256 checksums for corruption detection. Download manager maintains block bitmap (one bit per block) tracking completed blocks supporting resume from arbitrary block boundaries without re-downloading completed portions.

STAGED state writes downloaded artifact to staging partition while active image continues execution from active partition. Staging storage requires 1.5-2x artifact size accommodating decompression workspace for compressed artifacts (LZMA, Brotli compression ratios 30-50%). File system or raw partition writes depend on platform capabilities. Raw partition writes offer better power-loss resilience but complicate delta update application requiring in-place binary patching.

VERIFIED state validates cryptographic signatures and checksums against device-provisioned trust anchors. Signature verification uses device-provisioned public keys loaded from secure storage during boot. RSA-2048 signature verification completes in 50-200ms on 32-bit ARM Cortex-M4 processors (168 MHz) depending on implementation (software vs. hardware acceleration). ECDSA P-256 verification completes faster (10-50ms) but requires elliptic curve arithmetic library increasing code size 20-40KB. Devices reject artifacts failing verification and report failure reason codes to cloud for debugging.

READY state indicates artifact validation complete and installation can proceed when safety conditions permit. Device may remain in READY for hours or days waiting for safe installation window (vehicle parked overnight, HVAC idle period, water heater off-peak hours). READY state consumes storage holding staged artifact until installation or expiration timeout (typically 7-30 days depending on storage

constraints). Expired artifacts are purged automatically freeing staging storage for subsequent update attempts.

INSTALLING state copies verified artifact from staging to active partition. Flash write operations occur at sector boundaries (typically 4KB-64KB depending on NOR vs NAND flash technology). Write progress tracking at sector granularity enables recovery from power loss mid-installation. Installation completes in 30 seconds to 10 minutes depending on artifact size (512KB to 16MB range typical), flash write speed (50-200 KB/s for NOR flash, 500KB-2MB/s for managed NAND), and processor capability affecting decompression throughput.

COMMIT state finalizes installation by updating boot configuration pointer to reference new image and incrementing anti-rollback counter in secure storage. This operation should be atomic through single flash write to boot configuration sector when possible. On platforms lacking atomic write capability (bare NOR flash without transaction support), two-phase commit protocol writes new configuration with pending flag, verifies write success through readback, then clears pending flag. Boot loader interprets pending flag as installation in progress requiring validation before execution

B. Cryptographic Algorithm Recommendations

Selection of cryptographic algorithms for OTA update systems requires balancing security strength, computational performance on constrained embedded processors, code size footprint, and long-term algorithm viability. The following recommendations align with NIST SP 800-57 key management guidelines, ISO/SAE 21434 automotive cybersecurity requirements, and industry best practices validated in production deployments.

1) Signature Algorithms

RSA-PSS (Probabilistic Signature Scheme) with 2048-bit keys provides 112-bit security level sufficient for firmware artifacts with expected lifetimes under 15 years. RSA-PSS with 3072-bit keys provides 128-bit security level recommended for safety-critical ECUs or systems requiring protection beyond 2030. RSA signature generation in cloud HSMs executes in 1-5ms while device-side verification completes in 50-200ms on ARM Cortex-M4 class processors (168 MHz software implementation). RSA verification benefits from hardware acceleration: ARM Cortex-A series crypto extensions reduce verification to 10-30ms. Public key size: 2048-bit = 256 bytes, 3072-bit = 384 bytes. Signature size matches key size.

ECDSA (Elliptic Curve Digital Signature Algorithm) with P-256 curve provides 128-bit security level in compact format. ECDSA P-384 provides 192-bit security level for long-term protection. Device-side ECDSA P-256 verification executes in 10-50ms on ARM Cortex-M4 (software implementation) or 3-10ms with hardware crypto accelerators. Public key size: P-256 = 64 bytes, P-384 = 96 bytes. Signature size: P-256 = 64 bytes, P-384 = 96 bytes. ECDSA requires elliptic curve arithmetic library adding 20-40KB code size. Recommended for resource-constrained devices where signature size and verification performance matter more than code size.

EdDSA (Edwards-curve Digital Signature Algorithm) with Ed25519 curve provides 128-bit security level with deterministic signatures eliminating random number generator dependencies. Verification performance: 5-15ms on ARM Cortex-M4, 1-5ms with hardware acceleration. Public key size: 32 bytes. Signature size: 64 bytes. Code size similar to ECDSA (20-30KB library). EdDSA offers best performance-to-security ratio for modern implementations but has less mature embedded ecosystem than ECDSA/RSA.

2) Hash Functions

SHA-256 (256-bit Secure Hash Algorithm) provides 128-bit collision resistance suitable for firmware integrity verification. Hash computation: 5-20 MB/s on ARM Cortex-M4, 50-200 MB/s on ARM Cortex-A series. Hardware acceleration (ARM Crypto Extensions) increases throughput 3-10×. Output size: 32 bytes. Widely supported in embedded cryptographic libraries. Recommended for all artifact checksums and merkle tree construction.

SHA-384/SHA-512 from SHA-2 family provide 192-bit/256-bit collision resistance for long-term archival or safety-critical applications. SHA-512 performs better than SHA-256 on 64-bit processors but slower on 32-bit embedded systems. Output size: SHA-384 = 48 bytes, SHA-512 = 64 bytes. Use SHA-384 when higher security margin required without performance penalty of SHA-512 on 32-bit targets.
SHA-3 family (Keccak-based) offers alternative to SHA-2 with different security foundation. Performance on embedded processors: 2-10 MB/s (slower than SHA-256 without hardware acceleration). Use when cryptographic diversity desired (hedging against SHA-2 vulnerabilities) or when hardware SHA-3 acceleration available.

3) Transport Encryption

TLS 1.3 (RFC 8446) provides transport security for update artifact downloads. Mandatory cipher suites: TLS_AES_128_GCM_SHA256 (128-bit security), TLS_AES_256_GCM_SHA384 (256-bit security), TLS_CHACHA20_POLY1305_SHA256 (256-bit security with better performance on processors lacking AES hardware acceleration). Disable TLS 1.2 and earlier versions to eliminate known vulnerabilities. Certificate validation requires X.509 PKI infrastructure per RFC 5280 with OCSP stapling or CRL distribution for revocation checking.

4) Key Management

Signing keys: Generate and store in FIPS 140-2 Level 3 or Level 4 HSMs with multi-party control requiring M-of-N authorization (typically 2-of-3 or 3-of-5). Separate signing keys by criticality: production signing key (highest protection, air-gapped HSM with ceremony-based access), development/testing keys (standard HSM with role-based access), component-specific keys enabling granular revocation without affecting entire fleet.
Verification keys: Embed public keys in device firmware during manufacturing using secure provisioning process. Store in read-only memory (OTP fuses, secure flash with write-once protection) or ARM TrustZone secure storage preventing runtime modification. Support key rotation through multi-level verification: root key (embedded in hardware, validates intermediate certificates), intermediate keys (updateable through signed certificate chains), leaf keys (specific to component/version, shortest lifetime). Key rotation interval: Root keys 10-15 years (vehicle lifetime), intermediate keys 2-5 years, leaf keys 6-12 months or per-release.

5) Algorithm Selection Guidance

Consumer IoT devices (HVAC, water heaters): ECDSA P-256 + SHA-256 balances security and performance. Code size critical: RSA-2048 + SHA-256 requires smaller library. Automotive ECUs without crypto hardware: RSA-2048 + SHA-256 provides widest library support. Automotive ECUs with crypto accelerators: ECDSA P-256 + SHA-256 or EdDSA Ed25519 + SHA-256 maximizes performance. Safety-critical automotive (ASIL C/D): RSA-3072 + SHA-384 or ECDSA P-384 + SHA-384 provides higher security margin. Industrial equipment with long service life: RSA-3072 or ECDSA P-384 accommodates 20+ year operational lifetimes. Gateway/TCU devices: Support multiple algorithms enabling verification of updates for heterogeneous ECU populations using different signature schemes.

C. Local Policy Evaluation Engine

The policy engine interprets deployment intent constraints against current operational state. Engine parses policy predicates encoded in JSON or CBOR format and evaluates against telemetry sources including CAN bus messages (automotive), ModBus registers (industrial), or local sensor readings (consumer IoT).
Policy evaluation occurs every 30-300 seconds depending on device capability and power constraints. Vehicles typically evaluate every 60 seconds during ignition-on periods and every 300 seconds when parked. IoT devices with power constraints evaluate less frequently (5-15 minutes) trading policy responsiveness for power consumption. Battery-powered devices may evaluate only when awakened by scheduled timers or external events.
Precedence rules resolve conflicts when multiple policies apply simultaneously. Safety rules (priority 1000-1999) always take precedence over resource optimization rules (priority 3000-3999). Security rules (priority 2000-2999) override user preference settings (priority 5000-5999). Within same priority class, blocking failures prevent update execution while warnings permit execution with notification to cloud for monitoring

D. Rules Execution Engine

Rules evaluate predicates defined in deployment intent documents. Rule engine accesses telemetry through abstraction layers isolating rule logic from hardware-specific sensor interfaces. This abstraction permits policy reuse across hardware variants while mapping to platform-specific telemetry sources during runtime.
Vehicle speed rules query wheel speed sensors over CAN bus using standardized PIDs (0xB4-0xB7 for individual wheel speeds) or manufacturer-specific PIDs. Typical query retrieves 4-wheel speed values, validates sensor health bits in response frame, and computes vehicle velocity as maximum of four wheel speeds. Speed threshold is typically 0.5 km/h accounting for sensor noise and vehicle movement on slopes without driver input.
Battery state rules check voltage (PID 0x42) and state-of-charge (PID 0x5C for hybrid/EV). Voltage thresholds depend on battery chemistry: 12V lead-acid systems require 12.4V minimum (12.6V preferred indicating full charge); Lithium chemistries use chemistry-specific thresholds (3.0V per cell minimum, 3.7V nominal for Li-ion). SoC thresholds are typically 50-60% ensuring sufficient capacity for installation completion plus margin for unexpected power draw or installation duration variation.
Storage availability rules verify free space exceeds 1.5x artifact size. This accounts for staging area (1x artifact size), decompression workspace (0.3-0.5x for compressed artifacts), and temporary files during installation (0.1-0.2x for logs, partial writes, metadata). Insufficient storage triggers cleanup of expired artifacts, diagnostic logs, or cached data before retrying storage check.

E. Security and Anti-Rollback Manager

The security manager verifies signatures using RSA-PSS or ECDSA algorithms per RFC 4108 CMS specification. Signature verification occurs twice: before staging (preventing malicious artifacts from consuming storage) and before installation (defense-in-depth against staging storage corruption or time-of-check-to-time-of-use attacks).

Anti-rollback counters reside in one-time-programmable (OTP) memory, eFuse, or authenticated EEPROM depending on hardware security capabilities. Counter values are 32-bit or 64-bit unsigned integers incremented on each successful update. Devices compare incoming version numbers against stored counter and reject updates with version <= counter value. Counter overflow protection varies by implementation: 32-bit counters support 4.3 billion updates (sufficient for daily updates over 10 million device-years); 64-bit counters eliminate practical overflow concerns.
Emergency rollback scenarios requiring version downgrade (e.g., critical defect discovered post-deployment) use out-of-band mechanisms involving physical access to device or authorized dealer diagnostic tools. These mechanisms bypass normal anti-rollback enforcement through authenticated

commands requiring cryptographic proofs unavailable to remote attackers. This prevents remote downgrade attacks while permitting necessary rollback under controlled conditions with audit trail generation.

F. Recovery and Resilience Manager

Recovery manager implements fault tolerance using dual-bank flash architecture. Bank A contains currently executing image, Bank B contains backup image. Boot loader validates Bank A integrity on each startup; if validation fails (corrupted checksum, missing image header, invalid signature), boot loader automatically swaps to Bank B without user intervention.

Golden image protection maintains factory recovery image in read-only partition typically located in separate flash device or protected flash region with hardware write-protect enabled. This image cannot be updated through normal OTA mechanisms. Updates to golden image require physical access through secure dealer tools or manufacturing test fixtures. Golden image provides last-resort recovery when both Bank A and Bank B fail, though this requires user intervention or dealer service.

Automatic rollback triggers after 3 consecutive boot failures detected through persistent boot failure counter. Boot loader increments counter stored in EEPROM or battery-backed RAM on each boot attempt. Successful boot (defined as reaching application main() and executing for minimum duration, typically 30 seconds) resets counter to zero. Three failures indicate corrupted or defective image requiring rollback to backup bank. Some implementations use exponential backoff after each failure extending timeout before subsequent retry (1 second, 2 seconds, 4 seconds) allowing transient faults to clear.

Recovery time objectives vary by system criticality. Safety-critical automotive ECUs must recover to operational state within 30-60 seconds enabling vehicle operation after temporary power loss. Consumer IoT devices tolerate 2-5 minute recovery times permitting more extensive integrity checking and recovery procedures. Industrial equipment requirements vary by process criticality: safety systems require sub-minute recovery while monitoring equipment tolerates longer recovery periods.

VII. VEHICLE-SPECIFIC OTA LOGIC

Automotive systems require safety gating beyond consumer IoT devices due to functional safety requirements defined in ISO 26262 and multi-ECU coordination complexity. Vehicle update logic implements additional constraints ensuring updates occur only during safe operational states.

A. Vehicle State Gating

Update execution prerequisites specific to vehicles include:

- Vehicle velocity = 0 km/h: Verified through CAN bus wheel speed sensors queried via PIDs 0xB4-0xB7 for individual wheel speeds or manufacturer-specific PIDs providing computed vehicle speed. Rule evaluates max(FL_speed, FR_speed, RL_speed, RR_speed) < 0.5 km/h accounting for sensor noise (typically ±0.1-0.3 km/h) and vehicle movement on slopes without driver input. Sensor validation confirms all four sensors report consistent values within threshold (±2 km/h between sensors). Single sensor failure indicated by sensor-specific DTC or out-of-range value aborts update eligibility until sensor fault resolution through diagnostic procedures and DTC clearing.
- Transmission in Park: Queried via PID 0x0A45 for transmission range sensor state or manufacturer-specific transmission position PIDs. Automatic transmission must report Park (P) position corresponding to mechanical park pawl engagement. Manual transmission systems require neutral gear (queried via clutch position sensor and gear selector position) plus parking brake engagement verified via separate parking brake switch PID. Some ECU updates affecting transmission control logic require additional confirmation from transmission output shaft speed sensor (confirming zero rotation) and brake pressure sensor (confirming brake application) before proceeding with installation.
- Stable power supply: Battery voltage queried via PID 0x42 must remain between 12.4V-14.8V for 12V systems indicating alternator charging (engine running) or fully charged battery (engine off). State of

charge queried via PID 0x5C for hybrid/EV vehicles must indicate >= 60% charge. Installation duration typically requires 200-800 watt-hours depending on ECU count (single ECU update: 50-150 Wh; multi-ECU campaign: 500-1000 Wh), flash sizes, and update complexity. Multi-ECU updates updating 5-10 ECUs simultaneously require proportionally higher energy reserves. Insufficient battery capacity risks incomplete installation from power exhaustion during multi-ECU flashing sequences leaving vehicle in inconsistent state requiring dealer recovery.

- No critical DTCs: Critical diagnostic trouble codes queried via Mode 03 (confirmed DTCs) and Mode 07 (pending DTCs) include P0xxx powertrain codes indicating engine, transmission, or emissions system faults; C0xxx chassis codes indicating brake, steering, or suspension system faults; B0xxx body codes indicating airbag, seatbelt, or other safety system faults. Update deferral occurs when DTCs indicate potential safety system degradation that could compound during update-induced temporary functionality loss (e.g., ECU reboot, communication bus disruption). Non-critical informational codes do not block updates but are logged for correlation analysis with post-update behavior tracking whether updates trigger new fault conditions.

Thermal limits: ECU ambient temperature measured via internal temperature sensors (typical automotive ECUs include on-die temperature sensor) or engine coolant temperature sensor (PID 0x05 providing proxy for under-hood ambient temperature) must remain < 85°C for typical automotive-grade components. Flash write reliability degrades above specified temperature thresholds varying by flash technology: commercial-grade NOR flash typically rated to 85°C junction temperature; automotive-grade extends to 105°C; industrial-grade reaches 125°C. Industrial equipment deployed outdoors or in high-temperature manufacturing facilities may enforce stricter limits (60-70°C) depending on installation environment and component derating requirements

B. Multi-ECU Dependency Management

ECU dependencies form directed acyclic graphs (DAGs) where some ECUs provide foundational services (communication gateways, power management, bootloaders) required by dependent ECUs. Update orchestration follows topological ordering ensuring prerequisite ECUs update before dependents, preventing communication loss or dependency violations during multi-ECU campaigns.

Example dependency chain from typical vehicle architecture: Gateway ECU (provides CAN/Ethernet bridging and routing) → Infotainment Head Unit (coordinates audio/video systems) → Amplifier (audio processing) → Individual Speaker Controllers (per-channel control). Gateway must update first maintaining communication infrastructure supporting downstream ECU updates. Violating this ordering results in loss of connectivity to downstream ECUs requiring manual recovery through dealer diagnostic tools connected directly to affected ECUs via separate diagnostic CAN bus.

AUTOSAR specifications define dependency metadata including version compatibility matrices specifying which ECU software versions interoperate successfully, update prerequisites defining mandatory update ordering, and failure recovery strategies for partial update completion scenarios where some ECUs update successfully while others fail. Deployment intent documents encode these dependencies as directed graph structures with nodes representing ECUs (identified by address or unique ID) and edges representing dependency relationships (labeled with version constraints and criticality).

Graph traversal algorithms (typically depth-first search or topological sort) compute valid update orderings satisfying all dependency constraints before campaign initiation. Campaign validation phase rejects configurations containing circular dependencies as they create deadlock conditions preventing any update progression. If circular dependencies are detected during campaign configuration, engineering teams must resolve dependencies through architectural changes (breaking circular references through interface versioning) or staged multi-campaign deployments breaking circular references across campaign boundaries with compatibility shims maintaining operation between campaigns.

C. ASIL Classification Impact

Safety integrity levels defined in ISO 26262 affect update validation requirements and deployment processes. Higher ASIL classifications require more stringent validation before production deployment:

- ASIL A/B (low to moderate safety impact): Standard validation including automated regression testing with requirement coverage >= 80%, unit test coverage >= 80% statement coverage, and integration testing covering nominal and fault scenarios. Phased rollout with 1% → 10% → 50% → 100% progression over 7-14 days. Minimum success rate threshold 98% before phase progression. Phase advancement blocked if success rate falls below threshold or critical failures (defined as safety-relevant malfunctions) occur. Example ASIL A/B systems include infotainment, climate control, body control modules controlling non-safety-critical functions like interior lighting and door locks.
- ASIL C (high safety impact): Requires hardware-in-the-loop (HIL) testing simulating realistic vehicle operating conditions including sensor inputs (wheel speeds, steering angle, brake pressure), actuator outputs (throttle, brake modulation), and network traffic patterns (typical CAN bus loads, message timing). Extended 1% canary phase (7-14 days minimum) with detailed telemetry monitoring for failure patterns including degraded performance, communication errors, and unexpected behavior. Independent safety review by personnel uninvolved in development before progression to 10% phase. Minimum success rate 99.5%. Example ASIL C systems include electronic stability control (ESC), power steering (EPS), certain ADAS functions like adaptive cruise control.
- ASIL D (highest safety impact): Most stringent requirements including full vehicle integration testing on proving grounds under varied conditions (wet, dry, hot, cold, emergency maneuvers), third-party safety assessment by independent verification organization accredited for ISO 26262 compliance, and extended field monitoring (30+ days at 1% deployment minimum) before wider rollout. May require regulatory notification before deployment depending on jurisdiction (e.g., NHTSA in US, type approval authorities in EU). Minimum success rate 99.9% at each phase. Example ASIL D systems include primary braking control (ABS, ESC foundation brakes), airbag deployment logic, primary steering functions. Any field failure during canary phase triggers immediate campaign pause and comprehensive root cause investigation before retry attempts permitted

Evidence trails for safety-critical ECUs include: test execution records with timestamps, test configurations, stimulus sequences, and result codes; hardware-in-the-loop test configurations documenting simulated vehicle states and sensor values; approval records with reviewer identities, qualifications demonstrating ISO 26262 competence, and approval timestamps; field telemetry showing deployment progression, success/failure statistics, failure mode distribution across reason code categories; timeline visualization showing campaign phases and decision points. These evidence packages support regulatory audits and incident investigations demonstrating ISO 26262 compliance throughout update lifecycle from development through field deployment.

VIII. IN-VEHICLE DISTRIBUTION PLANE

Vehicle update architectures implement two-hop distribution protecting ECUs from direct internet exposure: Cloud → Vehicle Gateway → Interior ECUs. The gateway receives updates from cloud infrastructure over cellular or WiFi, then redistributes to interior ECUs over CAN or Automotive Ethernet networks. This architecture permits coordinated multi-ECU updates without requiring cellular modems in each ECU, reducing cost and complexity.

A. CAN-Based Distribution

Controller Area Network operates at 125 Kbps to 1 Mbps for Classical CAN with maximum 8-byte payload per frame. CAN-FD (CAN with Flexible Data-Rate) extends payload to 64 bytes and data phase speed to 2-5 Mbps while maintaining arbitration phase at Classical CAN speeds for backward compatibility with existing ECUs. Physical layer uses differential signaling on twisted pair cabling (CAN_H and CAN_L) with 120Ω termination resistors at both ends preventing signal reflections.

Update distribution over CAN uses UDS (Unified Diagnostic Services) protocol standardized in ISO 14229 or manufacturer-specific diagnostic protocols. Typical transfer rates achieve 15-30 KB/s for Classical CAN and 50-100 KB/s for CAN-FD accounting for protocol overhead including: service headers identifying request/response type, sequence counters tracking block ordering, acknowledgment frames confirming receipt, and flow control frames managing transfer pacing. A 2MB ECU flash image requires 60-120 seconds transfer time over Classical CAN or 20-40 seconds over CAN-FD, excluding installation duration (30-90 seconds for flash write and verification).

Artifacts are chunked into transfer blocks matching UDS block sequence counter limits (typically 256 blocks per transfer sequence requiring sequence counter rollover or transfer segmentation for large images) and CAN frame payload constraints. Each block contains: sequence counter for ordering (1-2 bytes), data payload (4-6 bytes per Classical CAN frame after protocol overhead, 50-60 bytes for CAN-FD), and block checksum for corruption detection. Transfer interruption mid-block requires retransmission from block start as partial blocks cannot be validated or stored.

Bus load management prevents update traffic from disrupting real-time control messages sharing the same physical CAN bus. Gateway monitors bus utilization calculated as (active transmission time / total time) averaged over sliding window (typically 1 second) and throttles update transfer when utilization exceeds configurable threshold (70-80% typical). Critical messages including brake requests (arbitration ID 0x100-0x1FF priority range), steering commands (0x200-0x2FF), and throttle control (0x300-0x3FF) use higher priority arbitration IDs (numerically lower values win arbitration) ensuring transmission even under heavy bus load from background update traffic.

Multi-drop CAN topology means all ECUs connected to bus receive all messages. Update sessions use addressed messaging where source (gateway) and destination (target ECU) addresses appear in message payload. Only the intended ECU processes transfer frames; others ignore them after arbitration ID filtering by CAN controller hardware. Session management includes: authentication via seed-key exchange where gateway sends random seed and ECU returns cryptographic hash proving key possession, session timeout monitoring with configurable inactivity limits (5-30 seconds typical), and keep-alive messaging during long transfers preventing session termination during multi-megabyte flashing operations lasting several minutes.

B. Ethernet-Based Distribution

Automotive Ethernet operates at 100 Mbps (100BASE-T1) or 1 Gbps (1000BASE-T1) over single unshielded twisted pair using PAM3 encoding (3-level pulse amplitude modulation) for 100 Mbps or PAM16 (16-level) for 1 Gbps achieving higher bandwidth than copper CAN while maintaining automotive-grade electromagnetic compatibility. Physical layer supports point-to-point links up to 15 meters (100BASE-T1) or 40 meters (1000BASE-T1) without repeaters. Switched Ethernet topologies use automotive-grade switches supporting IEEE 802.1Q VLANs for traffic segregation between safety-critical and non-critical data flows.

DoIP (Diagnostics over IP) encapsulates UDS diagnostic messages in TCP/IP or UDP/IP packets. TCP provides reliable ordered delivery with automatic retransmission for update transfers ensuring data integrity. UDP reduces overhead for status queries and keep-alive messaging where occasional packet loss is acceptable. Typical transfer rates achieve 5-10 MB/s accounting for TCP overhead (20-byte header per packet), IP fragmentation/reassembly for large transfers, and Ethernet frame headers (18-byte overhead for VLAN-tagged frames). A 10MB ECU flash transfers in 1-2 seconds plus installation time (30-60 seconds for flash write and verification operations).

Ethernet architectures support simultaneous multi-ECU updates through parallel TCP sessions. Gateway can update 4-8 ECUs concurrently depending on: network topology (daisy-chain topology creates bandwidth bottlenecks as downstream traffic flows through upstream links; star topology provides independent bandwidth per ECU), switch port count and backplane capacity, and available bandwidth after subtracting safety-critical traffic reservations. Concurrent updates reduce total campaign duration from sequential hours (10 ECUs × 2 minutes each = 20 minutes minimum) to parallel tens of minutes (10 ECUs in two batches of 5 = 4 minutes minimum plus orchestration overhead).

Quality of Service (QoS) mechanisms prioritize traffic classes per IEEE 802.1Q priority code point (PCP) field in VLAN tags. ADAS sensor data (camera, radar, lidar) and vehicle control messages receive highest priority (PCP 7/6 mapping to hardware queue 7 with strict priority scheduling). Infotainment and diagnostic traffic use medium priority (PCP 3/4). Update traffic uses lowest priority (PCP 1/2) preventing interference with safety-critical traffic during network congestion. Switches implement weighted round-robin or weighted fair queuing scheduling ensuring lower-priority traffic receives bandwidth when higher-priority queues are empty.

Time-Sensitive Networking (TSN) extensions provide deterministic latency for critical traffic while allowing best-effort update delivery during scheduled time windows. TSN gate control lists (GCL) schedule transmission windows: gates open for high-priority traffic during reserved time slots (e.g., 0-500 microseconds per 1ms cycle), gates close blocking lower-priority traffic during critical periods, gates open for best-effort traffic during remaining time. Update traffic transmits during best-effort windows (e.g., 600-1000 microseconds per cycle) accumulating bandwidth over many cycles. This ensures deterministic real-time behavior for vehicle control while permitting background update distribution without disrupting safety-critical operations.

C. Gateway Architecture

Vehicle gateways or telematics control units (TCU) implement critical infrastructure for OTA update distribution:

- Cellular connectivity: LTE Cat-1 (10 Mbps downlink, 5 Mbps uplink) provides basic telematics and update delivery for cost-sensitive applications. LTE Cat-M1 (1 Mbps downlink/uplink) optimizes for power consumption in battery-powered or hybrid vehicles. LTE Cat-4+ (150 Mbps downlink, 50 Mbps uplink) supports connected services including streaming media, real-time navigation, and over-the-air updates. 5G NR (New Radio) provides ultra-low latency (<10ms) and higher bandwidth (multi-Gbps) for future applications. Cellular data costs typically $2-5 per GB depending on carrier contracts, volume pricing, and geographical region, necessitating optimization through delta updates (60-90% size reduction) and compression algorithms (30-50% size reduction) reducing total data transfer.
- WiFi connectivity: 802.11ac (WiFi 5) provides 433-1300 Mbps using 80 MHz channels and MIMO. 802.11ax (WiFi 6) extends to 600-9600 Mbps using OFDMA and improved spectral efficiency. WiFi connectivity supports garage or dealer updates avoiding cellular data costs. Home WiFi permits full-image transfers (100MB-1GB+) impractical over cellular due to cost ($200-$5000 per vehicle for 1GB transfer) and time constraints (10-30 minutes at LTE Cat-1 speeds). Dual-band support (2.4 GHz for range, 5 GHz for throughput) provides connection redundancy and interference avoidance in congested RF environments.
- Storage: eMMC (embedded MultiMediaCard) 8-64 GB provides flash storage integrated on system-on-chip packages. UFS 2.1/3.0 (Universal Flash Storage) 16-128 GB offers higher performance (500-1000 MB/s sequential read, 200-500 MB/s write vs. eMMC 200-300 MB/s read, 50-150 MB/s write). Storage must accommodate: staged artifacts for multiple ECUs awaiting installation (10-50 GB depending on vehicle ECU count), operating system and middleware (1-5 GB), application software including maps and media (10-40 GB), and logged diagnostic data (1-5 GB). Wear leveling implemented by eMMC/UFS

controllers extends flash lifetime under frequent update write cycles (10,000-100,000 program-erase cycles per block for MLC NAND).

• Processing: ARM Cortex-A53/A72 quad-core or octa-core processors (1.0-2.0 GHz) provide computational capability for: signature verification executing RSA-2048 or ECDSA-P256 operations (50-200ms per signature on software implementation, 10-50ms with crypto accelerators), delta reconstruction executing binary differencing algorithms (xdelta, bsdiff consuming 2-3x RAM of output size), artifact decompression executing LZMA or Brotli algorithms (10-50 MB/s decompression throughput), and multi-ECU orchestration logic managing parallel update sessions. Cryptographic accelerators (ARM Crypto Extensions providing AES, SHA, and public-key operation instructions; dedicated crypto coprocessors) reduce signature verification time and CPU utilization enabling concurrent ECU updates without performance degradation.

• Network interfaces: CAN controllers (2-4 channels operating at 250 Kbps or 500 Kbps Classical CAN, 500 Kbps arbitration with 2-5 Mbps data phase for CAN-FD), Automotive Ethernet MAC/PHY (1-2 ports operating at 100BASE-T1 or 1000BASE-T1), LIN (Local Interconnect Network) controllers optional for low-speed sensors (19.2 Kbps single-wire protocol), FlexRay controllers optional on luxury vehicles (10 Mbps dual-channel time-triggered protocol). Gateway function bridges between network domains translating protocols and managing network timing ensuring messages from one domain forward correctly to others while maintaining real-time constraints.

IX. POLICY AND RULE MODEL

The architecture distinguishes policies (cloud-defined intent) from rules (device-executable predicates). This separation permits policy reuse across product families while adapting to platform-specific capabilities.

A. Policy Definitions

Policies describe update intent at abstraction levels comprehensible to operators and auditors without specifying implementation details. Examples: "Updates shall not install while vehicle is in motion", "HVAC firmware updates require stable AC power supply", "Installation shall occur only between 02:00-05:00 local time". Policy documents use JSON or CBOR format with versioning supporting evolution over product lifetime.

B. Rule Implementations

Rules translate policies into executable predicates. Vehicle motion policy maps to: query wheel speed sensors via CAN PIDs 0xB4-0xB7, verify all four sensors report < 0.5 km/h. Rules execute locally without cloud connectivity. Evaluation frequency depends on priority: safety rules every 30-60 seconds, resource rules every 5-15 minutes. Rule results include boolean decision, reason code, context values, and timestamp.

C. Precedence Resolution

Priority ordering: Safety (1000-1999) → Security (2000-2999) → Resource (3000-3999) → Dependency (4000-4999) → Policy (5000-5999). Higher priority rules override lower priority. Safety rules have absolute veto; single failure blocks execution. Within priority class: blocking failures prevent execution, warnings permit with notification, informational provides context.

D. Policy Distribution and Versioning

Policy documents distribute separately from firmware using same cryptographic signing. Policy updates occur more frequently (monthly) than firmware (quarterly) supporting operational flexibility. Devices maintain version history for compliance audit trails. Semantic versioning: major changes require device updates, minor add optional fields, patch correct defects.

X. SECURITY ARCHITECTURE

Defense-in-depth through multiple independent mechanisms at different layers prevents single points of failure.

A. Secure Boot Foundation

Boot chain anchored in hardware roots of trust: immutable boot ROM verifies bootloader signature using manufacturer public key; bootloader verifies kernel; kernel verifies applications. Each stage refuses execution on verification failure. Hardware roots vary: ARM TrustZone, Intel Boot Guard, dedicated security coprocessors (ST33, NXP EdgeLock). Tamper detection monitors voltage, clock, temperature; detected tampering triggers key erasure.

B. Artifact Signing and Verification

RSA-PSS (2048/3072-bit) or ECDSA (P-256/P-384) signatures generated in FIPS 140-2 Level 3 HSMs. Multi-person control requires quorum of authentication tokens (typically 3 of 5). Devices validate signatures before staging and before installation. Verification failures generate detailed error codes distinguishing signature mismatch, expired certificates, revoked keys.

C. Key Management

Key lifecycle: Generation in HSMs using certified RNG per NIST SP 800-90A. Distribution via X.509 certificates during manufacturing. Rotation annually or after 10,000-50,000 signatures with 30-90 day transition accepting both keys. Revocation via CRLs or OCSP with fail-secure policy rejecting unverifiable certificates after 30-day grace period. Emergency revocation broadcasts through multiple channels.

D. Deployment Intent Authentication

Signed deployment intent prevents unauthorized campaigns and tampering. Intent includes: target versions, constraints, rollout strategy, dependencies, signatures. Anti-replay: timestamps (7-30 day validity), 128-bit nonces, sequence numbers. Devices maintain persistent nonce cache (1000-10,000 entries, LRU eviction) detecting replay attempts.

E. Secure Communication

TLS 1.3 with mandatory forward secrecy, 1-RTT handshake, strong cipher suites only. Certificate pinning prevents CA compromise attacks. Mutual TLS: server certificates verify cloud authenticity, device certificates (provisioned during manufacturing) verify device authenticity eliminating password vulnerabilities.

XI. TESTING AND VALIDATION

Systematic testing addresses failure modes at component, integration, system, and field levels.

A. Power Loss Testing

- Deliberate power interruption at each phase validates recovery:
- Download phase - resume from last block using HTTP Range.
- Staging phase - discard partial staging, retry.
- Verification phase - repeat (idempotent).
- Installation phase - boot from backup, detect in-progress state, resume or abandon.
- Commit phase - two-phase commit ensures atomic update or rollback.
- Test battery states: sudden disconnection, gradual decay, voltage drop. Automotive: load dump, cold cranking, reverse polarity.

B. Network Interruption Testing

Network loss scenarios:

- Brief outages (1-5 seconds) - TCP retransmission.
- Extended offline (hours/days) - abandon transfer after timeout, cache intent, resume on reconnection.
- Intermittent connectivity - progressive download over multiple windows.
- Measure download efficiency: >95% bytes downloaded once.
- Simulate packet loss (0-5%), latency (50-500ms), bandwidth throttling (128 Kbps-10 Mbps).

XII. ARCHITECTURAL RECOMMENDATION MODEL

Decision framework maps system characteristics to governance patterns:

System Type	Architecture Pattern
Consumer IoT	Simplified governance, basic policy, cloud-initiated, device signature verification, single-bank flash with factory reset, minimum authentication, 1%→10%→100% over 3-7 days
Industrial IoT	Dual governance, local policy engine, time-windowed per production schedules, IEC 62443 authorization, dual-bank with auto-rollback, mutual TLS, 1%→5%→25%→100% over 7-14 days
Automotive IoT	Full distributed governance per ISO 26262, multi-layer safety gating, AUTOSAR multi-ECU orchestration, dual-bank + golden image, hardware root of trust, 1%→5%→25%→100% over 14-30 days with HIL
Network-Constrained	Compressed artifacts (30-50% reduction), delta updates (60-90% reduction), resumable transfers with block checksum, CDN with geographic optimization, cellular cost optimization via WiFi fallback
Safety-Critical Real-time	Strict operational gating (motion=0, no faults), dual-bank mandatory, auto-rollback after 3 boot failures, 30-60 second stability check, installation during scheduled maintenance only
Highly Regulated	Maker-checker workflows, comprehensive audit logging, third-party validation, evidence packages for compliance, retention per jurisdiction (5-10 years)

TABLE II. ARCHITECTURE PATTERN BASED ON SYSTEM TYPE

A. Implementation Trade-offs

Security vs. Convenience: Strong controls increase latency but reduce risk. Storage vs. Bandwidth: Delta updates reduce bandwidth but require more storage/CPU. Autonomy vs. Control: Local decisions enable offline operation but reduce cloud control. Speed vs. Safety: Rapid deployment addresses vulnerabilities quickly but increases validation risk.

B. Scaling Considerations

At scale, 0.1% failure rate across 1M devices = 1,000 manual recoveries. Infrastructure requirements: Peak download (10% concurrent × 1M devices × 50MB = 500GB burst requiring 100-500 Gbps CDN). Telemetry ingestion (1M devices × 10 updates/hour / 3600s = 2,778 msg/s average, 5,000-10,000 peak). Campaign orchestration requires optimized queries (indexed, materialized views, time-series databases).

C. Resource Requirements by Governance Depth

System resource requirements scale with governance depth complexity. The following table quantifies typical storage, bandwidth, and computational requirements enabling architects to size hardware platforms and estimate operational costs for different governance patterns.

Governance Depth	Flash Storage	Network Bandwidth	CPU Requirements
Level 0 (Minimal)	1.5× artifact (single bank + staging buffer). Example: 2MB firmware = 3MB total	Full artifact download. 2G/3G sufficient (100-500 kbps)	ARM Cortex-M0+ (48 MHz). RSA verify: 200-500ms. No crypto accel needed
Level 1 (Basic)	2× artifact (dual bank A/B). Add 64KB for metadata, state, logs. Example: 2MB firmware = 4.1MB total	Delta updates possible. 3G/4G recommended (500kbps-2Mbps). Reduces data 40-70%	ARM Cortex-M4 (168 MHz). RSA verify: 50-200ms. Basic policy eval: 10-50ms
Level 2 (Standard)	2.5× artifact (A/B + delta workspace). Add 256KB metadata, recovery, audit. Example: 8MB firmware = 20.3MB total	4G/LTE required (2-10 Mbps). Resumable downloads. Compressed artifacts	ARM Cortex-A5/A7 (400MHz-1GHz). Complex policy: 100-500ms. Multi-component orchestration
Level 3 (Advanced)	3× artifact (multiple components, rollback images). Add 512KB-1MB comprehensive logging. Example: 16MB firmware = 49MB total	4G/5G + WiFi offload (10-100 Mbps). Parallel downloads. CDN edge caching	ARM Cortex-A53+ (1-2 GHz quad-core). Crypto accel required. Gateway orchestration, dependency graphs
RAM Requirements	Level 0: 64-128KB	Level 1: 128-512KB	Level 2: 1-4MB. Level 3: 8-32MB

TABLE III. GOVERNANCE DEPTH

Notes: Storage requirements include decompression workspace (LZMA compressed artifacts expand 2-3× during processing requiring temporary RAM buffers). Delta updates using xdelta/bsdiff require RAM = 2-3× output size during reconstruction. Automotive gateways orchestrating 10-50 ECU updates require additional buffer pools (10-50MB) for queuing artifacts and managing concurrent sessions. Production systems should provision 20-30% safety margin above calculated minimums accommodating growth and unexpected scenarios

XIII. COMPARATIVE ANALYSIS WITH EXISTING OTA SOLUTIONS

This section contextualizes the proposed framework against established commercial and open-source OTA update solutions, highlighting architectural distinctions, feature coverage, and implementation trade-offs. The analysis focuses on solution categories addressing embedded automotive and IoT domains rather than mobile device management or enterprise software deployment systems.

A. Automotive-Focused Solutions

Uptane [6] provides secure software update framework specifically designed for automotive systems. Strengths include two-repository architecture (Director repository for targeting, Image repository for artifacts) preventing mix-and-match attacks, time-server integration for secure timestamps, and formal security analysis. Uptane focuses primarily on signature verification and repository compromise resistance but provides limited guidance on operational policy enforcement, multi-ECU dependency orchestration, or recovery mechanisms. The proposed framework extends Uptane's security foundation with comprehensive governance including operational safety gating, resource constraint enforcement, and production-validated failure recovery patterns.

Commercial automotive solutions from Tier-1 suppliers (Continental, Bosch, Aptiv) provide end-to-end OTA platforms integrated with vehicle architectures. These systems implement proprietary approaches to gateway orchestration, ECU update sequences, and diagnostic integration. Common architectural patterns include telematics control unit (TCU) serving as update gateway, UDS-based diagnostic protocols for ECU communication, and AUTOSAR-compliant software update interfaces. Limitations include vendor lock-in, limited cross-manufacturer interoperability, and constrained ability to customize policy enforcement for specific deployment scenarios. The proposed framework provides manufacturer-agnostic architectural patterns enabling integration across heterogeneous ECU suppliers while maintaining flexibility for OEM-specific policy requirements.

B. IoT and Embedded Solutions

SUIT (Software Updates for Internet of Things) framework [6], [41] defines manifest-based update procedures for constrained devices using CBOR encoding. SUIT manifests specify component versions, dependencies, installation sequences, and security credentials in compact binary format suitable for resource-limited processors. The framework excels in artifact metadata representation and provides standardized manifest format enabling cross-vendor compatibility. However, SUIT focuses on manifest structure rather than cloud-to-device governance architectures, policy evaluation engines, or operational safety constraints critical for safety-critical applications. The proposed framework complements SUIT by providing architectural context for policy-driven deployment intent, fleet management, and failure recovery absent from SUIT specifications.

The Update Framework (TUF) [43] addresses software supply-chain security through role-based key management, threshold signatures, and delegation mechanisms. TUF prevents freeze attacks (serving old metadata), mix-and-match attacks (combining incompatible components), and repository compromise through sophisticated cryptographic protocols. Originally designed for general-purpose software repositories (Python packages, Docker images), TUF has limited adoption in embedded automotive domains due to complexity overhead and metadata size incompatible with constrained devices. The proposed framework incorporates TUF's security principles (threshold signing, role separation) while optimizing for embedded constraints through simplified metadata structures and device-resident policy evaluation.

Android A/B system updates [27] implement seamless update pattern using dual system partitions enabling background installation without user interruption. A/B updates provide robust recovery through partition fallback but require double storage capacity (complete duplicate system image) infeasible for storage-constrained automotive ECUs. The proposed framework presents alternative transactional patterns (golden image recovery, differential updates) balancing reliability against storage constraints typical in automotive and industrial embedded systems.

C. Cloud Platform Services

AWS IoT Device Management, Azure IoT Hub Device Update, and Google Cloud IoT Core provide managed OTA services integrated with respective cloud ecosystems. These platforms offer artifact hosting, campaign management interfaces, fleet monitoring dashboards, and integration with device authentication services. Strengths include operational scalability, global CDN distribution, comprehensive logging and analytics, and integration with cloud-native security services (HSMs, certificate authorities, identity management). Limitations include vendor lock-in, requirement for constant cloud connectivity limiting autonomous device operation, and limited customization of policy enforcement logic beyond basic targeting rules. The proposed framework provides cloud-agnostic architectural patterns enabling organizations to implement equivalent functionality across cloud providers or on-premises infrastructure while maintaining device autonomy critical for intermittent connectivity scenarios.

D. Framework Positioning and Contributions

The proposed framework distinguishes itself through integration of security foundations (Uptane, TUF), IoT standards (SUIT), automotive specifications (AUTOSAR), and regulatory requirements (ISO 26262, ISO/SAE 21434, UNECE WP.29 [42]) into unified architectural patterns. Rather than presenting new cryptographic protocols or manifest formats, the framework synthesizes established standards with operational patterns validated across production systems managing millions of devices. Key differentiators include: (1) Dual governance model separating cloud deployment intent from device execution control enabling autonomous operation under intermittent connectivity. (2) Policy-driven constraint enforcement addressing operational safety, resource limitations, and regulatory compliance through extensible rule engine. (3) Two-hop distribution architecture for multi-ECU vehicles respecting in-vehicle network topology and security boundaries. (4) Comprehensive failure taxonomy and reason code system enabling fleet-wide analytics and systematic improvement. (5) Architectural recommendation model providing decision framework mapping system characteristics to appropriate governance depth.

The framework serves as implementation guide for system architects designing update systems requiring balance of security, safety, operational flexibility, and regulatory compliance. It complements rather than replaces existing solutions: organizations may implement Uptane signatures with framework's governance patterns, use SUIT manifests within framework's policy enforcement, or deploy framework patterns on commercial cloud platforms. This architectural layering enables selective adoption based on system requirements and existing infrastructure investments.

XIV. CONCLUSION

Software update architectures for embedded automotive and consumer systems require treatment as distributed governance platforms balancing fleet-level control against device-autonomous execution safety. This paper presented a policy-driven architectural framework synthesizing two decades of implementation experience across automotive infotainment platforms, manufacturing IoT systems, and consumer connected products. The framework grounds architectural decisions in established standards including ISO 26262 functional safety, ISO/SAE 21434 cybersecurity, AUTOSAR specifications, and NIST SP 800-193 firmware resiliency guidelines.

Core contributions address the distributed governance challenge through several architectural patterns. Dual governance layers separate cloud-defined deployment intent from device-resident execution control, permitting autonomous operation under intermittent connectivity while maintaining fleet-level oversight. Cryptographically signed deployment intent documents encode targeting metadata, execution constraints, and authorization proofs aligned with ISO/SAE 21434 requirements. Rule-based execution engines evaluate operational prerequisites including vehicle motion, battery state, thermal limits, and dependency satisfaction without requiring cloud communication. Transactional state machines manage lifecycle transitions through download, staging, verification, installation, and commit phases with recovery capabilities addressing power loss and system faults at each transition point.

The two-hop distribution architecture addresses multi-ECU vehicle systems where interior ECUs lack direct internet connectivity. Cloud-to-gateway communication uses cellular or WiFi networks while gateway-to-ECU communication uses CAN or Automotive Ethernet protocols. This architecture protects ECUs from direct internet exposure while supporting coordinated multi-ECU updates respecting dependency relationships defined in AUTOSAR specifications.

Policy schema specifications and reason code taxonomies provide standardized mechanisms for expressing update constraints and reporting execution decisions. Policy documents encode constraints including time windows, operational prerequisites, and safety requirements in platform-independent JSON or CBOR

format. Reason codes organized into safety, security, resource, dependency, policy, and execution categories enable fleet-wide analytics identifying systematic issues requiring policy refinement or engineering investigation.

The architectural recommendation model guides system architects through decision frameworks mapping operational constraints to appropriate governance patterns. Consumer IoT devices may implement simplified variants focusing on cryptographic verification and basic recovery mechanisms. Automotive systems require full governance including comprehensive safety gating, multi-ECU dependency orchestration, and regulatory audit trails. Industrial systems balance these extremes with time-windowed updates respecting production schedules and IEC 62443 compliance requirements.

The framework demonstrates applicability across domains sharing common requirements: safety-critical operational states, intermittent connectivity, heterogeneous hardware platforms, and regulatory compliance obligations. Modular architecture permits selective implementation based on system requirements. Consumer devices operating under lower safety criticality can adopt simplified governance reducing implementation complexity and operational overhead. Safety-critical automotive and industrial applications justify full framework implementation including comprehensive validation, audit logging, and maker-checker approval workflows.

Framework limitations include reliance on accurate operational state telemetry where sensor failures may provide incorrect safety assessments. Multi-ECU dependency management increases complexity in vehicles exceeding 100 ECUs requiring sophisticated graph-based orchestration algorithms. Comprehensive audit logging and maker-checker workflows introduce operational overhead including policy management effort, approval workflow latency, and telemetry infrastructure costs. These limitations represent acceptable trade-offs given safety and security benefits in critical applications where update failures risk physical harm, regulatory non-compliance, or security compromise.

Future research directions include integration with V2X communication enabling coordinated fleet behavior during updates. Vehicles could communicate update status to nearby vehicles preventing simultaneous updates that might cause traffic hazards or emergency response delays. Machine learning techniques could analyze historical reason code patterns predicting devices likely to fail updates before campaign execution, enabling preemptive remediation or targeted exclusion from rollout populations.
Secure multi-vendor update ecosystems present architectural challenges requiring trust models supporting third-party software distribution. Vehicle manufacturers must permit supplier-specific signing keys and authorization policies while maintaining oversight of fleet-wide update operations. Blockchain-based audit trails could provide tamper-evident update history supporting regulatory compliance demonstrations and incident investigations without centralized audit infrastructure vulnerable to manipulation.

GLOSSARY

- ASIL – Automotive Safety Integrity Level, classification scheme defined in ISO 26262 ranging from ASIL A (lowest) to ASIL D (highest) indicating required safety measures for automotive systems.
- AUTOSAR – AUTomotive Open System ARchitecture, standardized software architecture for automotive electronic control units providing specifications for software update mechanisms and diagnostic protocols.
- CAN – Controller Area Network, robust vehicle bus standard allowing microcontrollers and devices to communicate without a host computer, widely used in automotive applications.
- CBOR – Concise Binary Object Representation, data format providing compact serialization for constrained environments, used in IoT firmware update manifests per RFC 9019.

- CMS – Cryptographic Message Syntax, standard syntax for digitally signing, digesting, authenticating, or encrypting arbitrary message content per RFC 5652, used for secure artifact packaging.
- DoIP – Diagnostic over IP, AUTOSAR protocol enabling diagnostic services over IP-based networks including Ethernet, extending traditional CAN-based diagnostics.
- ECU – Electronic Control Unit, embedded system in automotive vehicles that controls electrical systems or subsystems, modern vehicles contain 50-150 ECUs.
- ECDSA – Elliptic Curve Digital Signature Algorithm, cryptographic signature scheme using elliptic curve cryptography offering equivalent security to RSA with smaller key sizes (256-bit ECDSA ≈ 3072-bit RSA).
- FOTA – Firmware Over-The-Air, remote firmware update mechanism delivering software updates wirelessly without physical device access, subset of OTA updates focusing on low-level firmware.
- HSM – Hardware Security Module, tamper-resistant hardware device providing cryptographic key generation, storage, and operations with physical security protections, used in cloud signing services.
- ISO 26262 – International standard for functional safety of electrical/electronic systems in production automobiles, defining systematic approach to development of safety-critical automotive systems.
- ISO/SAE 21434 – International standard for cybersecurity engineering of road vehicles, establishing requirements for threat analysis, risk assessment, and secure development throughout vehicle lifecycle.
- IoT – Internet of Things, network of physical objects embedded with sensors, software, and connectivity enabling data exchange, requiring secure update mechanisms for lifecycle management.
- JWS – JSON Web Signature, standard for digitally signing JSON payloads using JOSE (JSON Object Signing and Encryption) framework per RFC 7515.
- NIST SP 800-193 – Platform Firmware Resiliency Guidelines, NIST special publication defining requirements for firmware protection, detection, and recovery in computing platforms.
- OTA – Over-The-Air, wireless delivery of software updates, configuration changes, or encryption keys to devices without physical access, critical capability for connected products.
- PKI – Public Key Infrastructure, set of roles, policies, hardware, software, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates.
- RSA – Rivest-Shamir-Adleman, widely-used asymmetric cryptographic algorithm for digital signatures and encryption, common key sizes include 2048-bit and 4096-bit for signature operations.
- SUIT – Software Updates for Internet of Things, IETF framework defining manifest format and update procedures for constrained IoT devices per RFC 9124.
- TLS – Transport Layer Security, cryptographic protocol providing secure communication over networks, version 1.3 (RFC 8446) recommended for OTA update transport security.
- TCU – Telematics Control Unit, vehicle ECU providing wireless connectivity services including cellular data, GPS, and WiFi, typically serves as update gateway in two-hop architectures.
- TOCTOU – Time-Of-Check to Time-Of-Use, race condition vulnerability where system state changes between security check and resource use, mitigated through defense-in-depth verification.
- TrustZone – ARM security technology providing hardware-enforced isolation between secure and normal execution environments on ARM processors, enabling secure boot and key storage.
- UDS – Unified Diagnostic Services, diagnostic communication protocol used in automotive ECUs per ISO 14229, supporting firmware update and diagnostic operations over CAN and other networks.
- V2X – Vehicle-to-Everything, communication systems allowing vehicles to communicate with infrastructure, other vehicles, pedestrians, and networks, requiring coordination during software updates.
- VIN – Vehicle Identification Number, unique 17-character code assigned to each vehicle, used for targeting specific vehicles or vehicle populations in update campaigns

REFERENCES:

1. ISO 26262:2018, Road vehicles — Functional safety. International Organization for Standardization, 2018.
2. ISO/SAE 21434:2021, Road vehicles — Cybersecurity engineering. International Organization for Standardization / Society of Automotive Engineers, 2021.
3. AUTOSAR, "Specification of Software Update," AUTOSAR Classic Platform Release 4.4.0, 2020.
4. NIST Special Publication 800-193, "Platform Firmware Resiliency Guidelines," National Institute of Standards and Technology, May 2018.
5. J. Karthikeyan, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch, and J. Cappos, "Uptane: Securing Software Updates for Automobiles," in Proc. 14th Int. Conf. Embedded Security in Cars (escar), 2016.
6. H. Birkholz, C. Vigano, and C. Bormann, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest," RFC 9019, IETF, April 2022.
7. R. Housley, "Cryptographic Message Syntax (CMS) Algorithms for RSA PKCS #1 and Diffie-Hellman," RFC 4108, IETF, June 2005.
8. IEEE 802.3, "IEEE Standard for Ethernet," Institute of Electrical and Electronics Engineers, 2018.
9. ISO 11898-1:2015, Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signaling. International Organization for Standardization, 2015.
10. AUTOSAR, "Specification of Diagnostic over IP (DoIP)," AUTOSAR Classic Platform Release 4.4.0, 2020.
11. IEC 62443-3-3:2013, Industrial communication networks — Network and system security — Part 3-3: System security requirements and security levels. International Electrotechnical Commission, 2013.
12. J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, "Package Management Security," Technical Report TR2008-02, University of Arizona, 2008.
13. C. Percival, "Naive differences of executable code," 2003. [Online]. Available: http://www.daemonology.net/bsdiff/
14. J. MacDonald, "xdelta: A binary delta encoding algorithm," 2016. [Online]. Available: https://github.com/jmacd/xdelta
15. E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, IETF, August 2018.
16. C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," Black Hat USA, 2015.
17. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, IETF, May 2008.
18. ARM Limited, "ARM Security Technology - Building a Secure System using TrustZone Technology," ARM Technical White Paper, 2009.
19. M. Jones, J. Bradley, and N. Sakimura, "JSON Web Signature (JWS)," RFC 7515, IETF, May 2015.
20. IEEE 802.1, "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications," IEEE 802.1AS-2020, 2020.
21. C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and Other Botnets," IEEE Computer, vol. 50, no. 7, pp. 80-84, July 2017.
22. C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," RFC 7228, IETF, May 2014.

23. Google, "A/B (Seamless) System Updates," Android Open Source Project documentation, 2016. [Online]. Available: https://source.android.com/devices/tech/ota/ab
24. L. Kohnfelder and P. Garg, "The threats to our products," Microsoft Interface, Microsoft Corporation, April 1999.
25. R. Fielding, Y. Lafon, and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Range Requests," RFC 7233, IETF, June 2014.
26. S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in Proc. USENIX Security Symposium, 2011.
27. K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," in Proc. IEEE Symposium on Security and Privacy, 2010.
28. Cui and S. J. Stolfo, "A Quantitative Analysis of the Insecurity of Embedded Network Devices: Results of a Wide-Area Scan," in Proc. Annual Computer Security Applications Conference (ACSAC), 2010.
29. National Highway Traffic Safety Administration, "Cybersecurity Best Practices for Modern Vehicles," NHTSA Report DOT HS 812 333, October 2016.
30. Society of Automotive Engineers, "Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," SAE J3061, January 2016.
31. R. D. Pete, T. Garcia, and R. Tohidi, "Unified Diagnostic Services (UDS) Explained - A Controller Area Network Protocol," CSS Electronics White Paper, 2019.
32. Vector Informatik GmbH, "Security in Automotive Ethernet," Vector Technical Article Series, 2018.
33. Mukherjee, Z. Nejiati, and C. Runyon, "Security Challenges for Automotive Firmware Over-The-Air Updates," IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 8, pp. 11734-11748, August 2022.
34. H. Seudié, N. Coppola, and R. Fuchs, "Practical Considerations for Automotive Over-the-Air Software Updates," in Proc. IEEE International Conference on Connected Vehicles and Expo (ICCVE), pp. 1-6, 2021.
35. Y. Zhang, A. Kasahara, and J. Tan, "Secure Over-the-Air Software Updates in Connected Vehicles: A Survey," IEEE Access, vol. 8, pp. 214303-214319, 2020.
36. M. Aliwa, M. Cheminod, L. Durante, and A. Valenzano, "A Comprehensive Review on Automotive OTA Update Security," Journal of Systems Architecture, vol. 127, Article 102508, June 2022.
37. H. Moran, B. Tschofenig, and H. Birkholz, "A Firmware Update Architecture for Internet of Things," RFC 9019, IETF, April 2021.
38. T. Perner, F. Kargl, and S. Dietzel, "Secure Software Updates for Vehicles: A Survey," in Proc. ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec), pp. 189-200, 2023.
39. S. Dhillon, A. Al-Haddad, A. Tarrad, and L. Smalov, "AUTOSAR-Compliant Secure Boot and Firmware Update Framework," IEEE Transactions on Vehicular Technology, vol. 72, no. 3, pp. 2941-2955, March 2023.
40. L. Petit and R. Mayrhofer, "End-to-End Security for Automotive Software Updates using Updatable Public Keys," in Proc. ACM/IEEE Cyber-Physical System Security Workshop (CPSS), pp. 21-30, 2022.
41. N. Kobeissi, G. Nicolas, and K. Bhargavan, "Noise Explorer: Fully Automated Modeling and Verification for Arbitrary Noise Protocols," in Proc. IEEE European Symposium on Security and Privacy (EuroS&P), pp. 356-370, 2020.

42. UNECE, "UN Regulation No. 155 - Uniform provisions concerning the approval of vehicles with regards to cyber security and cyber security management system," World Forum for Harmonization of Vehicle Regulations, January 2021.
43. Sullivan, O. Haran, and E. Shay, "The Update Framework: Securing Software Distribution," in Proc. USENIX Security Symposium, pp. 1-16, 2020