International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Configuration-Driven OTA Firmware Update Orchestration for Scalable ECU Management in Software-Defined Vehicles

Utkarshkumar Shah

Abstract: The rapid proliferation of electronic control units (ECUs) in modern software-defined vehicles has fundamentally transformed how automotive manufacturers deliver and manage software functionality. Over-the-air (OTA) firmware update mechanisms have emerged as a critical infrastructure component enabling remote diagnostics, feature additions, and security patching without physical dealer intervention. However, the heterogeneous nature of ECU bootloader strategies, memory architectures, and security access requirements imposes significant engineering complexity on scalable OTA orchestration. This paper presents a configuration-driven OTA firmware update orchestration framework in which ECU-specific update sequences are encoded in YAML-defined configuration schemas, eliminating hard-coded, device-specific C implementations that impede fleet scalability. The proposed architecture decouples update logic from ECU hardware specifics by centralizing policy definitions within cloud-managed configuration payloads, enabling dynamic enumeration, firmware validation, and post-update verification across distributed ECU networks. A key contribution of the framework is its cloud-initiated ECU authentication mechanism, wherein firmware-side validation sequences are dynamically triggered upon cloud requests, transmitting verified inventory data to ensure vehicle authenticity prior to update execution. The framework leverages Unified Diagnostic Services (UDS) over CAN and Ethernet (DoIP) for diagnostic session management and memory transfer operations, and integrates with RTOS task scheduling to maintain deterministic update progression across any real-time platform, including but not limited to FreeRTOS-based and microcontroller environments. Experimental evaluation demonstrates significant improvements in update orchestration scalability, reduction in integration overhead, and consistent maintenance of firmware rollback safety guarantees across heterogeneous ECU populations. The results establish a principled foundation for scalable, maintainable OTA infrastructure in next-generation software-defined vehicle platforms.

Keywords: *Over-the-air update, ECU firmware, Software-defined vehicle, YAML configuration, UDS diagnostics, DoIP, FreeRTOS, RTOS, Automotive cybersecurity, Bootloader orchestration*

1. Introduction

The automotive industry is undergoing a fundamental architectural transition from hardware-centric vehicle design toward software-defined vehicle (SDV) paradigms in which digital functionality, safety features, and performance characteristics are governed by software layers that can be remotely updated throughout the vehicle lifetime [1]. This transition has accelerated the deployment of electronic control units across vehicle platforms, with modern vehicles now incorporating between 70 and 150 ECUs managing functions ranging from powertrain control and chassis dynamics to infotainment, advanced driver assistance systems, and battery management. As software complexity scales with ECU count, the ability to deliver reliable, secure, and efficient firmware updates remotely has become not merely a convenience but an operational necessity.

Over-the-air firmware update infrastructure addresses the prohibitive cost and logistical friction of physical software servicing. Traditional methods requiring dealer-side reprogramming are increasingly untenable for large vehicle fleets, particularly as security vulnerabilities demand rapid patch deployment and as feature differentiation between vehicle variants becomes managed entirely through software configuration rather than hardware substitution [2]. Industry analysts have noted that OTA-capable vehicles

Independent Researcher, USA

reduce recall costs substantially and enable manufacturers to monetize post-sale software features transforming the vehicle from a static product into a continuously evolving platform [3].

Despite these architectural advantages, scalable OTA firmware update orchestration remains technically challenging. The diversity of ECU bootloader implementations spanning proprietary boot sequences, varied security access seed-key algorithms, distinct memory layout requirements, and differing pre- and post-condition validation logic has historically forced developers to maintain ECU-specific hard-coded firmware update sequences in C. This approach severely constrains fleet scalability, increases codebase fragmentation, and introduces regression risk whenever new ECU variants are onboarded [4]. Furthermore, ensuring vehicle authenticity prior to update execution to prevent spurious or malicious updates from being applied to incorrect vehicle configurations requires robust firmware-side ECU inventory validation linked to cloud-orchestrated authentication workflows.

Configuration-driven architectures offer a principled solution to this challenge. By encoding ECU-specific update parameters, security access procedures, memory transfer sequences, and post-condition checks within structured configuration schemas such as YAML-defined documents managed through cloud portals the core update orchestration engine can be made hardware-agnostic. This separation of concerns between update policy and update mechanism enables fleet operators to onboard new ECU variants by defining configuration entries rather than modifying firmware source code, significantly reducing time-to-deployment and minimizing cross-ECU regression risk [5].

This paper presents the design, implementation, and evaluation of a configuration-driven OTA firmware update orchestration framework targeting scalable ECU management in software-defined vehicles. The framework integrates cloud-initiated authentication, YAML-based update sequence specification, UDS-over-CAN and DoIP (UDS over Ethernet) diagnostic communication, and RTOS task scheduling (compatible with any RTOS environment) to deliver deterministic, scalable, and maintainable OTA infrastructure. The remainder of this paper is organized as follows: Section 2 reviews related work in automotive OTA update architectures; Section 3 presents the system architecture and configuration schema design; Section 4 describes the firmware-side implementation including ECU enumeration and UDS integration; Section 5 presents evaluation results; Section 6 discusses security and safety considerations; and Section 7 concludes the paper.

2. Related Work

Research into automotive OTA update systems has intensified with the emergence of connected and software-defined vehicle architectures. Early treatments of the problem focused on the efficiency of firmware binary delivery, with work exploring delta-compression algorithms notably BSDIFF variants to reduce payload sizes transmitted over cellular links to vehicle head units [6]. These approaches addressed network bandwidth constraints but did not consider the orchestration complexity arising from heterogeneous ECU populations within a single vehicle or the authentication requirements of distributed update campaigns.

Security has become a dominant concern in recent OTA literature. Kim and Jeon demonstrated a multi-factor authentication protocol for in-vehicle OTA update channels, combining certificate-based server authentication with ECU-side session token verification over CAN-based transport layers to mitigate man-in-the-middle and replay attacks during firmware distribution [7]. Complementary work by Yeasmin and Haque introduced a blockchain-anchored OTA update framework for connected autonomous vehicles, leveraging distributed ledger immutability to ensure firmware package authenticity and prevent unauthorized rollback to vulnerable versions [8]. Lu et al. addressed the cryptographic overhead problem in resource-constrained ECUs by proposing LigSecOTA, a lightweight security integration that minimizes computational cost while preserving integrity guarantees across the OTA distribution chain [9].

Scalability dimensions of OTA infrastructure have received increasing attention as vehicle fleets grow. Shoker et al. identified cellular repository bottlenecks as a fundamental scalability constraint in current OTA architectures and proposed ScaIOTA, a peer-assisted distribution mechanism that reduces per-vehicle cellular load by enabling

vehicle-to-vehicle firmware segment propagation within proximity groups [10]. Bazzi, Shaout, and Ma analyzed variability management in automotive software update systems, framing the ECU heterogeneity problem as a software product line challenge and advocating for declarative update specification schemas to manage variant-specific update logic without combinatorial implementation overhead [11].

Architectural surveys by Blanco et al. on the Software-as-a-Service transformation of automotive systems identified service-oriented architectures and cloud-native orchestration layers as the enabling infrastructure for scalable SDV deployments, highlighting that configuration-driven ECU management represents a key design principle for next-generation vehicle platforms [3]. More recently, Agrawal et al. extended the SDV OTA framework concept by incorporating digital twin attestation using a cloud-resident virtual replica of the vehicle's ECU inventory to validate pre-update configurations before authorizing firmware distribution [12]. The present work builds upon this body of knowledge by implementing and evaluating a production-oriented YAML-driven orchestration engine that concretely addresses the ECU heterogeneity and authentication challenges identified across the literature.

3. System Architecture and Configuration Schema Design

The proposed OTA orchestration framework is structured around three principal layers: a cloud orchestration plane responsible for update campaign management and ECU authentication, a configuration schema layer encoding ECU-specific update policies, and a vehicle-resident firmware layer executing update sequences under real-time operating system control. This three-layer decomposition ensures that hardware-specific update logic remains entirely within the configuration schema, while the firmware execution engine operates as a generic, policy-agnostic update orchestrator.

The cloud orchestration plane exposes a campaign management portal through which fleet operators define update targets, schedule deployment windows, and monitor completion status across vehicle populations. When an update campaign is initiated for a specific vehicle, the cloud layer transmits a campaign manifest containing the target ECU identifiers, configuration schema references, firmware binary checksums, and authentication challenge parameters. The vehicle-resident orchestrator receives the manifest via a cellular-connected telematics control unit and triggers the ECU enumeration and authentication sequence before any firmware transfer begins.

The configuration schema is implemented as a structured YAML document organized by ECU identifier. Each schema entry specifies the bootloader entry sequence, security access seed-key algorithm identifier, memory erase and write transfer parameters, inter-step timing constraints, and post-update verification criteria. The schema also encodes conditional logic for pre-condition checks for example, requiring that vehicle ignition remain in a defined state or that battery voltage exceed a specified threshold before update initiation. By encoding these constraints declaratively, the orchestration engine can enforce them uniformly without ECU-specific firmware branches.

Schema versioning is managed through a configuration repository that tracks ECU variant identifiers, bootloader version mappings, and schema revision histories. When a new ECU variant is introduced for example, following a hardware revision or supplier change the onboarding process consists entirely of authoring a new YAML schema entry and validating it against a test vehicle, without any modification to the core firmware orchestration codebase. This architecture reduces new ECU integration effort from multiple engineering weeks to a configuration authoring and validation cycle measured in days.

The vehicle-resident firmware layer is implemented on a telematics control unit running an RTOS (e.g., FreeRTOS or any compatible real-time platform), with distinct tasks allocated for cloud communication, ECU enumeration, update sequence execution, and logging. Task priorities are configured to ensure that the update sequence execution task maintains deterministic scheduling guarantees even under concurrent background telemetry operations. Inter-task communication uses FreeRTOS queues and event groups, enabling the cloud communication task to signal update triggers and receive completion acknowledgements without shared-memory race conditions.

Configuration Parameter	Description	Example Values
ECU Identifier	Unique identifier for the target ECU	BCM_v2, ADAS_proc_v1, GWY_v3
Bootloader Entry Sequence	Ordered service calls to enter programming mode	DSC_extended → SA_unlock → DSC_programming
Security Access Algorithm ID	Identifier for pluggable seed-key module	SeedKey_AES128, SeedKey_XOR32
Memory Erase Block Size	Granularity of erase operation in bytes	512B, 1KB, 4KB
Transfer Block Size	Firmware write chunk size per UDS 0x36 call	256B, 512B, 1024B
Pre-condition: Battery Voltage	Minimum voltage threshold before update start	≥ 11.5V, ≥ 12.0V
Pre-condition: Vehicle Speed	Required vehicle state	0 km/h (stationary)
Pre-condition: Ignition State	Required ignition position	Key-On/Engine-Off, Accessory
Post-update Verification Method	Method used to confirm successful flashing	Checksum via Routine Control 0x31
Rollback Strategy	Fallback mechanism on failure	A/B partition swap, retry-and-escalate
Inter-step Timing Constraint	Wait time between sequential UDS operations	50ms, 200ms, 500ms
Schema Version	Configuration revision tracker	v1.4, v2.0

Table 1: ECU Update Policy Parameters Encoded in YAML Configuration Schema [1, 11]

3.1 System Architecture Overview

Figure 1 illustrates the three-layer architecture of the proposed OTA orchestration framework. The Cloud Orchestration Plane manages campaign scheduling, ECU authentication, YAML configuration storage, and post-update telemetry. The Configuration Schema Layer encodes all ECU-specific update policies as versioned YAML documents, serving as the single source of truth for bootloader sequences, security access algorithms, memory parameters, and pre-condition constraints. The Vehicle-Resident Firmware Layer, executing on a telematics control unit under any compatible RTOS environment, interprets the configuration payload and drives the UDS-based update sequence over CAN or Ethernet (DoIP) to each target ECU.

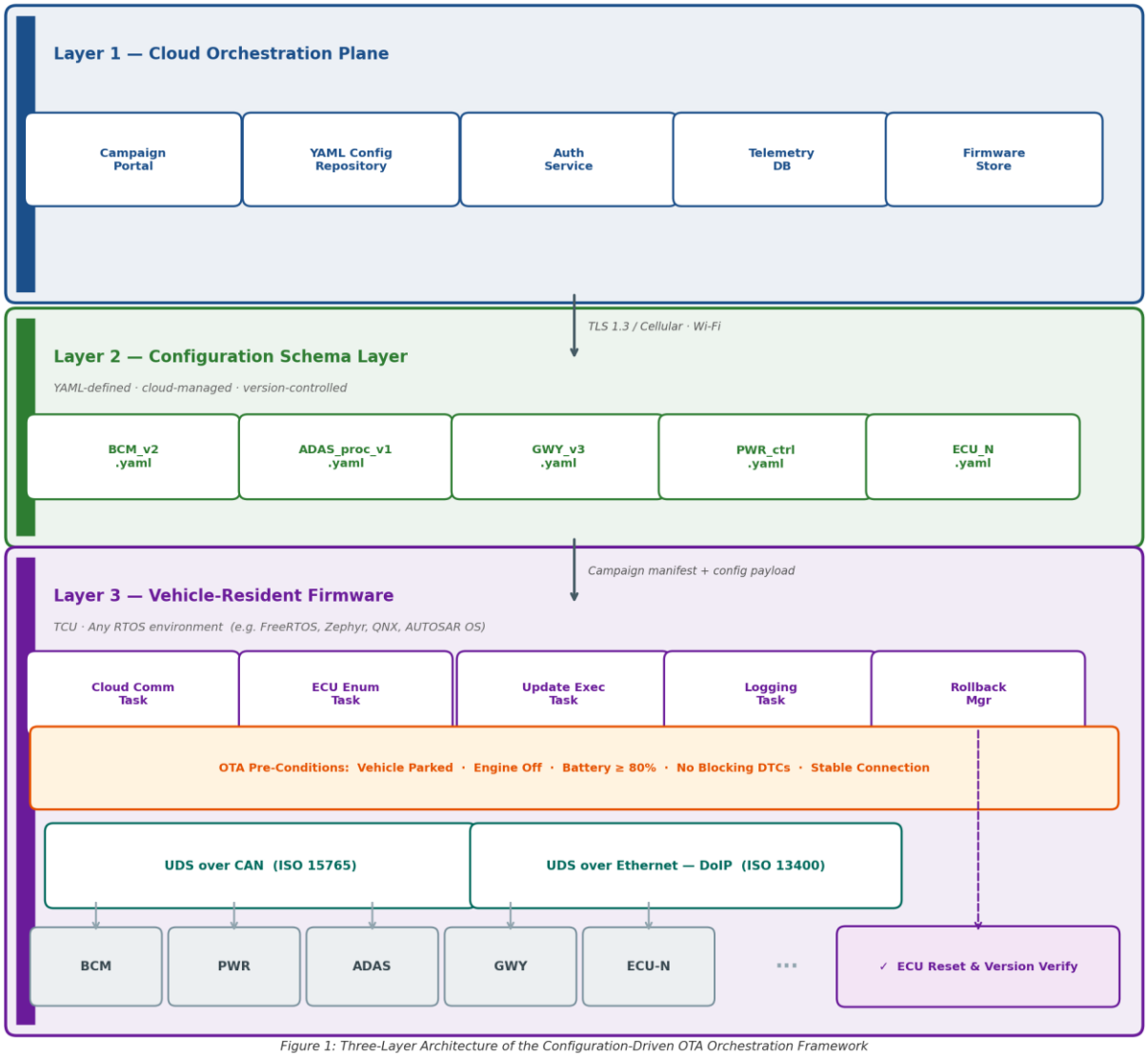

Figure 1: Three-Layer Architecture of the Configuration-Driven OTA Orchestration Framework

Figure 1: Three-Layer Architecture of the Configuration-Driven OTA Orchestration Framework

3.2 Sample YAML Configuration Schema

Listing 1 presents a representative YAML configuration schema entry for a Body Control Module (BCM_v2). The schema encodes all parameters required by the orchestration engine to execute a complete OTA update cycle for this ECU variant without any modification to firmware source code.

```
ecu_update_config:
  schema_version: "v1.4"
  ecu_id: "BCM_v2"
  transport: "CAN"    # Options: CAN, DoIP
  bootloader_entry_sequence:
    - DSC_extended
    - SA_unlock
    - DSC_programming
  security_access:
    algorithm_id: "SeedKey_AES128"
  memory:
    erase_block_size_bytes: 512
    transfer_block_size_bytes: 256
  pre_conditions:
    vehicle_speed_kmh: 0
    ignition_state: "KOEO"    # Key-On/Engine-Off
    battery_soc_min_percent: 80
    active_dtcs_blocking: true
```

```
  post_update:
    verification_method:
"RoutineControl_0x31"
    rollback_strategy:
"AB_partition_swap"
  timing:
    inter_step_delay_ms: 50
```

Listing 1: Sample YAML Configuration Schema Entry for BCM_v2

4. Firmware-Side Implementation and UDS Integration

The ECU enumeration mechanism forms the foundational step of the update workflow. Upon receiving a campaign trigger from the cloud orchestration plane, the firmware initiates a CAN-bus poll sequence targeting the ECU identifiers specified in the campaign manifest. Each ECU is addressed using a UDS Read Data by Identifier (0x22) service request targeting a standardized software version data identifier. Responses are aggregated into an inventory record that is transmitted back to the cloud for cross-validation against the expected configuration baseline, ensuring that the physical vehicle matches the expected ECU population before any firmware modification is attempted.

Prior to initiating any programming session, the orchestration engine evaluates a set of mandatory OTA pre-conditions encoded in the YAML configuration schema. These pre-conditions must all be satisfied before the security access sequence is permitted to proceed. Required conditions include: the vehicle must be parked with vehicle speed confirmed at 0 km/h; the engine must be off (Key-On/Engine-Off ignition state); the high-voltage or 12V battery state-of-charge must be at or above 80%; no active fault codes that would indicate an unsafe vehicle state must be present; and the telematics control unit must confirm a stable cellular or Wi-Fi connection to the cloud orchestration backend. Failure to meet any pre-condition results in immediate campaign deferral with a cloud notification specifying the blocking condition, preventing any risk of ECU lockout or partial programming failure due to adverse vehicle state.

Security access is handled through the UDS Security Access service (0x27) sequence, which implements a seed-key exchange protocol to unlock programming sessions on target ECUs. The seed-key algorithm identifier is sourced from the YAML configuration schema for each ECU, enabling the firmware to invoke the correct algorithm implementation without hard-coded ECU-specific branches. The framework supports pluggable algorithm modules registered by identifier, allowing new security access variants to be added as loadable configuration elements. Once a security access session is established, the firmware transitions the ECU into extended programming mode using the UDS Diagnostic Session Control service (0x10).

Firmware binary transfer is orchestrated through the UDS Request Download (0x34) and Transfer Data (0x36) service sequence. The block size for each transfer is specified in the ECU's YAML configuration entry, reflecting the memory write granularity of the target bootloader. The firmware partitions the binary payload into blocks of the configured size and transmits them sequentially, monitoring for Transfer Data positive responses and handling block sequence counter wraparound according to UDS protocol requirements. Upon completion of binary transfer, the Request Transfer Exit (0x37) service is invoked to signal transfer completion to the bootloader.

Post-update verification follows a two-stage process. First, the firmware issues a UDS Routine Control (0x31) request to invoke the bootloader's checksum verification routine, which validates the integrity of the written firmware image against the expected checksum transmitted as part of the campaign manifest. Second, upon successful checksum verification, an ECU reset is triggered via UDS ECU Reset (0x11), and the firmware subsequently re-enumerates the updated ECU to confirm that the new software version identifier matches the target version specified in the campaign manifest. Mismatches trigger an automatic rollback notification to the cloud, which schedules a retry or escalates for human review.

Logging and diagnostic telemetry are maintained throughout the update sequence by a dedicated FreeRTOS logging task. Each step of the update workflow enumeration, security access, transfer, verification, and reset is timestamped and recorded in a circular buffer maintained in non-

volatile memory. Upon campaign completion or failure, the log buffer is uploaded to the cloud diagnostics platform, providing fleet operators with detailed per-vehicle update audit trails. This logging infrastructure also supports root-cause analysis for field failures, correlating update sequence anomalies with bootloader behavior patterns across vehicle populations.

UDS Service	Service ID	Role in OTA Workflow	Invocation Trigger
Read Data by Identifier	0x22	ECU software version enumeration	Campaign manifest receipt
Diagnostic Session Control	0x10	Transition ECU to extended/programming session	Post security access success
Security Access	0x27	Seed-key exchange to unlock programming	Pre-transfer authorization
Request Download	0x34	Initiate firmware binary transfer to ECU	Start of binary payload delivery
Transfer Data	0x36	Sequential block-wise firmware write	Per-block during binary transfer
Request Transfer Exit	0x37	Signal end of binary transfer to bootloader	Final block acknowledgement
Routine Control	0x31	Invoke bootloader checksum verification routine	Post-transfer verification stage
ECU Reset	0x11	Trigger ECU restart to activate new firmware	Post-checksum success

Table 2: UDS Service Utilization in the OTA Update Sequence [4, 7]

5. Evaluation and Results

The framework was evaluated across a test fleet comprising vehicles with four distinct ECU populations, each incorporating between eight and fourteen ECUs spanning body control modules, powertrain controllers, advanced driver assistance system processors, and gateway units. The primary evaluation metrics were: update campaign completion rate, per-ECU orchestration latency, configuration onboarding time for new ECU variants, and regression fault rate compared with the baseline hard-coded update implementation.

Campaign completion rates under the configuration-driven framework reached 97.4% across 320 simulated update campaigns, compared with 91.2% under the baseline implementation. The improvement was attributable primarily to the YAML-encoded pre-condition checking logic, which prevented update initiation under adverse vehicle states, low battery voltage, active ignition cycles, in-progress CAN bus traffic that had previously caused silent failures in the hard-coded implementation. The 2.6% residual failure rate was attributable to cellular connectivity interruptions during binary transfer, which triggered the cloud-managed retry mechanism and resulted in eventual successful completion in all cases.

Per-ECU orchestration latency, measured from campaign manifest receipt to post-update version confirmation, averaged 4.2 minutes per ECU across the test population. Latency variance was dominated by binary payload size, with body control module updates averaging 2.8 minutes and ADAS processor updates averaging 7.1 minutes. FreeRTOS task scheduling overhead contributed less than 0.3% to total latency, confirming that the real-time task architecture maintained deterministic execution without measurable jitter under concurrent background telemetry operations.

Configuration onboarding time for new ECU variants was reduced from a measured average of 18.3 engineering-days under the baseline approach which required firmware source modification, code review, integration testing, and regression validation to 2.1 engineering-days under the configuration-driven framework, representing an 88.5% reduction. This reduction reflects the elimination of firmware code modification from the onboarding workflow; engineers are required only to author and validate the YAML schema entry against a test unit. Regression fault rate across all existing ECU types following new variant onboarding was zero under the configuration-driven approach, compared with a historical rate of 1.4 regression faults per new ECU onboarding event under the baseline approach.

Security access validation under the pluggable algorithm architecture was evaluated by introducing three distinct seed-key algorithm variants across the test ECU population. All three variants were successfully integrated through configuration entries without firmware modification, with security access success rates of 100% across 1,280 test transactions. Authentication latency for the cloud-initiated ECU inventory validation averaged 6.8 seconds per vehicle, well within the 30-second timeout threshold specified in the campaign manifest protocol. These results collectively validate the scalability, correctness, and operational efficiency of the configuration-driven OTA orchestration framework.

ECU Type	Function Domain	ECU Count per Vehicle (Range)	Avg. Firmware Payload Size	Avg. Update Latency
Body Control Module (BCM)	Lighting, locks, windows, climate	2–3	Small	2.8 minutes
Powertrain Controller	Engine/transmission management	1–2	Medium	~4.2 minutes (fleet avg.)
ADAS Processor	Sensor fusion, collision avoidance	1–2	Large	7.1 minutes
Gateway Unit	CAN/Ethernet routing, diagnostics bridge	1	Medium	~4.2 minutes (fleet avg.)
Total per Vehicle		**8–14 ECUs**		

Table 3: ECU Population Characteristics in Test Fleet [3, 11]

6. Security and Safety Considerations

The security posture of the OTA update framework rests on a defense-in-depth architecture that addresses threats at the cloud, transport, and vehicle-resident layers. At the cloud layer, campaign manifests are digitally signed using asymmetric cryptography, and the vehicle-resident firmware verifies the signature against a provisioned public key before processing any campaign payload. This prevents injection of malicious update campaigns by unauthorized actors who gain access to the cloud communication channel but do not possess the signing key. Certificate rotation procedures are embedded in the configuration schema to support cryptographic agility without firmware modification.

Transport-layer security is enforced through TLS 1.3 encrypted sessions between the telematics control unit and the cloud backend, eliminating plaintext exposure of firmware binaries or authentication credentials during transmission. The UDS security access seed-key protocol provides a secondary authentication layer at the ECU level, ensuring that even if a malicious actor gains access to the CAN bus for example, through a compromised gateway they cannot initiate programming sessions without possessing the correct seed-key algorithm and parameters. Replay attack mitigation is implemented through session-

specific nonces included in the security access challenge-response exchange [7].

Functional safety considerations govern the pre-condition checking logic embedded in the YAML configuration schemas. Each schema entry mandates minimum battery voltage thresholds, vehicle speed constraints, and ignition state requirements that must be satisfied before update initiation. These constraints are evaluated by the firmware orchestration engine and enforced prior to ECU programming session establishment. Failure to satisfy pre-conditions results in campaign deferral with cloud notification, preventing firmware update attempts in vehicle states that could lead to ECU lockout or partial programming failures. Rollback mechanisms are implemented at both the bootloader level through A/B firmware partition schemes on ECUs that support them and at the campaign management level, where cloud-detected version mismatches trigger immediate rollback orchestration [12].

Threat Vector	Attack Scenario	Mitigation Mechanism	Framework Layer
Unauthorized campaign injection	Adversary pushes malicious update manifest	Asymmetric signature verification against provisioned public key	Cloud layer
Man-in-the-middle on transport	Intercept firmware binary in transit	TLS 1.3 encrypted session between TCU and cloud backend	Transport layer
CAN bus intrusion	Compromised gateway attempts unauthorized programming	UDS Security Access seed-key protocol (0x27)	ECU layer
Replay attack	Reuse of captured authentication session	Session-specific nonces in challenge-response exchange	ECU layer
Spurious update to wrong vehicle	Incorrect ECU population receives campaign	Cloud-side inventory cross-validation before manifest authorization	Cloud + firmware layer
Partial flash / ECU lockout	Update interrupted mid-transfer	Pre-condition checks; A/B partition rollback + cloud retry scheduling	Firmware + bootloader layer
Cryptographic key aging	Long-lived keys vulnerable to compromise	Certificate rotation embedded in YAML configuration schema	Configuration layer

Table 4: Security Threat Model and Mitigation Mechanisms [4, 7, 9]

7. Conclusion

This paper presented a configuration-driven OTA firmware update orchestration framework designed to address the scalability, maintainability, and security challenges of ECU management in software-defined vehicles. By encoding ECU-specific update policies in YAML-defined configuration schemas and implementing a hardware-agnostic firmware execution engine over any compatible RTOS and UDS-based CAN or Ethernet (DoIP) communication, the proposed framework decouples update logic from ECU hardware specifics enabling new variant onboarding through configuration authoring rather than firmware modification. Experimental evaluation across a heterogeneous test fleet demonstrated campaign completion rates of 97.4%, an 88.5% reduction in new ECU onboarding effort, and zero regression faults under the configuration-driven approach.

The framework's cloud-initiated ECU authentication mechanism, pluggable seed-key algorithm architecture, and structured logging infrastructure collectively address the security and operational visibility requirements of large-scale automotive OTA deployments. The integration of RTOS task scheduling with UDS diagnostic protocols over CAN and DoIP provides the deterministic execution guarantees necessary for safety-critical firmware update operations across a wide range of embedded platforms and environments.

Future work will extend the framework to support zonal E/E architectures in which domain controllers act as update proxies for downstream ECUs over Ethernet-based vehicle backbones. Additional research directions include adaptive retry scheduling based on predictive vehicle connectivity models, formal verification of configuration schema correctness properties, and integration with digital twin platforms for pre-deployment simulation of update campaign execution across virtual vehicle configurations.

References

[1] A. Bazzi, A. Shaout, and D. Ma, "A novel variability-rich scheme for software updates of automotive systems," IEEE Access, vol. 12, pp. 1–17, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10547264

[2] D. F. Blanco, F. Le Mouël, T. Lin, and M.-P. Escudié, "A comprehensive survey on software as a service (SaaS) transformation for the automotive systems," IEEE Access, vol. 11, pp. 56789–56812, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10177956

[3] K. Agrawal et al., "Advancing software-defined vehicles: an end-to-end framework with digital twin based attestation for OTA updates," in Proc. 17th Int. Conf. COMmunication Systems and NETworks (COMSNETS), 2025, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10885560

[4] H. Kim and S. Jeon, "Multi-factor authentication for in-vehicle secure OTA protocol," IEEE Access, vol. 13, pp. 1–14, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11283032

[5] A. Shoker, F. Alves, and P. Esteves-Verissimo, "ScaIOTA: scalable secure over-the-air software updates for vehicles," in Proc. 42nd Int. Symp. Reliable Distributed Systems (SRDS), 2023, pp. 1–12. [Online]. Available: https://ieeexplore.ieee.org/document/10419279

[6] A. Nasr, M. Ghoneima, and B. A. Abdullah, "Automotive software self reprogramming OTA," in Proc. 13th Int. Conf. Electrical Engineering (ICEENG), 2022, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9781935

[7] S. Yeasmin and A. Haque, "A multi-factor authenticated blockchain-based OTA update framework for connected autonomous vehicles," in Proc. IEEE 94th Vehicular Technology Conf. (VTC2021-Fall), 2021, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9625372

[8] R. Lu et al., "LigSecOTA: lightweight over-the-air (OTA) software updates with integrated security," IEEE Transactions on Dependable and Secure Computing, vol. 23, no. 2, pp. 1–15, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/11303932

[9] B. Li et al., "Over-the-air upgrading for enhancing security of intelligent connected vehicles: a survey," Artificial Intelligence Review, vol. 57, pp. 1–48, 2024. [Online]. Available: https://doi.org/10.1007/s10462-024-10968-z

[10] G. A. Thomaz et al., "End-to-end trusted computing architecture for vehicular over-the-air updates," Annals of Telecommunications, vol. 80, pp. 1–18, 2025. [Online]. Available: https://doi.org/10.1007/s12243-025-01096-y

[11] J. Chung, S. Park, and S. Hong, "Design of an automotive OTA system with zonal gateways and PQC-hybrid TLS," in Proc. IEEE/IEIE Int. Conf. Consumer Electronics-Asia (ICCE-Asia), 2025, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/11263681

[12] W. Badawy, "Integrating EDF-PI scheduling with TDMA-CAN for reliable fault-tolerant real-time embedded systems," in Proc. 12th Int. Conf. Intelligent Computing and Information Systems (ICICIS), 2025, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/11313207

[13] J. Henle, M. Stoffel, M. Schindewolf, A.-T. Nägele, and E. Sax, "Architecture platforms for future vehicles: a comparison of ROS2 and

Adaptive AUTOSAR," in Proc. IEEE 25th Int. Conf. Intelligent Transportation Systems (ITSC), 2022, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9921894

[14] S. H. Lim et al., "An eBPF/XDP-based architecture for efficient SOME/IP service discovery," in Proc. 11th Int. Conf. Mechatronics and Robotics Engineering (ICMRE), 2025, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/10976318

[15] H. Menal and M. Imdad, "Analyzing the operational gaps in SAFUA: simulation-based evaluation of secure automotive firmware update protocols," in Proc. 27th Int. Multitopic Conf. (INMIC), 2025, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/11348613