

USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Beyond Classical PKI: A Quantum-Safe Authentication Framework for Automotive OTA Updates

Wei-Lun Lin, Ting-Yu Chen, and Arijit Karati, *National Sun Yat-sen University*

https://www.usenix.org/conference/vehiclesec26/presentation/lin

This paper is included in the Proceedings of the
4th USENIX Symposium on Vehicle Security and Privacy.

August 10–11, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-56-4

Open access to the Proceedings of the
4th USENIX Symposium on Vehicle Security and Privacy
is sponsored by USENIX.



Beyond Classical PKI: A Quantum-Safe Authentication Framework for Automotive OTA Updates*

Wei-Lun Lin
National Sun Yat-sen University

Ting-Yu Chen
National Sun Yat-sen University

Arijit Karati
National Sun Yat-sen University
arijit.karati@mail.cse.nsysu.edu.tw

Abstract

The automotive industry is transitioning to software-defined architectures, making Over-the-Air (OTA) updates essential for managing the long-term lifecycles of modern vehicles, which often exceed a decade. Although Public Key Cryptography (PKC) currently secures these updates, it remains vulnerable to emerging quantum threats. While Post-Quantum Cryptography (PQC) is a feasible solution, its improper adoption in the automotive sector may lead to larger key storage requirements and higher computational costs, thereby hindering the security of resource-constrained devices. To address long-term deployment concerns and comply with vehicle software update requirements, we propose a quantum-safe authentication protocol to deliver software OTA updates securely over the vehicle-to-cloud (V2C) link. Our protocol employs Leighton-Micali Signatures (LMS) for Original Equipment Manufacturer (OEM) server identification and the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) to establish session keys for OTA updates. To mitigate server-side bottlenecks in bursty-request scenarios, we introduce a one-time ticket-gating approach that filters out invalid requests, preventing flooding or DoS attacks against the OEM server. Security analysis demonstrates that our protocol preserves authenticity, confidentiality, integrity, and anti-rollback properties while resisting critical threats, including replay and quantum attacks. Comprehensive performance results show that it achieves more security traits with adequate functional overhead compared to existing works in the context of automotive systems.

1 Introduction

In recent years, the automotive industry has experienced a paradigm shift, with traditional mechanical vehicles evolving into advanced mobile computing platforms. This adaptation has enabled a range of cutting-edge capabilities, including Advanced Driver-Assistance Systems (ADAS), immersive infotainment, and energy management systems for electric vehicles (EVs). The complexity of automotive software correlates with an increasing demand for regular maintenance and feature enhancements. As a result, the industry is shifting from conventional, in-person service models to efficient wireless software delivery. To meet these evolving requirements, Over-the-Air (OTA) update technology has become the standard for many contemporary automotive systems. OTA updates simplify remote software deployment, thereby significantly reducing operational costs for manufacturers and alleviating the necessity for consumers to make repeated visits to service centers [14]. This transition to wireless transmission across wide-area networks necessitates a strong reliance on effective security frameworks to preserve the integrity of vehicle firmware and the safety of its occupants.

Modern EVs rely on wireless connectivity (e.g., cellular/5G or Wi-Fi) to receive updates via public networks. To ensure software safety, the Public Key Cryptography (PKC), such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) for data confidentiality, and Digital Signature Algorithm (DSA) for software integrity. However, attackers may exploit network connectivity by eavesdropping on traffic, modifying transmitted data, and replaying observed messages if the security mechanism is insufficiently robust [10, 28]. For instance, works [1, 13, 35] show that remote attackers may exploit In-vehicle Infotainment (IVI) systems, telematics, or other wireless interfaces, and inject crafted messages to affect in-vehicle communication, such as the Controller Area Network (CAN). This flaw may allow attackers to leak sensitive data, including vehicle location and driving record, to attackers.

1.1 OTA Workflow and Challenges

Figure 1 shows an overview of OTA workflow, where **(a)** the OEM Server prepares an update package at the backend, uploads it to a cloud server, and notifies target vehicles; **(b)**

*This work was supported in part by the National Science and Technology Council (NSTC) under Grants NSTC 114-2221-E-110-042-MY3 and 114-2634-F-110-001-MBK.

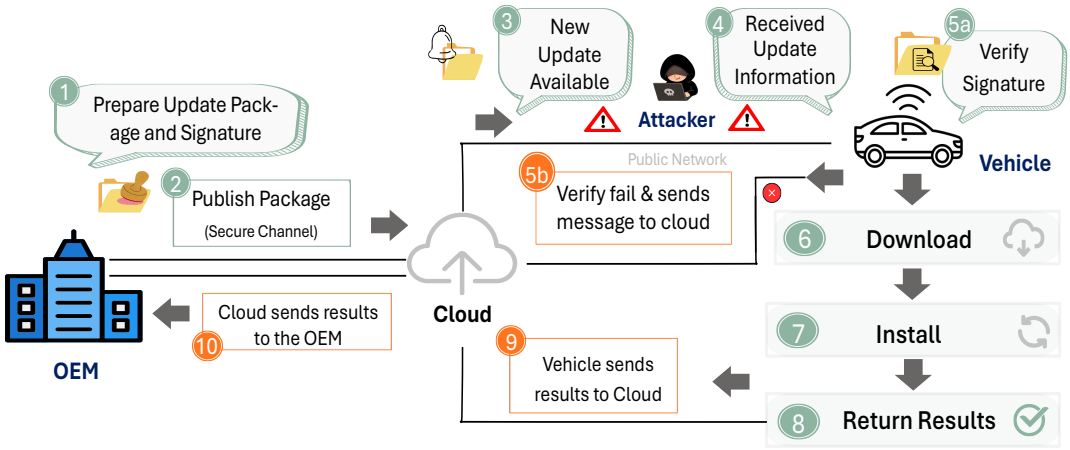


Figure 1: Overview of Over-the-Air (OTA) communication threats

after receiving the notification, vehicles verify the package and download it before installation, typically by checking a cryptographic hash and validating a digital signature; **(c)** after the install is complete, the vehicle will return the results to the OEM Server. For safety, vehicles should only install *trusted* software. Any software that fails verification is not accepted, so security in public network transmission is a priority.

1.1.1 Regulatory Industry and Technical Context

Contemporary vehicle-to-cloud (V2C)-based OTA updates rely on PKC-based security, with ECC used for key management and digital signatures (e.g., ECDSA) to ensure software integrity and accountability. Payload encryption secures the payload during transmission, but it is not sufficient. Vehicles require verifiable proof that the notification is from authentic OEM servers. The OTA updates involve a complex process for interconnected ECUs, including IVIs and safety-critical systems, such as ADAS. Halder et al. [14] demonstrated that this multi-node architecture is highly vulnerable, and a vulnerable ECU may lead to critical system failures. Thus, a security protocol must hold: (a) *Authentication and Integrity* to ensure the software is from a genuine OEM server; (b) *confidentiality* to protect intellectual property and sensitive firmware; and (c) *Freshness* to prevent replay attacks. In the absence of these traits, malicious nodes can replay old packages, leading to resource flooding or Denial-of-Service (DoS) attacks that jeopardize the vehicle network and the OEM infrastructure.

OTA updates are a necessary long-term maintenance in the automotive industry. On the regulation side, the UN-ECE/WP.29 UN R156 [29] deals with software updates and the Software Update Management System (SUMS), while the UN R155 [9] deals with vehicle cybersecurity and the Cybersecurity Management System (CSMS). At a more engineering level, ISO 24089 defines requirements and recommendations for the software update engineering, and the objectives of UN R156 are similar. Similar to UN R155, ISO/SAE 21434 [9] specifies cybersecurity engineering requirements across the vehicle lifecycle. ISO 26262 updates remain relevant to safety-related E/E functions and are up to date. Benyahya et al. [5] have shown that In-vehicle R156, ISO 24089, and Out-of-vehicle ISO 21177 for V2X are often part of industry regulations. The increasing adoption of EVs [36] and the growing complexity of IVI software stacks [16, 32] result in numerous update requests that can strain OEM infrastructure, leading to compute pressure and potential DoS vulnerabilities during bursty release cycles. The fundamental framework for this ecosystem is Uptane, proposed by Kuppusamy et al. [17]. *Uptane* is the first comprehensive update framework designed for compromise resilience, ensuring that vehicles install authorized, fresh, and compatible images even if parts of the infrastructure, such as a repository or an in-vehicle ECU, are compromised. The Uptane v2.1.0[1] provides a comprehensive security model centered on three pillars:

i. *Trusted Software Identification:* Utilizing signed metadata and Image/Director repositories to mitigate rollback, freeze, and mix-and-match attacks.

ii. *In-Vehicle ECU Management:* Leveraging secure time and Primary/Secondary verification models to ensure safe internal distribution.

iii. *Operational Interoperability:* Providing Protocols, Operations, Usage, and Formats (POUFs) to support standardized implementation across different OEMs.

Uptane accommodates confidentiality[2] either as end-to-end or through classical transport layers like TLS. It has exhibited adaptability to emerging threats, including experimental support for PQC (e.g., SLH-DSA, a.k.a. SPHINCS+), since 2022.

[1] https://uptane.org/docs/2.1.0/standard/uptane-standard
[2] https://uptane.org/docs/latest/standard/uptane-standard#431-read-updates

While Uptane provides a software update framework for G1 (Authenticated announcement), G2 (Integrity, Confidentiality), and G3 (Anti-rollback), it remains transport-agnostic. Consequently, it does not provide the protocol-level optimizations, specifically regarding session setup and handshake overhead, required to mitigate the substantial performance and latency penalties inherent in PQC-based V2C communication.

We develop a secure protocol to safeguard the wireless V2C delivery (Uptane's Type (i) deployment). While we adopt Uptane's security goals G1-G3, we specifically focus on G4 (Server-side DoS resilience) and G5 (Low-latency session setup) against quantum adversaries. Our protocol implements a ticket-gating cryptographic mechanism to mitigate resource exhaustion during the heavy PQC handshake. Besides, it minimizes the number of online exchanges to prevent session loss in high-mobility conditions with irregular connectivity.

1.1.2 Security Goals

Considering specific vehicle and OTA issues [14, 17], as well as the aforementioned practical constraints that motivate our security goals. Particularly, we aim for (i) authenticated release announcements, (ii) software package integrity and confidentiality, (iii) replay and anti-rollback resistance, (iv) backend robustness to withstand burst requests, and (v) low-latency session setup. We summarize our key security goals below.

- **G1 (Authenticated announcement).** V accepts a release announcement only if it originates from the legitimate OEM backend S and has not been modified.
- **G2 (Software package integrity and confidentiality).** The software package ensures integrity to prevent tampering and confidentiality to protect intellectual property.
- **G3 (Anti-replay and Anti-rollback).** Fresh sessions reject replayed messages; updates are only accepted if their version number exceeds the locally stored maximum.
- **G4 (Server-side DoS resilience).** The server executes an expensive session setup only after an authorization check succeeds, enabling early rejection during a flood.
- **G5 (Low latency session setup).** Since a vehicle is a moving device, avoiding packet loss during transmission requires mitigating the online authentication.

1.2 Our Contributions

In addition to the challenges of external software validation, we propose a lightweight quantum-resistant authentication for secure OTA updates. The design prioritizes OEM authenticity verification while maintaining a 1-RTT online authentication to mitigate network latency and server strain during high-concurrency scenarios. For long-term security, the protocol incorporates post-quantum cryptographic primitives.

Following NIST guidance and recommendation, we adopt LMS for signed announcements [8] Moreover, ML-KEM is used to establish an ephemeral shared secret [23]. To further reduce server-side computing pressure under request flooding, We add a one-time ticket-gating mechanism based on a Merkle root R_{ID} and a hash-based one-time signature (OTS). The server checks the ticket (Merkle proof, OTS verification, and spent flag). After key establishment, symmetric cryptography, such as ASCON-AEAD [33], is used to efficiently protect large OTA payloads. This structure keeps heavy payload protection in the data plane and keeps the authentication process simple. The proposed protocol has the following features:

- **Quantum-safe and server authentication:** The vehicle only accepts the announcement if it verifies success under pk^{sig}. This reduces future quantum risks while preserving authenticated OEM-origin announcements.
- **Software package integrity and confidentiality** By integrating the NIST algorithm, we employ ASCON for AEAD to make the receiver verify TAG^{AEAD} to ensure the software package is not tampered with, and to provide confidentiality during transmission.
- **Mitigate OEM server computing pressure:** The ticket mechanism is designed to handle invalid vehicle requests and mitigate server pressure during short bursts of requests. Most out-of-list floods are filtered.
- **Feasibility of Deployment and 1-RTT:** The online exchange uses lightweight primitives, including HKDF, HMAC, hashing, ticket checks, and ASCON-AEAD to efficiently protect large OTA payloads. Then, we achieve key confirmation in 1-RTT, which is excellent for practical deployment.

Our protocol achieves deployment feasibility and operational efficiency through a multi-stage verification process. Once the server publishes a signed update announcement, the vehicle verifies the version, software authenticity, and announcement integrity. Next, the vehicle selects an unused OTS secret key (sk_x) and executes a lightweight authentication to derive a shared secret and confirm the key. By checking the OTS ticket first, the server can reject invalid requests early, saving computational resources. This online exchange completes within 1-RTT, and payload transmission uses the lightweight ASCON AEAD cipher for maximum efficiency.

2 Related Work

Table 1 groups existing OTA methods into three scopes: Type 1 – *OEM/Repository to Vehicle*, prioritizing secure public transmission; Type 2–*OEM to ECU*, addressing gateway distrust and update management; and Type 3–*End-to-End framework*, covering the entire pipeline from repository to ECU.

Table 1: Related Secure OTA Update Mechanisms comparison

Work	Year	Type	Method	Advantages	Drawbacks
Mbakoyiannis et al. [22]	2019	Type 3	Role-separated	∗ Low single trust ∗ Replay resistance	∗ Deployment complexity ∗ Time dependency
Manna et al. [20]	2021	Type 1	Attribute-Based Encryption on Embedded	∗ Measured overhead ∗ Practical integration	∗ Key revocation maintenance ∗ Key distribution overhead
Yeasmin et al. [37]	2021	Type 1	MFA + blockchain-based OTA framework	∗ Decentralization ∗ Distributed System	∗ Consensus latency ∗ Operational complexity
Ghosal et al. [12]	2022	Type 1	Ciphertext-Policy ABE	∗ Cloud secrecy ∗ Scalable scheduling	∗ Policy/ABE management ∗ ABE overhead
Plappert et al. [26]	2023	Type 2	ECU attestation	∗ Integrity assurance	∗ Hardware requirement ∗ Provisioning cost
Bazzi et al. [4]	2023	Type 2	Merkle-tree	∗ Fine-grained updates ∗ Efficient verification	∗ Tree state maintenance
Shoker et al. [31]	2023	Type 3	Update-station	∗ Lower cellular bandwidth ∗ Mitigate server bottleneck	∗ New operator boundary ∗ Station storage needed
Wang et al. [34]	2024	Type 1	PKI hybrid encryption + AES encryption	∗ Mature primitives ∗ Integrity verification	∗ Classical PKI reliance ∗ Verification failure risk
Kovacevic et al. [19]	2024	Type 1	Decentralized Identifier Framework	∗ Decentralization ∗ Explicit revocation	∗ Integration complexity ∗ Ecosystem maturity
Kim et al. [18]	2025	Type 3	Multi-Factor Authentication	∗ Gateway independence ∗ Deployment practicality	∗ Credential overhead
Chung et al. [7]	2025	Type 3	Zonal hierarchical + PQC-hybrid TLS	∗ PQC-ready zonal OTA architecture	∗ Relies on hardware ∗ Infrastructure requirement
Lu et al. [21]	2025	Type 3	One-machine-one-certificate OTA	∗ Integrated security ∗ Lightweight overhead	∗ Gateway trust assumption ∗ Certificate management

Type 1: Backend-to-vehicle OTA security. **Type 2**: ECU-level authentication or verification. **Type 3**: Hierarchical OTA distribution via gateway, TCU, primary ECU, or zonal gateway.

Manna et al. [20] evaluate attribute-based encryption (ABE) on an automotive-grade embedded platform, showing the practical cost of using ABE to protect OTA update confidentiality even when updates are stored and delivered via an untrusted cloud. Ghosal et al. [12] propose STRIDE, which uses CP-ABE to enforce fine-grained access control for OTA packages. Yeasmin et al. [37] designed a multi-factor authenticated OTA framework in which the blockchain ledger provides decentralized validation/auditability of update records. [12] propose STRIDE, which uses CP-ABE to enforce fine-grained access control for OTA packages and further designs a scheduling algorithm to improve update throughput under dynamic cellular traffic while maintaining end-to-end security, including confidentiality and freshness. Wang et al. [34] select an RSA signature with AES to protect OTA payload transmission. Kovacevic et al. [19] proposed a decentralized identifier (DID)-based OTA framework that uses verifiable credentials and a distributed ledger, emphasizing decentralization with explicit revocation and traceable, accountable update authorization.

Plappert et al. [26] use TPM 2.0-based ECU attestation with DICE support to provide a hardware root of trust; the backend can verify ECU integrity before accepting and installing OTA updates. Bazzi et al. [4] propose MT-SOTA. This Merkle-tree-based scheme enables fine-grained updates for independent software clusters, improving update efficiency while keeping integrity/compatibility checks lightweight, and it is compatible with hash-based signatures for quantum-resilient settings.

Shoker et al. [31] propose ScalOTA, which introduces update stations integrated with EV charging stations to mitigate the cellular repository bottleneck and reduce download latency, while formalizing an end-to-end chain of trust across OEMs/suppliers/stations/ECUs. Chung et al. [7] design an automotive OTA system with zonal gateways and PQC-hybrid TLS, focusing on a zonal hierarchical architecture and a PQC-ready secure channel for future deployment. Mbakoyiannis et al. [22] devised a role-separated secure firmware workflow (e.g., update manager, firmware server, key server, timestamp server), where role separation helps reduce single points of trust and improves replay resistance. Kim et al. [18] propose an in-vehicle OTA protocol based on multi-factor authentication, in which vehicles directly establish mutual authentication with the server without relying on a gateway, thereby improving the practicality of deployment in some architectures.

Quantum Threat Discussion: Vehicles often remain in service for over a decade; OTA security should account for long-term cryptographic risks. Shor's algorithm shows that a sufficiently capable quantum computer can solve integer factoring efficiently, which erodes the security foundation of RSA-type public-key cryptography and weakens long-term confidence in widely deployed PKC trust anchors used for

authentication and key establishment in OTA systems [6, 24]. Regarding the migration requirement, Fernandez et al. [11] discuss quantum-resistant cryptosystems for IoT and explain why performance and resource overhead must be considered alongside security. Shim et al. [30] compare PQ signature families and emphasize long-term security in domains with long product lifetimes; they also suggest that the transition may take years and should be planned in phases. In the real world, most vehicles last 10 or more years. Thus, the OTA must consider long-term deployment; it is better to plan PQC migration early and keep a practical design.

PQC was proposed to remain secure even against quantum adversaries [25, 30]. Motivated by this, the National Institute of Standards and Technology (NIST) has released PQC standards and recommendations, including LMS for firmware signing [8], ML-KEM for post-quantum key establishment [23], and ASCON-AEAD for large-scale data protection [33]. As we know, OTA packages may contain both firmware and higher-layer software components, and post-quantum key establishment, such as ML-KEM, is also relevant in OTA settings [23]. However, a recent study [30] shows that PQC often introduces higher cost, and many schemes use larger keys. These overheads increase bandwidth usage, and more computing time will delay the update. For resource-limited telematics units, this is not a small issue. In an OTA process, stronger cryptography can directly lead to longer update times and a worse user experience. Therefore, balancing security and efficiency is necessary, not optional.

3 Preliminaries

This section lists symbols in Table 2 and the cryptographic primitives and assumptions used throughout the paper. We formalize LMS as a stateful signature primitive, model ML-KEM as a KEM abstraction, and state the underlying hardness assumptions referenced in the protocol and security analysis.

3.1 Leighton–Micali Signature

Let pk^{sig} and sk^{sig} be the public and secret keys, and σ denotes a Leighton–Micali Signature (LMS). LMS signature σ includes a one-time leaf index i and the corresponding Merkle authentication path π_i used by LMS.Verify.

Definition 1 (Leighton–Micali Signature (LMS))**.** The LMS consists of signing and verification algorithms for $b \in \{0,1\}$:

$$\sigma \leftarrow \mathsf{LMS.Sign}(sk^{sig}, m), \quad b \leftarrow \mathsf{LMS.Verify}(pk^{sig}, m, \sigma)$$

Statefulness. LMS is stateful: each one-time leaf index must be used at most once. The signer maintains a persistent state idx and updates it after each signature:

$$\mathsf{idx} \leftarrow \mathsf{idx} + 1 \tag{1}$$

Security notion. Assuming correct state management (no OTS key reuse), we treat LMS as EUF-CMA secure.

Table 2: Summary of Notations

Symbol	Description
S, V, ID	OEM Server & Vehicle & Vehicle ID
N_S, N_V	Fresh nonces (server, vehicle)
$T_{1\sim8}, M_{1\sim8}$	Timestamp & Transmission Message
Ver_S, Ver_V	Version & local maximum version
$H(\cdot), \|$	Hash function & concatenation
Dst	authentication transcript hash
ek^{eph}, dk^{eph}	Server ephemeral ML-KEM key pair
cti, ss	index i ML-KEM ciphertext & shared secret
hs	Handshake secret for online authentication
sk^{sig}, pk^{sig}	LMS signing key pair
σ	LMS signature on announcement
MD_S	Signed metadata
HKDF	HMAC-based Key Derivation Function
Len	Application-key length
K_S^{kc}, K_R	Key-confirmation keys
TAG_S, TAG_R	Key-confirmation tags & Recovery tags
K_s, K_v	Session Key
ASCON	Lightweight AEAD (ASCON-AEAD)
$Pack, Res, \Delta T$	OTA payload & Vehicle install results, & $T_{1\sim8}$ tolerant
ct^{Pack}, ct^{Res}	Ciphertext (ct) of payload & ct of install results
TAG^{AEAD}	ASCON tag
OTS	One-Time Signature for ticket proof
R_{ID}	Merkle root of ticket public keys
pk_x, sk_x	Ticket one-time public/secret key pair (index)
$path_x$	Merkle authentication path for pk_x
Sig_x	OTS signature for ticket (proof of sk_x)

3.2 Module-Lattice-Based Key-Encapsulation Mechanism and Hardness Assumptions

For the Key-Encapsulation Mechanism (KEM) component, the server's ephemeral key pair is denoted by $(KEMpk_S^e, KEMsk_S^e)$; cti and ss denote the encapsulation ciphertext and the derived shared secret, respectively. $H(\cdot)$ denotes a cryptographic hash function. For the MLWE assumption, q, n, and k denote the modulus, ring dimension, and module rank, and $\mathsf{KEM}.*$ denotes the KEM algorithms.

Definition 2 (Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM))**.** It has the following algorithms:

$$\begin{aligned}(pk^e, sk^e) &\leftarrow \mathsf{KEM.KeyGen}(),\\ (cti, ss) &\leftarrow \mathsf{KEM.Encaps}(pk^e),\\ ss &\leftarrow \mathsf{KEM.Decaps}(sk^e, cti).\end{aligned}$$

For honestly generated inputs, if $(pk^e, sk^e) \leftarrow \mathsf{KEM.KeyGen}()$ and $(cti, ss) \leftarrow \mathsf{KEM.Encaps}(pk^e)$, then $\mathsf{KEM.Decaps}(sk^e, cti) = ss$.

We consider ML-KEM as an IND-CCA2-secure KEM.

Definition 3 (Decision-MLWE)**.** Let q be a modulus and let $R_q = \mathbb{Z}_q[x]/(x^n+1)$. Sample $A \xleftarrow{\$} R_q^{k\times k}$, $s \xleftarrow{\$} R_q^k$, and $e \xleftarrow{\$} \chi^k$

from a noise distribution χ. Let $b := As + e \in R_q^k$. The decision-MLWE assumption states that (A, b) is computationally indistinguishable from (A, u) where $u \xleftarrow{\$} R_q^k$.

Definition 4 (Hash one-wayness and second-preimage resistance)**.** Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a cryptographic hash. We assume no PPT adversary can find (i) a preimage for a random hash output, or (ii) $x' \neq x$ such that $H(x') = H(x)$, with non-negligible probability.

The ML-KEM is IND-CCA2 secure under the MLWE assumption (see Definition 3). Besides, the LMS is EUF-CMA secure under the hash assumptions (see Definition 4) and correct state management.

3.3 System Model and Adversary Model

In this section, we will discuss the system model, protected assets, trust/provisioning assumptions, and the adversary model.

3.3.1 System Model

The OTA authentication framework and system model are illustrated in Fig. 2. It consists of the following entities.

- **Vehicle OTA Client (V):** A vehicle-side client (e.g., a TCU) receives the release announcement, verifies OEM authenticity, executes a 1-RTT online exchange, and installs the update only after cryptographic verification.
- **Cloud/CDN (delivery layer, optional):** The Cloud/CDN functions as a transparent delivery layer, facilitating the storage and distribution (caching/relaying) of OTA payloads. Security is maintained through end-to-end cryptographic trust between the Source (S) and V_i, ensuring that the delivery infrastructure remains outside the trust boundary.

OTA Workflow and Two-Plane Structure. As depicted in Fig. 1, the workflow has two planes. *Control plane: S* broadcasts an LMS-signed announcement carrying minimal metadata $MD_S = \{Ver_S, ek^{eph}\}$; after accepting the announcement, V runs a 1-RTT exchange to obtain a fresh session secret with explicit key confirmation. *Data plane: S* encrypts the OTA payload under an AEAD key derived from the session secret, and V installs only after verification.

3.3.2 Adversary Model

We adopt a Dolev-Yao-style network attacker A operating over the public channel. The adversary can eavesdrop on, modify, inject, replay, delay, drop, reorder, and interleave messages across concurrent sessions. The attacker may also create bursty request patterns to increase backend workload.

Our DoS discussion focuses on *compute-exhaustion* against the OEM backend, where the goal is to force the server to check whether the vehicle is valid or not. Accordingly, our design places ticket validation before server-side to enable early rejection. Volumetric network DDoS attacks, wireless jamming, and physical compromise of ECUs are outside our scope. These can be handled by deployment controls (e.g., rate limiting, filtering, redundancy) orthogonal to our protocol.

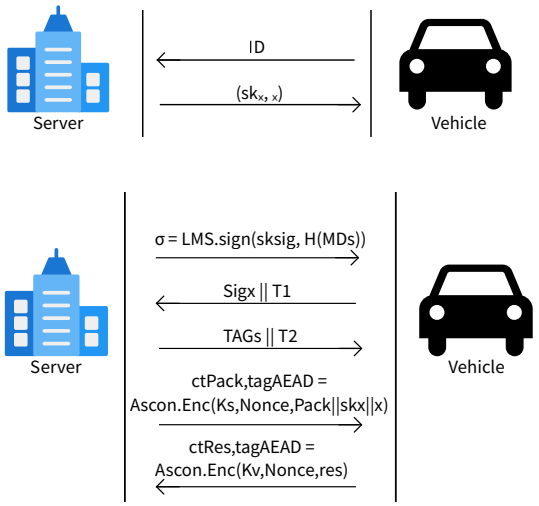


Figure 2: Registration and authentication process

3.3.3 OTA Resource-Pressure Setting

In practical OTA deployments, releases are often published within short time windows, and many vehicles connect concurrently after receiving notifications. We therefore consider a bursty workload in which $n \in \{100, 500, 1000\}$ vehicles request updates during the same period. This setting stresses both the public network path and the OEM backend, where delays and congestion can occur simultaneously. Our primary objective under this setting is to keep the online authentication lightweight (1-RTT with key confirmation), and to reduce server-side, expensive work under heavy load by validating tickets before running the full authentication phase.

4 Proposed Protocol

As shown in Figs. 3-5, our design separates the control plane and the data plane for OTA authentication and OTA payload transmission. In the control plane, we integrate the LMS post-quantum signature algorithm to sign the release announcement MD_S; these signatures are performed offline; ML-KEM to establish an ephemeral shared secret.

Moreover, HKDF for key derivation, HMAC for explicit key confirmation; in the data plane, we select NIST AEAD ASCON to protect the OTA payload, and the entire design is divided into four phases. For the network factor such as packet loss problem happened in phase 4, we strictly consider this

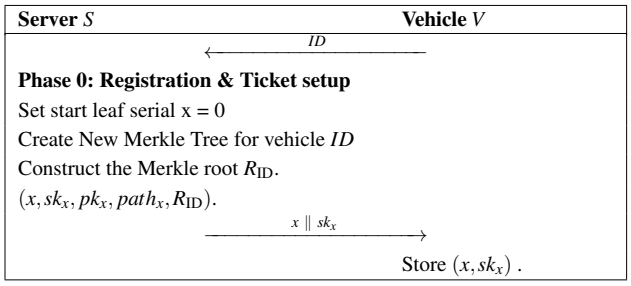


Figure 3: Registration phase over a secure link

kind of situation, we design the ticket recovery mechanism for the specific V, the concept is V select valid hs and execute HMAC to generate TAG_R then sending his $H(ID)$ to S that S can identify V is legal or not, and implement the recovery process. All long-term secret keys, including the master secret key, are stored in TPM 2.0, which provides hardware-backed secure key storage.

Registration and Ticket Setup: In phase 0, V reveals its personal information, as defined by the vehicle ID ID it sends to S. Then S create a unique Merkle tree that only for specific vehicle ID, then transmission the ticket mechanism, as we define the V as a signing party, and OEM Server is a verifying party, hence if the V cannot not demonstrate Sig_x under sk_x then we define that it is not valid V, and will reject OTA update in advanced in the authentication phase, as we define phase 3.

4.1 Offline Update Announcement

In Phase 1 (Offline signature and update announcement), the S generates an ephemeral ML-KEM key pair $(dk^{eph}, ek^{eph}) = \mathsf{ML\text{-}KEM.Gen}()$, the OEM server S prepares a signed metadata object $MD_S = \{Ver_S, ek^{eph}\}$, where Ver_S is the target software version. The key point is that the V must be able to determine whether this release package is genuine before incurring any costs for the online authentication.

For the signing phase, S uses the stateful LMS signature. Each LMS leaf key is one-time; hence, S maintains a signing state (e.g., a leaf index) and advances it with each release. If frequent releases are expected, S can provision a larger LMS tree or rotate to a fresh LMS instance once the current tree approaches exhaustion. This behavior keeps the protocol message flow unchanged and remains a deployment requirement. To preserve the one-time property under reboot and prevent state rollback or key exposure, S protects sk^{sig} and its state in a secure environment (e.g., TPM-backed storage). Then S is computed and signed offline:

$$\sigma = \mathsf{LMS.Sign}(sk^{sig}, H(MD_S))$$

The OEM server sends the update announcement.

$$S \rightarrow V: \; M_1 = (\sigma \parallel MD_S)$$

All of these steps in the OEM server are performed offline.

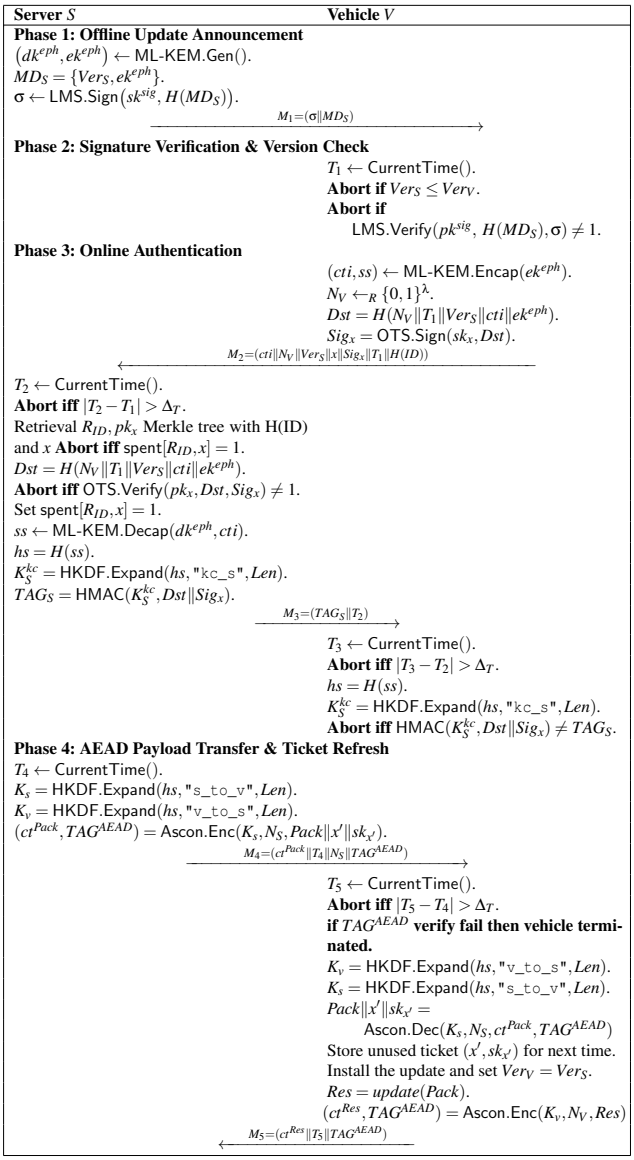


Figure 4: Quantum-safe ticket-gated server authentication

4.2 Signature Verification & Version Check

In Phases 2 (Version check and signature verification), V first generates T_1 and checks the version to avoid anti-rollback, then selects pk^{sig} as a trusted public key to verify σ under pk^{sig}. If verification fails, V rejects the update immediately.

4.3 Online Authentication

In phase 3 (Online Authentication), after accepting the signed announcement, V gets $(cti, ss) = \mathsf{ML\text{-}KEM.Encap}(ek^{eph})$ first, and creates a fresh nonce N_V. The ciphertext cti is sent to S, and only the owner of dk^{eph} can decapsulate cti and get the same shared secret ss under IND-CCA2 security. To ensure both parties agree on which session they are in, we define

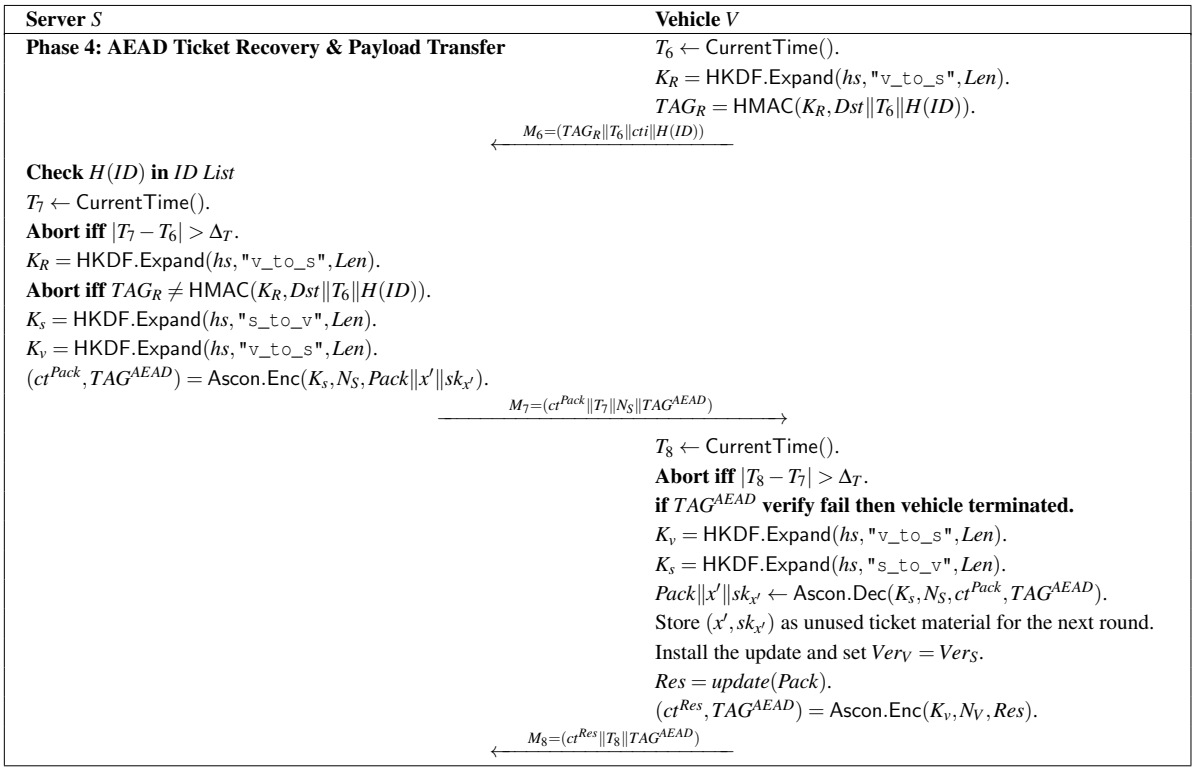


Figure 5: Ticket recovery mechanism and AEAD payload transfer

a transcript hash. For ticket proof, the V computes a one-time signature: $Sig_x = \mathsf{OTS.Sign}(sk_x, Dst)$. It sends ($V \rightarrow S$): $M_2 = (cti \| N_V \| Ver_S \| x \| Sig_x \| T_1 \| H(ID))$.

After receiving M_2, S generates T_2 and checks whether $|T_2 - T_1| \leq \Delta T$; otherwise, it aborts immediately. Before doing the whole authentication operation, the S checks the ticket in a lightweight way: (1) checks whether the specific leaf is used or not, then it reconstructs the transcript:

- Merkle proof check for (R_{ID}, x, pk_x) under R_{ID},
- one-time use check $\mathsf{spent}[R_{ID}, x] = 0$,
- S computes:

$$Dst = H(N_V \| T_1 \| Ver_S \| cti \| ek^{eph})$$

$$\mathsf{OTS.Verify}(pk_x, Dst, Sig_x) = 1$$

If any check fails, the S aborts silently. If it passes, S sets $\mathsf{spent}[R_{ID}, x] = 1$. $ss = \mathsf{ML\text{-}KEM.Decap}(dk^{eph}, cti)$, and derives $hs = H(ss)$. The S derives its confirmation keys

$$K_S^{kc} = \mathsf{HKDF\text{-}Expand}(hs, \texttt{"kc_s"}, Len)$$

If verification succeeds, the S computes $TAG_S = \mathsf{HMAC}(K_S^{kc}, Dst \| Sig_x)$, and sends:

$$S \rightarrow V: \; M_3 = (TAG_S \| T_2).$$

Finally, the V creates T_3, performs timestamp verification, and verifies TAG_S using its locally derived key material; if verification fails, it will be terminated immediately. After this, both parties share (ss, hs) and have a common transcript Dst that binds Phases 1-3. Ticket gating also improves robustness against flooding, since invalid clients are dropped before the authentication operation completes.

4.4 Payload Transfer and Ticket Refresh

In phase 4, we use the established authentication secret hs to generate the K_s and K_v to protect the data plane. Both parties derive directional application keys:

$$K_s = \mathsf{HKDF.Expand}(hs, \texttt{"s_to_v"}, Len),$$

$$K_v = \mathsf{HKDF.Expand}(hs, \texttt{"v_to_s"}, Len),$$

Next, S delivers the update over ASCON-AEAD.

$$\mathsf{Ascon.Enc}\big(K_s, N_S, \; Pack \| x' \| sk'_{x'}\big).$$

To amortize the session and avoid an extra round, S bundles the OTA payload with the next-round ticket material under the same protected channel:

$$S \rightarrow V : M_4 = (ct^{Pack} \| T_4 \| N_S \| TAG^{AEAD})$$

After decryption, V returns an install status message

$$V \rightarrow S: \; M_5 = (ct^{Res} \| T_5 \| TAG^{AEAD})$$

AEAD Ticket Recovery and Payload Transfer: For the ticket recovery (Fig. 5), V sends $M_6 = (TAG_R \| T_6 \| ct_i \| H(ID))$ to S, where TAG_R is a HMAC generated using hs. S checks H(ID) in ID list and verifies the timestamp. Then, it builds TAG_R to execute the HMAC verification to ensure that TAG_R is generated by $H(ID)$. Next, it executes the ASCON AEAD scheme with ticket-update mechanism $sk_{x'}, x'$ to V with $M_7 = (ct^{Pack} \| T_7 \| N_S \| TAG^{AEAD})$. Upon receive, V verifies timestamp and TAG^{AEAD}, then builds (K_s, K_v) to run $ASCON.Dec$ under K_s and installs it. Next, it performs $ASCON.Enc$ under K_v and returns $M_8 = (ct^{Res} \| T_8 \| TAG^{AEAD})$.

5 Security Analysis

This section presents formal security models and proofs for mutual authentication and session key agreement. Let Π_V^i and Π_S^j denote the i-th instance of vehicle V and the j-th instance of server S, respectively. The adversary $\mathcal{A}$ is a probabilistic polynomial-time (PPT) adversary that fully controls the public channel. Therefore, $\mathcal{A}$ may eavesdrop, replay, delay, block, modify, and inject public-channel messages. We give proofs using the stated security models, in which the simulator $\mathcal{S}$ interacts with the polynomial-time opponent $\mathcal{A}$.

1) Security Model for Mutual Authentication: $\mathcal{A}$ outputs a target vehicle V^* to $\mathcal{S}$ before the game starts.

Setup Phase: $\mathcal{S}$ initializes the public parameters, generates the LMS signing key pair (pk^{sig}, sk^{sig}), and prepares the signed announcement metadata $MD_S = (Ver_S, ek^{eph})$, generates the server ephemeral ML-KEM key pair (ek^{eph}, dk^{eph}), and maintains the server-side ticket state $\mathsf{spent}[R_{ID}, x]$ together with the per-vehicle ticket records indexed by $H(ID)$.

Training: $\mathcal{A}$ can query the following oracles to $\mathcal{S}$:

- **Registration:** On input a vehicle identity ID, $\mathcal{S}$ simulates the secure provisioning phase and bootstrap ticket issuance for that vehicle.
- **Authentication:** On input a registered vehicle V, $\mathcal{S}$ returns the public messages exchanged during an honest execution of the proposed protocol for that vehicle.

Challenge: $\mathcal{A}$ sends challenges to $\mathcal{S}$ by impersonating the legal server S toward the target vehicle V^* (Theorem 1) or by impersonating a legal V_i toward the honest S (Theorem 2).

Guess: $\mathcal{S}$ uses the final forgery generated by $\mathcal{A}$ to contradict the corresponding primitive or the required hidden ticket-binding in the proposed protocol.

Theorem 1 (Server Impersonation)**.** *If the LMS is EUF-CMA-safe, then our protocol is secure against server impersonation.*

Proof. Assume that $\mathcal{A}$ is a PPT attacker with a non-negligible advantage against the proposed protocol. Then one can construct a simulator $\mathcal{S}$ that breaks the EUF-CMA security of LMS with non-negligible probability.

1) Setup Phase: $\mathcal{S}$ receives the LMS public verification key pk^{sig} from the LMS challenger. Then, $\mathcal{S}$ sets pk^{sig} as the system public key, prepares the remaining public parameters of the proposed protocol, and sends them to $\mathcal{A}$.

2) Training: $\mathcal{A}$ may query the registration and authentication oracles. $\mathcal{S}$ answers all registration queries consistently. For each honest announcement, $\mathcal{S}$ queries the LMS signing oracle on $H(MD_S)$ and returns the valid announcement

$$M_1 = (\sigma \| MD_S).$$

Hence, $\mathcal{A}$ may observe valid server-originated announcements, but the signing key sk^{sig} is never revealed.

3) Challenge: To make the target vehicle V^* accept a forged server-originated announcement, $\mathcal{A}$ must output

$$M_1^* = (\sigma^* \| MD_S^*)$$

such that $\mathsf{LMS.Verify}\big(pk^{sig}, H(MD_S^*), \sigma^*\big) = 1$. If M_1^* is accepted by the target vehicle and $H(MD_S^*)$ was not queried by the signing oracle before the signing, then $\mathcal{A}$ has produced a valid LMS forgery.

4) Guess: $\mathcal{S}$ outputs $\big(H(MD_S^*), \sigma^*\big)$ to the LMS challenger. Hence, if $\mathcal{A}$ succeeds with non-negligible probability, then $\mathcal{S}$ breaks the EUF-CMA security of LMS with non-negligible probability. Since the LMS is EUF-CMA secure, the proposed protocol is secure against server impersonation. ■

Theorem 2 (Vehicle Impersonation)**.** *If the challenged unused ticket remains hidden, the OTS scheme is EUF-CMA secure, and the Merkle authentication structure remains binding under the adopted hash function. The proposed protocol is secure against vehicle impersonation.*

Proof. Assume that $\mathcal{A}$ is a PPT attacker with a non-negligible advantage against the proposed protocol. Then one shows that the ticket-gating binding of the proposed protocol is violated with non-negligible probability.

1) Setup Phase: $\mathcal{S}$ prepares the public parameters, the server-side ticket records, the per-vehicle Merkle root R_{ID}, and the state table $\mathsf{spent}[R_{ID}, x]$ of the proposed protocol. For the target vehicle V^*, the corresponding challenged unused ticket secret sk_x^* remains hidden.

2) Training: $\mathcal{A}$ may issue the stated registration and authentication queries and may observe valid public transcripts generated by honest executions. Therefore, $\mathcal{A}$ may obtain the public values contained in protocol transcripts, including

$$M_1,\ ct_i,\ N_V,\ Ver_S,\ x,\ T_1,\ H(ID),\ Dst.$$

However, the challenged unused ticket sk_x^* is not revealed.

3) Challenge: $\mathcal{A}$ sends a forged request to the honest server:

$$M_2^* = (ct_i^* \| N_V^* \| Ver_S^* \| x^* \| Sig_{x^*} \| T_1^* \| H(ID^*)).$$

On getting M_2^*, the server checks the freshness window and locates the corresponding ticket tree by $H(ID^*)$. It then finds

$$Dst^* = H(N_V{}^* \,\|\, T_1{}^* \,\|\, Ver_S{}^* \,\|\, ct_i{}^* \,\|\, ek^{eph}),$$

and retrieves the stored ticket record corresponding to $(H(ID^*), x^*)$, and finally verifies $\mathsf{spent}[R_{ID^*}, x^*] = 0$, and

$$\mathsf{Merkle.Verify}(R_{ID^*}, x^*, leaf_{x^*}, path_{x^*}) = 1$$

$$\mathsf{OTS.Verify}(pk_{x^*}, Dst^*, Sig_{x^*}) = 1.$$

Thus, if $\mathcal{A}$ makes the server accept a forged request, then at least one of the following must occur:

1. $\mathcal{A}$ produces a valid OTS forgery on Dst^* under pk_{x^*};
2. $\mathcal{A}$ constructs a false Merkle authentication that still verifies under R_{ID^*};
3. $\mathcal{A}$ reuses an already accepted ticket.

4) Guess: The first event contradicts the EUF-CMA security of the OTS scheme. The second event contradicts the binding property of the Merkle authentication structure under the adopted hash function. The server-side state transition prevents the third event

$$\mathsf{spent}[R_{ID}, x] : 0 \to 1 \tag{2}$$

after one successful acceptance. Therefore, if $\mathcal{A}$ nevertheless makes the server accept a forged request, $\mathcal{A}$ must violate either OTS unforgeability or the Merkle/hash binding with non-negligible probability. Given these assumptions, the proposed protocol is secure against vehicle impersonation. ■

Theorem 3 (Session Key Indistinguishability)**.** *If ML-KEM is IND-CCA2 secure, $H(\cdot)$ behaves as a secure key-compression function, and HKDF behaves as a secure pseudorandom key-derivation function, then no PPT adversary can distinguish the derived session keys from random values with non-negligible advantage.*

Proof. Assume that $\mathcal{A}$ is a PPT attacker with a non-negligible advantage against the proposed protocol. Then one can construct a simulator $\mathcal{S}$ that breaks the security of the underlying key-establishment chain with non-negligible probability.

1) Setup Phase: $\mathcal{S}$ receives the challenge ML-KEM public key and embeds it as the server ephemeral public key ek^{eph} in the public parameters of our protocol. Then, $\mathcal{S}$ prepares the remaining public parameters and sends them to $\mathcal{A}$.

2) Training: $\mathcal{A}$ may issue the stated registration and authentication queries and may observe valid public transcripts generated by honest executions. However, the challenged decapsulation key dk^{eph}, the challenged shared secret ss^*, and the corresponding derived key material remains hidden.

3) Challenge: $\mathcal{S}$ runs an honest target execution and returns the public challenge transcript. Let

$$(ct_i^*, ss^*) = \mathsf{ML\text{-}KEM.Encap}(ek^{eph}),$$

and let $hs^* = H(ss^*)$,

$$K_S^{kc*} = \mathsf{HKDF.Expand}(hs^*, \texttt{"kc_s"}, Len),$$

$$K_v^* = \mathsf{HKDF.Expand}(hs^*, \texttt{"v_to_s"}, Len),$$

$$K_s^* = \mathsf{HKDF.Expand}(hs^*, \texttt{"s_to_v"}, Len).$$

If $\mathcal{A}$ distinguishes the real challenge-session key material from random with non-negligible probability, then at least one of the following must hold:

1. $\mathcal{A}$ distinguishes the challenged ML-KEM shared secret from random;
2. $\mathcal{A}$ distinguishes $hs^* = H(ss^*)$ from a random value;
3. $\mathcal{A}$ distinguishes the HKDF-expanded keys from independent random strings.

4) Guess: $\mathcal{S}$ uses the final guess made by $\mathcal{A}$ to contradict the IND-CCA2 of ML-KEM or the PRF-style security of HKDF. Hence, the session keys derived in our protocol are computationally indistinguishable from random values. ■

5.1 Informal Security Analysis

Recall that the authentication transcript digest is $Dst = H(N_V \,\|\, T_1 \,\|\, Ver_S \,\|\, ct_i \,\|\, ek^{eph})$.

Proposition 1 (Authentication)**.** *Our protocol holds mutual authentication between the OEM server S and the vehicle V.*

Proof. First, V authenticates S by checking $\mathsf{LMS.Verify}(pk^{sig}, H(MD_S), \sigma) = 1$. Hence, a fake server cannot forge a valid announcement without breaking the LMS. Next, after the online exchange, V accepts only if $TAG_S = \mathsf{HMAC}(K_S^{kc}, Dst \,\|\, Sig_x)$, where K_S^{kc} is derived from the shared secret ss. Thus, V confirms that the sender knows the correct session-dependent secret and that the response is bound to the same ticket-authenticated session context. On the other hand, S authenticates V only if $\mathsf{spent}[R_{ID}, x] = 0$, $\mathsf{Merkle.Verify}(R_{ID}, x, leaf_x, path_x) = 1$, and $\mathsf{OTS.Verify}(pk_x, Dst, Sig_x) = 1$. Since only a legitimate vehicle possesses the unused one-time secret key sk_x, a forged requester cannot produce a valid Sig_x on Dst without breaking the OTS scheme. Therefore, both sides authenticate each other, thereby achieving mutual authentication. ■

Proposition 2 (Confidentiality)**.** *Our protocol preserves the confidentiality of the OTA payload over the public channel.*

Proof. After the vehicle receives the signed announcement, it computes $(ct_i, ss) = \mathsf{ML\text{-}KEM.Encap}(ek^{eph})$, while the server computes $ss = \mathsf{ML\text{-}KEM.Decap}(dk^{eph}, ct_i)$. Then both sides derive $hs = H(ss)$,

$$K_s = \mathsf{HKDF.Expand}(hs, \texttt{"s_to_v"}, Len),$$

$$K_v = \mathsf{HKDF.Expand}(hs, \texttt{"v_to_s"}, Len).$$

The OTA payload is protected by

$$(ct^{Pack}, TAG^{AEAD}) = \mathsf{Ascon.Enc}(K_s, N_S, Pack \| x' \| sk_{x'}).$$

Hence, without recovering the hidden shared secret ss, $\mathcal{A}$ cannot derive the AEAD key and cannot reveal the OTA package contents, including the next-round ticket material $(x', sk_{x'})$. Therefore, payload confidentiality is preserved. ■

Proposition 3 (Integrity)**.** *The proposed protocol preserves the integrity of the announcement, the online authentication messages, and the OTA payload.*

Proof. LMS protects the announcement's integrity:

$$\sigma = \mathsf{LMS.Sign}(sk^{sig}, H(MD_S)).$$

Thus, any modification to MD_S causes

$$\mathsf{LMS.Verify}(pk^{sig}, H(MD_S), \sigma) = 0.$$

The ticket proof and key-confirmation tag protect the integrity of online authentication. In particular, $\mathsf{OTS.Verify}(pk_x, Dst, Sig_x) = 1$ binds $(N_V, T_1, Ver_S, ct_i, ek^{eph})$ to the vehicle ticket, and

$$TAG_S = \mathsf{HMAC}(K_S^{kc}, Dst \| Sig_x)$$

binds the server response to the same authenticated session context. Hence, tampering with the online messages changes Dst or invalidates the tag/signature verification.

Finally, the payload integrity is protected by ASCON-AEAD. If an adversary modifies the ciphertext, nonce, or tag, the receiver-side AEAD verification fails. ■

Proposition 4 (Freshness and Replay Resistance)**.** *Our protocol resists replay attacks and preserves message freshness.*

Proof. The protocol binds fresh values into the transcript

$$Dst = H(N_V \,\|\, T_1 \,\|\, Ver_S \,\|\, ct_i \,\|\, ek^{eph}).$$

Since N_V is freshly generated by the vehicle, ct_i is freshly generated by ML-KEM encapsulation, and T_1 is checked under the freshness window at the server, different sessions produce different transcript digests. In addition, the server enforces

$$\mathsf{spent}[R_{ID}, x] = 0$$

before accepting a ticket. After one successful acceptance, the state changes to $\mathsf{spent}[R_{ID}, x] = 1$. Therefore, replaying old online messages or reusing an old ticket cannot make the server accept them again. Hence, the proposed protocol preserves freshness and resists replay attacks. ■

Proposition 5 (Anti-Rollback)**.** *The proposed protocol resists rollback attacks.*

Proof. The advertised version value Ver_S is included in the signed announcement. Hence, an attacker cannot replace the version field without invalidating the LMS signature on $H(MD_S)$. Moreover, upon receiving the announcement, the vehicle compares the advertised version with its locally stored maximum-accepted version, Ver_V. The vehicle proceeds only if $Ver_S > Ver_V$. If the received version is at least as old as the vehicle aborts. After a successful installation, the vehicle updates $Ver_V \leftarrow Ver_S$. Thus, replaying an old but previously valid OTA package still cannot force the vehicle to downgrade. Hence, the protocol is resistant to rollback attacks. ■

Proposition 6 (Quantum Resistance)**.** *The proposed protocol provides quantum-resistant protection, assuming the security of the primitives it uses.*

Proof. The protocol does not rely on RSA, ECC, or Diffie–Hellman in its core authentication and key-establishment path. Instead, it uses LMS for authenticated announcement and ML-KEM for shared-secret establishment. Thus, the protocol avoids the classical public-key assumptions that are directly threatened by Shor's algorithm, and achieves quantum-resistant OTA protection. ■

Proposition 7 (Forward Secrecy)**.** *Assume that each release uses a fresh ephemeral ML-KEM key pair (ek^{eph}, dk^{eph}) and that the old dk^{eph} is erased after use. Then the proposed protocol provides forward secrecy against later compromise of the long-term authentication key.*

Proof. The long-term key sk^{sig} is used only to sign $H(MD_S)$, while traffic keys are derived from the ephemeral shared secret

$$ss = \mathsf{ML\text{-}KEM.Decap}(dk^{eph}, ct_i).$$

The ticket secret sk_x is used only for one-time vehicle authentication on Dst and is not used as input to the traffic-key derivation. Hence, later compromise of long-term authentication material alone does not reveal prior session keys.

If each release uses a fresh (ek^{eph}, dk^{eph}) and the old dk^{eph} is erased after the release window, then compromise of the long-term authentication key in the future does not reveal previously established traffic keys K_S^{kc}, K_s, K_v. Therefore, the protocol provides forward secrecy against the later compromise of the long-term authentication key. ■

6 Empirical Performance Analysis

Experimental Setting: Our experimental testbed emulates the vehicle-side TCU using a Raspberry Pi 3 Model B+, which comprises a 1.4 GHz Broadcom BCM2837B0 SoC and 1 GB of LPDDR2 RAM. The OEM server includes a GTX 1660 GPU, an i5-10400F processor, and 16 GB of DDR4-2400

Table 3: Online Computation Cost Comparison of Related OTA Works

Protocol	Method	Computational costs (one session)
Work [34]	PKI hybrid encryption and signature-based verification	$T_H + T_{\text{RSA.Dec}} + T_{\text{AES.Dec}} + T_{\text{RSA.Verify}}$
Work [18]	PBKDF2-based ECU-level OTA authentication	$T_{\text{PBKDF2}} + T_{\text{HKDF}} + T_{\text{HMAC}} + T_{\text{Sig.Sign}} + 2T_{\text{Sig.Verify}} + 3T_H$
Work [12]	STRIDE: CP-ABE-based secure OTA	$T_{\text{CP-ABE.Dec}} + T_{\text{HashVerify}}$
Work [20]	CP-ABE-protected OTA update with RSA signature	$T_H + T_{\text{RSA.Verify}} + T_{\text{CP-ABE.Dec}}$
Ours	LMS authentication and ticket-gating mechanism	$T_{\text{LMS.Verify}} + T_{\text{KEM.Encap}} + T_{\text{OTS.Sign}} + T_{\text{MerkleVerify}} + T_{\text{OTS.Verify}} + T_{\text{KEM.Decap}} + 4T_{\text{HKDF}} + 2T_{\text{HMAC}} + 6T_H$

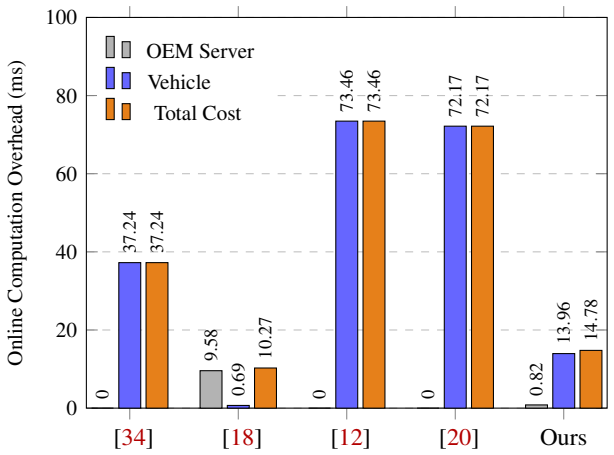


Figure 6: Comparison with related security protocols

memory. All systems utilize 32 GB Class 10 microSD cards for persistent storage and firmware image buffering.

We first compare the per-session computational cost of the proposed protocol with that of related work. We then estimate the OTA download time at fleet scale with concurrency. Finally, we measure the protocol runtime and report its relative delay as a percentage of download time.

6.1 Simulation via Raspberry Pi 3 Model B+

Comparison with Related Security Protocols: Table 3 provides a theoretical estimation of per-session computation costs of related works. All data is shown in Fig. 6 is a simulation from Raspberry Pi 3B+ and PC, comparing the per-session computational cost of our protocol with related protocols. The selected baseline paper includes mechanisms to verify and protect OTA software updates. In-vehicle secure OTA design based on multi-factor authentication, Public Key Infrastructure (PKI) hybrid encryption, STRIDE baseline, and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) for software update. Our protocol achieves competitive per-session online cost while additionally supporting post-quantum mutual authentication and ticket-gated early request filtering to discard invalid requests and avoid DoS or flooding issues, simulating an average computational online overhead 0.82 ms and 13.96 ms on the server and vehicle sides.

Table 4: Comparison of Security Key Sizes (bytes)

Scheme	Private Key	Public Key
Ed25519	32	32
ECDSA P-256	32	32
Falcon	1281	897
ML-DSA (Dilithium)	2528	1312
LMS (RFC 8554)	64	60

Storage Cost: As shown in Table 4, the table compares PKC and post-quantum signatures, including Ed25519, ECDSA, Falcon, ML-DSA, and LMS. This study [3] demonstrates that most currently post-quantum signature algorithms are highly demanding in terms of storage costs for public and secret keys; however, LMS has a very small key size for both public and private keys, which is why we select it.

Security Comparison & Protocol Advantage Discussion: The selected baselines, as shown in Table 5, span different types of OTA-related work: four security-oriented schemes, three efficiency-oriented schemes, and the Uptane framework. The comparison shows that although security-oriented works generally provide better protection than efficiency-oriented ones, the Uptane framework [17], which serves as the high-level orchestrator for authentication, integrity, freshness, and anti-rollback, is also transport-agnostic. Besides, it explicitly relies on transport-layer security (TLS) or encrypted payloads to satisfy data confidentiality. As a result, it only provides a partial solution to quantum resistance (supporting signatures such as SLH-DSA since 2022). It does not provide any built-in support for Forward Secrecy, Server-Side DoS Resilience, and Low-Latency Handshake Setup. Our design addresses these gaps by separating the authentication phase from the transmission of the OTA payload. This guarantees that initial verification takes only a minimal amount of time in the event of an authentication failure. In contrast, current works, such as [34], combine RSA-based authentication with AES-protected payload transmission. If authentication fails, substantial overhead may already have been incurred due to superfluous decryption operations. Through the optimization of the V2C delivery phase (Uptane's Type (i) deployment), our protocol accomplishes additional G4 (Server-side DoS resilience) through a ticket-gating mechanism and G5 (Low-

Table 5: Security Features Comparison of Related Works

Security Property	Work [34]	Work [18]	Work [12]	Work [20]	Work [15]	Work [2]	Work [27]	Work [17]	**Ours**
Mutual Authentication	×	✓	×	×	×	×	×	✓	✓
Confidentiality	✓	✓	✓	✓	×	×	×	✓	✓
Integrity	✓	✓	✓	✓	×	✓	×	✓	✓
Freshness (Anti-Replay)	△	✓	✓	×	×	×	×	✓	✓
Anti-Rollback	×	✓	×	✓	×	×	×	✓	✓
Forward Secrecy	×	×	×	×	×	×	×	×	✓
Quantum-Resistance	×	×	×	×	×	×	×	△	✓
Server-Side DoS Resilience	×	×	×	×	×	×	×	×	✓
Low-Latency Handshake Setup	✓	✓	✓	✓	×	×	×	×	✓

(✓: considered, △: partially considered, ×: not addressed)

Table 6: Evaluation of OTA Payload Download Time (sec) and Throughput (Mbps)

Payload	**10**	**20**	**50**	**70**	**100**	**500**	**1000**
1MB	1.37 (7.31)	2.60 (7.70)	5.69 (8.80)	7.90 (8.86)	13.48 (7.42)	56.24 (8.89)	114.81 (8.71)
10MB	7.51 (13.32)	14.07 (14.22)	34.47 (14.51)	48.07 (14.56)	91.77 (10.90)	352.66 (14.18)	699.60 (14.29)
50MB	31.32 (15.97)	61.29 (16.32)	155.61 (16.07)	213.48 (16.39)	350.27 (14.27)	1550.94 (16.12)	3055.39 (16.36)
100MB	62.82 (15.92)	123.10 (16.25)	304.29 (16.43)	447.71 (15.64)	627.96 (15.92)	3028.60 (16.51)	6066.83 (16.48)
200MB	165.02 (12.12)	360.71 (11.09)	764.49 (13.08)	892.66 (15.68)	1216.56 (16.44)	6134.59 (16.30)	12027.40 (16.63)
300MB	183.11 (16.38)	358.15 (16.75)	892.17 (16.81)	1252.84 (16.76)	1797.80 (16.69)	9068.84 (16.54)	18719.52 (16.03)

Table 7: Evaluation of Proposed Protocol Consumption (ms)

Vehicles	**10**	**20**	**50**	**70**	**100**	**500**	**1000**
Ours	148	296	739	1035	1478	7390	14780

latency session setup) by reducing online exchanges, even in high-mobility environments with intermittent connectivity.

OTA Download-Time Model: As shown in Table 6, we simulate concurrent OTA payload downloads with vehicles $v \in \{10, 20, 50, 70, 100, 500, 1000\}$, where the payload size per vehicle is $s \in \{1, 10, 50, 100, 200, 300\}$ MB. The download makespan is mainly considered by the aggregate throughput TP (MB/s). Thus, the time in Table 6 can be estimated as:

$$t_{\text{OTA}} \approx \frac{v \cdot s}{TP} \tag{3}$$

Protocol Runtime Measurement: As shown in Table 7, we measure the time overhead of the proposed protocol alone, excluding the OTA payload transfer. This isolates the pure security cost and enables us to report how much of the end-to-end OTA time is spent on the protocol.

Protocol Delay Percentage: Table 8, derived from Tables 6-7, shows protocol delay of OTA download time:

$$P(\%) = 100 \cdot \frac{t_{\text{proto}}}{t_{\text{OTA}}} \tag{4}$$

Table 8: Evaluation of the Proposed Protocol Overhead as a Percentage (%) of Payload Download Time

Payload	**10**	**20**	**50**	**70**	**100**	**500**	**1000**
1MB	10.80	11.38	12.99	13.10	10.96	13.14	12.87
10MB	1.97	2.10	2.14	2.15	1.61	2.10	2.11
50MB	0.47	0.48	0.47	0.48	0.42	0.48	0.48
100MB	0.24	0.24	0.24	0.23	0.24	0.24	0.24
200MB	0.09	0.08	0.10	0.12	0.12	0.12	0.12
300MB	0.08	0.08	0.08	0.08	0.08	0.08	0.08

where t_{proto} is from Table 7 and t_{OTA} is from Table 6. For example, with 100 vehicles and a 100 MB payload, $t_{\text{proto}} = 1.47$ sec and $t_{\text{OTA}} = 627.96$ sec, so $P = 0.24\%$. Once the payload exceeds 100 MB, the percentage drops to a negligible level.

7 Conclusion and Future Work

In this paper, we present a quantum-safe server authentication protocol for vehicular OTA updates, using LMS for announcement authenticity and an IND-CCA2-secure KEM to establish session keys with low RTT. To reduce OEM server pressure under bursty requests, we add a one-time ticket-gating mechanism that allows the server to filter unauthorized requests before running the expensive operation. We implement the protocol on a Raspberry Pi 3 Model B+ testbed and measure both protocol execution time and OTA payload download time. Results show that the protocol overhead becomes neg-

ligible as the payload grows, dropping to about 0.10–0.12% for 200–300 MB updates in our settings, which suggests the design is practical for real deployments where updates are typically hundreds of MB. As future work, we plan to evaluate in more realistic ECU and network conditions and build a more comprehensive deployment-level simulation.

Acknowledgments

This work was supported in part by the National Science and Technology Council (NSTC) under Grants NSTC 114-2221-E-110-042-MY3 and 114-2634-F-110-001-MBK, and assisted by the Cryptology and Network Security Laboratory at National Sun Yat-sen University (CANSEC-Lab@NSYSU). The authors would also like to thank the anonymous reviewers and our paper shepherd for their valuable feedback and guidance during the revision process.

Artifact Availability

https://doi.org/10.5281/zenodo.19682385

References

[1] Rafael Abreu, Frederico Branco, Manuel JCS Reis, and Carlos Serôdio. Cybersecurity in connected and autonomous vehicles: A systematic review of automotive security. *IEEE Access*, 2025.

[2] Nicholas Ayres, Lipika Deka, and Daniel Paluszczyszyn. Continuous automotive software updates through container image layers. *Electronics*, 10(6):739, 2021.

[3] Gustavo Banegas, Koen Zandberg, Emmanuel Baccelli, Adrian Herrmann, and Benjamin Smith. Quantum-resistant software update security on low-power networked embedded devices. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 872–891. Springer, 2022.

[4] Abir Bazzi, Adnan Shaout, and Di Ma. MT-SOTA: A merkle-tree-based approach for secure software updates over the air in automotive systems. *Applied Sciences*, 13(16), 2023.

[5] Meriem Benyahya, Anastasija Collen, and Niels Alexander Nijdam. Analyses on standards and regulations for connected and automated vehicles: Identifying the certifications roadmap. *Transportation Engineering*, 14:100205, 2023.

[6] Vaishali Bhatia and KR Ramkumar. An efficient quantum computing technique for cracking RSA using Shor's algorithm. In *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*, pages 89–94, 2020.

[7] Jiseong Chung, Sungkwon Park, and Songnam Hong. Design of an automotive OTA system with zonal gateways and PQC-hybrid TLS. In *2025 IEEE/IEIE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–6, 2025.

[8] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. NIST Special Publication 800-208: Recommendation for stateful hash-based signature schemes, 2020.

[9] Gianpiero Costantino, Marco De Vincenzi, and Ilaria Matteucci. A comparative analysis of UNECE WP. 29 R155 and ISO/SAE 21434. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 340–347, 2022.

[10] Popy Das, Md. Rashid Al Asif, Sohely Jahan, Kawsar Ahmed, Francis M. Bui, and Rahamatullah Khondoker. STRIDE-based cybersecurity threat modeling, risk assessment and treatment of an in-vehicle infotainment system. *Vehicles*, 6(3):1140–1163, 2024.

[11] Tiago M Fernández-Caramés. From pre-quantum to post-quantum iot security: A survey on quantum-resistant cryptosystems for the internet of things. *IEEE Internet of Things Journal*, 7(7):6457–6480, 2019.

[12] Anirban Ghosal, Suman Halder, and Mauro Conti. Secure over-the-air software update for connected vehicles. *Computer Networks*, 218:109394, 2022.

[13] Tian Guan, Yi Han, Nan Kang, Ningye Tang, Xu Chen, and Shu Wang. An overview of vehicular cybersecurity for intelligent connected vehicles. *Sustainability*, 14(9):5211, 2022.

[14] Subir Halder, Amrita Ghosal, and Mauro Conti. Secure over-the-air software updates in connected vehicles: A survey. *Computer Networks*, 178:107343, 2020.

[15] Roland Herberth, Sidney Körper, Tim Stiesch, Frank Gauterin, and Oliver Bringmann. Automated scheduling for optimal parallelization to reduce the duration of vehicle software updates. *IEEE Transactions on Vehicular Technology*, 68(3):2921–2933, 2019.

[16] Seonghoon Jeong, Minsoo Ryu, Hyunjae Kang, and Huy Kang Kim. Infotainment system matters: Understanding the impact and implications of in-vehicle infotainment system hacking with automotive grade linux. In *13th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 201–212, 2023.

[17] Trishank Karthik, Akan Brown, Sebastien Awwad, Damon McCoy, Russ Bielawski, Cameron Mott, Sam Lauzon, André Weimerskirch, and Justin Cappos. Uptane: Securing software updates for automobiles. In *Intl. Conf. on Embedded Security in Car*, volume 11, 2016.

[18] Hyeonseong Kim and Sanghoon Jeon. Multi-factor authentication for in-vehicle secure OTA protocol. *IEEE Access*, 13:208626–208641, 2025.

[19] Ana Kovacevic and Nenad Gligoric. Enhancing security of automotive ota firmware updates via decentralized identifiers and distributed ledger technology. *Electronics*, 13(23):4640, 2024.

[20] Michele La Manna, Luigi Treccozzi, Pericle Perazzo, Sergio Saponara, and Gianluca Dini. Performance evaluation of attribute-based encryption in automotive embedded platform for secure software over-the-air update. *Sensors*, 21(2), 2021.

[21] Ruiqi Lu, Guoqi Xie, Lida Huang, Xinzhong Liu, Wei Xu, Jianmei Lei, and Junqiang Jiang. LigSecOTA: Lightweight over-the-air (OTA) software updates with integrated security. *IEEE Transactions on Dependable and Secure Computing*, pages 1–15, 2025.

[22] Dimitris Mbakoyiannis, Othon Tomoutzoglou, and George Kornaros. Secure over-the-air firmware updating for automotive electronic control units. In *34th ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 174–181, 2019.

[23] NIST. Module-lattice-based key-encapsulation mechanism standard. FIPS 203, 2024. Available: https://doi.org/10.6028/NIST.FIPS.203.

[24] Ugochukwu Orji and CH Ugwuishiwu. An overview of quantum cryptography and shor's algorithm. *International Journal of Advanced Trends in Computer Science and Engineering*, 2020.

[25] Sebastian Paul, Patrik Scheible, and Friedrich Wiemer. Towards post-quantum security for cyber-physical systems: Integrating PQC into industrial M2M communication1. *Journal of Computer Security*, 30(4):623–653, 2022.

[26] Christian Plappert and Andreas Fuchs. Secure and lightweight ecu attestations for resilient over-the-air updates in connected vehicles. In *39th Annual Computer Security Applications Conference (ACSAC)*, pages 283–297, 2023.

[27] Muhammad Hamza Sarwar, Munam Ali Shah, Muhammad Umair, and Syad Hassnain Faraz. Network of ecus software update in future vehicles. In *2019 IEEE 25th International Conference on Automation and Computing (ICAC)*, pages 1–5, 2019.

[28] Roshan Sedar, Charalampos Kalalas, Francisco Vázquez-Gallego, Luis Alonso, and Jesus Alonso-Zarate. A comprehensive survey of V2X cybersecurity mechanisms and future research paths. *IEEE Open Journal of the Communications Society*, 4:325–391, 2023.

[29] Jaewan Seo, Jiwon Kwak, and Seungjoo Kim. Development for high-assurance software update management system complying with UN R156. *IEEE Access*, 12:135811–135830, 2024.

[30] Kyung-Ah Shim. A survey on post-quantum public-key signature schemes for secure vehicular communications. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):14025–14042, 2021.

[31] Ali Shoker, Fernando Alves, and Paulo Esteves-Verissimo. Scaiota: Scalable secure over-the-air software updates for vehicles. In *2023 IEEE 42nd International Symposium on Reliable Distributed Systems (SRDS)*, pages 151–161, 2023.

[32] Junko Takahashi, Makoto Iwamura, and Masashi Tanaka. Security threat analysis of automotive infotainment systems. In *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, pages 1–7, 2020.

[33] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey. Ascon-based lightweight cryptography standards for constrained devices. In *NIST Special Publication (SP) NIST SP 800–232 ipd*. National Institute of Standards and Technology, 2024.

[34] Jingxuan Wang, Rui Chen, Haiou Wen, Yaxiang Chen, Qianping Hou, and Wendong Duan. A secure and efficient vehicle ota software update system: Design and validation. In *2024 IEEE 6th International Conference on Electronic Engineering and Informatics (EEI)*, pages 1176–1180, 2024.

[35] Jun Yeon Won, Wenzhuo Wang, Keith Redmill, and Zhiqiang Lin. CarPlay at risk: Unveiling security threats of third-party infotainment adapters. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)*, pages 143–159, 2025.

[36] Minfeng Wu and Wen Chen. Forecast of electric vehicle sales in the world and china based on PCA-GRNN. *Sustainability*, 14(4):2206, 2022.

[37] Sadia Yeasmin and Anwar Haque. A multi-factor authenticated blockchain-based ota update framework for connected autonomous vehicles. In *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, pages 1–6, 2021.



USENIX VehicleSec '26 Artifact Appendix: Beyond Classical PKI: A Quantum-Safe Authentication Framework for Automotive OTA Updates

Wei-Lun Lin
National Sun Yat-sen University

Ting-Yu Chen
National Sun Yat-sen University

Arijit Karati
National Sun Yat-sen University
arijit.karati@mail.cse.nsysu.edu.tw

A Artifact Appendix

The artifact considers two PowerShell sessions on a single device: shell 1 as a server and shell 2 as a vehicle. The entire experimental process is divided into three segments: (1) **Set-up**, open two PowerShell *with Administrator and all shells into WSL* in the PC, (2) **Install dependencies one time on any shell**, and (3) **Replicate the entire experimental process** and simulate the recovery mechanism. We design a drop Ascon, and when executing drop Ascon, one can see a packet-loss message, execute the recovery mechanism, and receive the payload and the next ticket $\langle sk_x, x \rangle$.

A.1 Abstract

We provide hardware and software dependencies, setup and installation, a basic test description, and demonstrate that our artifacts match our paper claims. We also provide instructions on what these artifacts need in README.txt.

A.2 Description & Requirements

Description & Requirements. We define shell 1 (wsl) as the OEM server and shell 2 (wsl) as the vehicle side. Both the OEM server and the vehicle side operate on a single host machine. The project is located at `C:\VS_ticket` and accessed in WSL as `/mnt/c/VS_ticket`.

A.2.1 Security, Privacy, and Ethical Concerns

This artifact should be executed only in a private local network. Evaluators should not expose the ports used to the public network or run the artifact on a public real IP. This prototype should not be connected to any production system, real vehicle, or other real-world operational environment.

A.2.2 How to Access

The evaluated artifact is available as an archived Zenodo record with a stable DOI. Our artifact's DOI is: https://doi.org/10.5281/zenodo.19682385. This URL includes the artifacts zip file and the demo video zip.

A.2.3 Badges

Artifacts Available, Artifacts Functional, Results Reproduced.

A.2.4 Hardware Dependencies

We using single PC to simulate our artifacts, a Windows 10 64-bit with an Intel core i5-10400f CPU, and 16 GB RAM.

A.2.5 Software Dependencies

(1) For the wsl setting, open PowerShell in Windows with administrator, run **wsl –install**, and reboot the system (2), Feel free to choose any shell (shell 1, shell 2) to install software library only once. The required third-party libraries are `Ascon-c`, `hash-sigs`, `liboqs`, and `liboqs-install`. The artifact is compiled once in the wsl environment.

A.2.6 Benchmarks

None.

A.3 Set-up

Set-up. Shell 1 as an OEM server and shell 2 as a vehicle.

A.3.1 Installation

Extracting the `VS_ticket.zip` archive into `C:\`, will automatically create the `C:\VS_ticket` directory. Then, install the required system packages and dependencies.

A.3.2 Basic Test

The basic test includes registration, authentication, Ascon payload phase.

Note: Instruction *–vehicle-id 123000*, if you registered vehicle ID 123000, you should use the same ID in Step 2 (online

authentication), Step 3 (payload transmission) on both side. Optional (Recovery Mechanism) also provided. All *ip* in the artifacts are setting same: 127.0.0.1.

Step 1: Registration. On the server side (PC/wsl), run:

```
./build/registration_server \
  --state-prefix local_eval \
  --host 127.0.0.1 \
  --port 6000 \
  --versions 60 \
  --once
```

On the vehicle side (shell 2), run:

```
./build/registration_vehicle \
  --state-prefix local_eval \
  --server-ip 127.0.0.1 \
  --server-port 6000 \
  --vehicle-id 123000
```

Expect: You will type the height of the Merkle tree on Shell 1 and see the Registration ID on both sides.

Step 2: Online Authentication, Version 1. On the server side (shell 1), run:

```
./build/control_server \
  --state-prefix local_eval \
  --host 127.0.0.1 \
  --port 6000 \
  --version 1 \
  --delta-seconds 300 \
  --once
```

On the vehicle side (shell 2), run:

```
./build/control_vehicle \
  --state-prefix local_eval \
  --server-ip 127.0.0.1 \
  --server-port 6000 \
  --version 1 \
  --delta-seconds 300 \
  --vehicle-id 123000
```

Expect: You will see the online authentication phase.

Step 3: Payload Transmission, Version 1. On the server side (shell 1), execute:

```
./build/payload_server \
  --state-prefix local_eval \
  --host 127.0.0.1 \
  --port 6000 \
  --version 1 \
  --payload-bytes 1048576 \
  --once
```

On the vehicle side (shell 2), execute:

```
./build/payload_vehicle \
  --state-prefix local_eval \
  --server-ip 127.0.0.1 \
  --server-port 6000 \
  --version 1 \
  --vehicle-id 123000
```

Expect: You will see the Ascon encryption transmission.

A.4 Evaluation workflow

It demonstrates various phases: (1) *Registration* (2) *Online authentication* (3) *Payload transmission & ticket refresh* and when packet loss happen in (3) are as follow: (2) *Online authentication* (3) *Drop Ascon payload & ticket refresh* (4) *Recovery Ascon payload & ticket refresh*.

A.4.1 Major claims

C1: The artifact achieves registration phase, authentication phase, payload transmission. (E1).

C2: It simulates the recovery mechanism successfully. (E2).

C3: The multiple times simulation. (E3).

A.4.2 Experiments

(E1) [Proposed protocol] *[20 human minutes + one PC]–multiple times OTA Workflow, Version 1:*
⋆ **Preparation:** Open two powershell with wsl.
⋆ **Execution:** Step 1: Registration, Step 2: Online authentication, and Step 3: Payload dispatch & ticket refresh.
⋆ **Results:** It shows that the server takes nearly 0.83 ms, and the vehicle takes 3.23 ms in a 20-time multiple test.

(E2) [Recovery mechanism] *[10 human minutes + one PC]–Dropped Payload and Recovery Workflow, Version 2:* We show the process in a demo video accessed from Zenodo.
⋆ **Preparation:** Open two powershell with wsl.
⋆ **Execution:** The workflow is online authentication, drops payloads, and invokes the recovery mechanism.
⋆ **Results:** We demonstrate that even if we deliberately drop the payload, the recovery mechanism still finishes the payload transmission with the next-ticket sk_x, x.

(E3) [Repeat simulation of proposed protocol] [5 human-minutes + one PC]: Multiple-run evaluation.
⋆ **Preparation:** Open two powershell with wsl.
⋆ **Execution:** Full steps are as follows: Step 1: Registration. In this step, both Shell 1 and Shell 2 need to input the *same vehicle ID and IP* in the instruction.
⋆ **Results:** You will expect to see multiple results on the screen and finally get the averaged data.

A.5 Notes on Reusability

A reset operation is provided to clean the registration list. Running the commands below on Shell 1 and Shell 2 removes the registration-related state and restores a clean environment.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing, and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.