

## Article

# Enhancing Vehicle IoT Security with PQC: A Lightweight Approach for Encrypted Sensor Data Transmission

Jackson Diaz-Gorriñ \*  and Candido Caballero-Gil 

Department of Computer Science and Systems, University of La Laguna, 38270 La Laguna, Spain; ccabgil@ull.edu.es

\* Correspondence: jhdiazgo@ull.edu.es

## Abstract

Cybersecurity threats are evolving constantly, and the arrival of quantum computing raises serious doubts about whether today's cryptographic methods will hold up over time. This concern has motivated interest in algorithms designed to resist future attacks, with CRYSTALS-Kyber emerging as a practical candidate and forming the basis of an NIST post-quantum standard. This study focuses on protecting data exchanged between a vehicle sensor suite and cloud services over the Message Queuing Telemetry Transport protocol. Performance must remain acceptable; therefore, attention centers on lightweight and efficient execution while leveraging the board's hardware capabilities to keep latency and resource usage low. Adding this layer of post-quantum encryption helps limit the exposure of critical telemetry and control data to sophisticated adversaries. It also aims to preserve integrity and confidentiality in vehicular communications as the Internet of Things becomes increasingly connected. This approach maintains a practical balance between forward-looking security and real-world deployability.

**Keywords:** cybersecurity; internet of things; connected vehicles; edge computing; post-quantum cryptography; network security

## 1. Introduction

The Internet of Things (IoT) enables the interconnection of physical devices equipped with sensors, actuators, and communication capabilities, facilitating data collection, exchange, and processing over the Internet [1]. In the transportation sector, IoT adoption has led to connected vehicles and intelligent transportation systems (ITSs), which support real-time monitoring, data-driven decision making, and remote control, thereby enhancing safety, efficiency, and user experience.

Modern vehicles integrate diverse sensors that monitor critical parameters such as tire pressure, fuel level, engine oil condition, and vehicle speed [2]. Continuous monitoring improves road safety, prevents mechanical failures, and reduces operational costs, making sensor-based vehicle health monitoring a key component of connected vehicle platforms.

Beyond local monitoring, vehicular data is frequently transmitted to cloud-based IoT platforms for centralized storage and analysis [3]. This enables fleet managers, service providers, and vehicle owners to access real-time vehicle status, receive alerts for anomalies, and analyze historical trends. The integration of GPS data further supports route tracking [4], trip reconstruction, and fleet optimization, with applications extending to security, such as theft detection and unauthorized usage monitoring.



Academic Editor: Yan (Mia) Zhang

Received: 14 December 2025

Revised: 30 January 2026

Accepted: 3 February 2026

Published: 4 February 2026

**Copyright:** © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

The sensitive nature of vehicular data, including location, driving behavior, and system diagnostics, makes secure transmission essential [5]. Encryption and authentication mechanisms are crucial for ensuring the confidentiality, integrity, and authenticity of data exchanged between vehicles, roadside units, and cloud servers. This requirement is particularly important in emergency scenarios, where timely and reliable communication can directly impact safety.

The emergence of quantum computing threatens classical public-key cryptography, including RSA and elliptic curve algorithms, which are widely used in vehicular IoT systems [6,7]. post-quantum cryptography (PQC) is therefore essential to protect against future quantum attacks [8]. At the same time, vehicular IoT devices face strict constraints in terms of computation, memory, and energy, necessitating lightweight PQC solutions suitable for embedded environments [9].

Protocols such as Message Queuing Telemetry Transport (MQTT) are commonly used for sensor-to-cloud communication. Integrating PQC algorithms like CRYSTALS-Kyber offers quantum-resistant key establishment, enabling the secure derivation of symmetric session keys that are then used to encrypt MQTT payloads, while maintaining low latency and efficient resource utilization, particularly on embedded platforms such as the NVIDIA Jetson Orin Nano, which is well-suited for automotive applications.

The contributions of this work are threefold. First, a post-quantum secure communication approach for vehicular sensor-to-cloud data exchange over the MQTT protocol is presented, addressing the limitations of classical cryptographic mechanisms in the context of emerging quantum threats. Second, the CRYSTALS-Kyber post-quantum Key Encapsulation Mechanism (KEM) is integrated to securely establish symmetric session keys, which are subsequently used to encrypt MQTT payloads at the application layer, ensuring confidentiality and efficiency on resource-constrained devices. Finally, the proposed solution is implemented and experimentally evaluated on a resource-constrained embedded platform, namely the NVIDIA Jetson Orin Nano, with an assessment of its impact on latency, computational overhead, and resource utilization in realistic vehicular IoT scenarios.

This work is structured as follows. Section 1 presents the Introduction. Section 2 covers the necessary preliminaries. Section 3 summarizes related work in post-quantum vehicular IoT security. Section 4 presents the main aspects of MQTT-based encrypted sensor data transmission. Section 5 details the proposed implementation and experiments. Section 6 discusses the experimental results. Finally, the last section presents our concluding remarks.

## 2. Preliminaries

### 2.1. Jetson Orin Nano

The Jetson Orin Nano [10] from NVIDIA is a compact and energy-efficient board well suited for automotive integration. It offers 8 GB of RAM and enough processing power to run continuously inside a vehicle. In practice it can act as a local gateway, collecting and pre-processing data from sensors such as temperature probes, accelerometers and GPS modules. Some tasks demand heavy computation, and others only brief bursts of work. The Orin Nano handles both without drawing much power.

### 2.2. Message Queuing Telemetry Transport Protocol

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol suited to many IoT deployments [11]. It was designed for networks with limited bandwidth and variable latency. As a result, overhead stays low and communication remains reliable even under poor conditions.

Some of the key features of MQTT are:

- **Lightweight Design:** MQTT is optimized for constrained environments with limited processing power and bandwidth.
- **Quality of Service [12] Levels:** The protocol provides three distinct levels 0 (at most once), 1 (at least once), and 2 (exactly once) allowing developers to tailor the reliability of message delivery to meet the specific requirements of their applications.
- **Last Will and Testament [13]:** This feature allows clients to specify a message that the broker should send to a designated topic if the client unexpectedly disconnects, enhancing network reliability.
- **Retained Messages:** When a message is published with the retained flag, the broker retains the last message on a topic, ensuring new subscribers receive the most recent data immediately upon subscribing.

### 2.3. Post-Quantum Cryptography

PQC [14] is a new class of cryptographic methods built to stay secure even once powerful quantum computers arrive. Today's common systems, like RSA or elliptic-curve cryptography, rely on mathematical puzzles that are hard for ordinary computers but could be solved much faster by quantum machines, so researchers are preparing replacements now to avoid a future where encrypted data becomes easy to break.

PQC algorithms are built on mathematical problems that remain hard for quantum hardware. Quantum computers work with qubits and run special algorithms such as Shor's algorithm, which can solve certain tasks far more efficiently than classical machines. For that reason, post-quantum schemes choose mathematical foundations that resist those quantum shortcuts. The objective remains unchanged. It is to protect confidentiality and to prevent forgery of identities and keys while relying on mathematics that stays robust in a quantum era.

One leading example is CRYSTALS-Kyber [9], a lattice-based algorithm developed through the NIST post-quantum standardization effort. Kyber provides public-key encryption and a Key Encapsulation Mechanism (KEM) that make it practical for securing communications today while remaining resistant to quantum attacks. In short, Kyber and other PQC tools are the forward-looking building blocks we need to keep digital conversations and data safe as computing technology evolves.

Some of the key features of PQC are:

- **Lattice-Based Security:** Kyber is built on the hardness of specific lattice problems, particularly the Learning With Errors (LWE) problem. This provides a strong theoretical foundation, as current research suggests that lattice-based problems are hard to solve even with quantum algorithms.
- **Efficiency:** One of the main advantages of Kyber is its efficiency in terms of computational speed and resource usage. It has been optimized to perform well on a variety of platforms, including low-power devices often found in IoT applications.
- **Key Encapsulation Mechanism [15]:** Kyber offers a KEM, which allows parties to securely exchange symmetric keys over an insecure channel. This is critical for establishing secure communication between devices.
- **Public-Key Encryption:** In addition to its use as a KEM, Kyber supports public-key encryption, enabling secure message transmission without requiring prior key exchange.
- **Standardization:** Kyber serves as the basis for the NIST Multi-Layer Key Encapsulation Mechanism (ML-KEM) standard. Its selection indicates strong confidence in its security and practicality.

- Variants and Security Levels: Kyber comes in several variants with different security levels (e.g., Kyber-512, Kyber-768, Kyber-1024), allowing users to choose an appropriate level of security based on their specific needs and threat models.
- Real-World Applicability: Due to its efficiency and security features, Kyber is being actively considered for use in various applications, including secure communication protocols, data encryption, and digital signatures in a post-quantum era.

#### 2.4. Encryption and Decryption with Kyber

Kyber is used as a KEM to establish a shared secret, which is then employed as the key for a symmetric cipher, AES-128-GCM, to encrypt MQTT payloads. This approach ensures both quantum-resistant key exchange and efficient encryption of application-layer data.

##### 2.4.1. Key Generation

A key pair (public key  $pk$  and private key  $sk$ ) is generated through a key generation function  $KeyGen$ .

##### 2.4.2. Encapsulation

The shared secret is derived via the encapsulation function

$$ss = \text{Encaps}(pk)$$

This shared secret is then used as the key for a symmetric cipher to encrypt the actual MQTT payload.

##### 2.4.3. Decapsulation

The shared secret is recovered via

$$ss = \text{Decaps}(sk, ct)$$

This secret key is then used to decrypt the MQTT payload securely on the receiver side.

#### 2.5. Message Authentication

Message authentication and replay prevention can be achieved through a Hash-based Message Authentication Code (HMAC). In particular, the hash function SHAKE-128, of the NIST SHA-3 hash family, can be used as part of an HMAC as follows:

$$\text{hash} = \text{SHAKE-128}(ct \parallel \text{id}_{\text{msg}})$$

where  $\parallel$  denotes concatenation, and  $\text{id}_{\text{msg}}$  is a unique identifier assigned to each transmitted message. This ensures that each message has a unique hash, preventing replay attacks and enabling secure message authentication.

### 3. Related Work

Post-quantum cryptography (PQC) has been increasingly studied as a solution to secure IoT communications against potential quantum attacks. Several works have explored different PQC algorithms, protocols, and deployment scenarios, focusing on efficiency, resource constraints, and practical applicability.

The work by Samandari and Gritti [16] investigates the use of the post-quantum digital signature scheme CRYSTALS-Dilithium alongside the CRYSTALS-Kyber Key Encapsulation Mechanism (KEM) to provide authentication in the MQTT protocol. Their study evaluates CPU, memory, and disk usage on resource-constrained IoT devices, comparing KEM-based authentication with traditional mechanisms. Their results show that KEMs can significantly reduce computational overhead, achieving a speed increase of 25 ms and CPU savings

of 71%, with minimal memory impact. This study highlights practical trade-offs when adopting PQC for lightweight messaging protocols.

Ehsan et al. [17] perform a comparative analysis of CRYSTALS-Kyber and NTRU KEMs for low-power IoT devices, considering security, computational efficiency, and memory requirements. They also conduct a case study applying these KEMs to a low-power embedded device, evaluating performance in real-world scenarios. Their results provide guidance on selecting suitable PQC schemes for constrained IoT environments, emphasizing that even when quantum computers are not widely accessible, IoT devices still require protection against future quantum attacks.

Cruz-Piris et al. [18] examine the impact of PQC in Industrial IoT scenarios with resource-constrained devices. They design and implement a framework for extensive experiments measuring execution time, energy consumption, and computational load across different PQC algorithms. Their findings demonstrate the feasibility of transitioning from classical cryptography to PQC in IIoT nodes while quantifying the performance overhead introduced by these algorithms.

Almutairi and Sheldon [19] present a systematic review of PQC integration in lightweight IoT protocols such as MQTT and CoAP. They analyze network resilience and protocol-level performance under lossy and high-latency conditions, highlighting challenges in non-standardized PQC integration, resource limitations, and potential side-channel vulnerabilities. The review identifies gaps in our understanding of the performance of CRYSTALS-Kyber and other PQC schemes under constrained network conditions.

Kwala et al. [20] propose an optimized Module-LWE key-exchange protocol for IoT devices that eliminates the traditional encapsulation–decapsulation structure used in schemes like Kyber. By employing modular rounding and Peikert’s error reconciliation technique, their protocol reduces computational cycles, communication size, and memory footprint. Benchmarking with Open Quantum Safe data across Kyber512, Kyber768, and Kyber1024 levels demonstrates improved energy efficiency and lightweight operation for embedded IoT systems.

The present work provides a practical implementation of CRYSTALS-Kyber for secure sensor-to-cloud data transmission in vehicular IoT systems using the MQTT protocol. The proposed approach is evaluated on a Jetson Orin Nano, demonstrating low-latency, real-time encrypted communication suitable for vehicle-grade embedded platforms. This work integrates a lattice-based PQC KEM with MQTT communication and experimental validation in a realistic vehicular environment, highlighting its novelty in addressing both quantum-resistant security and resource constraints in automotive IoT.

Table 1 summarizes relevant studies on PQC in IoT environments. It compares whether each work implements the CRYSTALS-Kyber Key Encapsulation Mechanism (KEM), uses the MQTT protocol for communication, protects the message payloads (Payload Security), targets resource-constrained embedded devices, and addresses vehicular scenarios. This comparison highlights the novelty of the present work, which combines Kyber-based payload encryption, MQTT communication, and experimental validation on an embedded vehicular platform.

**Table 1.** Comparison of PQC implementations in IoT studies.

| Reference                | Kyber | MQTT | Payload Security | Embedded Device | Vehicular |
|--------------------------|-------|------|------------------|-----------------|-----------|
| Samandari & Gritti [16]  | ✓     | ✓    | ✗                | ✓               | ✗         |
| Ehsan et al. [17]        | ✓     | ✗    | ✗                | ✓               | ✗         |
| Cruz-Piris et al. [18]   | ✓     | ✗    | ✗                | ✓               | ✗         |
| Almutairi & Sheldon [19] | ✓     | ✓    | ✗                | ✗               | ✗         |
| Kwala et al. [20]        | ✗     | ✗    | ✓                | ✓               | ✗         |
| This Work                | ✓     | ✓    | ✓                | ✓               | ✓         |

**Note:** ✓ indicates the feature is considered in the study, while ✗ indicates it is not.

#### 4. PQC for MQTT Sensor Data Transmission

MQTT is widely used because it is simple and efficient, but that same simplicity leaves important security gaps. By default the protocol sends messages in plain text, so anyone who intercepts the traffic can read sensitive information. Without extra protections, MQTT is therefore a poor fit for scenarios that demand confidentiality and strong access control.

Encrypting MQTT traffic with Transport Layer Security can close that gap, yet TLS often requires more processing power and energy than many IoT devices can spare [21]. For small sensors and battery-powered modules, the overhead of full TLS may be impractical, which forces designers to choose between security and device lifetime.

Kyber offers a middle ground by providing a quantum-resistant key-exchange mechanism that is relatively light on resources. Using Kyber to establish session keys for end-to-end encryption keeps message contents private while avoiding the full cost of TLS, making secure MQTT communication realistic on constrained hardware. In short, integrating Kyber helps protect MQTT deployments against both today's threats and the quantum risks on the horizon, provided that implementations carefully manage performance, memory and key lifecycle.

MQTT lacks built-in mechanisms for strong mutual authentication, leaving it vulnerable to Man-in-the-Middle (MitM) attacks [22], where an attacker can intercept or manipulate messages. Moreover, traditional asymmetric methods used for authentication, such as RSA [23] or Diffie–Hellman [24], are anticipated to be susceptible to quantum decryption, posing a significant risk to the integrity of communications between devices.

The integration of Kyber enhances MQTT security by providing a lattice-based public-key infrastructure that is resilient to quantum decryption. By utilizing Kyber for key exchange within the MQTT framework, both parties can authenticate securely. This integration mitigates the risk of Man-in-the-Middle attacks, fostering mutual trust and ensuring the integrity of the connection, thus preventing attackers from intercepting or altering messages between the client and the broker.

The MQTT protocol often relies on weak or optional authentication. In many deployments, devices connect to brokers without thorough credential checks. That gap makes it easier for an attacker to gain unauthorized access or to impersonate a legitimate device.

Kyber can be applied to establish strong authentication that resists quantum attacks. By integrating Kyber, devices exchange public keys that remain secure even against future quantum computers. This enables authentication that is reliable and difficult to tamper with.

The net effect is a clear improvement in MQTT security. Only authorized devices should be able to connect and exchange messages on the network. This protection remains effective in a post-quantum era.

Replay attacks [25] represent a clear threat to MQTT. Intercepted messages can be stored and later retransmitted by an attacker to alter system behavior. Because the protocol

often lacks unique identifiers or reliable timestamps, it offers little defense against the replay of otherwise legitimate messages, and this can lead to unauthorized actions.

Kyber can help mitigate this weakness by enabling the creation of unique session keys for encrypting each MQTT message. Each key is valid only for a single session, which increases message freshness and supports stronger authentication. That makes it much harder for an attacker to reuse captured messages.

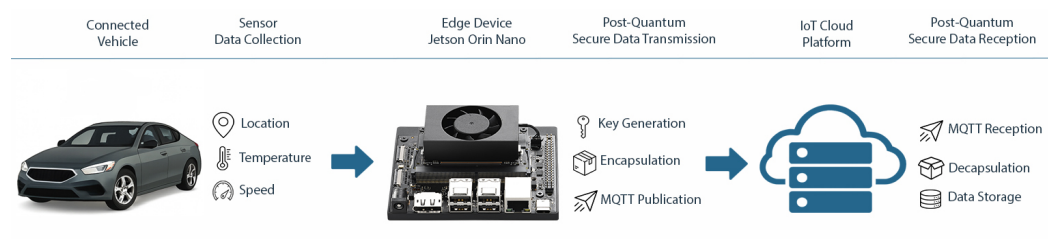
The practical outcome is a marked improvement in resilience against replay threats. Ensuring a distinct cryptographic signature per session prevents reuse of message data and helps keep only authorized operations on the network.

## 5. Implementation and Experiments

### 5.1. System Overview

This subsection describes the architecture and operational workflow of the proposed post-quantum secure MQTT-based vehicular IoT system. The objective is to clarify how CRYSTALS-Kyber is integrated into the data transmission pipeline and how cryptographic operations interact with MQTT communication and cloud storage.

Figure 1 illustrates the overall system architecture. The system is organized into three functional layers, comprising an edge layer based on the Jetson Orin Nano platform where vehicular sensor data are collected and encrypted, an MQTT-based communication layer that includes an MQTT broker responsible for message routing, and a cloud layer in which authorized subscribers decrypt the received data and store it in a time-series database.



**Figure 1.** Workflow of the proposed post-quantum secure MQTT-based vehicular IoT system.

At the edge layer, multiple onboard sensors generate telemetry data that must be transmitted securely to the cloud. Prior to transmission, the Jetson device performs the CRYSTALS-Kyber Key Encapsulation Mechanism to establish a shared secret with the receiving endpoint. This process involves three cryptographic phases, key generation, encapsulation, and decapsulation, which are distributed across the communicating endpoints. Key generation is executed during session initialization, while encapsulation is performed before data transmission. Decapsulation is carried out on the receiving side to recover the shared secret.

Once the shared secret is established, it is used to protect the confidentiality of sensor payloads prior to publication via MQTT. The encrypted payload is then transmitted through the MQTT broker using standard publish/subscribe mechanisms. Importantly, the integration of PQC does not alter the core MQTT protocol logic, preserving compatibility with existing MQTT infrastructures.

On the cloud side, encrypted messages are received, decrypted using the decapsulated shared secret, and stored in an InfluxDB time-series database, version 2.7.12. Integrity checks are performed at the cloud layer to confirm that the transmitted data remains unaltered during transport. The stored data can subsequently be queried and analyzed in near real time, enabling timely processing of vehicular telemetry streams.

This architecture decouples cryptographic processing from network transmission, allowing cryptographic costs to be evaluated independently from communication latency.

It also ensures that post-quantum security is applied transparently within the MQTT workflow, without introducing significant protocol overhead. The proposed design therefore provides a practical and scalable approach for deploying post-quantum secure communication in vehicular IoT environments with strict energy, latency, and thermal constraints.

### 5.2. Edge Layer

A Jetson board serves as the main data processing hub for the vehicle. It interfaces with temperature probes, speed sensors and GPS modules. Together these components provide continuous real-time insight into vehicle performance and the surrounding environment.

A script developed in Python 3.12.3 runs automatically every minute under a task scheduler such as cron on Linux systems. Each run reads the most recent sensor values, packages them into a structured payload and encrypts the result with the Kyber algorithm. Encryption is applied before transmission to protect data while in transit.

Encrypted payloads are published to a Mosquitto MQTT broker 2.0.18, which manages and routes messages across the IoT network. Mosquitto was chosen because it is open-source, lightweight and compliant with MQTT standards. These traits make it well suited to the resource-constrained environments common in many IoT deployments.

Kyber-512 was selected due to its design goals of low computational complexity and suitability for resource-constrained embedded platforms. The broker helps ensure reliable delivery and supports the real-time updates required for vehicle monitoring [26].

Devices operate as clients while the broker manages message delivery. Publishers post data to topics, and subscribers receive only the information relevant to them. This separation simplifies scaling and preserves flexibility as devices join or leave the network. In practice MQTT streamlines telemetry collection and distributes events across large fleets of sensors and actuators. Dispersed signals become coherent information that enables effective monitoring and control.

Connectivity is one of Jetson's strong points; the module includes a system on chip that pairs an ARM CPU with a GPU and can be fitted with a 5G modem and a SIM card for cellular access. The implemented configuration enables real-time transmission of telemetry to cloud-based IoT platforms, allowing secure and reliable connections that permit remote monitoring and advanced analytics. Vehicle sensor data can be processed off board, and the resulting analyses may prompt maintenance actions, generate alerts, or inform decisions at the fleet level.

The implemented architecture provides timely access to vehicular data while adding a layer of security through end-to-end encryption [27].

### 5.3. Cloud Layer

An IoT cloud environment was created by provisioning a virtual machine on Amazon Web Services [28] running an Ubuntu Server Ubuntu 24.04.3 LTS. Mosquitto was installed to receive messages forwarded from the Jetson, and InfluxDB was deployed to store and query time-series sensor data efficiently.

The application logic was written in Python 3.12.3 to handle incoming telemetry. Scheduled tasks pick up messages, decrypt payloads with the Kyber scheme and prepare the data for storage and analysis. Libraries such as Paho-MQTT 1.6.1 and a Kyber implementation were added to manage MQTT communication and encryption, resulting in a reliable, secure pipeline for processing vehicle telemetry.

### 5.4. Data Collection

The data collection system links a Jetson board with a set of vehicle sensors. Temperature probes, speed sensors that track velocity, GPS modules for precise positioning, and tire

pressure monitors feed the platform continuously. Door status and other discrete signals are also captured.

Readings arrive in real time and often change rapidly. Some values update every second, and others less frequently. The mix of continuous and event-driven data gives a fuller picture of vehicle behavior and the surrounding environment. This stream of information supports situational awareness and practical decision making. Alerts can be raised when thresholds are crossed, and trends can be followed over time. The result is a more informed view of vehicle state and a stronger basis for timely actions.

### 5.5. Experimental Methodology

All experiments were conducted on a Jetson Orin Nano running Ubuntu Linux 22.04. Execution times for key generation, encryption, and decryption were measured using high-resolution system timers available in the Linux environment.

Although Kyber-512 was selected for the end-to-end vehicular sensor-to-cloud pipeline due to its balance between security and efficiency, additional experiments were performed using Kyber-768 and Kyber-1024 in isolation. These supplementary measurements aim to analyze scalability and security–performance trade-offs across different NIST security levels on the same hardware platform.

### 5.6. Key Pair Generation

Before any data is sent, a public- and private-key pair is generated using the Kyber algorithm. This step secures communications, a critical requirement in IoT settings where data privacy matters greatly.

The key generation rests on the Module Learning With Errors problem, often abbreviated to M-LWE, which gives a solid theoretical defense against quantum attacks. The implementation follows Kyber’s Key Encapsulation Mechanism, or KEM, a structure designed to establish a symmetric key over an insecure channel in a secure way.

The chosen configuration uses the Kyber-512 variant, aligned with NIST Security Level 1. This selection balances strong protection with the need for lightweight, efficient operation suitable for the Jetson Orin Nano hardware.

Additional experiments with Kyber-768 and Kyber-1024 were planned to assess scalability across different NIST security levels.

The generation involves computationally intensive steps inherent to the lattice-based scheme:

1. Matrix Generation: Generation of the public matrix  $A$ .
2. Noise Sampling: Sampling of the secret polynomial  $s$  and the error polynomial  $e$  from specialized distributions.
3. Public-Key Derivation: Calculation of the public key  $t$ , which is derived from the matrix  $A$  and the secret components.

The Kyber algorithm is based on asymmetric cryptography, which involves the use of two distinct keys:

- **Public Key:** This key is intended for distribution and can be shared openly. In our system, it is used only to encapsulate a symmetric key via Kyber KEM. The symmetric key is then used to encrypt the sensor payload. The public key does not directly encrypt data.
- **Private Key:** This key must remain confidential and securely stored on the device that generated the key pair, in this case, Jetson. The private key is essential for decrypting messages that have been encrypted with the public key.

The private key is never shared or transmitted. Its main functions include:

- Decryption: The receiving client uses the private key to decapsulate the symmetric key sent by the Jetson, which is then used to decrypt the sensor payload.
- Digital Signatures: If the system implements digital signatures, the private key can be used to sign messages, enabling recipients to verify the authenticity of the data.

The generation of the key pair is executed using a dedicated library for Kyber (see Algorithm 1).

---

**Algorithm 1** Generate Public–Private Key Pair using Kyber-512

---

```

1: security_parameter ← [Kyber-512]
2: key_pair ← KeyPair.generate(security_parameter)
3: public_key ← key_pair.public_key
4: private_key ← key_pair.private_key

```

---

In this example:

- The Keys.generate() method creates a new key pair, where key\_pair contains both the public and private keys.
- The public\_key variable stores the public key, while the private\_key variable retains the private key.

### 5.7. Configuration

Following the generation of the public key, it is essential to configure the IoT cloud platform to securely receive and utilize this key for decrypting incoming data. The public key is used locally on the Jetson device to encapsulate a symmetric key via Kyber KEM. The resulting encapsulated key, along with the encrypted sensor payload, is sent to the MQTT broker.

This structured approach not only helps in data organization but also prepares the data for the encryption process.

### 5.8. Data Encryption

The sensor payload is never encrypted directly with the public key. Instead, it is protected using a post-quantum secure key exchange with CRYSTALS-Kyber. Kyber serves as a Key Encapsulation Mechanism (KEM) to establish a shared symmetric key between the Jetson and the cloud platform, as detailed in Algorithm 2. This symmetric key is then used with a cipher AES-128-GCM to encrypt the sensor payload.

On the Jetson Orin Nano, the encapsulation and decapsulation operations complete in milliseconds, enabling low-latency end-to-end communication. The short execution times also imply low energy consumption per operation, which is advantageous for power-sensitive IoT deployments.

Kyber's high efficiency is achieved through optimized polynomial arithmetic operating in a specific ring structure. The most time-consuming operation in Kyber, polynomial multiplication, is accelerated by leveraging the Number Theoretic Transform (NTT) and its inverse (INTT). The NTT is crucial for enhancing overall computational speed and supports Kyber's advantages in computational efficiency and resource usage on platforms such as the Jetson Orin Nano. System performance relies heavily on how efficiently these internal mathematical routines are executed on the Jetson's underlying hardware architecture.

**Algorithm 2** Encrypt Data

---

```

1: function ENCRYPT DATA(data_string, public_key)
2:   sym_key, encapsulated  $\leftarrow$  Kyber.encapsulate(public_key)
3:   data_bytes  $\leftarrow$  Encode(data_string)
4:   encrypted_data  $\leftarrow$  SymmetricEncrypt(data_bytes, sym_key)
5:   return encapsulated, encrypted_data
6: end function

```

---

*5.9. Transmission of Encrypted Data*

Once a shared symmetric key is established via Kyber encapsulation on the Jetson, it is used to encrypt the sensor payload. The encapsulated key (encapsulated) is included with the encrypted payload (encrypted\_data) in the same MQTT message. The broker simply forwards this message and does not have the ability to decrypt it. Only authorized subscribers possessing the corresponding private key can decapsulate the key and decrypt the payload to access the sensor metrics.

Keeping the payload encrypted before and during transmission preserves confidentiality. Delivery depends on factors such as message quality of service, retry policies, and local buffering on the device. Network interruptions and broker downtime can interrupt delivery, so simple recovery measures are advisable.

Authorized subscribers can retrieve and decrypt the messages. Proper key management and access control remain essential to prevent unauthorized access. The next sections describe implementation details and the mechanisms used to improve robustness and fault tolerance.

*5.10. Setup of MQTT Client*

Communication with the Mosquitto broker starts by creating an MQTT client on the device. This client opens and keeps the connection alive and takes care of publishing the encrypted payloads. In Python the paho-mqtt library is a common choice because it is straightforward to use and supports the main MQTT features.

First the client is instantiated and then configured with the broker IP address and port, as shown in Algorithm 3. After connecting, the client handles reconnection attempts, callback functions and message acknowledgments according to the chosen quality of service. These behaviors help maintain a steady data flow even when the network is unreliable.

Simple recovery measures improve robustness. For example, retry logic and local buffering reduce the chance of data loss during short outages. Proper handling of callbacks and acknowledgments also makes delivery more predictable.

Overall the client acts as the bridge between the Jetson and the broker. It publishes encrypted messages, manages delivery semantics and helps keep the system resilient under real-world conditions.

**Algorithm 3** Initialize MQTT Client

---

```

1: broker_address  $\leftarrow$  [broker_ip_address]
2: broker_port  $\leftarrow$  [broker_port]
3: mqtt_client.connect(broker_address, broker_port)

```

---

This code snippet initializes an MQTT client using the Paho MQTT library in Python. It begins by importing the necessary library and defining the address and port of the MQTT broker. The variable broker\_address holds the IP address of the broker, while port is designated for the port number, which is typically the default MQTT port. Finally, an instance of the MQTT client is created using mqtt.client(), preparing it for further configuration and communication with the specified broker.

### 5.11. Establishing the Connection

After configuring the client, the next step is to establish a connection to the MQTT broker. The connect method is utilized to initiate this connection, followed by the loop\_start method to handle network traffic and callbacks in the background (see Algorithm 4).

---

#### Algorithm 4 Connect to MQTT Broker and Start Loop

---

```

1: broker_address ← [broker_ip_address]
2: broker_port ← [broker_port]
3: mqtt_client ← mqtt.Client()
4: mqtt_client.connect(broker_address, broker_port)
5: mqtt_client.loop_start()

```

---

This code establishes a connection to the broker, allowing the client to publish messages and subscribe to topics. The loop\_start method runs a loop in a separate thread, enabling asynchronous message handling.

### 5.12. Publishing the Encrypted Data

After establishing the connection, the encrypted data can be published to a designated topic on the MQTT broker. Topics are hierarchical strings that allow for organized message distribution and subscription. The following code snippet illustrates how to publish the encrypted\_data to a specified topic, for example, vehicle/sensors/encrypted (see Algorithm 5).

---

#### Algorithm 5 Publish Encrypted Data to MQTT Topic

---

```

1: payload ← Package(encapsulated, encrypted_data)
2: mqtt_client.publish(topic, payload)

```

---

In this algorithm, the publish method sends the encrypted\_data to the specified topic. The broker handles the delivery of this message to all clients that have subscribed to that topic, facilitating efficient data sharing.

### 5.13. Handling Connection and Publish Events

The MQTT protocol includes callback functions that can be defined to handle various events, such as successful connections, message arrivals, or publish confirmations. By setting these callbacks, developers can ensure that the application behaves correctly in response to different scenarios. For instance, a callback can be added to confirm the successful publication of a message (see Algorithm 6).

---

#### Algorithm 6 Setup Callback for Successful Publish

---

```

1: mqtt_client.on_publish ← OnPublish

```

---

This algorithm defines a simple callback function that prints a confirmation message when the publication is successful. Assigning this function to the on\_publish attribute of the client allows the application to respond to publish events appropriately.

### 5.14. Disconnection

After publishing the data, it is important to disconnect the client from the MQTT broker cleanly. This can be accomplished using the disconnect method, followed by stopping the MQTT loop.

Algorithm 7 ensures the client properly terminates its connection to the broker, freeing resources and ensuring that all messages have been processed.

**Algorithm 7** Disconnect from MQTT Broker and Stop Loop

---

```

1: mqtt_client.disconnect()
2: mqtt_client.loop_stop()

```

---

*5.15. Data Reception and Decryption*

On the receiving end, clients retrieve the encapsulated key sent from the Jetson device and use the Kyber decapsulation function to derive the shared symmetric key. This symmetric key is then used to decrypt the actual sensor payload using a secure symmetric cipher AES-128-GCM, ensuring confidentiality and integrity.

*5.16. Subscription to MQTT Topics*

Clients first connect to Mosquitto and subscribe to the topic that carries the sensor data. Subscribing makes sure only relevant messages are received. The broker forwards published packets to each active subscriber. MQTT has a light footprint and fits devices with limited CPU memory and power. This keeps overhead low and helps preserve battery life. In real deployments, keeping session state small and limiting retained messages improves responsiveness and reduces unexpected load.

*5.17. Retrieving Encrypted Messages*

Once subscribed, the client will receive messages published to the designated topic. These messages contain the encrypted sensor data generated by Jetson. Each message includes not only the encrypted data but also any relevant metadata, such as timestamps or sensor identifiers, which aid in data management and organization.

*5.18. Decryption Process*

The decryption of the received data is performed using the private key generated during the initial key exchange. The private key must be securely stored and accessed by the receiving client to ensure the confidentiality of the decryption process (see Algorithm 8). The decryption process can be broken down into the following steps:

- **Retrieval of the Private Key:** The receiving client securely retrieves its private key, which was generated during the key pair establishment. This key is crucial for decrypting any messages encrypted with the corresponding public key.
- **Decryption Algorithm Execution:** The client uses the Kyber decryption algorithm, which takes the encrypted data and the private key as inputs. The decryption process can be implemented using a dedicated library for Kyber.
- **Verification and Validation:** After decryption, the client may implement verification mechanisms to ensure the integrity and authenticity of the received data. This can involve checking for any tampering or corruption during transmission. Techniques such as hash verification or digital signatures can be employed for this purpose.

**Algorithm 8** Decrypt Data

---

```

1: function DECRYPTDATA(encapsulated, encrypted_data, private_key)
2:   sym_key ← Kyber.decapsulate(encapsulated, private_key)
3:   decrypted_data ← SymmetricDecrypt(encrypted_data, sym_key)
4:   return decrypted_data
5: end function

```

---

This algorithm illustrates the fundamental operation where the decrypt function processes the encrypted data using the private key to yield the original sensor information.

### 5.19. Storing Decrypted Data

Once decryption succeeds, the original sensor readings can be processed further or archived for later analysis. In the implemented system, the decrypted records are written to InfluxDB, which is a time-series database optimized for timestamped data. This arrangement improves retrieval speed. It also enables longitudinal analysis and supports real-time monitoring and prompt decision making in IoT deployments.

## 6. Results and Discussion

This section presents and analyzes the experimental results obtained from the integration of CRYSTALS-Kyber into an MQTT-based vehicular IoT pipeline deployed on a Jetson Orin Nano platform. The evaluation focuses on computational performance, energy consumption, and end-to-end transmission latency.

The implementation combines MQTT with Kyber to send data from Jetson devices to an IoT cloud platform. MQTT is a lightweight messaging protocol that reduces overhead on resource-constrained devices. Multiple sensors on a Jetson board can reliably send data to the cloud, while energy consumption and bandwidth use remain low.

Our experiments measured the time required for key generation, encapsulation, and decapsulation. Energy consumption per operation was estimated from CPU utilization and operation duration using standard power models for the Jetson Orin Nano. The results indicate that Kyber provides strong protection against quantum attacks and runs efficiently on low-power devices, balancing security and performance in this environment.

Encrypted data arriving in the cloud platform was decrypted and stored in InfluxDB without any loss of integrity. Retrieval tests confirm that time-series data can be queried quickly for further analysis, supporting timely processing of sensor streams.

Based on these positive outcomes, several improvements are proposed to further optimize performance and security in the context of PQC. Priority should be given to developing advanced authentication mechanisms adapted to the properties of post-quantum algorithms. Stronger authentication restricts publishing and subscribing to authorized devices and reduces the risk of unauthorized access. Exploring additional post-quantum algorithms that could complement or strengthen the Kyber-based implementation is also advisable, as combining multiple cryptographic techniques improves resilience to future vulnerabilities. Enhancing monitoring and logging of transmissions enables early detection of anomalies, with careful balancing of logging depth against storage and privacy requirements. Cryptographic choices should be regularly reassessed as quantum computing advances, with updates prioritized according to risk and practical impact.

### *Cryptographic Performance Evaluation*

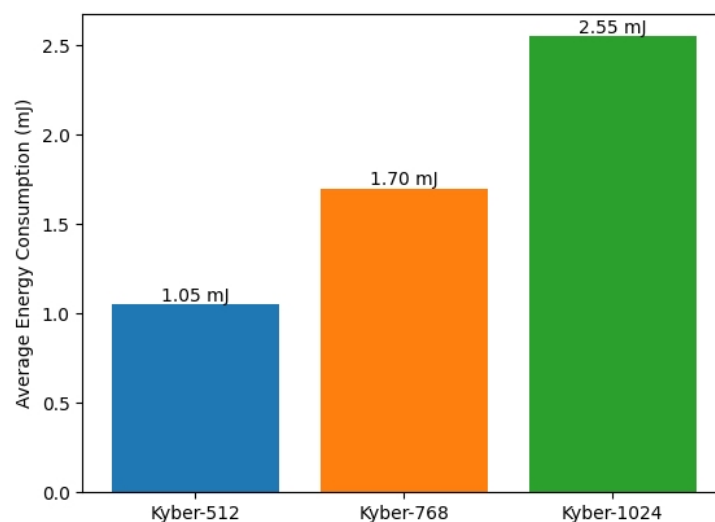
The computational performance of CRYSTALS-Kyber was evaluated on a Jetson Orin Nano platform by measuring the execution time of key generation, encapsulation, and decapsulation operations across the three NIST security levels. Each measurement is reported as the average of 1000 runs.

Table 2 summarizes the measured execution times for all operations. As expected, key generation requires the longest processing time, followed by decapsulation and encapsulation. Kyber-512 exhibits the lowest computational latency, making it particularly suitable for low-latency, energy-sensitive IoT deployments.

**Table 2.** Execution time for Kyber operations on Jetson Orin Nano.

| Kyber Variant | Key Generation (ms) | Encapsulation (ms) | Decapsulation (ms) |
|---------------|---------------------|--------------------|--------------------|
| Kyber-512     | 0.52                | 0.19               | 0.23               |
| Kyber-768     | 0.88                | 0.31               | 0.36               |
| Kyber-1024    | 1.38                | 0.46               | 0.53               |

Figure 2 presents the average energy consumption for encapsulation and decapsulation operations across the three Kyber variants, computed from CPU power profiles and measured execution times. The measured average energy per operation is 1.05 mJ for Kyber-512, 1.70 mJ for Kyber-768, and 2.55 mJ for Kyber-1024. Key generation, being more computationally intensive, consumes more energy, with approximate averages of 3.2 mJ, 5.1 mJ, and 7.0 mJ for Kyber-512, Kyber-768, and Kyber-1024, respectively. As the figure shows, energy consumption increases with the security level, reflecting the additional computational effort required for larger key sizes. Kyber-512 remains the most energy-efficient variant for encapsulation and decapsulation, while Kyber-1024 offers higher security at the cost of greater energy usage.

**Figure 2.** Average energy consumption for encapsulation and decapsulation operations per Kyber variant on Jetson Orin Nano.

Experimental monitoring indicated that Kyber operations impose a moderate computational load on the Jetson Orin Nano. Average CPU utilization varied with the security level and operation type. For key generation, CPU utilization averaged 38%, 48%, and 58% for Kyber-512, Kyber-768, and Kyber-1024, respectively. For encapsulation and decapsulation, CPU utilization was slightly lower, averaging 33%, 43%, and 53% for Kyber-512, Kyber-768, and Kyber-1024, respectively. These results confirm a lightweight computational footprint, suitable for real-time, energy-constrained vehicular IoT systems without adversely affecting system performance.

Network transmission time depends primarily on connectivity and payload size and remains effectively constant across all variants. Consequently, network overhead dominates end-to-end latency, and the integration of PQC introduces only minimal impact on real-time MQTT communication.

The experimental results support the efficient deployment of CRYSTALS-Kyber on the Jetson Orin Nano for secure MQTT-based vehicular telemetry. Key observations include the following:

- Kyber-512 provides the best trade-off between computational latency, energy consumption, and security for resource-constrained embedded platforms.
- Higher-security variants (Kyber-768 and Kyber-1024) remain feasible but introduce longer processing times and higher energy usage, which may be less suitable for latency-sensitive applications.
- MQTT transmission dominates end-to-end latency; cryptographic operations introduce negligible overhead relative to network delays.

Although Kyber-1024 demonstrates feasible end-to-end performance, it was not selected for deployment due to its higher energy consumption and increased cryptographic latency compared to Kyber-512, which offers a more suitable balance for resource-constrained and energy-sensitive IoT deployments.

These results confirm the lightweight and efficient operation of the proposed MQTT-Kyber system on the Jetson Orin Nano. Key generation, encapsulation, and decapsulation complete in milliseconds with low energy consumption, validating the practical feasibility of post-quantum secure communication in vehicular IoT scenarios with strict energy and thermal constraints.

## 7. Conclusions

This work presents the integration of the MQTT protocol with the CRYSTALS-Kyber PQC scheme as a secure communication framework for vehicular IoT applications. The experimental evaluation demonstrates that sensor data can be transmitted, decrypted, and stored reliably on an IoT cloud platform while maintaining strong post-quantum security. These results support the feasibility of the proposed approach and confirm that it achieves an effective balance between security requirements and operational constraints for resource-limited embedded devices.

The system's ability to perform end-to-end secure data handling, including key establishment, encryption, transmission, and decryption, confirms its operational integrity. This highlights its potential applicability in real-world scenarios such as smart cities, autonomous vehicles, and industrial IoT, where low-latency, energy-efficient, and secure communication is critical. The combination of a lightweight messaging protocol with post-quantum encryption ensures suitability for resource-constrained environments.

Further improvements, such as enhanced authentication, key management, and monitoring mechanisms, can reduce the system's attack surface and strengthen overall resilience. Optimizations in performance and resource usage will also support low-latency operation on embedded platforms, ensuring practical deployment in vehicular and IoT settings.

Future work will explore additional PQC algorithms to complement or enhance the Kyber-based implementation. Ongoing reassessment and adaptation to advances in quantum computing will be essential to maintain confidentiality and trust across interconnected IoT systems. Continued refinement of the proposed approach will ensure that vehicular and industrial IoT deployments remain secure, efficient, and resilient in the face of emerging cyber-physical threats.

**Author Contributions:** Conceptualization, J.D.-G. and C.C.-G.; methodology, J.D.-G. and C.C.-G.; software, J.D.-G. and C.C.-G.; validation, J.D.-G. and C.C.-G.; formal analysis, J.D.-G. and C.C.-G.; investigation, J.D.-G. and C.C.-G.; resources, J.D.-G. and C.C.-G.; data curation, J.D.-G. and C.C.-G.; writing—original draft preparation, J.D.-G. and C.C.-G.; writing—review and editing, J.D.-G. and C.C.-G.; visualization, J.D.-G. and C.C.-G.; supervision, J.D.-G. and C.C.-G.; project administration, J.D.-G. and C.C.-G. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the Recovery, Transformation and Resilience Plan, financed by the European Union (NextGenerationEU) under the strategic project C064/23 SCITALA (INCIBE-ULL).

**Data Availability Statement:** The data are available on request from the corresponding author.

**Acknowledgments:** This research has been possible thanks to support from the SCITALA C064/23 project, the Cybersecurity Chair at the University of La Laguna, INCIBE, and the European Union NextGenerationEU/PRTR.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Mansour, M.; Gamal, A.; Ahmed, A.I.; Said, L.A.; Elbaz, A.; Herencsar, N.; Soltan, A. Internet of Things: A Comprehensive Overview on Protocols, Architectures, Technologies, Simulation Tools, and Future Directions. *Energies* **2023**, *16*, 3465. <https://doi.org/10.3390/en16083465>.
2. Abdelkader, G.; Elgazzar, K.; Khamis, A. Connected Vehicles: Technology Review, State of the Art, Challenges and Opportunities. *Sensors* **2021**, *21*, 7712. <https://doi.org/10.3390/s21227712>.
3. Nurudin, A.A.S.; Zarlis, M. Monitoring applications for vehicles based on Internet of Things (IoT) using the MQTT protocol. *Procedia Comput. Sci.* **2023**, *227*, 73–82. <https://doi.org/10.1016/j.procs.2023.10.504>.
4. Al-Wesabi, F.N.; Alshahrani, A.; Alanazi, R.; Alshahrani, M.M.; Sorour, S.; Alghamdi, A.M.; Alamri, M.Z.; Alanazi, S. Integrated Communication and Location Monitoring System for Vehicle Monitoring via Smartphone Calls. *Alex. Eng. J.* **2025**, *128*, 1159–1167. <https://doi.org/10.1016/j.aej.2025.08.035>.
5. De Vincenzi, M.; Moore, J.; Smith, B.; Sarma, S.E.; Matteucci, I. Security risks and designs in the connected vehicle ecosystem: In-vehicle and edge platforms. *IEEE Open J. Veh. Technol.* **2025**, *6*, 442–454. <https://doi.org/10.1109/OJVT.2024.3524088>.
6. Cherkaoui Dekkaki, K.; Tasic, I.; Cano, M.-D. Exploring Post-Quantum Cryptography: Review and Directions for the Transition Process. *Technologies* **2024**, *12*, 241. <https://doi.org/10.3390/technologies12120241>.
7. Lohmiller, N.; Kaniewski, S.; Menth, M.; Heer, T. A survey of post-quantum cryptography migration in vehicles. *IEEE Access* **2025**, *13*, 10160–10176. <https://doi.org/10.1109/ACCESS.2025.3528562>.
8. Sharath, H.A.; Vrindavanam, J.; Dana, S.; Prasad, S.N. Quantum-resilient cryptography: A survey on classical and quantum algorithms. *IEEE Access* **2025**, *13*, 172854–172877. <https://doi.org/10.1109/ACCESS.2025.3612982>.
9. Nguyen, T.-H.; Dam, D.-T.; Duong, P.-P.; Kieu-Do-Nguyen, B.; Pham, C.-K.; Hoang, T.-T. Efficient hardware implementation of the lightweight CRYSTALS-Kyber. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2025**, *72*, 610–622. <https://doi.org/10.1109/TCSI.2024.3443238>.
10. Scalcon, F.P.; Tahal, R.; Ahrabi, M.; Huangfu, Y.; Ahmed, R.; Nahid-Mobarakeh, B.; Shirani, S.; Vidal, C.; Emadi, A. AI-Powered Video Monitoring: Assessing the NVIDIA Jetson Orin Devices for Edge Computing Applications. In *2024 IEEE Transportation Electrification Conference and Expo (ITEC)*; IEEE: New York, NY, USA, 2024; pp. 1–6. <https://doi.org/10.1109/ITEC60657.2024.10598994>.
11. Laghari, S.U.A.; Li, W.; Manickam, S.; Nanda, P.; Al-Ani, A.K.; Karuppayah, S. Securing MQTT Ecosystem: Exploring Vulnerabilities, Mitigations, and Future Trajectories. *IEEE Access* **2024**, *12*, 139273–139289. <https://doi.org/10.1109/ACCESS.2024.3412030>.
12. Shinji, Y.; Shiimoto, K. QoS in IoT-Based River Flood Monitoring System Using MQTT Brokers in Tandem Connection. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*; IEEE: New York, NY, USA, 2022; pp. 1–2. <https://doi.org/10.1109/WF-IoT54382.2022.10152261>.
13. Paul, G.A.; Jagnani, Y.; Supraja, P. Improving Fault Tolerance and Tackling Broker Failure in MQTT through Blockchain. In *2023 Third International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*; IEEE: New York, NY, USA, 2023; pp. 1–6. <https://doi.org/10.1109/ICAECT57570.2023.10118032>.
14. Ricci, S.; Dobias, P.; Malina, L.; Hajny, J.; Jedlicka, P. Hybrid Keys in Practice: Combining Classical, Quantum and Post-Quantum Cryptography. *IEEE Access* **2024**, *12*, 23206–23219. <https://doi.org/10.1109/ACCESS.2024.3364520>.
15. Parameswarath, R.P.; Sikdar, B. Quantum-Safe Authentication Protocol Using Post-Quantum Key Encapsulation Mechanism for Transportation Systems. In *2024 IEEE 4th International Conference on Sustainable Energy and Future Electric Transportation (SEFET)*; IEEE: New York, NY, USA, 2024; pp. 1–6. <https://doi.org/10.1109/SEFET61574.2024.10717975>.
16. Samandari, J.; Gritti, C. Post-Quantum Authentication in the MQTT Protocol. *J. Cybersecur. Priv.* **2023**, *3*, 416–434. <https://doi.org/10.3390/jcp3030021>.
17. Ehsan, M.A.; Alayed, W.; Rehman, A.U.; Hassan, W. u.; Zeeshan, A. Post-Quantum KEMs for IoT: A Study of Kyber and NTRU. *Symmetry* **2025**, *17*, 881. <https://doi.org/10.3390/sym17060881>.

18. Cruz-Piris, L.; Marín-López, A.; Álvarez-Campana, M.; Sanz, M.; Moreno, J.I.; Arroyo, D. Measuring the impact of post quantum cryptography in Industrial IoT scenarios. *Internet Things* **2025**, *34*, 101793. <https://doi.org/10.1016/j.iot.2025.101793>.
19. Almutairi, M.; Sheldon, F.T. Resilience of Post-Quantum Cryptography in Lightweight IoT Protocols: A Systematic Review. *Eng* **2025**, *6*, 346. <https://doi.org/10.3390/eng6120346>.
20. Kwala, A.K.; Mishra, A.; Kumar, A. Securing IoT communication: A module-LWE key exchange protocol with efficient error reconciliation technique. *Discov. Internet Things* **2025**, *5*, 84. <https://doi.org/10.1007/s43926-025-00176-8>.
21. Chung, J.; Vljajic, N. Survey of Remote TLS Vulnerability Scanning Tools and Snapshot of TLS Use in Banking Sector. In *2022 IEEE Conference on Communications and Network Security (CNS)*; IEEE: New York, NY, USA, 2022; pp. 1–2. <https://doi.org/10.1109/CNS56114.2022.9947230>.
22. Thankappan, M.; Rifà-Pous, H.; Garrigues, C. A Signature-Based Wireless Intrusion Detection System Framework for Multi-Channel Man-in-the-Middle Attacks Against Protected Wi-Fi Networks. *IEEE Access* **2024**, *12*, 23096–23121. <https://doi.org/10.1109/ACCESS.2024.3362803>.
23. Sun, H.-M.; Wu, M.-E.; Ting, W.-C.; Hinek, M.J. Dual RSA and Its Security Analysis. *IEEE Trans. Inf. Theory* **2007**, *53*, 2922–2933. <https://doi.org/10.1109/TIT.2007.901248>.
24. Li, N. Research on Diffie-Hellman Key Exchange Protocol. In *2010 2nd International Conference on Computer Engineering and Technology*; IEEE: New York, NY, USA, 2010; pp. V4-634–V4-637. <https://doi.org/10.1109/ICCET.2010.5485276>.
25. Guo, H.; Sun, J.; Pang, Z.-H. Analysis of Replay Attacks With Countermeasure for State Estimation of Cyber-Physical Systems. *IEEE Trans. Circuits Syst. II Express Briefs* **2024**, *71*, 206–210. <https://doi.org/10.1109/TCSII.2023.3302151>.
26. Hwang, K.; Lee, J.M.; Jung, I.H.; Lee, D.-H. Modification of Mosquitto Broker for Delivery of Urgent MQTT Message. In *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*; IEEE: New York, NY, USA, 2019; pp. 166–167. <https://doi.org/10.1109/ECICE47484.2019.8942800>.
27. Tripathi, P.; Miraz, M.H.; Joshi, S. Comparative Analysis of MongoDB and InfluxDB for Time Series Data Management in IoT Environments. In *2023 International Conference on Computing, Networking, Telecommunications & Engineering Sciences Applications (CoNTESA)*; IEEE: New York, NY, USA, 2023; pp. 39–42. <https://doi.org/10.1109/CoNTESA61248.2023.10384962>.
28. Rizvi, S.R.; Killough, B.; Cherry, A.; Gowda, S. Lessons Learned and Cost Analysis of Hosting a Full Stack Open Data Cube Application on AWS. In *IGARSS 2018—2018 IEEE International Geoscience and Remote Sensing Symposium*; IEEE: New York, NY, USA, 2018; pp. 8643–8646. <https://doi.org/10.1109/IGARSS.2018.8518084>.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.