future internet

MDPI

Article

HECS4MQTT: A Multi-Layer Security Framework for Lightweight and Robust Encryption in Healthcare IoT Communications

Saud Alharbi [1,*], Wasan Awad [2] and David Bell [1,*]

[1] Department of Computer Science, Brunel University, Uxbridge UB8 3PH, UK
[2] College of Information Technology, Ahlia University, Manama 10878, Bahrain; wawad@ahlia.edu.bh
* Correspondence: saud.alharbi@brunel.ac.uk (S.A.); david.bell@brunel.ac.uk (D.B.)

Abstract

Internet of Things (IoT) technology in healthcare has enabled innovative services that enhance patient monitoring, diagnostics and medical data management. However, securing sensitive health data while maintaining system efficiency of resource-constrained IoT devices remains a critical challenge. This work presents a comprehensive end-to-end IoT security framework for healthcare environments, addressing encryption at two key levels: lightweight encryption at the edge for resource-constrained devices and robust end-to-end encryption when transmitting data to the cloud via MQTT cloud brokers. The proposed system leverages multi-broker MQTT architecture to optimize resource utilization and enhance message reliability. At the edge, lightweight cryptographic techniques ensure low-latency encryption before transmitting data via a secure MQTT broker hosted within the hospital infrastructure. To safeguard data as it moves beyond the hospital to the cloud, stronger end-to-end encryption are applied to ensure end-to-end security, such as AES-256 and TLS 1.3, to ensure confidentiality and resilience over untrusted networks. A proof-of-concept Python 3.10 -based MQTT implementation is developed using open-source technologies. Security and performance evaluations demonstrate the feasibility of the multi-layer encryption approach, effectively balancing computational overhead with data protection. Security and performance evaluations demonstrate that our novel HECS4MQTT (Health Edge Cloud Security for MQTT) framework achieves a unique balance between efficiency and security. Unlike existing solutions that either impose high computational overhead at the edge or rely solely on transport-layer protection, HECS4MQTT introduces a layered encryption strategy that decouples edge and cloud security requirements. This design minimizes processing delays on constrained devices while maintaining strong cryptographic protection when data crosses trust boundaries. The framework also introduces a lightweight bridge component for re-encryption and integrity enforcement, thereby reducing broker compromise risk and supporting compliance with healthcare security regulations. Our HECS4MQTT framework offers a scalable, adaptable, and trust-separated security model, ensuring enhanced confidentiality, integrity, and availability of healthcare data while remaining suitable for deployment in real-world, latency-sensitive, and resource-limited medical environments.

Keywords: Internet of Things (IoT); MQTT; healthcare security; lightweight cryptography; end-to-end encryption; cloud security; multi-broker architecture

Academic Editors: Di Lu and Yichuan Wang

Received: 28 May 2025
Revised: 20 June 2025
Accepted: 27 June 2025
Published: 30 June 2025

Citation: Alharbi, S.; Awad, W.; Bell, D. HECS4MQTT: A Multi-Layer Security Framework for Lightweight and Robust Encryption in Healthcare IoT Communications. *Future Internet* **2025**, *17*, 298. https://doi.org/10.3390/fi17070298

1. Introduction

Internet of Things (IoT) adoption in healthcare has revolutionized the delivery of medical services, enabling real-time monitoring, personalized care, and remote diagnostics. Wearable devices, implantable sensors, and mobile health applications now form a connected ecosystem that continuously generates sensitive health data. These data are transmitted across networks to cloud platforms or healthcare provider dashboards for processing and intervention. However, the inherent resource constraints of IoT devices and the critical sensitivity of health-related data present significant challenges to ensuring security, privacy, and system efficiency.

Conventional security protocols, such as TLS and IPsec, while robust, are often unsuitable for constrained environments due to their computational and memory overhead [1–3]. In many IoT deployments, particularly in e-health scenarios, devices operate with limited processing power, battery capacity, and communication bandwidth. As a result, there is a growing demand for lightweight cryptographic techniques that can ensure data confidentiality and integrity without degrading device performance or user experience. For instance, wearable heart rate monitors or portable glucose sensors typically rely on low-power microcontrollers, which lack the resources to efficiently perform full TLS handshakes.

This research presents a secure, scalable IoT communication framework tailored for e-health environments. The proposed system incorporates multi-layer encryption strategies and a multi-broker MQTT architecture to ensure efficient, end-to-end protection of patient data across distributed hospital settings. At the edge, lightweight encryption schemes are used to minimize latency and preserve device resources. Data is first transmitted to a local broker within the hospital network and then securely forwarded to a centralized cloud MQTT broker, where more computationally intensive security operations are applied. Finally, authorized medical staff access the data through a secure web or mobile application, acting as a subscriber. In this work, we extend our previous evaluation of lightweight encryption schemes presented in [4] by designing a secure, dual-layer MQTT communication architecture tailored for healthcare IoT systems. The proposed framework integrates Salsa20-Blake2b encryption at the edge with AES-256 over TLS 1.3 for secure transmission to the cloud. A custom bridge application is introduced to seamlessly manage the cryptographic transition between edge and cloud environments, ensuring end-to-end data confidentiality while maintaining performance efficiency.

To evaluate the effectiveness of the proposed approach, the system was implemented using multiple encryption schemes and benchmarked in terms of encryption time, memory usage, CPU cycles, and throughput. The results demonstrate the feasibility of secure, real-time data transmission in healthcare settings using layered encryption without relying solely on heavy TLS-based solutions.

The paper is organized as follows. Section 2 presents the theoretical foundations of lightweight cryptography and MQTT-based communication. Section 3 reviews related work in the field of IoT encryption and MQTT security. Section 4 introduces the proposed system architecture. Section 5 details the implementation of the publisher and subscriber modules. Section 6 presents experimental results, performance and security analysis. Finally, Section 7 concludes the paper and outlines future work.

2. Theoretical Background

2.1. Message Queue Telemetry Transport (MQTT)

The Message Queuing Telemetry Transport (MQTT) protocol [5] is a lightweight, open-standard publish–subscribe messaging protocol designed for constrained environments and low-bandwidth, high-latency networks. Standardized by OASIS and ISO/IEC 20922, MQTT is widely adopted in Internet of Things (IoT) applications to facilitate efficient

communication between distributed devices and services. Figure 1 illustrates a standard MQTT communication model secured by TLS. In this architecture, the MQTT Publisher sends messages on a specific topic (e.g., Topic1) to the MQTT Broker over a TLS-secured session. The MQTT Subscriber, which has previously subscribed to the same topic, also communicates with the broker via another TLS session, ensuring that both data transmission and topic subscriptions are encrypted. This setup ensures the confidentiality and integrity of the messages exchanged over potentially insecure networks.

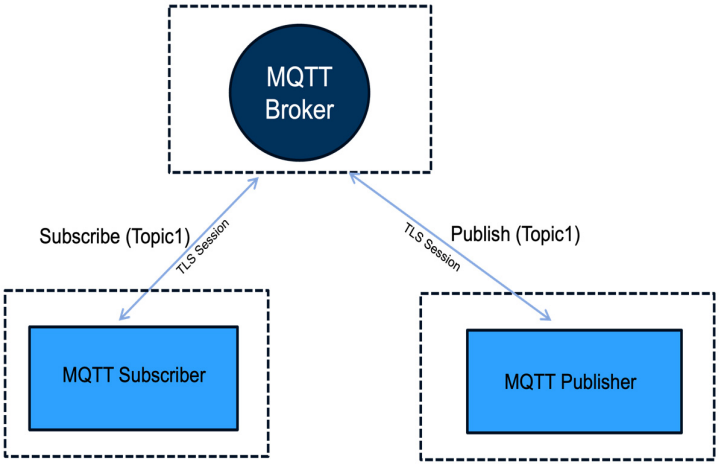

Figure 1. MQTT protocol.

MQTT operates [5] using a client–broker–client architecture. Publishers send messages to a centralized broker, which then distributes them to all subscribed clients based on hierarchical topic filters. This model ensures decoupling between data producers and consumers, enhancing scalability, interoperability, and system modularity. However, MQTT inherently lacks direct end-to-end communication between clients. MQTT Version 5.0 [5] introduces several protocol-level enhancements that address limitations in previous versions and improve flexibility, robustness, and extensibility [5]. Notable features include:

1. Reason Codes: provide explicit acknowledgment feedback, enabling more granular error reporting and diagnostic capability.
2. Shared Subscriptions: allow multiple clients to share a single subscription, supporting distributed processing and load balancing.
3. Message Expiry Intervals: define the validity period of a message, preventing outdated data from being delivered.
4. Topic Aliases: replace long topic strings with shorter aliases to reduce payload size, which is beneficial for low-bandwidth links.
5. User Properties: enable the transmission of custom metadata through MQTT packets for application-specific use cases.

Security in MQTT is typically implemented through Transport Layer Security (TLS), which ensures confidentiality, integrity, and authentication of transmitted data. However, MQTT itself does not enforce any specific security mechanisms, placing responsibility on the application layer to implement key management, access control, and encryption policies appropriate for the deployment context. For full protocol specification and implementation guidance, the official OASIS documentation of MQTT Version 5.0 serves as the authoritative reference [5].

2.2. Lightweight Cryptography for MQTT

Lightweight cryptography has gained prominence as a practical solution for addressing the limitations of resource-constrained IoT devices, including limited processing power,

restricted memory, and energy constraints. Lightweight cryptography refers to encryption techniques designed to minimize resource usage while maintaining acceptable levels of security [4]. Algorithms such as ChaCha20, Salsa20, SIMON, SPECK, and TinyAES are particularly well-suited for edge environments due to their reduced key scheduling complexity, smaller code footprint, and lower computational requirements. These schemes often trade off marginal cryptographic strength or feature sets in exchange for practical performance on constrained platforms.

Authenticated encryption schemes [4], particularly AEAD (Authenticated Encryption with Associated Data) modes like AES-GCM and ChaCha20-Poly1305, are critical in such applications as they ensure both confidentiality and message integrity. The choice of cipher must balance security requirements with device capabilities, especially when continuous data transmission is involved, such as in real-time patient monitoring scenarios.

3. Related Work

Securing communication in resource-constrained IoT environments, particularly in healthcare, has received increasing attention in recent years. Traditional encryption approaches, such as TLS or IPsec, offer robust security but remain impractical for low-power devices due to high computational overhead and memory consumption [1–3]. This limitation has motivated the development and evaluation of lightweight cryptographic algorithms optimized for embedded systems.

3.1. Lightweight Encryption in IoT

Singh et al. [6] proposed the SMQTT and SMQTT-SN security schemes, which utilize Key Policy and Ciphertext Policy Attribute-Based Encryption (KP/CP-ABE) in combination with lightweight Elliptic Curve Cryptography. Their work included a security analysis demonstrating robustness against several attacks. However, the study lacked a comprehensive performance evaluation, as it did not offer a comparative analysis with alternative protocols. Pal et al. [7] and Wang et al. [8] proposed IoT encryption using Attribute-Based Encryption (ABE) to ensure data confidentiality. However, these approaches impose considerable overhead due to the reliance on the ABE protocol, which may limit their suitability for resource-constrained publisher devices. Another ABE-based solution was presented by Bisne et al. [9]. It was compared with other AES protocols with different modes and demonstrated better results compared to AES. Iqbal et al. [10] proposed a new security scheme for MQTT by integrating MQTT with the ARIA encryption algorithm for securing the MQTT payload and MbedTLS for network tunnel encryption. Some other security approaches were related to improving AES by deriving new protocols from the standard AES, such as the work presented in [11–14]. Hamad et al. [15] presented a new enhanced protocol called SEEMQTT, which is proposed to ensure end-to-end data confidentiality, integrity, and authorization between publishers and subscribers. The framework establishes a secure communication channel between clients and KeyStore servers using Identity-Based Encryption (IBE). Buccafurri et al. [16] introduced MQTT-I, a protocol designed to provide end-to-end data flow integrity in MQTT systems, even in the presence of an untrusted broker. Leveraging Merkle Hash Trees, the approach ensures integrity across dynamic, multi-topic data streams without requiring direct coordination between publishers and subscribers. Although, it was proven by authors, the scope of this study is limited to data integrity. Li et al. [17] proposed iTLS, a lightweight secure transport protocol tailored for resource-constrained IoT environments, which employs identity-based encryption to reduce handshake complexity and session overhead. Unlike traditional TLS, iTLS employs identity-based cryptography (IBC) to enable certificate-free, implicit mutual authentication and early key generation, allowing encrypted data to be sent without additional round trips.

Compatible with TLS 1.3, iTLS reduces network overhead by over 61% and handshake latency by at least 60%, offering strong security and efficiency for low-power IoT networks. Although iTLS is more lightweight than traditional TLS, it still requires the use of AES, ChaCha20, or a similar encryption scheme for payload-level protection to ensure true end-to-end security. This is particularly important when the MQTT broker is untrusted or compromised—a risk that becomes more significant in cloud-hosted broker environments.

The introduction of authenticated encryption schemes has further expanded the design space for secure IoT communication. Sadio et al. [18] proposed securing MQTT and MQTT-SN communication in constrained environments using the ChaCha20-Poly1305 authenticated encryption with associated data (AEAD) scheme. Their approach leverages ChaCha20 as a lightweight stream cipher for confidentiality, combined with Poly1305 as a one-time message authenticator to ensure data integrity and authenticity. A similar improved security scheme was proposed by Alharbi et al. [4], which employs Salsa20 for confidentiality and Blake2b for message integrity. A comprehensive performance evaluation was conducted against several encryption protocols, including AES variants, TinyAES-HMAC, ASCON, SIMON-HMAC, SPECK-HMAC, and ChaCha20-Poly1305, using both 128-bit and 256-bit keys where applicable. The results demonstrate that the proposed scheme achieves lower CPU cycles, faster encryption time, and higher throughput. While the scheme in [4] focused on evaluating the performance of a lightweight encryption configuration using Salsa20 and Blake2b at the edge, our proposed HECS4MQTT framework extends this by integrating it into a multi-layer architecture that separates edge and cloud security. HECS4MQTT framework incorporates multi-broker MQTT communication and introduces a bridge component that re-encrypts data with AES-256 and TLS 1.3 before transmission to the cloud.

Performance benchmarking of encryption algorithms is essential in the context of IoT hardware because of the tight constraints imposed by device hardware. For example, Wijayanto et al. [19] conducted a comparative analysis of the AES, Grain V1, and RC4 encryption algorithms, evaluating both their security and performance effectiveness. The study confirmed that all three algorithms successfully mitigated passive sniffing threats. Additionally, a cryptanalytic assessment was performed to estimate the time required to break each algorithm's key, revealing that AES offered the highest level of resistance and thus the strongest security among the three. Performance benchmarks showed that RC4 delivered the fastest encryption and decryption times. Al-Ani et al. [20] conducted an in-depth evaluation of the MQTT protocol by experimentally assessing the impact of various cryptographic techniques—specifically AES-CBC, RSA, and an ECC-AES hybrid scheme—on processing time and message size. The findings indicated that payload encryption consistently led to increased processing latency and larger message sizes. These results underscore the trade-offs between security strength and communication efficiency when integrating cryptographic methods into MQTT-based systems. The existing body of literature on securing MQTT-based communication in IoT environments consistently emphasizes the challenge of achieving a balance between robust security and the operational constraints of resource-limited devices [1–3]. A wide range of encryption protocols—ranging from traditional symmetric schemes to modern lightweight cryptography—have been proposed and evaluated to address these challenges. However, each solution presents inherent trade-offs. Protocols offering stronger cryptographic guarantees often introduce increased computational complexity, memory consumption, and energy usage, which can negatively impact the performance and responsiveness of constrained nodes [1–3]. Conversely, schemes optimized for low overhead may compromise on cryptographic strength.

3.2. MQTT Multi-Broker Related Work

Multi-broker MQTT architectures have been investigated to improve the resilience and scalability of MQTT-based IoT environments. Kurdi et al. [21] proposed a multi-tier MQTT architecture based on fog computing, where each MQTT broker is equipped with an authentication manager responsible for mutual authentication between entities. Their approach introduces a lightweight authentication scheme that combines hash functions with XOR operations, targeting reduced storage and communication overhead. Their evaluation reports an 89% reduction in storage overhead and a 23% reduction in communication overhead, highlighting its efficiency and scalability in industrial IoT scenarios. The focus and contributions of the current work differ in both scope and depth. Rather than addressing authentication as the primary security concern, this research centers on the confidentiality and integrity of IoT e-health data at the edge and end-to-end security when data moves to the cloud. Also, there are other papers that introduced a similar multi-layer MQTT without focusing on encryption, like the work presented in [22,23].

HECS4MQTT addresses a clear gap in current work, where existing solutions either employ strong encryption mechanisms such as TLS, AES, or identity-based encryption that are unsuitable for resource-constrained IoT devices or rely solely on lightweight encryption schemes that lack sufficient protection when data moves beyond the edge to less trusted environments such as the cloud. Compared to existing works, our proposed architecture integrates lightweight encryption at the edge—where IoT devices operate under strict resource constraints—and applies robust end-to-end encryption when transmitting data from hospital premises to the cloud. This layered encryption approach is embedded within a scalable MQTT framework, where multiple brokers are deployed to localize traffic and reduce bottlenecks across distributed hospital networks. In our system, each local broker is responsible for managing encrypted communication from resource-constrained publishers, enabling low-latency and energy-efficient data transmission. As data transitions from the edge to external cloud platforms, additional encryption layers are applied, ensuring end-to-end security across potentially untrusted network segments. This separation of encryption responsibilities between the edge and the cloud ensures that encryption protocols are tailored to the capabilities of each system component. This layered design mitigates risks associated with centralized processing, reduces latency, and enhances message reliability. The use of multi-broker coordination further enables healthcare providers to access encrypted patient data from any authorized endpoint while maintaining compliance with privacy regulations and ensuring cryptographic resilience throughout the data lifecycle.

While many of these studies evaluate individual encryption methods or architectural components, relatively few integrate a full-stack, real-time system that combines layered encryption, MQTT-based transmission, and multi-level broker infrastructure tailored for healthcare environments. This work aims to address this gap by proposing and implementing a complete secure communication pipeline—from IoT publisher nodes through local and cloud brokers to end-user subscriber applications—while benchmarking multiple lightweight encryption schemes under realistic operational conditions.

3.3. Limitation of Lightweight Encryption and the Need for Layered Security

While lightweight encryption schemes have been widely recognized as effective solutions for securing resource-constrained IoT devices, their scope of applicability remains largely confined to local or edge-level communications. These cryptographic mechanisms, designed to minimize CPU cycles, memory usage, and energy consumption, are optimized to operate within the tight hardware limitations of IoT nodes such as wearables, sensors, and embedded devices. Algorithms like ChaCha20, TinyAES, SIMON, and SPECK demonstrate

efficiency on such platforms by reducing computational complexity and supporting real-time data processing.

However, as IoT ecosystems scale and data transmission extends beyond the local premises of edge environments toward remote cloud infrastructures, the security landscape changes considerably. Communication across public or semi-trusted networks introduces increased exposure to sophisticated attack vectors, including man-in-the-middle attacks, replay attacks, and key compromise risks. In such scenarios, the lightweight encryption schemes that prioritize performance over cryptographic strength may no longer offer sufficient resilience against these threats. This renders lightweight encryption alone inadequate for securing sensitive health data or critical IoT applications once data moves beyond the trusted local domain. Conversely, applying robust encryption schemes—such as AES-256 with GCM mode, RSA, or Elliptic Curve Cryptography (ECC) with large key sizes—provides stronger protection and compliance with regulatory standards like HIPAA and GDPR for data in transit across untrusted environments. These algorithms offer higher security margins, resistance to brute-force attacks, and support for advanced features like mutual authentication and key exchange. However, their computational requirements, larger memory footprint, and higher energy consumption make them impractical for direct deployment on constrained IoT devices, especially at the edge, where processing resources are limited.

This performance-security dichotomy highlights a critical trade-off in IoT environments. Lightweight encryption satisfies performance demands at the device level but falls short when higher-grade security is required for wide-area or cloud communication. On the other hand, robust cryptographic protocols ensure stronger data protection but at the cost of rendering constrained devices inoperable due to resource exhaustion. To address this challenge, there is a need for a layered encryption approach that applies lightweight cryptography for local, edge-level communication while enforcing stronger encryption mechanisms at the boundary between edge systems and cloud services. Such a design allows for optimized resource utilization at the device level without compromising the overall security of the system when sensitive data exits the local trusted environment.

This work embraces this layered security philosophy by integrating lightweight encryption at the edge and robust end-to-end encryption during data transmission to the cloud through a multi-broker MQTT architecture. This design ensures that security policies are adaptable to the context of data transmission, providing a balanced solution between cryptographic strength and resource efficiency. The main contributions of this study are the following:

1. Designing and implementing a dual-layer encryption architecture that integrates lightweight encryption at the edge and strong encryption (AES-256 with TLS 1.3) when transmitting data to the cloud, ensuring confidentiality, integrity, and performance across the entire data pipeline.
2. Introducing a custom bridge application that operates at the boundary between trusted (hospital) and untrusted (cloud) environments. The bridge functions as both subscriber and publisher—decrypting edge-encrypted messages and re-encrypting them using cloud-grade protocols before forwarding—effectively decoupling transport-level and application-level encryption.
3. Demonstrating an architecture that enhances security without increasing complexity at the edge, thereby preserving the performance benefits of Salsa20-Blake2b while ensuring data remains protected when it leaves the local network.
4. Validating the system through an end-to-end implementation, including local and cloud Mosquitto brokers, certificate-based TLS setup, and performance evaluation,

showcasing its applicability to real-world healthcare IoT deployments where both performance and privacy are critical.

4. Proposed Architecture

This section introduces secure and scalable end-to-end architecture for IoT-based e-health monitoring systems. The proposed framework is designed to ensure data confidentiality and reliability across constrained environments by employing layered encryption mechanisms and a multi-broker MQTT infrastructure. It supports direct communication from IoT sensing nodes to cloud-based systems and ultimately to healthcare professionals, enabling real-time decision-making while maintaining strict privacy and security standards.

4.1. High-Level Architecture Overview

The system architecture, illustrated in Figure 2, is composed of four core layers: the Perception Layer, the Local MQTT Broker, the Cloud Integration Layer, and the Application Layer. Each layer has a distinct responsibility in the secure acquisition, transmission, processing, and presentation of healthcare data, from sensor-level interactions to cloud-based storage and clinical dashboards.

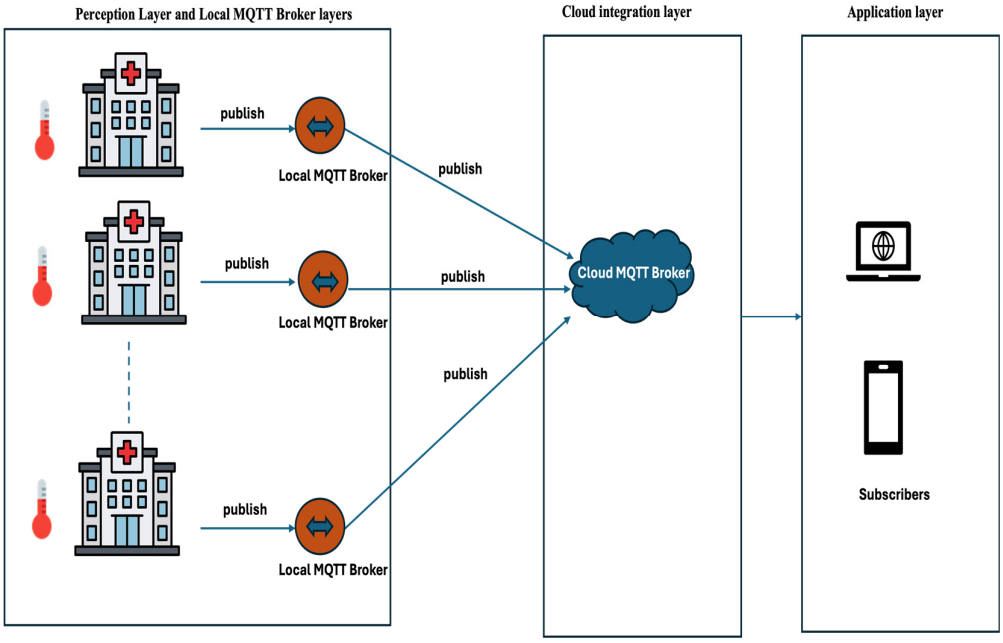

Figure 2. End-to-end secure IoT architecture for e-health applications using layered encryption and multi-broker MQTT.

4.2. Perception Layer

The Perception Layer comprises wearable medical devices equipped with physiological sensors capable of monitoring vital signs such as heart rate, body temperature, and blood oxygen saturation. The IoT node handles both sensing and computation tasks, performing lightweight encryption on collected data using ciphers suitable for constrained environments, such as ChaCha20 or Salsa20. Once encrypted, the data is transmitted wirelessly using MQTT over TLS, with the microcontroller acting as the MQTT publisher. This direct publishing model eliminates the need for a separate edge gateway, reducing system complexity and improving responsiveness.

To enhance system robustness in case of local broker failure, edge device publishers implement automatic reconnection logic. Upon detecting a disconnection, the MQTT client on the publisher side iteratively attempts to reconnect to a predefined list of alternative

local brokers (e.g., covering different floors or subnets within the hospital). This fallback list is configured during the device provisioning phase.

4.3. Local MQTT Broker

In the proposed architecture, a locally deployed MQTT broker is utilized as an intermediate security and routing layer within the hospital network. This broker, implemented using the Eclipse Mosquitto platform, runs on a dedicated edge server or embedded device such as a Raspberry Pi. Its primary role is to receive data from resource-constrained IoT publishers—such as wearable or diagnostic devices—which encrypt the data payload using lightweight cryptographic algorithms (e.g., Salsa20-Blake2b). The local broker functions in conjunction with a custom-built bridge application that acts as a security gateway. The bridge application subscribes to the local broker and is responsible for decrypting incoming lightweight-encrypted messages and re-encrypting the payload using AES-256 before forwarding them to the cloud-based MQTT broker over a TLS 1.3 channel. Resource efficiency is prioritized at the edge, and stronger cryptographic guarantees are enforced when data leaves the hospital premises.

A critical component of the proposed multi-layer security framework is the secure and efficient management of cryptographic keys. To meet the dual requirements of resource-constrained IoT edge devices and compliance-driven cloud communication, a hybrid key management model is adopted. At the edge, IoT devices encrypt payloads using lightweight ciphers (Salsa20 for confidentiality and Blake2b for integrity) with pre-provisioned static symmetric keys. This design minimizes computational and memory overhead, aligning with the limited resources of typical embedded medical devices. The symmetric keys for AES-256 encryption are securely created and managed using Amazon Web Services Key Management Service (AWS KMS) [24]. During the initial system deployment phase, each hospital or medical environment will provision its infrastructure, registering all local MQTT brokers and subscribers as legitimate devices within AWS KMS, allowing only authenticated and authorized devices to request symmetric keys for encryption and decryption. In addition to the symmetric encryption applied at the application layer, the proposed framework enforces transport-layer security using TLS 1.3 to protect all communications between internal system components, particularly between the local MQTT broker, cloud MQTT broker and subscriber. To ensure mutual authentication and prevent unauthorized access, the framework utilizes X.509 certificates issued by trusted Certificate Authorities (CAs). These certificates are securely installed and configured at three locations within the architecture: local broker, cloud broker and subscriber.

To ensure real-time monitoring of system health and broker availability, the proposed framework includes a heartbeat failure detection mechanism at the local MQTT broker level. Each local broker is configured to periodically publish a lightweight “I am online” status message to a cloud-facing topic (e.g., hospital/floor1/broker/status). This message is sent at a fixed interval (e.g., every 10 s) and includes metadata such as: broker identifier (e.g., hostname or floor-specific ID), current timestamp, status code (e.g., “online”, “degraded”). The cloud-side subscriber component subscribes to all registered broker status topics (e.g., hospital/+/broker/status) and updates an internal status table reflecting broker health. If a subscriber does not receive a heartbeat from a broker within the configured timeout threshold (e.g., 2× heartbeat interval), it flags the broker as unavailable or disconnected and notifies the user interface layer. This information is visualized in the system’s cloud dashboard, enabling administrators to monitor all local brokers and identify broker failures in real time.

4.4. Cloud Integration Layer

As messages move beyond the hospital network, they enter the Cloud Integration Layer, which comprises a cloud-hosted MQTT broker capable of supporting large-scale distributed systems. Before transmission to the cloud, data is re-encrypted using stronger end-to-end encryption protocols such as AES-256. The cloud broker is integrated with identity and access management services, enabling dynamic control over who can access specific patient data. In addition, cloud-native services, such as intrusion detection systems and data logging modules, provide a robust layer of protection and traceability for transmitted information.

4.5. Application Layer

The final layer is the Application Layer, where healthcare professionals access patient data through a secure, web-based dashboard. The dashboard acts as an MQTT subscriber, receiving real-time updates from the cloud broker through WebSocket connections. The dashboard presents encrypted health metrics with live updates and supports role-based access control to restrict user permissions according to clinical responsibilities. Communication between the web application and the cloud is conducted over HTTPS, and all user actions are logged to ensure transparency and accountability.

5. Implementation of Secure IoT-Based Communication System

This section presents the implementation of the secure publisher–subscriber system designed to operate within the proposed e-health IoT architecture. The design focuses on encryption at the edge using constrained devices and efficient MQTT-based data delivery to a cloud-based application for healthcare providers. The system enables modular encryption benchmarking and real-time performance monitoring, supporting integration with both local and cloud MQTT brokers. The diagram shown in Figure 3 presents the traffic flow from the IoT constraint node to the subscriber node.

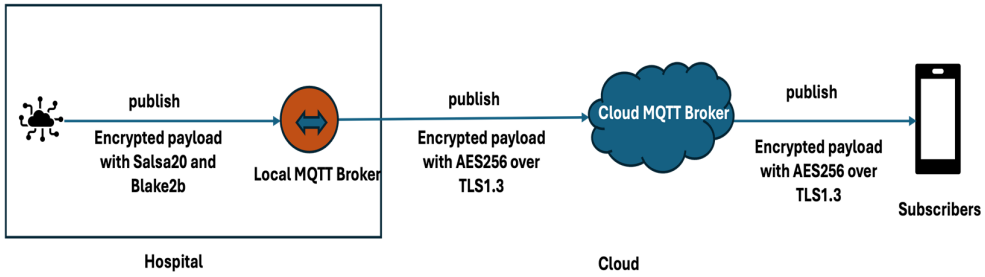

Figure 3. Traffic flow.

5.1. Publisher Implementation at the IoT Node

The publisher emulates a typical IoT edge node, such as a wearable medical device, that collects and transmits patient-related data. In this implementation, a static image file is used to represent sensed data for evaluation purposes. The image data is loaded in binary format and encrypted using one of several available cryptographic schemes prior to transmission. The supported encryption algorithms include ChaCha20-Poly1305, AES in EAX, GCM, and CCM8 modes, TinyAES in CTR mode, Salsa20 with Blake2b, Ascon, and lightweight ciphers such as SIMON and SPECK. Each encryption routine uses either a fixed 128-bit or 256-bit key, defined in advance. For authenticated encryption schemes, nonce and tag management are handled within the function, and the encrypted payload is base64 encoded for safe transmission via MQTT.

The publisher component connects to the local MQTT broker over a non-TLS channel (port 1883), reflecting real-world scenarios where IoT devices operate in trusted local

environments with limited TLS capability. TLS support is included for both OpenSSL and WolfSSL, although this is deactivated by default. The publisher constructs a JSON message containing the encrypted image and a timestamp indicating the moment of publication. Messages are sent to the MQTT broker on a predefined topic, simulating real-time data flow from an IoT health sensor.

To assess the performance of each encryption scheme under constrained conditions, the implementation integrates system-level monitoring. The Linux perf tool is executed as a subprocess to capture hardware-level statistics, including task clock, CPU cycles, instructions, cache references, and cache misses. In parallel, a separate thread utilizes the psutil library to track memory usage, measuring real-time resident set size (RSS) and peak memory allocation during the transmission phase. The publisher is designed to run for a fixed number of iterations, transmitting a defined number of encrypted messages at regular intervals. Upon completion, the system calculates throughput based on the total size of encrypted data sent over the elapsed time. All metrics—timing, CPU usage, cache behavior, memory usage, and throughput—are aggregated and logged in a structured CSV format for subsequent analysis.

Algorithm 1 outlines the core functionality of the publisher module within the proposed framework.

Algorithm 1: Pseudocode For Publisher Component

Input: Image directory path, number of images x, encryption method
Output: Encrypted messages published to MQTT topic, log performance
Begin
Connect to Local MQTT Broker with TLS
For x number of images do
Load image data from file
Generate timestamp
EncryptedPayload = Encrypt_Payload(image_data, Encryption_Method)
Construct JSON message with ‘timestamp’ and ‘content’
Publish message to topic
Wait fixed interval before next message
Log performance metrics (latency, throughput, CPU stats)
End For
Save performance data in cvs file
Disconnect from Broker
End

Algorithm 2 outlines the core functionality of the encryption function.

Algorithm 2: Pseudocode For Encryption Function

Input: Image directory path, number of images x, encryption method
Output: Encrypted messages published to MQTT topic, log
Begin
function Encrypt_Payload(data, method)
If method =AES-GCM:
Generate nonce

Algorithm 2: *Cont.*

```
        Encrypt data with AES in GCM mode
        Concatenate nonce + tag + ciphertext
        Encode in base64 and return
        If method = Salsa20-HMAC:
        Generate nonce
        Encrypt data with Salsa20
        Generate HMAC using Blake2b
        Concatenate nonce + ciphertext + HMAC
        Encode in base64 and return
    End
```

Other methods, such as ChaCha20-Poly1305 and CCM8, follow similar encryption and tagging generation patterns.

5.2. Local MQTT Broker and Bridge Application

The local broker, implemented using Mosquitto, serves as a lightweight, internal MQTT hub that receives encrypted messages from the publisher. It provides basic message queuing and forwarding capabilities without any transformation or decryption of the payload. This broker acts as an intermediary that simplifies edge communication and enables local scalability by allowing multiple publishers to offload encryption tasks while forwarding to a common cloud path via the bridge. The broker configuration includes a listener on port 1883 for local (non-TLS) connections and defines the topic structure (e.g., image/test) used to separate and organize published image data.

The same MQTT local broker is running a custom bridge application that was developed to securely transfer encrypted IoT data from the local MQTT broker, hosted within the hospital network, to a cloud-based broker. The bridge acts as a middleware layer with dual MQTT roles. It subscribes to encrypted messages published to the local broker by edge IoT devices—typically resource-constrained nodes that encrypt their payloads using lightweight symmetric encryption algorithms such as Salsa20, in conjunction with the Blake2b hashing function for message integrity. Upon receiving these messages, the bridge decrypts the payload locally using a pre-shared symmetric key and nonce structure.

Following successful decryption, the bridge performs re-encryption of the payload using AES-256. This transformation ensures that sensitive healthcare data is protected with stronger cryptographic guarantees before being transmitted across untrusted networks. The re-encrypted data is then published to the cloud MQTT broker over a secure TLS 1.3 channel, leveraging a self-signed X.509 certificate for broker authentication.

Functionally, the bridge application encapsulates the following operations:

1. MQTT Subscriber: subscribes to a designated topic on the local Mosquitto broker to retrieve incoming encrypted messages.
2. Decryption Module: applies the Salsa20 cipher to extract the original message content.
3. Re-encryption Module: uses AES in GCM mode with a 256-bit key to ensure confidentiality and integrity.
4. MQTT Publisher: publishes the transformed message to the cloud broker on a mirrored topic, using UTF-8 encoding and base64 to preserve structure.

Algorithm 3 outlines the core functionality of the local broker component.

Algorithm 3: Pseudocode For Local Broker Component

Input: Encrypted MQTT messages from edge devices
Output: Re-encrypted messages published to cloud broker
Begin
function Bridge_ReEncrypt_Forward()
Connect to Local MQTT Broker as Subscriber
Connect to Cloud MQTT Broker as Publisher
Subscribe to Topic from Edge Devices
For each incoming message do
Extract timestamp and encrypted payload
DecryptedData = Decrypt_Salsa20_Blake2b(encrypted_payload)
ReEncryptedPayload = Encrypt_AES_GCM(DecryptedData)
Construct JSON with original timestamp and re-encrypted content
Publish to cloud broker
End for
End

Each hospital can deploy an identical bridge that independently manages encryption, translation and secure routing, by incorporating both application-layer and transport-layer encryption. This design decouples resource-efficient encryption at the edge from the computationally heavier encryption required for cloud integration, allowing edge devices to remain lightweight and energy-efficient while maintaining compliance with cloud security standards and providing architectural scalability.

5.3. Cloud MQTT Broker and Subscriber

The cloud broker is hosted in a secure environment (e.g., AWS) and configured to accept TLS connections from external clients such as the bridge. It functions identically to the local broker in terms of topic management but enforces additional security policies. The subscriber connects to this cloud broker, receives the forwarded encrypted messages, and is responsible for:

- Verifying the authenticity of the payload using the MAC.
- Decrypting the ciphertext using the shared symmetric key.
- Measuring latency and decryption time.

The subscriber reconstructs the original binary image from the decrypted payload and stores it locally for further analysis. Decryption operations mirror the encryption protocol used by the publisher and require proper nonce extraction and HMAC validation. Only if the integrity check passes is the decryption process executed. The subscriber also logs message arrival time, processing delays, and total messages received. These metrics help evaluate end-to-end system performance and protocol efficiency in realistic deployment scenarios.

Algorithm 4 outlines the core functionality of the subscriber component.

Algorithm 4: Pseudocode For Subscriber Component

Input: Encrypted MQTT messages from edge devices
Output: Re-encrypted messages published to cloud broker
Begin
Connect to Cloud MQTT Broker
Subscribe to Topic

Algorithm 4: *Cont.*

For each received message do
Extract 'timestamp' and 'encrypted content'
DecryptPayload = Decrypt_AES_GCM(encrypted_content)
Compute latency = CurrentTime − Timestamp
Save decrypted image
Log metrics (latency, throughput)
End for
End

All cryptographic schemes were implemented using well-established, open-source cryptographic libraries—primarily PyCryptodome, a widely-used and actively maintained Python library that supports a broad range of symmetric encryption algorithms, AEAD modes, and secure random number generation. Here are some sample imports taken from the source code:

```
from Crypto.Cipher import AES, ChaCha20,ChaCha20_Poly1305,Salsa20
from Crypto.Util.Padding import pad,unpad
from Crypto.Random import get_random_bytes
```

6. Performance and Security Analysis

Table 1 summarizes the metrics used to evaluate the proposed layered encryption framework. The metrics are divided into two categories: security-related metrics, which assess how well the system protects sensitive healthcare data, and performance metrics, which assess the efficiency of the encryption techniques when applied to resource-constrained IoT devices.

Table 1. Security and performance metrics.

Metric	Tool/Method	Evaluated Layer	Expected Result
Confidentiality	NIST STS, Wireshark	Edge and Cloud	High entropy, secure ciphertext
Integrity	HMAC tampering, TLS integrity	Edge and Cloud	Tampering is detected
Broker compromise resilience	Payload capture test	Cloud broker	Data unreadable, MAC fails
CPU usage	perf, psutil	Publisher (constraint node)	Salsa20\Blake2b is better compared to other protocols
Encryption time	perf, psutil	Publisher (constraint node)	Salsa20\Blake2b is better compared to other protocols
Throughput	perf, psutil	Publisher (constraint node)	Salsa20\Blake2b is better compared to other protocols

Each metric is listed alongside the tools or methods used to measure it, the specific layer of the architecture where the evaluation was applied (either edge or cloud), and the expected outcome based on the design goals.

6.1. Testbed

To evaluate the proposed multi-layer encryption framework under realistic IoT healthcare conditions, we extended our original testbed in this study [4] to include a local MQTT broker deployed within the hospital (edge) environment. This modification allows us

to simulate the layered security model: lightweight encryption at the edge and robust encryption when data leaves the hospital premises toward the cloud. The revised testbed is illustrated in Figure 4. A similar testbed using the Ubuntu operating system has been adopted in several research studies. For example, Fan et al. [25] proposed a data protection protocol for MQTT based on hierarchical ID-based encryption and evaluated its performance using Python libraries on an Ubuntu 18.04.4 LTS system with an Intel Core i9-9940X 3.30 GHz processor.

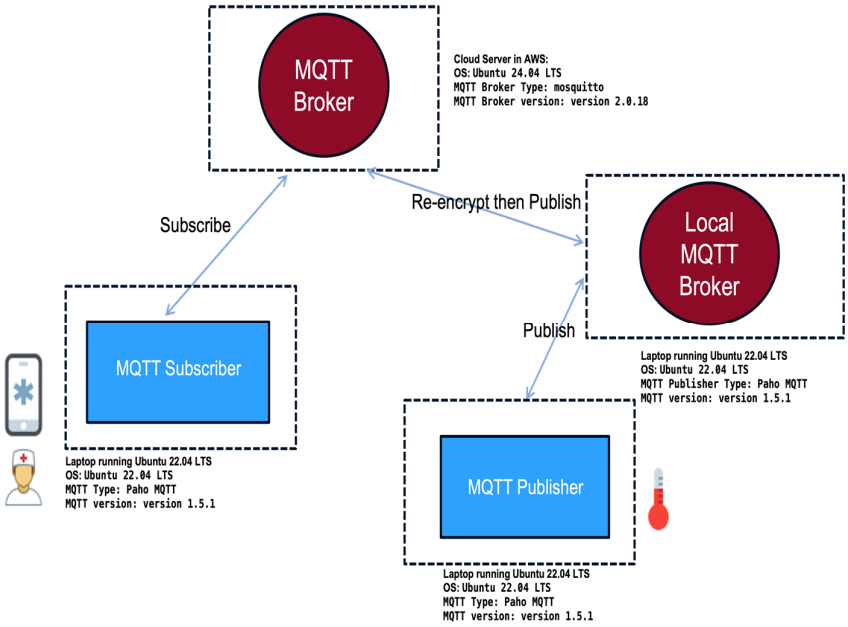

Figure 4. Testbed hardware and software components.

The testbed consists of three main roles: the MQTT Publisher, the Local MQTT Broker, and the MQTT Subscriber, all deployed on laptops running Ubuntu 22.04 LTS. Each device is equipped with an Intel Core i5 CPU, 8 GB of RAM, and utilizes the Paho MQTT library version 1.5.1. The publisher encrypts image data using lightweight ciphers (e.g., Salsa20-Blake2b) and sends it to the local MQTT broker. The Local Broker, hosted on an identical laptop, receives and decrypts the incoming messages. A bridge component re-encrypts them using AES-256 and publishes them over a TLS 1.3 channel to the Cloud MQTT Broker, which is hosted on an AWS instance running Ubuntu 24.04 LTS and Mosquitto version 2.0.18. The MQTT Subscriber, located on a fourth laptop with the same hardware and software stack as the publisher, subscribes to the cloud broker and receives the re-encrypted data for further validation and performance analysis.

6.2. Security Evaluation

In the context of encrypted IoT traffic, particularly for privacy-sensitive domains such as healthcare, the preservation of statistical unpredictability in ciphertext is essential. To assess this, we applied the Approximate Entropy (ApEn) test from the NIST Statistical Test Suite (STS) [26], which measures the frequency and predictability of overlapping patterns in a binary sequence. A high entropy score indicates that the ciphertext lacks detectable regularity—an essential criterion for resisting ciphertext-only attacks and for maintaining the indistinguishability of encrypted data.

Each tested protocol (e.g., CCM8, EAX, GCM, Salsa20, TinyAES) was used to encrypt identical payloads, representing typical IoT data (e.g., JPEG images from medical sensors). The resulting ciphertexts were subjected to the ApEn test across multiple iterations. The p-values generated by the test were analyzed to determine whether the output se-

quences exhibit sufficient randomness. A *p*-value greater than 0.01, as per NIST criteria [26], suggests that the ciphertext passes the test and is statistically indistinguishable from a random sequence.

The proposed framework uses Salsa20-Blake2b at the edge layer for its efficiency and cryptographic robustness. NIST test results show that Salsa20 achieves high *p*-values in Approximate Entropy, typically above 0.40, and in some cases exceeding 0.90, confirming that it produces statistically unpredictable ciphertext. Table 2 presents the Approximate Entropy *p*-values obtained for the Salsa20-Blake2b encryption scheme over eight test rounds. For each round, the ciphertext was captured and analyzed using the NIST Statistical Test Suite to evaluate its randomness properties. The average *p*-values observed across other protocols were as follows: AES-CCM8—0.457, AES-EAX—0.403, AES-GCM—0.846, TinyAES—0.296 and Salsa20-Blake2b—0.607. These results highlight the suitability of Salsa20-Blake2b for resource-constrained edge devices, offering high entropy with low computational cost, while also affirming the robust randomness characteristics of AES-GCM, making it an effective choice for cloud-layer encryption within the proposed multi-broker architecture.

Table 2. *p*-values in approximate entropy for Salsa20-Blake2b.

Protocol	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7	Round 8
Salsa20-Blake2b	0.88381	0.283651	0.987717	0.411331	0.928822	0.696591	0.213525	0.452711

To ensure message integrity and authentication at the edge, the proposed framework utilizes the Blake2b hash function in conjunction with the Salsa20 encryption scheme. Blake2b is a cryptographically secure and efficient hash function, well-suited for constrained devices due to its speed and simplicity. At the publishing node, each outgoing message is encrypted using Salsa20 and appended with a keyed HMAC generated using Blake2b. As shown in Figure 5, when the ciphertext was altered or transmitted with an invalid or mismatched HMAC, the bridge failed to authenticate the message, generating repeated errors: "HMAC verification failed at local broker." Similarly, AES-GCM authentication tag and TLS1.3 MAC are utilized to ensure data integrity when the data leave the local broker.

Figure 5. Detecting tampering at local broker.

To assess the resilience of the proposed HECS4MQTT multi-layer encryption framework, we conduct a structured security analysis based on the MITRE ATT&CK framework [27]. This methodology allows us to map real-world adversarial tactics and techniques to potential vulnerabilities in the system and to evaluate the effectiveness of our mitigation strategies. HECS4MQTT demonstrates robust protection against several prevalent attack vectors, particularly in the domains of network interception, credential compromise, and payload tampering.

One of the most significant strengths of the framework lies in its defense against network-based attacks, including traffic interception and man-in-the-middle (MitM) attacks.

By employing Salsa20-Blake2b encryption at the edge layer and AES-256 encryption with TLS 1.3 at the cloud layer, HECS4MQTT ensures that sensitive healthcare data is encrypted during the entire transmission path. The use of authenticated encryption further protects against data manipulation and unauthorized modifications (aligned with MITRE technique T1565—Data Manipulation).

Furthermore, the integration of TLS 1.3 with X.509 certificates issued by trusted Certificate Authorities (CAs) at the local broker, cloud broker and subscriber mitigates threats associated with unauthorized access or spoofed brokers (e.g., T1557—Man-in-the-Middle). This is reinforced by mutual certificate validation and strict topic-based access control at MQTT brokers, reducing the risk of command-and-control misuse via the application layer (e.g., T1071.001—MQTT protocol abuse).

The framework also mitigates risks related to credential brute-forcing and password compromise (T1110—Brute Force) through the use of pre-provisioned symmetric keys at the edge and planned integration with AWS Key Management Service (KMS) at the bridge and cloud layers.

6.3. Performance Evaluation

Performance evaluation is undertaken with six encryption protocols—Salsa20, ChaCha20-Poly1305, AES-GCM, CCM8, EAX, and TinyAES—across three payload sizes (1 KB, 5 KB, and 10 KB). The evaluation considers average throughput, CPU cycles, and encryption time, offering a granular view of how each protocol scales with data volume.

The experimental results illustrated in this section provide a comprehensive overview of the performance of the proposed HECS4MQTT framework across its core components: publisher, local broker bridge, and subscriber.

Figures 6 and 7 capture the behavior of the publisher. Figure 6 presents the execution of a test cycle where the publisher transmits encrypted image payloads using a selected protocol (ChaCha20-Poly1305 in this run). The system iterates through multiple encryption protocols, and for each run, 100 images are published. Figure 7 reports detailed performance metrics such as CPU cycles, encryption time, throughput, and memory usage (RSS). These measurements are essential for evaluating the computational overhead of each encryption protocol on constrained devices.

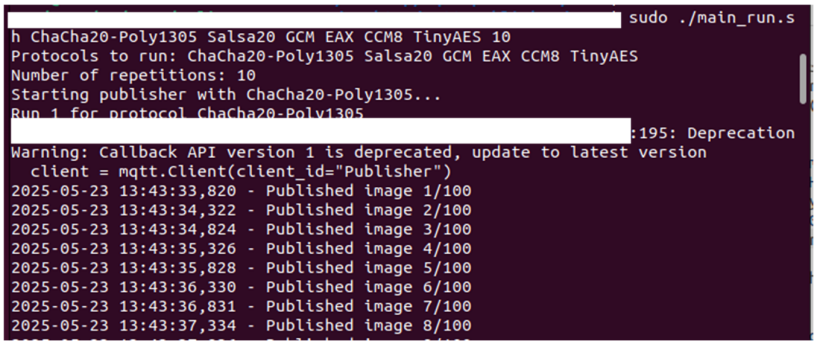

Figure 6. Publisher testing.

Figure 8 shows the terminal output of the local broker bridge. It confirms successful connection to the local MQTT broker and logs the message transformation and forwarding activity to the cloud broker. Each message is decrypted (Salsa20), re-encrypted (AES256), and securely forwarded over TLS 1.3, highlighting the bridge's role in ensuring layered encryption and secure transition beyond the edge network. Figure 9 presents the subscriber's behavior. It logs the receipt of published messages, indicating correct end-to-end delivery. Together, these figures validate the operational integrity and performance of the multi-layer

encryption approach, confirming that the proposed framework supports secure, scalable, and efficient MQTT-based communication in IoT healthcare environments.

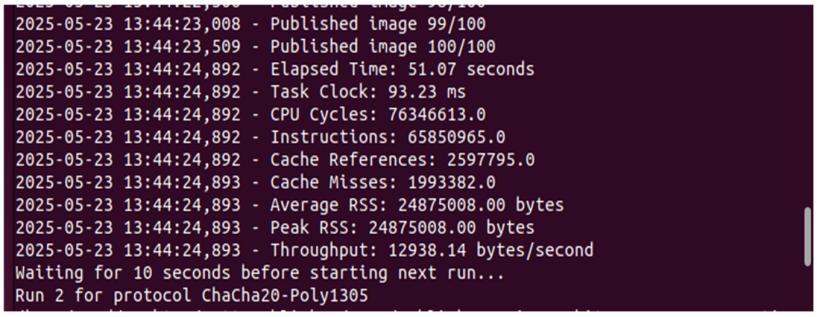

Figure 7. Publisher result per run.

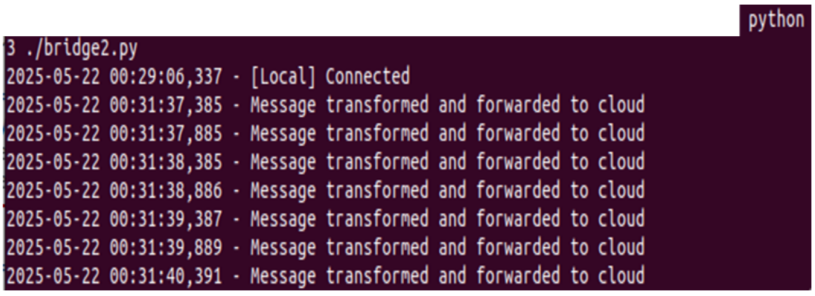

Figure 8. Local broker.

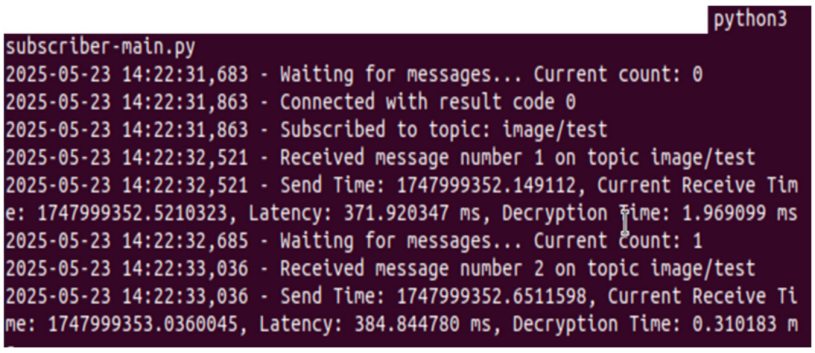

Figure 9. Subscriber.

Throughput is a critical measure in time-sensitive applications, such as continuous patient data streaming. For all image sizes, Salsa20 consistently maintains the highest throughput, particularly notable in the 10 KB scenario, where it achieves over 26,000 bytes/s. Throughput generally declines with smaller payloads, as observed in 1 KB tests, where all protocols show reduced data rates due to fixed overheads becoming more prominent. Figure 10 illustrates the average throughput of various encryption protocols across three image sizes (1 KB, 5 KB, and 10 KB), showing that Salsa20 maintains consistently high performance.

Encryption time reflects latency and responsiveness, essential for real-time IoT devices. Across all image sizes, Salsa20 and TinyAES offer the fastest processing times for 1 KB and 5 KB payloads, while EAX consistently exhibits the highest latency, especially noticeable at 10 KB, where it exceeds 135ms. These results highlight the scalability benefits of Salsa20 and raise concerns over EAX's suitability for edge scenarios. Figure 11 presents the average encryption time, highlighting that Salsa20 and TinyAES offer the lowest latency.

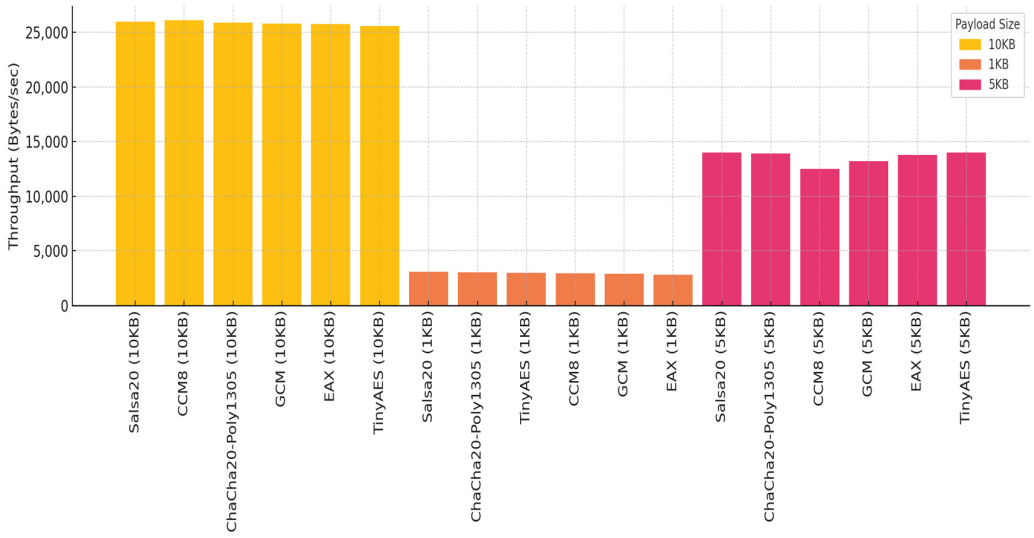

Figure 10. Average throughput by protocol and image size.

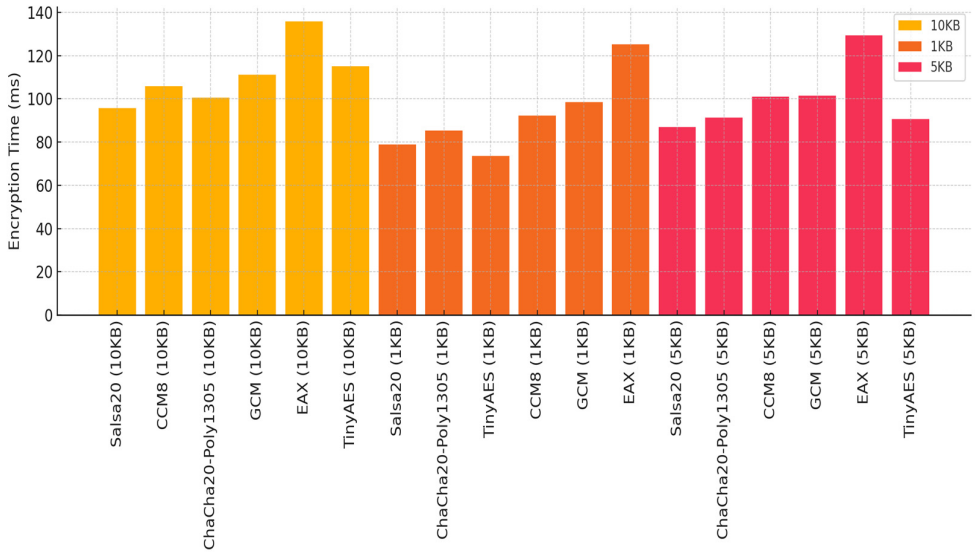

Figure 11. Average encryption by protocol and image size.

Average CPU cycles reveal how demanding each protocol is on device processors, a key constraint in IoT environments. Salsa20 again proves efficient, requiring the fewest cycles across all payloads. Figure 12 displays the average CPU cycles consumed by each protocol.

Pereira et al. [28] studied the performance evaluation of e-health applications, noting the limitations of existing studies and the lack of realistic measurements in the context of IoT communications. Importantly, they state that a latency below 500 milliseconds is required for emergency and rapid response e-health applications, particularly in scenarios involving event-driven alarms where immediate action is critical. In contrast, less time-sensitive applications, such as those involved in electrocardiogram (ECG) monitoring, can typically tolerate latency of up to 1 s without compromising clinical utility. All latency measurements obtained during our experimental evaluation were consistently below 500 milliseconds, thereby demonstrating the effectiveness of the HECS4MQTT framework.

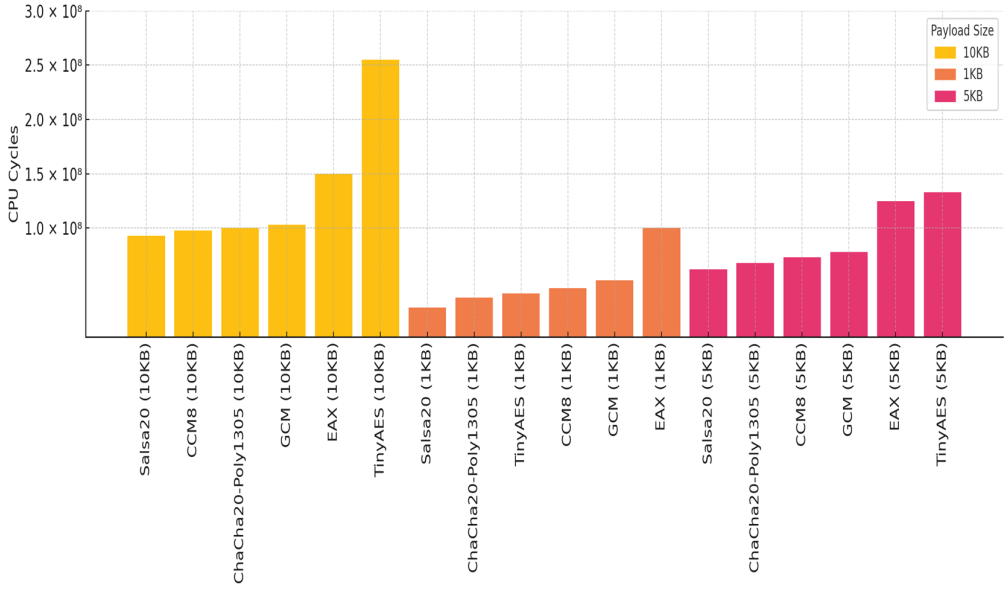

Figure 12. CPU cycles by protocol and image size.

7. Conclusions and Discussion

This study has presented a secure and scalable MQTT-based communication framework tailored for healthcare IoT environments. The proposed system leverages a multi-broker layered encryption model, combining lightweight encryption at the edge (using Salsa20 and Blake2b) with robust cryptographic protections (AES-256 and TLS 1.3) as data moves toward the cloud. A key architectural advancement is the introduction of a custom bridge application, which performs in-place cryptographic translation, decoupling edge-layer encryption from transport-layer security without imposing computational burdens on resource-constrained devices.

A comparison with traditional TLS-only architectures highlights the security and operational benefits of the proposed approach. As shown in Table 3, TLS-only models rely on a centralized broker to handle all client traffic, which, while offering basic confidentiality and metadata protection, exposes decrypted payloads at the broker layer. In contrast, the multi-broker architecture preserves end-to-end confidentiality by maintaining encryption throughout the data lifecycle. Furthermore, this model reduces the impact of broker compromise, avoids TLS-related overhead on IoT endpoints, and introduces cryptographic adaptability suited for varied operational environments, particularly in healthcare, where privacy, performance, and reliability are paramount.

The modular separation of trust zones—from edge nodes within hospital networks to cloud infrastructures—also enhances auditability and security governance. Each broker operates independently, which promotes scalability across multiple hospitals or clinics while retaining a consistent security posture. This makes the proposed design not only technically feasible but also strategically aligned with large-scale health data systems that must comply with privacy regulations.

While the current architecture demonstrates strong performance and security guarantees, several areas offer opportunities for future enhancement:

1. Dynamic Key Management: future research could explore the integration of lightweight key exchange protocols or identity-based encryption to reduce the reliance on pre-shared keys at the edge.
2. Mutual TLS Authentication: enforcing client authentication via mutual TLS in the cloud broker could further enhance access control and mitigate impersonation risks.

3. Intrusion Detection Integration: the bridge application can be extended with lightweight anomaly detection modules to monitor traffic patterns and detect signs of compromise or data leakage.
4. Interoperability Testing: extending the framework to support additional brokers, such as HiveMQ or EMQX, and testing interoperability in hybrid deployments would validate its practical adaptability.

Table 3. Comparison of traditional TLS-MQTT with multi-broker layered encryption.

Aspect	TLS Only (Single Broker)	Multi-Broker Layered Encryption
Architecture	Single MQTT broker with TLS connection from all clients	Local broker (in hospital) + cloud broker, with edge and cloud layers using different encryption schemes
Encryption at Edge	TLS (standard AES, handshake overhead)	Lightweight encryption (e.g., Salsa20-Blake2b) optimized for resource-constrained devices
Data Security Beyond the Broker	Not preserved—broker decrypts payload	Preserved—data re-encrypted (e.g., AES-256 + TLS 1.3) before leaving trusted environment
Metadata Protection	Protected (TLS encrypts topics, headers, etc.)	Fully protected via TLS on outer layer; internal encryption protects payload end-to-end
End-to-End Confidentiality	Broken at broker level—broker sees plaintext payload	Maintained from publisher to subscriber through layered encryption
Broker Compromise Risk	High—one compromised broker exposes all client data	Reduced—layered encryption limits exposure even if a broker is compromised
TLS Overhead on IoT Devices	Significant (handshake, certificate validation)	Avoided—edge uses lightweight symmetric encryption tailored to constrained devices
Key Management Complexity	Centralized certificate-based authentication	Multi-key architecture (light at edge, stronger in cloud) with symmetric/shared keys
Adaptability for Healthcare	Less suitable for edge devices (wearables, implants)	Highly adaptable—tailored crypto for edge, strong security when data moves to public/cloud infrastructure
Compliance and Auditability	Requires TLS logs and endpoint audit trails	Clear separation of trust domains (edge, hospital, cloud) allows enhanced security controls and logging
Scalability Across Hospitals	Single point of failure	Multi-broker allows local independence and central management in cloud

With the rise of quantum computing, existing cryptographic schemes—especially RSA and ECC—face potential vulnerabilities. Further extension of this work could incorporate post-quantum algorithms, such as lattice-based key exchange, particularly at the cloud layer, where resources are more flexible. Additionally, 6G+ networks will enable ultra-low latency and edge intelligence, making the proposed multi-broker architecture well-suited for integration with 6G features like network slicing and secure edge processing. Future work could include investigating the integration of quantum-resilient cryptography and assessing the framework's adaptability to 6G-enabled healthcare IoT systems.

7.1. Positioning Within Current Research

Recent studies in IoT security have explored various lightweight encryption methods [4–14,16–20]. However, most existing works either focus solely on lightweight schemes at the device level or employ TLS in monolithic, single-broker architectures that do not preserve end-to-end confidentiality. This research builds on prior findings by integrating both dimensions—efficient edge-layer encryption and robust cloud-layer protection—through a modular, multi-broker MQTT framework.

7.2. Limitations of This Work

While the proposed system demonstrates strong performance and security under standard operating conditions, several limitations must be acknowledged to inform future research and practical deployments. We acknowledge the following: (1) static key management—the current implementation uses pre-shared symmetric keys at the edge, which may present management and scalability challenges in large-scale deployments; (2) limited broker interoperability—the system was primarily tested using a specific MQTT broker implementation, and broader compatibility with other brokers (e.g., HiveMQ, EMQX) has not yet been fully validated; and (3) no real-time attack simulation—the system was evaluated under normal operation scenarios, and future evaluations should include adversarial testing for various attack surfaces.

7.3. Future Research Directions

Building on the proposed design, future research could explore the following:

1. Integration of post-quantum cryptographic algorithms, especially lattice-based key exchanges, to ensure long-term cryptographic resilience.
2. Adapting the framework to 6G-enabled infrastructures, leveraging edge intelligence, ultra-low latency, and secure slicing for healthcare-specific use cases.
3. Development of lightweight dynamic key exchange protocols, such as EDHOC or identity-based encryption, to reduce reliance on static keys.
4. Embedding intrusion detection capabilities within the bridge application to detect anomalies and enhance runtime security.
5. Extending performance evaluation to battery-powered IoT devices, including wearables and implantables, in real hospital or smart clinic settings.

In conclusion, this work offers a practical and scalable contribution toward securing healthcare IoT systems. By combining efficient cryptographic techniques with strategic architectural layering, it addresses the core challenges of securing sensitive health data without compromising performance or scalability.

Author Contributions: Conceptualization, S.A. and D.B.; methodology, S.A. and D.B.; software, S.A.; validation, S.A., D.B. and W.A.; investigation, S.A.; resources, S.A.; data curation, S.A.; writing—original draft preparation, S.A.; writing—review and editing, S.A., D.B. and W.A.; visualization, S.A.; supervision, D.B.; project administration, S.A.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original data presented in the study are openly available in FigShare at https://doi.org/10.6084/m9.figshare.c.7836779.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Abbasi, M.; Plaza-Hernandez, M.; Prieto, J.; Corchado, J.M. Security in the Internet of Things Application Layer: Requirements, Threats, and Solutions. *IEEE Access* **2022**, *10*, 97197–97216. [CrossRef]
2. Harbi, Y.; Aliouat, Z.; Refoufi, A.; Harous, S. Recent Security Trends in Internet of Things: A Comprehensive Survey. *IEEE Access* **2021**, *9*, 113292–113314. [CrossRef]
3. Khan, M.N.; Rao, A.; Camtepe, S. Lightweight Cryptographic Protocols for IoT Constrained Devices: A Survey. *IEEE Internet Things J.* **2020**, *8*, 4132–4156. [CrossRef]
4. Alharbi, S.; Bell, D.; Awad, W. Lightweight Security Scheme for Internet of Things Encryption. *Appl. Math. Inf. Sci.* **2025**, *19*, 751–760. [CrossRef]
5. OASIS Standard. MQTT Version 5.0. OASIS. 2019. Available online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed on 1 June 2025).
6. Singh, M.; Rajan, M.A.; Shivraj, V.L.; Balamuralidhar, P. Secure MQTT for Internet of Things (IoT). In Proceedings of the 5th International Conference on Communication Systems and Networks Technologies, Gwalior, India, 4–6 April 2015; pp. 746–751.
7. Pal, P.; Lauer, G.; Khoury, J.; Hoff, N.; Loyall, J. P3S: A privacy preserving publish-subscribe middleware. In Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Montreal, QC, Canada, 3–7 December 2012; pp. 476–495.
8. Wang, J.; Zhang, J.; Schooler, E.M.; Ion, M. Performance evaluation of attribute-based encryption: Toward data privacy in the IoT. In Proceedings of the IEEE International Conference on Communications (ICC), Sydney, Australia, 10–14 June 2014; pp. 725–730.
9. Bisne, L.; Parmar, M. Composite secure MQTT for Internet of Things using ABE and dynamic S-box AES. In Proceedings of the 2017 Innovations in Power and Advanced Computing Technologies (IPACT), Vellore, India, 21–22 April 2017; pp. 1–5. [CrossRef]
10. Iqbal, M.; Laksmono, A.M.A.; Prihatno, A.T.; Pratama, D.; Jeong, B.; Kim, H. Enhancing IoT security: Integrating MQTT with ARIA Cipher 256 algorithm cryptography and mbedTLS. In Proceedings of the 2023 International Conference on Platform Technology and Service (PlatCon), Busan, Republic of Korea, 16–18 August 2023; pp. 91–96. [CrossRef]
11. Hintaw, J.; Manickam, S.; Karuppayah, S.; Aladaileh, M.A.; Aboalmaaly, M.F.; Laghari, S.U.A. A Robust Security Scheme Based on Enhanced Symmetric Algorithm for MQTT in the Internet of Things. *IEEE Access* **2023**, *11*, 43019–43040. [CrossRef]
12. Hosseinkhani, R.; Javadi, H.H.S. Using cipher key to generate dynamic S-box in AES cipher system. *Int. J. Comput. Sci. Secur.* **2012**, *6*, 19–28.
13. Kazlauskas, K.; Kazlauskas, J. Key-dependent S-box generation in AES block cipher system. *Informatica* **2009**, *20*, 23–34. [CrossRef]
14. Al-Mamun, A.; Rahman, S.S.M.; Ahmed Shaon, T.; Hossain, M. Security analysis of AES and enhancing its security by modifying S-box with an additional byte. *Int. J. Comput. Netw. Commun.* **2017**, *9*, 69–88. [CrossRef]
15. Hamad, M.; Finkenzeller, A.; Liu, H.; Lauinger, J.; Prevelakis, V.; Steinhorst, S. SEEMQTT: Secure End-to-End MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation. *IEEE Internet Things J.* **2022**, *10*, 3384–3406. [CrossRef]
16. Buccafurri, F.; De Angelis, V.; Lazzaro, S. MQTT-I: Achieving End-to-End Data Flow Integrity in MQTT. *IEEE Trans. Dependable Secur. Comput.* **2024**, *21*, 4717–4734. [CrossRef]
17. Li, P.; Su, J.; Wang, X. iTLS: Lightweight Transport-Layer Security Protocol for IoT with Minimal Latency and Perfect Forward Secrecy. *IEEE Internet Things J.* **2020**, *7*, 6828–6841. [CrossRef]
18. Sadio, O.; Ngom, I.; Lishou, C. Lightweight Security Scheme for MQTT/MQTT-SN Protocol. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019. [CrossRef]
19. Wijayanto, A.; Nugrahani, S.S.; Wardani, D.W.; Cahyono, H.D.; Setiadi, H. Performance Comparison of AES, Grain V1, and RC4 Algorithms on the MQTT Protocol. In Proceedings of the International Conference on Information Technology, Computer and Electrical Engineering, Semarang, Indonesia, 31 August–1 September 2023. [CrossRef]
20. Al-Ani, A.; Shen, W.K.; Al-Ani, A.K.; Laghari, S.A.; Elejla, O.E. Evaluating Security of MQTT Protocol in Internet of Things. In Proceedings of the 2023 IEEE Canadian Conference on Electrical and Computer Engineering, Regina, SK, Canada, 24–27 September 2023. [CrossRef]
21. Kurdi, H.; Thayananthan, V. A Multi-Tier MQTT Architecture with Multiple Brokers Based on Fog Computing for Securing Industrial IoT. *Appl. Sci.* **2022**, *12*, 7173. [CrossRef]
22. Veeramanikandan, M.; Sankaranarayanan, S. Publish/Subscribe Based Multi-Tier Edge Computational Model in Internet of Things for Latency Reduction. *J. Parallel Distrib. Comput.* **2019**, *127*, 18–27.
23. Pham, V.N.; Nguyen, V.D.; Nguyen, T.D.T.; Huh, E.N. Efficient Edge-Cloud Publish/Subscribe Broker Overlay Networks to Support Latency-Sensitive Wide-Scale Iot Applications. *Symmetry* **2020**, *12*, 3. [CrossRef]
24. AWS Key Management Service. Available online: https://docs.aws.amazon.com/kms/latest/developerguide/overview.html (accessed on 1 June 2025).

25. Fan, C.; Shie, C.; Tseng, H. An Efficient Data Protection Scheme Based on Hierarchical ID-Based Encryption for MQTT. *ACM Trans. Sens. Netw. (TOSN)* **2023**, *19*, 21. [CrossRef]
26. NIST STS Documentation. 2010. Available online: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final (accessed on 1 June 2025).
27. MITRE ATT & CK Framework. Available online: https://attack.mitre.org/ (accessed on 1 June 2025).
28. Pereira, C.; Pinto, A.; Ferreira, D.; Aguiar, A. Experimental Characterization of Mobile IoT Application Latency. *IEEE Internet Things J.* **2017**, *4*, 1082–1094. [CrossRef]