An Enhanced MQTT Communication Protocol for Privacy Preservation in Industrial Internet of Things (IIoT) Systems

[1]Onwuachu Uzochukwu Christian., [2]Opuh Jude Iwedike

[1]Department of computer science, Imo State university, Owerri, Nigeria

[2]Department of computer science, Southern delta university Ozoro Nigeria

DOI: https://doi.org/10.51584/IJRIAS.2026.11010055

Received: 10 January 2026; Accepted: 15 January 2026; Published: 03 February 2026

ABSTRACT

This paper addresses the security shortcomings of MQTT in Industrial IoT by designing, implementing, and evaluating a secure MQTT prototype that balances confidentiality, integrity, and usability. Using an Object-Oriented Analysis and Design Methodology (OOADM) guided by UML artifacts, the work decomposes the system into modular classes and enforces layered security aligned with the OSI and client–server models. Implemented in Python with Tkinter GUIs and Mosquitto as the MQTT broker, and supporting libraries (paho-mqtt, pycryptodome, bcrypt, hashlib, base64, socket) for secure messaging, encryption, authentication, and IP tracking. MQTT Explorer was used for real-time visualization of message flows, encryption consistency, and topic activity. The system integrates cryptographic techniques such as AES-CBC encryption with random IVs, HMAC-SHA256 integrity checks, bcrypt password hashing, and an OTP email verification/recovery flow (Gmail SMTP). Role-Based Access Control, account lockout policies, and audit logging (user, role, IP, timestamp, message state) provide operational safeguards. Experimental deployment and validation were conducted in a controlled virtual environment. Kali Linux running in VirtualBox provided the platform for penetration testing, and the subscriber was executed on a Kali instance in UserLAnd with GUI access through R-VNC to emulate a realistic IIoT endpoint. Security evaluation and Penetration tools included: John the Ripper for offline password cracking, Bettercap for man-in-the-middle (MITM) testing and traffic manipulation, Wireshark for packet capture and protocol analysis, and Nmap/Zenmap for port and service enumeration. These tools verified the system's resilience against common attacks, including unauthorized topic publishing, credential compromise, and message interception. Results demonstrate that plaintext MQTT (1883) is trivially intercepted and modifiable while TLS (8883) prevents passive decryption without trust compromise, and that the combined cryptographic and access-control measures significantly reduce practical attack surfaces for IIoT deployments.

Keywords: MQTT broker, Privacy Preservation, Security, Penetration Tool and Packets

INTRODUCTION

The increasing adoption of Internet of Things (IoT) technologies in various sectors such as healthcare, smart industries, and smart homes has created a demand for secure, reliable, and privacy-preserving communication protocols. Among the existing messaging protocols, the Message Queuing Telemetry Transport (MQTT) has gained wide acceptance due to its lightweight nature and suitability for resource-constrained devices. However, despite its efficiency, MQTT lacks robust built-in mechanisms for authentication, privacy preservation, and resilience against common cyber threats. This limitation necessitates the design of enhanced frameworks that integrate security features without compromising performance. This paper presents a prototype system that strengthens MQTT communication using advanced security techniques. It integrates AES-128 encryption**,** bcrypt-based password protection**,** HMAC verification**,** TLS (Transport Layer Security)**,** and OTP verification via Gmail SMTP to achieve confidentiality, integrity, and strong user authentication. Furthermore, it implements role-based access control**,** audit logging, and account lockout mechanisms**,** ensuring resilience against brute-force attacks and unauthorized publishing.

The system is not a conventional mobile application; rather, it is a desktop-based application with a Tkinter graphical user interface (GUI) running on a PC that functions as both broker host and publisher client. The subscriber client is deployed on a mobile device using the UserLAnd Ubuntu environment in combination with RVNC remote desktop access. This distributed configuration demonstrates the system as a client–server IoT communication model that validates security concepts in a real-world test environment. Additionally, the project is modeled conceptually using both the OSI Reference Model, which explains the layered communication and security mapping, and the Client–Server Model, which represents the interactive relationship between broker, publisher, and subscriber. By combining these theoretical models with practical implementation, the system contributes to bridging the gap between abstract IoT security frameworks and functional prototypes. (Roman et al., 2013). To secure user credentials, the system employs bcrypt hashing, a slow hashing algorithm that incorporates a random salt to protect passwords. Salting ensures that even if two users have the same password, their stored hashes appear different, making it significantly more difficult for attackers to perform rainbow table or dictionary-based attacks. During registration, each user's password is hashed with a salt and stored securely in a local JSON-based user database (Li & Palanisamy, 2019).

Message confidentiality is ensured using the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode. Each message is encrypted using a 128-bit symmetric AES key and a randomly generated Initialization Vector (IV). The IV introduces randomness into the first encryption block, ensuring that even identical plaintext messages produce different ciphertexts. This significantly enhances data privacy when compared to simpler modes like Electronic Codebook (ECB). To prevent message tampering and to guarantee that messages are not altered during transmission, the system incorporates Hash-based Message Authentication Code (HMAC) with the SHA-256 hashing algorithm (Alotaibi et al., 2024). The publisher generates a HMAC digest of the message using a shared secret key. Upon receipt, the subscriber recomputes the HMAC and compares it to the original. If the digests match, the message is considered authentic and untampered. The system also incorporates a Role-Based Access Control (RBAC) mechanism, where each user is assigned a role (e.g., admin, sensor, viewer) during registration. These roles determine the specific topics a user is permitted to publish to or subscribe from, enforcing the principle of least privilege. This prevents unauthorized users from accessing or interfering with sensitive MQTT topics. A One-Time Password (OTP) verification mechanism is implemented to enhance security during both user registration and password recovery. During sign-up, new users must verify their identity using a randomly generated OTP that is displayed directly within the GUI. The same mechanism is employed during password reset, where an OTP is locally generated and presented on the interface to validate the recovery attempt (Li & Palanisamy, 2019). Each OTP expires after 30 seconds, and a cooldown period (e.g., 30 minutes) is enforced before a new OTP can be requested. This GUI-based verification approach maintains strong authentication without relying on external SMS services. In addition, the system enforces an account lockout policy that temporarily disables user access after three failed login attempts, protecting against brute-force login attacks.

Comprehensive audit logging is enabled in both the publisher and subscriber modules. Logged data includes user login events, role assignments, topic access, timestamps, IP addresses, and encrypted message content. These logs support system accountability, real-time monitoring, and post-incident forensic analysis — especially in security-sensitive environments such as Industrial IoT (IIoT). To visualize and validate message flow during development and testing, the system leverages MQTT Explorer, a third-party graphical MQTT client tool. MQTT Explorer provides a real-time hierarchical view of all published and subscribed topics, payloads, and retained messages. It was instrumental in verifying message structure, topic organization, and successful decryption during system simulations. The entire system is developed using Python 3.13, with Tkinter for GUI design, Mosquitto as the MQTT broker, and supporting libraries including paho-mqtt, pycryptodome, bcrypt, and socket (Li & Palanisamy, 2019). Coding and testing were carried out using Sublime Text. By simulating multiple user interactions, encrypted payload exchanges, and access-controlled communication, the system demonstrates the practical feasibility of deploying secure MQTT-based protocols tailored for privacy, scalability, and efficient operation in real-world IoT deployments. This paper delivers a robust and functional prototype that showcases the effectiveness of integrating privacy-preserving techniques into MQTT communication. The combination of encryption, authentication, role enforcement, OTP verification, audit logging, and testing tools presents a holistic approach to addressing the core security challenges facing modern IoT infrastructures.

REVIEW OF RELATED WORKS.

Alaba et al. (2017): Internet of Things Security: A Survey, conducted a comprehensive survey on IoT security and identified MQTT as a vulnerable protocol due to its lack of native security features. Their study highlighted key risks such as replay attacks, unauthorized publishing, and eavesdropping. The authors advocated for the integration of lightweight cryptographic protocols and access control mechanisms as essential components in securing IoT systems. This directly aligns with the present study's use of AES, HMAC, and RBAC.

Das et al. (2016): A Dynamic ID-based Remote User Authentication Scheme proposed a dynamic ID-based authentication mechanism to secure remote users. Their study addressed password-based vulnerabilities but lacked integration with lightweight IoT protocols like MQTT. Insights from this work informed the OTP-based login system implemented in the current project, ensuring secure user authentication across multiple devices.

Davis (2019): Perceived Usefulness, Perceived Ease of Use, and User Acceptance. Davis introduced the Technology Acceptance Model (TAM) to evaluate user adoption of information systems. Although early and conceptual, this model provided guidance for designing a user-friendly MQTT GUI. Usability considerations were incorporated to enhance system adoption without compromising security.

Ferraiolo, et al. (2001): Role-Based Access Control, established the foundation of Role-Based Access Control (RBAC), defining structured permission allocation. While the model does not cover encryption or transport security, it directly influenced the implementation of topic-based role access in the MQTT GUI, ensuring only authorized users can publish or subscribe to specific topics.

Frankel, et.el. (2003): The AES-CBC Cipher Algorithm and Its Use with IPsec, Frankel demonstrated AES-CBC encryption with randomized IVs to prevent replay and ciphertext pattern attacks. Their findings guided the per-message IV generation and AES-128 encryption applied in the publisher and subscriber scripts, ensuring secure MQTT message transmission.

Gefen et al. (2003): Trust and TAM in Online Shopping, integrated trust modeling with TAM to assess user acceptance in online platforms. Although focused on e-commerce, their framework emphasized trust and perceived security—concepts adopted in the MQTT GUI design, where OTP verification, hashed passwords, and role-based access foster user confidence.

Hunkeler et al. (2008): MQTT-S—A Publish/Subscribe Protocol for Wireless Sensor Networks, introduced MQTT-S, a lightweight publish/subscribe protocol for resource-constrained networks. While QoS limitations exist, their work validated MQTT as a suitable communication protocol for IIoT, forming the basis for the messaging architecture in this study.

Lin et al. (2017): A Survey on Internet of Things: Architecture, Enabling Technologies, Security, and Applications surveyed IoT architectures, enabling technologies, and security challenges. Though primarily conceptual, their findings influenced the overall system design, including MQTT selection, TLS integration, and AES encryption in a multi-device environment.

Naik (2017): Choice of Effective Messaging Protocols for IoT Systems, Naik evaluated MQTT, CoAP, AMQP, and HTTP for IoT communication. The study highlighted MQTT's lightweight nature and suitability for constrained devices but noted minimal security evaluation. These insights reinforced the choice of MQTT, combined with AES/TLS, in the present system.

Niruntasukrat et al. (2016): Authorization Mechanism for MQTT-based Internet of Things. proposed topic-level authorization and transport encryption for MQTT clients. Their approach validated bcrypt password hashing and TLS for secure communication, directly informing the secure login and role-based topic access implemented in this study.

Roman et al. (2013): Securing the Internet of Things, outlined conceptual IoT security frameworks, emphasizing encryption, authentication, and access control. These principles guided the integration of AES, TLS, OTP, and RBAC in the MQTT GUI system.

Venkatesh & Bala (2008): Technology Acceptance Model 3, expanded TAM3 to incorporate interventions affecting adoption. Their study supported the inclusion of user-friendly design elements in the MQTT GUI, balancing usability with security mechanisms.

Zeadally et al. (2013): Towards Privacy Protection in Smart Grid, investigated privacy-preserving mechanisms for smart grid communications, highlighting encryption, authentication, and logging. These findings informed the integration of audit logging, OTP recovery, and Differential Privacy techniques in the current prototype.

Russo et al. (2022): Analysis on Functionalities and Security Features of Internet of Things Related Protocols. analyzed IoT protocols including MQTT, CoAP, and AMQP, highlighting their security strengths and gaps. Their work reinforced the importance of combining TLS, AES, and authentication mechanisms to secure communications in IIoT systems.

Santos et al. (2021): Secure MQTT Broker for IIoT Applications. examined techniques to harden MQTT brokers for industrial IoT, including authentication, session management, and topic access control. Their recommendations guided the broker configuration in this study, ensuring robust server-side security.

Segarra et al. (2020): MQT-TZ: Hardening IoT Brokers Using ARM TrustZone, proposed ARM TrustZone for broker isolation and secure execution. While hardware-specific, the study highlighted best practices for access control and broker security, informing the security architecture of the MQTT broker in this project.

Singh et al. (2015): Secure MQTT for Internet of Things (IoT), explored encryption and authentication in MQTT for IoT devices, validating AES and TLS integration. Their findings directly support the encryption and transport security measures adopted in this study.

Sun et al. (2016): Data Security and Privacy in Cloud Computing, examined encryption, access control, and privacy mechanisms in cloud-based IoT. Their recommendations influenced secure data handling and message storage practices in the MQTT GUI, maintaining confidentiality for local and remote communications.

Vaccari et al. (2020): SlowITe: A Novel Denial of Service Attack Affecting MQTT, highlighted DoS vulnerabilities in MQTT brokers. This prompted the inclusion of authentication checks, session monitoring, and throttling mechanisms in the system to mitigate potential denial-of-service threats.

MATERIALS AND METHODS

The proposed system is a secure, privacy-preserving MQTT communication framework designed for IIoT. It extends the standard MQTT architecture by integrating application-layer encryption, integrity checks, robust authentication, transport-layer encryption, role-based access control, and detailed audit logging. It was implemented as a Python prototype with a Tkinter GUI and validated in a cross-device testbed: a PC hosting broker + publisher, and a mobile phone running Ubuntu via UserLAnd acting as the subscriber (GUI accessed remotely through RVNC).

A. Components of the Proposed System

The proposed system comprises modular components described below:

1. Graphical User Interface (Presentation Layer)

   i. Implemented with Tkinter. Screens: Login, Sign-up, OTP verification, Forgot Password, Role selection, Topic selection, Publish, Subscribe.

ii. Provides user feedback: OTP sent, OTP expired, lockout, publish success/failure, TLS connection errors, HMAC mismatch alerts.

2. Authentication & OTP Module (Application Layer)

i. bcrypt for password hashing.

ii. OTP generation (6-digit), expiry 30s, resend cooldown (e.g., 30 minutes).

iii. Gmail SMTP integration using smtplib/ssl for OTP delivery. (Implementation notes: use app-specific password or OAuth in production.)

iv. Lockout policy: 3 failed attempts → 5-minute lockout.

3. Encryption & Integrity Module (Application Layer)

i. AES-CBC with a fresh random IV per message. IV || ciphertext || HMAC are concatenated and Base64-encoded for safe MQTT transport.

ii. HMAC-SHA256 over (IV + ciphertext) for integrity.

4. RBAC Manager (Application Layer)

i. Maps roles to permitted topics; GUI shows only allowed topics. Prevents publish or subscribe actions outside permissions.

5. MQTT Communication Layer (Infrastructure)

i. Mosquitto configured with TLS (port 8883) using ca.crt and server/client certs; clients connect using TLS to ensure confidentiality and server authenticity.

ii. Paho-MQTT clients perform TLS handshake.

iii. Encrypted payloads are published as Base64 strings.

6. Audit Logger (Storage/Infrastructure)

i. Append-only text logs capturing events: timestamp, username, role, action (login/publish/subscribe), topic, IP address, and payload metadata. Logs enable post-incident analysis and system monitoring.

7. Cross-Device Integration

i. Subscriber runs on Android using UserLAnd (Ubuntu) for a Linux user-space, with GUI forwarded via RVNC. This validates mobile usability without building native apps.

B. Functional Analysis of the Proposed System

1. User Registration & OTP

User supplies username, email, password, role. System hashes password with bcrypt and stores user record in users.json. System sends a time-bound OTP via Gmail SMTP and requires verification to complete registration.

2. Authentication & Lockout

On login, password is verified against bcrypt hash. Failed attempts increment the attempts counter; on the 3rd consecutive failure, system sets lock_time (current time + 300 seconds). Locked accounts show remaining lockout time. Forgot Password triggers OTP email for password reset.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue I January 2026

3. Role Enforcement & Topic Control

After authentication, GUI shows only topics permitted by RBAC. Attempting to publish to a forbidden topic produces an access denied error.

4. Message Encryption & Publishing

Before publish: generate IV, AES-CBC encrypt plaintext, compute HMAC over (IV + ciphertext), assemble payload = IV || ciphertext || MAC, Base64 encode, then publish over a TLS-secured connection.

5. Broker Handling

Mosquitto receives the encrypted payload and forwards it to subscribers. Broker operates with TLS, rejecting clients that fail certificate validation (unless intentionally configured insecure).

6. Reception, Verification & Decryption

Subscriber receives Base64 payload, decodes, extracts IV/ciphertext/MAC, verifies HMAC using shared HMAC key, then decrypts ciphertext with AES key and IV. On HMAC mismatch or decryption failure, the subscriber discards message and logs the event.

7. Audit Logging

Each critical event (registration, login success/failure, publish, subscribe, OTP events, lockouts) is appended to audit logs with IP (determined via socket), timestamp, role, and action.

8. Cross-Device Flow

Publisher (PC) publishes to Mosquitto (PC). Subscriber (mobile via UserLAnd) connects using the PC's reachable IP (or local network routing) with TLS. GUI rendered via RVNC allows user to interact as if native.

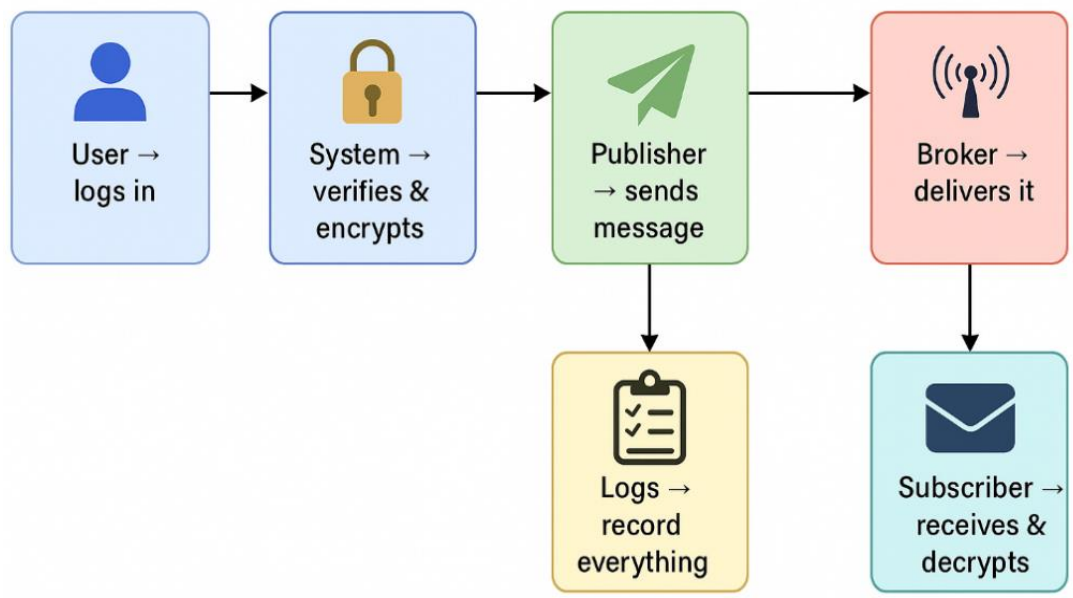

Figure 1: Functional analysis of the Proposed System

C. The Proposed System Block

The block diagram below illustrates the architecture of the proposed secure MQTT-based IIoT communication system. The system includes:

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue I January 2026

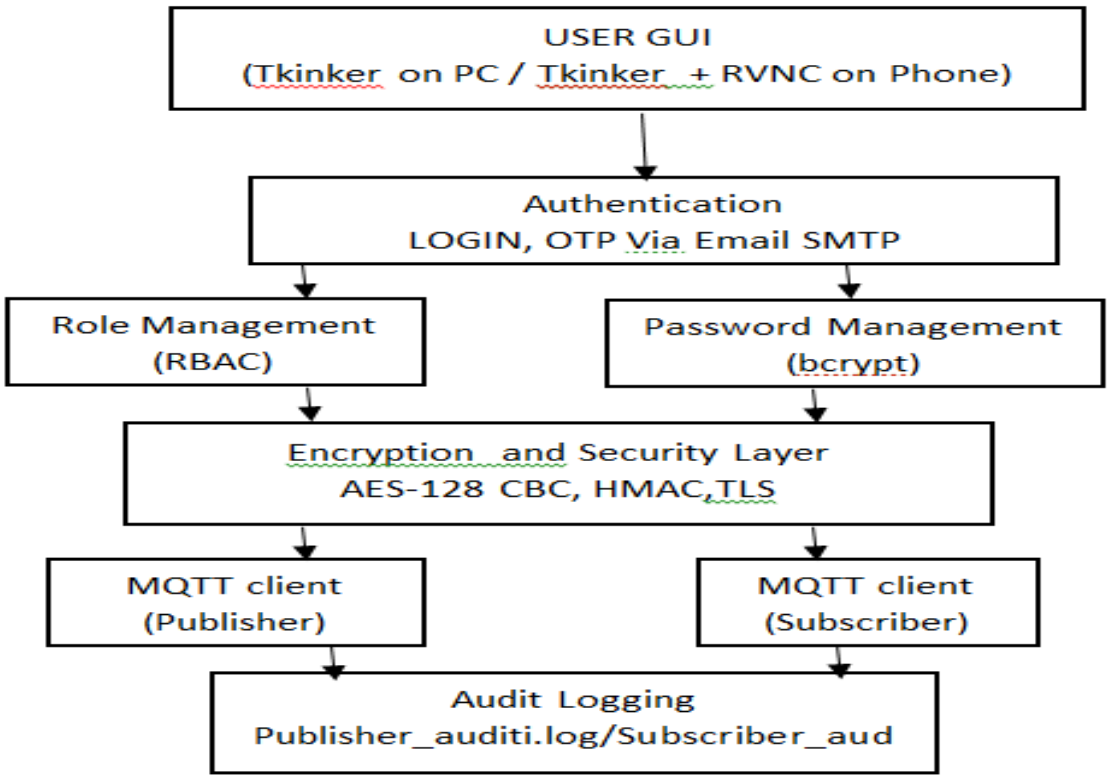

Figure 2: Block Diagram of the Proposed System

Figure 2. (Block Diagram) illustrates the data and control flow between these components:

1. Client (Presentation): Tkinter GUI initiates registration/login/publish/subscribe actions.

2. Auth & OTP Service: On registration or password recovery, GUI requests OTP; the system sends OTP via Gmail SMTP.

3. Encryption Module: Messages from GUI are passed to the encryption module (AES-CBC + HMAC), which produces a Base64 payload.

4. MQTT Broker (Mosquitto with TLS): Receives and routes encrypted payloads; enforces TLS on connections.

5. Subscriber (Cross-device): Authenticates, receives encrypted payload, verifies HMAC, decrypts and displays message.

6. Audit Logger: All events are recorded with IP address, timestamp, and role.

Note: the block diagram explicitly shows the TLS boundary around broker–client communication and an Email SMTP arrow from the OTP service to external email infrastructure (Gmail).

D. System Data Modelling Tools

The project used OOADM and UML to model system behavior and structure. UML artifacts produced include:

1. Use Case Diagram: models actor interactions such as Register, Authenticate, Publish, Subscribe, Reset Password (OTP).

2. Sequence Diagrams: show runtime flows — e.g., login with OTP, publish message (encrypt → publish → broker → subscriber → decrypt), TLS handshake flow between client and broker, and OTP email delivery timeline (generate → send via SMTP → user enters OTP).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue I January 2026

3. Class Diagram: captures static structure (User, Authenticator, OTPManager, MQTTPublisher, MQTTSubscriber, RBACManager, AuditLogger, GUI classes).

4. High-Level Architecture Diagram: maps the Presentation / Application / Infrastructure layers and shows TLS and SMTP boundaries.

These diagrams were used to generate the Python classes and to verify responsibilities and message flows during implementation. This demonstrates how users (e.g., Admin, Subscriber) interact with the system to perform actions such as login, publish messages, and reset passwords

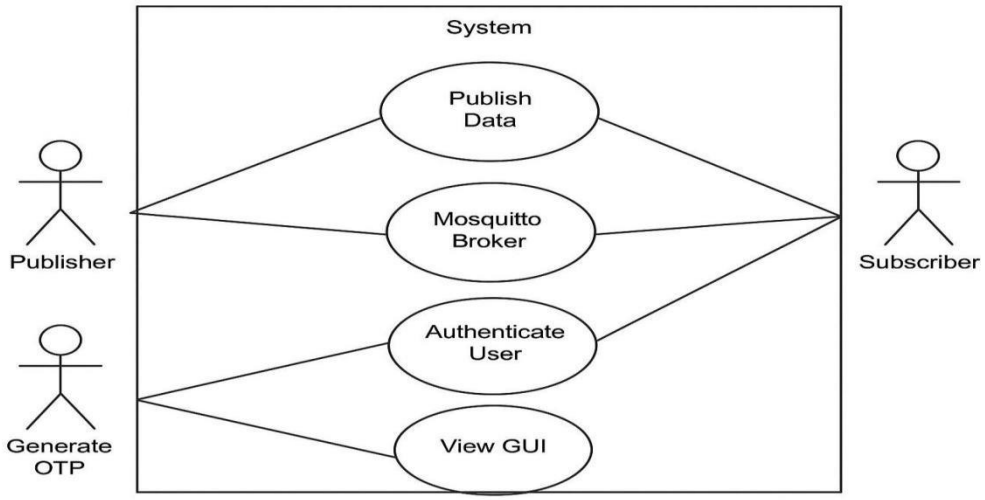

Figure3: The UML Use Case diagram of the proposed system.

This Use Case Diagram models the interactions between external actors (users) and the proposed IIoT communication system. Publisher: Sends encrypted data to the system. Subscriber: Receives and decrypts data from allowed topics. Generate OTP (User): Triggers OTP generation for password reset and secure verification. Publish Data: Action initiated by the Publisher to send encrypted information through the Mosquitto broker. Mosquitto Broker: Acts as the central message-routing mechanism between publishers and subscribers. Authenticate User: Used by any actor who attempts login; involves checking hashed credentials and account status. View GUI: Allows users to interact visually with the system via a graphical interface. Subscribe to Data: Lets the Subscriber receive messages only from permitted topics, enforced via RBAC. This diagram emphasizes the security-aware flow of actions and shows how users interact with distinct modules (authentication, messaging, GUI) in a structured and modular system.

This UML Sequence Diagram models in figure 4 shows the flow of interactions in the enhanced IIoT communication system.

Key Lifelines:

1. Publisher: Sends messages securely to the broker.
2. Mosquitto Broker: Routes encrypted messages.
3. Subscriber: Requests data and handles decryption.
4. Authentication Service: Verifies user credentials.
5. GUI: Interfaces between the user and system actions.

Message Flow:

1. Publisher → TLS publish → Mosquitto Broker: Initiates the flow by sending encrypted data.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue I January 2026

2. Mosquitto Broker → Subscriber: Forwards the message.

3. Subscriber → Authentication Service: Authenticates the user before processing.

4. Authentication Service → Subscriber: Returns success/failure based on bcrypt password check.

5. Subscriber → Mosquitto Broker: Verifies access via RBAC.

6. Subscriber: Decrypts the message using AES and verifies with HMAC.

7. Subscriber → GUI → Audit logging: Displays the result to the user and stores it.

Outlines the flow of messages between objects such as the GUI, authentication module, broker, and subscriber.

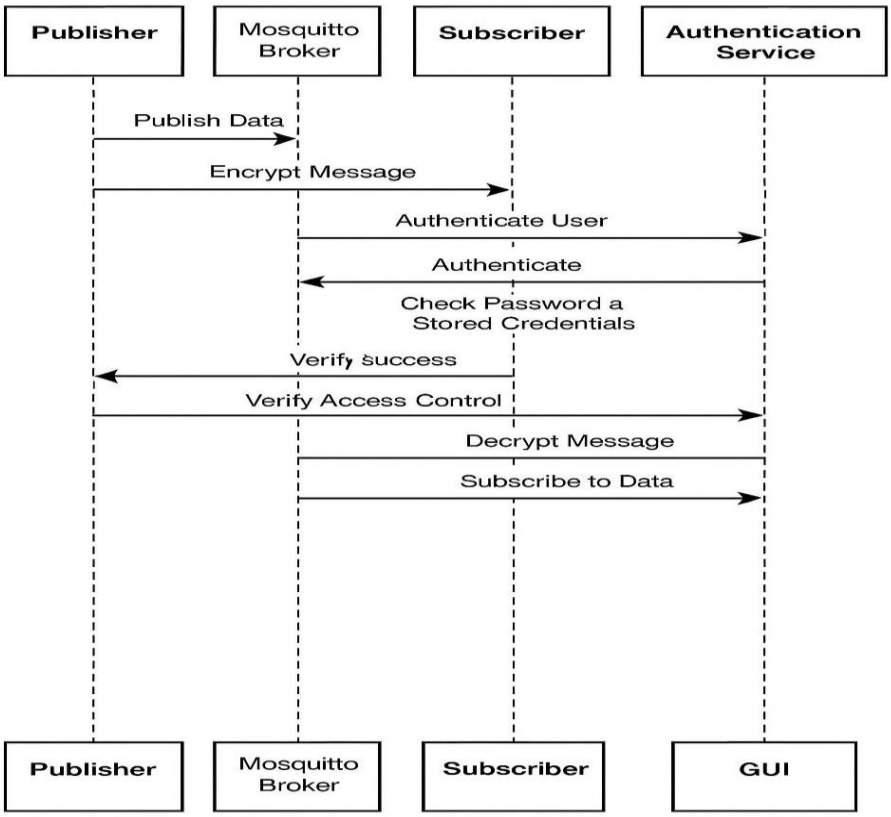

Figure 4: The UML sequence diagram of the proposed system

Class Diagram

Depicts static structure by showing classes, their attributes, methods, and relationships (It shows the main classes such as User, OTPVerifier, MQTTPublisher) and how they relate to each other

Key classes and responsibilities:

1. User: attributes (username, bcrypt_hash, role, email, attempts, lock_time), methods (verify_password(), is_locked()).
2. Authenticator: login(), generate_otp(), send_otp_via_gmail(), validate_otp(), reset_password().
3. MQTTPublisher: encrypt_message(), generate_hmac(), publish_tls().
4. MQTTSubscriber: subscribe_tls(), verify_hmac(), decrypt_message().

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue I January 2026

5. RBACManager: get_allowed_topics(), enforce().
6. AuditLogger: log_event(ip, user, role, action, topic).
7. GUI: orchestrates user interactions and presents TLS/OTP errors.

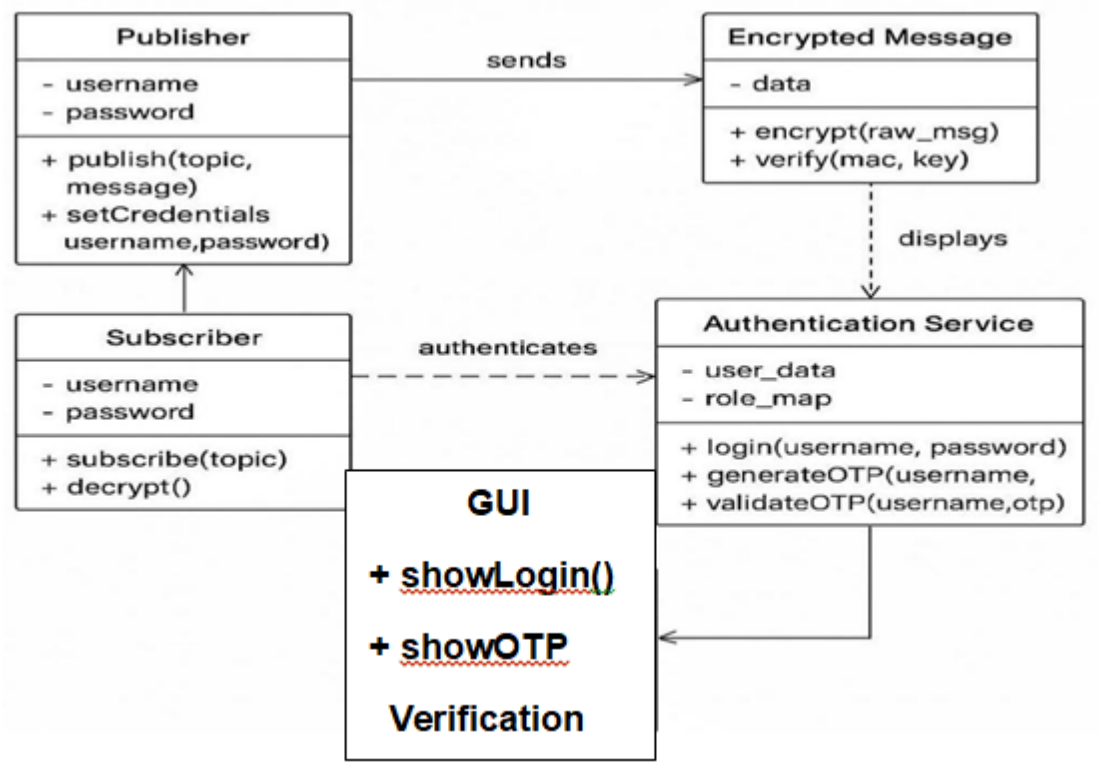

Figure 5.: The UML Class Diagram of the Proposed System

Figure 6. (High-Level Model) below organizes functional components and information flows:

1. Core Secure MQTT System: Mosquitto broker (TLS) at the center.
2. Publisher Module (PC): GUI → Authenticator → Encryption → Publish (TLS).
3. Subscriber Module (Mobile via UserLAnd+RVNC): Connect (TLS) → Receive → HMAC verify → AES decrypt → Display.
4. Auth/OTP Service: handles OTP generation and delivery via Gmail SMTP.
5. Storage: users.json for credentials; audit logs for events.
6. Monitoring & Dev Tools: MQTT Explorer for runtime observation; Sublime/Nano for code editing.

All inter-module flows are secured: message payloads are encrypted and integrity-protected at the Application layer; TLS secures Transport layer; GUI operates at Presentation layer.

The model maps onto the OSI model as follows:

1. Application Layer: MQTT, GUI actions, OTP, encryption logic.
2. Presentation Layer (within Application context): payload format (IV || ciphertext || MAC), Base64 encoding/decoding, and serialization of audit entries.
3. Transport Layer: TLS securing TCP between clients and broker (port 8883).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue I January 2026

4. Network / Data Link / Physical Layers: IP addressing and underlying Wi-Fi/Ethernet connectivity enabling PC↔mobile communication.

The high-level model demonstrates how data paths are protected end-to-end and how roles and policies control access to sensitive topics.

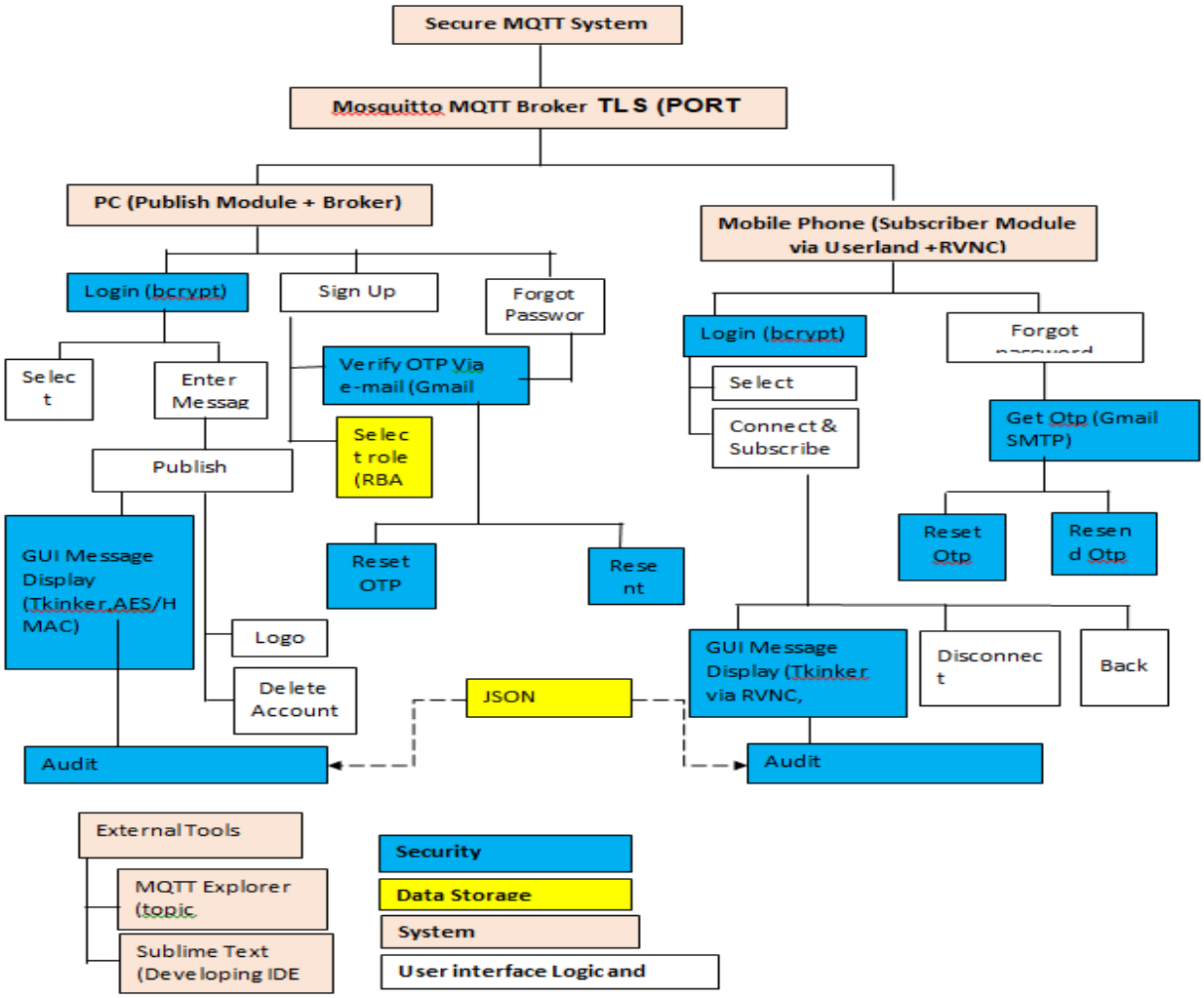

Figure 6. High Level Model of a Secure MQTT System

Experiment and Result

This section documents the security evaluation performed on the prototype MQTT system. The goal of this phase was to validate that the implemented controls (bcrypt password hashing, AES-CBC + HMAC, TLS on Mosquitto, RBAC, OTP workflows and audit logging) provide effective defenses against common attacks in IIoT contexts. Penetration testing was performed in a controlled lab environment using the deployed components (Windows PC: broker & publisher; Android device: subscriber via UserLAnd + RVNC; Kali in VirtualBox for attacker tooling).

Tools used

i. **John the Ripper** — offline password/crack testing against bcrypt hashes.

ii. **Nmap / Zenmap** — port and service discovery, TLS cipher checks.

iii. **Wireshark** — packet capture and protocol analysis for 1883/8883 traffic.

iv. **Bettercap** — ARP spoofing, MITM experiments and traffic redirection/tampering.

v. **Supporting**: MQTT Explorer (topic monitoring), Mosquitto (broker), Kali (testbed).

A. Password / Credentials Testing (John the Ripper)
Procedure: bcrypt password hashes were exported from users.json and tested with John the Ripper using standard wordlists and incremental modes (CPU-bound cracking).
Outcome: With reasonable bcrypt cost factors and per-user salts, cracking times were impractical for real-world passwords. The test validated bcrypt's effectiveness for offline attack resistance.
Recommendation: Enforce a minimum password complexity policy and consider periodic rehashing with a higher bcrypt work factor as hardware capability grows.

B. Port & Service Scanning (Nmap)
Procedure: Nmap scans were run against the broker host to enumerate listening ports and services, confirm exposed ports (1883, 8883), and inspect TLS properties when enabled.
Outcome: The correct operational configuration exposes TLS (8883) for secure client connections; plaintext 1883 was present only during specific test modes. Nmap was useful to confirm no extra services were unintentionally exposed.
Recommendation: In production, disable plaintext 1883 entirely and restrict access to 8883 via firewall rules (allow only required client subnets). Periodically run Nmap/automated scans as part of maintenance.

C. Packet Capture & Traffic Analysis (Wireshark)
Procedure: Wireshark captures were taken while running three modes: (a) 1883 plaintext, (b) 1883 with application-layer AES-CBC + HMAC, and (c) 8883 TLS. Captures focused on payload visibility and handshake traces.
Outcome:

i. **1883 (no AES/HMAC):** Plaintext payloads visible on the wire; topic and message content readable.

ii. **1883 + AES/HMAC:** Payloads were unreadable in raw capture; however, topic metadata and TCP/IP flow remained visible. HMAC verification at the subscriber detected any tampering.

iii. **8883 TLS:** Entire TLS record encrypted; packet captures show TLS handshake and encrypted application data; payloads unreadable without keys.
Interpretation: AES+HMAC protects payload confidentiality and integrity, but transport metadata is still observable. TLS provides stronger transport-layer confidentiality and integrity, protecting broader metadata as well.

D. **Man-in-the-Middle Experiments (Bettercap)**
Procedure: On Kali, Bettercap was used to ARP-spoof the Windows host/gateway and redirect broker-bound connections to a local proxy for controlled tampering. Experiments attempted to read and inject PUBLISH messages in both plaintext and protected modes.
Outcome:

- **Plaintext 1883:** Bettercap could read and inject arbitrary PUBLISH messages (shows complete risk of MITM on plaintext).

- **1883 with AES+HMAC (application-layer):** Bettercap could not silently alter payloads without invalidating the HMAC; malformed messages were detected and rejected by the subscriber.

- **8883 TLS:** Bettercap could not decrypt or tamper with the TLS stream unless a trusted CA/certificate was installed on the client.
Recommendation: Use TLS (8883) as primary defense. If TLS cannot be used, combine application-layer AES+HMAC with network controls — — but accept that some metadata (topics, IPs) will remain observable.

RESULT DISCUSION

1. **Plaintext 1883 is insecure** — avoid enabling in production (open to eavesdropping and easy MITM injection).

2. **AES-CBC + HMAC on 1883** — provides payload confidentiality and tamper detection; however it does not hide transport metadata (topics, IPs, ports). Use only as a fallback when TLS is not available.

3. **TLS on 8883** — best practice: use TLS for broker–client communication. TLS protects payload and many transport-level metadata elements and prevents passive/active interception without installed trusted certs.

4. **bcrypt** — effective for password hardening; enforce password complexity and rotate bcrypt parameters as needed.

5. **Operational hygiene** — disable unused ports (1883), secure CA/private keys, rotate keys and certificates periodically, and restrict broker access via firewall rules.

6. **Logging & detection** — audit logs (publisher/subscriber) and periodic packet captures are useful for forensic analysis; integrate automated alerting where possible.

7. **Limitations** — AES+HMAC requires careful packet buffering/handling to deal with TCP segmentation; MITM experiments involving TLS require client certificate/trust changes to succeed and thus do not represent a transparent attack in correctly configured environments.

Table 1: Test Matrix (brief)

Test Mode	Tool	Result
1883 plaintext	Wireshark/Bettercap	Payloads readable; MITM injection trivial
1883 + AES/HMAC	Wireshark/ Bettercap	Payload unreadable; tampering detected (HMAC)
8883 (TLS)	Wireshark/Bettercap	Payload encrypted (TLS); MITM prevented without trust chain
Password cracking	John the Ripper	bcrypt hashes resistant under current parameters; complexity remains key
Port enumeration	Nmap	Verified exposed services; recommended disabling 1883 in production

Testing Outcome

All major features passed their test cases, confirming that the system behaved as intended under both normal and edge conditions. The TLS-secured MQTT system demonstrated robustness, correctness, and responsiveness during cross-device experiments between PC (broker/publisher) and mobile (subscriber). Audit logs accurately captured all user activities, proving accountability.

CONCLUSION

The project demonstrated the feasibility of building a secure, lightweight, and privacy-preserving MQTT communication system suitable for IIoT applications. By integrating cryptographic safeguards (AES-CBC, HMAC, bcrypt) with authentication, transport-level TLS, RBAC, and auditability, the system effectively mitigated threats such as impersonation, unauthorized topic publishing, and message interception. The

comparative evaluation of 1883 without security, 1883 with AES/HMAC, and 8883 with TLS confirmed the incremental strength of layered protections, with TLS providing the most robust confidentiality. The success of the cross-device demonstration—broker/publisher on PC and subscriber on mobile (via UserLAnd and RVNC)—validates the portability, modularity, and usability of the system. This shows that lightweight industrial protocols like MQTT can be fortified without losing efficiency or accessibility, balancing security with real-world usability.

Further Works

While this project successfully met its objectives, future enhancements can focus on:

1. Biometric Authentication – replacing or augmenting passwords with fingerprint or facial recognition.

2. Blockchain Integration – ensuring immutable audit logs and verifiable trust among multiple stakeholders.

3. Homomorphic Encryption – enabling computations on encrypted industrial data.

4. Real Deployment Testing – scaling the system in live IIoT infrastructure to assess performance under real workloads.

1. Zero-Knowledge Proofs (ZKPs) – to strengthen authentication without revealing credentials.

REFERENCES

1. Alaba, F. A., Othman, M., Hashem, I. A. T., & Alotaibi, F. (2017). Internet of Things security: A survey. Journal of Network and Computer Applications, 88, 10–28. https://doi.org/10.1016/j.jnca.2017.04.002
2. Li, C., & Palanisamy, B. (2019). Privacy in Internet of Things: From principles to technologies. IEEE Internet of Things Journal, 6(1), 488–505. https://doi.org/10.1109/JIOT.2018.2877831
3. Roman, R., Zhou, J., & Lopez, J. (2013). On the features and challenges of security and privacy in distributed Internet of Things. Computer Networks, 57(10), 2266–2279. https://doi.org/10.1016/j.comnet.2012.12.018
4. Russo, M., et al. (2022). Analysis on functionalities and security features of Internet of Things related protocols. Wireless Networks. https://doi.org/10.1007/s11276-022-02999-7
5. Santos, J., et al. (2021). Secure MQTT broker for IIoT applications. IEEE Internet of Things Journal, 8(18), 14433–14444. https://doi.org/10.1109/JIOT.2021.3067994
6. Segarra, C., Delgado-Gonzalo, R., & Schiavoni, V. (2020). MQT-TZ: Hardening IoT brokers using ARM TrustZone. arXiv. https://doi.org/10.48550/arXiv.2007.12442
7. Singh, M., Rajan, M., Shivraj, V., & Balamuralidhar, P. (2015). Secure MQTT for Internet of Things (IoT). In 2015 Fifth International Conference on Communication Systems and Network Technologies (pp. 746–751). IEEE. https://doi.org/10.1109/CSNT.2015.160
8. Sun, Y., Zhang, Y., Xiong, Y., & Zhu, H. (2016). Data security and privacy in cloud computing. International Journal of Distributed Sensor Networks, 12(4), 1–9. https://doi.org/10.1155/2016/2944075
9. Vaccari, I., Aiello, M., & Cambiaso, E. (2020). SlowITe: A novel denial of service attack affecting MQTT. Sensors, 20(10), 2932. https://doi.org/10.3390/s20102932
10. Das, M. L., Saxena, N., & Gulati, V. P. (2016). A dynamic ID-based remote user authentication scheme. IEEE Transactions on Consumer Electronics, 50(2), 629–631.
11. Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly, 13(3), 319–340.
12. Ferraiolo, D. F., Kuhn, D. R., & Chandramouli, R. (2001). Role-Based Access Control. Artech House.
13. Frankel, S., Glenn, R., & Kelly, S. (2003). The AES-CBC cipher algorithm and its use with IPsec (RFC 3602). Internet Engineering Task Force.
14. Gefen, D., Karahanna, E., & Straub, D. W. (2003). Trust and TAM in online shopping: An integrated model. MIS Quarterly, 27(1), 51–90.

15. Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops, 791–798.
16. Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., & Zhao, W. (2017). A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications. IEEE Internet of Things Journal, 4(5), 1125–1142.
17. Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP, and HTTP. 2017 IEEE International Systems Engineering Symposium (ISSE), 1–7.
18. Niruntasukrat, A., Issariyapat, C., Pongpaibool, P., Phonphoem, A., & Thiemjarus, S. (2016). Authorization mechanism for MQTT-based Internet of Things. In 2016 IEEE International Conference on Communications Workshops (ICC) (pp. 290–295). IEEE.
19. Roman, R., Najera, P., & Lopez, J. (2013). Securing the Internet of Things. Computer, 44(9), 51–58.
20. Venkatesh, V., & Bala, H. (2008). Technology Acceptance Model 3 and a research agenda on interventions. Decision Sciences, 39(2), 273–315.
21. Zeadally, S., Pathan, A. S. K., & Alcaraz, C. (2013). Towards privacy protection in smart grid. Wireless Personal Communications, 73(1), 23–50.