On Cross-Domain Security Architecture for MQTT-SN in Constrained IoT Systems

Hemant Gupta

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Doctorate of Philosophy
in
Electrical Engineering and Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The rapid growth of limited Internet of Things (IoT) devices has made it even more important to have lightweight, reliable, and secure communication protocols that can operate in environments with limited processing power, memory, and energy. MQTT-SN is a version of MQTT that works with sensor networks that do not use IP. It has a good publish-subscribe model, but lacks key security features, making IoT applications vulnerable. MQTT-SN inherits almost all the properties and features of MQTT, along with its vulnerabilities, and the MQTT-SN specification does not specify how to address the security requirements of MQTT-SN components. This gap drives a thorough examination of MQTT-SN's security in real-world deployments critical to safety and privacy.

This thesis introduces a cohesive security framework for MQTT-SN, formulated and corroborated within three illustrative IoT domains: Vehicle-to-Vehicle and Vehicle-to-Emergency-Services (V2S) communication, smart lock systems, and continuous glucose monitoring (CGM) healthcare devices.

The thesis presents an adapted SREP/SQUARE methodology specifically designed for IoT ecosystems, facilitating the systematic identification of assets, threats, and security requirements. This framework is utilized for all three use cases, generating comprehensive threat models and prioritized security requirements that account for the limitations and risks associated with constrained devices and publish-subscribe architectures.

Later, the thesis presents comprehensive security architectures for V2S, smart lock, and CGM systems. These include protocols for onboarding, key establishment, message protection, and access control, in detail. We use Scyther for formal verification, timing and energy analysis on limited hardware, and comparisons with existing technologies and commercial products to test these architectures. The results show that the proposed designs offer strong authentication, privacy, integrity, authorization, and replay protection.

This work improves the security of IoT communication by providing the first complete, cross-domain security architecture for MQTT-SN. It shows that MQTT-SN can be a strong, scalable, and efficient way for different IoT applications to communicate, as long as it is properly secured. These applications can include smart homes, healthcare, and vehicle safety systems.

Acknowledgements

I would like to thank my family and all my friends who supported me physically and mentally. Finally, I express my most profound appreciation to Prof. Amiya Nayak. He provided guidance, support, advice, and ideas throughout my thesis work. I would not have been able to complete my program if it were not for him.

Contents

List of Tables

List of Figures

Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks.

AEAD Authenticated Encryption with Associated Data.

AEBS Advanced Emergency Braking System.

AES-CCM AES Counter with CBC-MAC.

AES-CMAC AES Cipher-based Message Authentication Code.

AMQP Advanced Message Queuing Protocol.

ARM Advanced RISC Machine.

AVR Atmel AVR Microcontroller.

BLE Bluetooth Low Energy.

C-V2X Cellular Vehicle-to-Everything.

CEP Complex Event Processing.

CGM Continuous Glucose Monitoring.

ClientID MQTT Client Identifier.

CoAP Constrained Application Protocol.

CS Charging Station.

DoS Denial of Service.

DRDoS Distributed Reflection Denial of Service.

DSRC Dedicated Short Range Communication.

DTLS Datagram Transport Layer Security.

DUP Duplicate Flag (MQTT).

ECDH Elliptic Curve Diffie–Hellman.

ECDHE Elliptic Curve Diffie–Hellman Ephemeral.

ECDSA Elliptic Curve Digital Signature Algorithm.

EV Electric Vehicle.

GB Gigabyte.

GHz Gigahertz.

HMAC Hash-based Message Authentication Code.

IDS Intrusion Detection System.

IoT Internet of Things.

IP Internet Protocol.

KB Kilobyte.

kHz Kilohertz.

MAC Message Authentication Code.

MB Megabyte.

MHz Megahertz.

MITM Man-in-the-Middle.

MQTT Message Queuing Telemetry Transport.

MQTT-SN Message Queuing Telemetry Transport for Sensor Networks.

NDN Named Data Networking.

NFC Near Field Communication.

noOS No Operating System.

OOB Out-of-Band.

PERS Personal Emergency Response System.

PIC Peripheral Interface Controller.

PID Proportional–Integral–Derivative Controller.

PKI Public-Key Infrastructure.

PUBCOMP MQTT Publish Complete.

PUBREC MQTT Publish Received.

PUBREL MQTT Publish Release.

QoR Quality of Robustness.

QoS Quality of Service.

RAM Random Access Memory.

RPL IPv6 Routing Protocol for Low-Power and Lossy Networks.

RSU Roadside Unit.

SKKE Symmetric-Key Key Establishment.

SOS Save Our Souls.

SQUARE Security Quality Requirements Engineering.

SREP Security Requirements Engineering Process.

STK Short-Term Key.

TCP Transmission Control Protocol.

TIR Time in Range.

TK Temporary Key.

TLS Transport Layer Security.

UDP User Datagram Protocol.

URI Uniform Resource Identifier.

V2S Vehicle-to-Emergency-Services.

V2V Vehicle-to-Vehicle.

VIN Vehicle Identification Number.

XMPP Extensible Messaging and Presence Protocol.

Chapter 1

Introduction

1.1 Overview

The Internet of Thing (IoT) is involved in many aspects of our daily life. The IoT provides network connectivity to everyday objects [6]. Kevin Ashton coined the term of IoT in 1999 at a conference [7]. After 1999, there was a sudden increase in smart devices, convenient to use and cheap. With the increase in the number of devices, IoT has become an exciting domain for research. Researchers and professionals easily understand the concept of IoT and many research efforts aim to resolve challenges associated with security, privacy and trust of IoT. Some of these challenges also depend on devices' heterogeneity, device identity, bootstrapping, device management, and device-to-device communication. These IoT devices allow each physical object to share information through the Internet, creating more threats for user data and business information. Imagine if these devices have vulnerabilities from the time of their manufacturing, then any adversary may exploit them and use them to spy on us or obtain our data. It would be worse if IoT applications used in critical environments such as smart vehicles, smart home, healthcare, or nuclear reactors were compromised. Many security solutions have been suggested by academics

and deployed by professionals for IoT. However, there is still a considerable gap in the security of constrained IoT devices [8].

The IoT distributed systems demand more flexible communication models. In today's IoT era, end-devices are loosely coupled (ensuring that a single device's failure does not lead to widespread failure) and have very little interaction with each other; the publish-subscribe paradigm is designed with the consideration of such a system. There are many publish-subscribe systems available since the 1980s [3]. With the evolving Internet communications infrastructure, constrained devices require appropriate communication mechanisms in different IoT domains such as healthcare (e.g., remote people monitoring), smart grid, home automation, smart cities (e.g., smart lighting system, pollution monitoring) and vehicular communication. IoT devices use lightweight messaging protocols to communicate with each other in order to save battery, such as MQTT (Message Queuing Telemetry Transport) [9], CoAP (Constrained Application Protocol) [10], XMPP (Extensible Messaging and Presence Protocol) [11], and AMQP (Advanced Message Queuing Protocol) [12]. MQTT-SN [13] is another application layer publish-subscribe protocol designed for sensor networks.

MQTT-SN protocol was introduced in 2008 [13] and standardized in 2013 by OASIS for wireless sensor networks [14]. MQTT-SN is used in IoT environments for constrained devices, which do not have IP-addressing capabilities. MQTT-SN improves efficiency in data transmission as it has a small message size compared to other messaging protocols. MQTT-SN also consumes less power as devices are in the publish-subscribe model, so there is no continuous information polling. In addition, it also provides a keep-alive procedure that allows devices to go to sleep when needed and receive any information waiting for them when they wake up. Therefore, it is a good fit for battery-operated constrained IoT devices.

MQTT-SN is a variant of the MQTT publish-subscribe protocol [14], and the MQTT-SN specification is derived from the MQTT specification; therefore, it inherits almost all the properties and features of MQTT along with its vulnerabilities [13][14]. The MQTT specification also provides client authentication through *username/password* for clients to authenticate themselves to the broker [15]; but this feature has been left out from the MQTT-SN specification [14]. Although, client authentication using *username/password* has been considered in the latest working draft of MQTT-SN [16]. The MQTT-SN specification [14] is designed explicitly for devices communicating over ZigBee, and it lacks security features the same as MQTT, such as mutual authentication, authorization, confidentiality, and others [17]. Multiple studies have been conducted on attacks based on MQTT design issues [18] [19].

The work in this thesis is on communication aspects of IoT devices, mainly application layer publish-subscribe protocols for constrained IoT devices [20]. Our primary research focuses on designing a novel security architecture for MQTT-SN for constrained IoT devices.

1.2 Motivation

The rapid growth of constrained IoT devices—battery-powered, low-memory, often without IP connectivity—creates a space where traditional Internet protocols and security mechanisms simply do not fit. As you note, many of these devices "only communicate using link layer protocol, so protocols that go higher (e.g., up to Network layer) can not be used with these constrained devices. Instead, these constrained devices use BLE or ZigBee as link layer protocol for communication." This immediately rules out a large class of IP-centric protocols and TLS-based solutions for end-to-end protection.

MQTT-SN stands out here because it is explicitly designed for sensor networks and

non-IP environments. It preserves the publish–subscribe model of MQTT while adapting it to devices that lack full TCP/IP stacks and have strict energy and memory constraints compared to:

- **MQTT:** MQTT assumes TCP/IP and typically relies on TLS for security. This is often too heavy for Class III–V devices with tens of kilobytes of RAM and no Wi-Fi. MQTT-SN, by contrast, can run over ZigBee or BLE and uses compact headers and topic IDs instead of long topic strings, reducing bandwidth and energy usage.

- **CoAP:** CoAP is optimized for constrained devices but is request–response oriented and tightly coupled to UDP/IP. It fits well for RESTful interactions but is less natural for loosely coupled, event-driven publish–subscribe scenarios where brokers decouple publishers and subscribers.

- **AMQP:** AMQP is feature-rich and heavyweight, targeting enterprise messaging with reliable queues and complex routing. Its overhead and dependency on full network stacks make it unsuitable for deeply constrained, battery-operated nodes.

- **XMPP:** XMPP is XML-based and chat-oriented, with significant parsing and bandwidth overhead. Even with optimizations, it is not competitive with MQTT-SN for tiny devices on low-rate, low-power links.

In addition, MQTT-SN inherits MQTT's strengths and weaknesses: "MQTT-SN is a variant of the MQTT publish-subscribe protocol, and the MQTT-SN specification is derived from the MQTT specification; therefore, it inherits almost all the properties and features of MQTT along with its vulnerabilities." This combination—excellent fit for constrained, non-IP environments but with largely unspecified or weak security—creates a very specific and important research gap.

Existing works often assume that the link between constrained end-devices and gateways is "secure enough," typically by relying on ZigBee or BLE link-layer encryption or pre-shared secrets. "When developers rely on encryption provided by link layer protocol such as ZigBee, if link-layer encryption fails... the developers are left in a situation where what they rely on does not deliver what they hope." That makes MQTT-SN an ideal focal point: it is the right communication model for constrained IoT, but it lacks a principled, application-layer security architecture that can survive link-layer compromise and heterogeneous deployments across smart vehicles, smart homes, and healthcare. In detailed summary,

1. MQTT-SN is uniquely positioned for constrained, non-IP IoT (ZigBee/BLE, battery-powered, low-memory). Many constrained devices have no user input/output interface, no pre-shared secret for establishing trust between devices, and low computational capabilities; therefore, it is hard to implement public-key operations. In addition, many smart consumer devices are constrained devices. Therefore, the solutions deployed for current MQTT, such as TLS, will not suit MQTT-SN.

2. Existing protocols (MQTT, CoAP, XMPP, AMQP) either assume IP/TCP, are too heavy, or do not match the publish–subscribe interaction pattern needed in many IoT domains. Many constrained devices only communicate using layer protocol, so protocols that go higher (e.g., up to Network layer) cannot be used with these constrained devices. Instead, these constrained devices use BLE (Bluetooth low energy) [21] or ZigBee [22] as link layer protocol for communication. As per our literature review, MQTT-SN is the only application layer messaging protocol that supports communication over link-layer protocols such as ZigBee or BLE.

3. Security is underspecified and fragile, especially on the client–gateway–broker path, and current practice over-trusts link-layer security. Based on our literature review,

in proposed security solutions for MQTT-SN, the gateway device and end-device, and the link between them mostly assumed secure based on some pre-shared secret or other means [17] [23] [24]. The developers rely on ZigBee security to secure the connection between the constrained end-device and the gateway device. However, when developers rely on encryption provided by link layer protocol such as ZigBee, if link-layer encryption fails, such as in the case of IoT Goes Nuclear [22], the developers are left in a situation where what they rely on does not deliver what they hope. Therefore, we need a novel key management technique for key establishment at the application layer for MQTT-SN rather than relying on link layer protocol security.

4. MQTT-SN was standardized in 2013 by OASIS. However, it was not accepted in the market for a long time. On the other hand, MQTT is used in many deployed products. Recently, a few companies like Hive-MQ [25], uBlox [26] and others are providing MQTT-SN services. Furthermore, a few researchers and professionals started designing a new standard draft for MQTT-SN [27], which aligns with the MQTTv5 protocol specification [15]. Thus, the current trend shows the rise of research interest in MQTT-SN.

Real-world, safety- and privacy-critical use cases—V2V/V2S distress signaling, smart locks, and CGM—demand stronger, formally reasoned security than what MQTT-SN currently offers. Given this, focusing the thesis on MQTT-SN is not just reasonable—it's strategically powerful. This presents an exciting opportunity to show the usage of MQTT-SN in different fields of IoT. It also shows a gap in the understanding of actual security requirements and the security procedures implemented. It also shows us the opportunity to design and implement the security architecture for MQTT-SN, which can be used in multiple domains (i.e., *Smart Lock* (we use this term in the proposal to

refer to the smart lock use case), *Vehicles* (in sending emergency signals), and continuous glucose monitoring(i.e., CGM) from three different IoT domains (i.e., smart homes, smart vehicles, and smart healthcare)).

1.3 Problem Statement

The core problem addressed in this thesis is that MQTT-SN—designed for constrained, non-IP IoT devices—lacks a complete, domain-independent security architecture, leaving real-world IoT deployments vulnerable to attacks that threaten safety, privacy, and system reliability. Although MQTT-SN is the only lightweight publish-subscribe protocol that operates directly over link-layer technologies such as ZigBee and BLE, its specification omits essential security mechanisms including mutual authentication, secure onboarding, authorization, confidentiality, integrity protection, replay resistance, and robust key management. As a result, developers often rely solely on link-layer encryption or ad-hoc pre-shared secrets, which fail under device compromise, key leakage, or protocol-level attacks. This gap becomes critical in safety- and privacy-sensitive IoT domains—such as Vehicle-to-Emergency-Services (V2S) distress signaling, smart lock systems, and continuous glucose monitoring (CGM)—where constrained devices must communicate securely despite limited memory, processing power, and energy budgets. Existing research provides fragmented countermeasures, but no unified methodology exists for defining security requirements, modeling threats, and designing end-to-end architectures specifically for MQTT-SN across heterogeneous IoT ecosystems. Therefore, the thesis addresses the following overarching problem: MQTT-SN lacks a comprehensive, formally validated, cross-domain security architecture capable of protecting constrained IoT devices against modern threats, especially when link-layer security fails. There is no unified framework to systematically derive security requirements, design se-

cure MQTT-SN communication flows, and evaluate their correctness, performance, and applicability across diverse IoT domains.

1.4 Research Objective

The objective of this thesis is related to the security of MQTT-SN (MQTT for sensor networks) with the help of three use cases: vehicle communication, smart home, and smart healthcare.

- Identify and define comprehensive security requirements for three distinct IoT use cases: Vehicle-to-Vehicle (V2V) communication, Smart lock systems and Continuous Glucose Monitoring (CGM) healthcare devices.
- Design and implement domain-tailored, end-to-end security architectures for MQTT-SN that address the identified requirements in constrained environments.
- Evaluate the proposed architectures through formal verification, performance analysis, and comparative studies to validate security, feasibility, and scalability.

1.5 Contributions

The contributions as part of this thesis are:

- A unified, IoT-specific security requirements framework that can be applied to any MQTT-SN-based system. We focus on defining security requirements for a key aspect of V2V and Smart Lock communication—an area still lacking a comprehensive framework. We apply a modified SREP and SQUARE methodology to develop a threat model for vehicular communication using the MQTT-SN publish/subscribe protocol. This contribution details the revised approach and the rationale behind

establishing security requirements for the V2V and Smart Lock use cases. [This contribution is addressed in Chapter 4.]

- Three complete, formally analyzed security architectures for V2S, Smart Lock, and CGM systems. We focus on the most vulnerable part of MQTT-SN—the client-to-gateway link via the forwarder—often overlooked in research. We implemented a complete security architecture for a V2S prototype using MQTT-SN to transmit distress signals in remote areas without connectivity [5]. [The V2S architecture is presented in Chapter 5, the CGM architecture in Chapter 6, the Smart Lock architecture in Chapter 7].

- Comprehensive validation through formal verification, timing/energy analysis on constrained hardware, and comparative evaluation against existing systems. We provide a rigorous evaluation using Scyther, microcontroller-level timing benchmarks, energy impact analysis, and comparisons with commercial smart locks, CGM devices, and V2V technologies—demonstrating feasibility and security guarantees. [This contribution is developed in Chapter 8.]

By applying the proposed security methodology and architectures across three distinct IoT domains, the thesis provides the first cross-domain, generalizable security framework for MQTT-SN. This demonstrates that MQTT-SN can be made secure and scalable for healthcare, smart home, and vehicular systems when supported by appropriate application-layer security mechanisms.

1.6 Outline

Organization of thesis is as follows:

- Chapter 2 provides foundational knowledge required for the thesis. It covers constrained IoT devices, publish-subscribe systems, MQTT and MQTT-SN, ZigBee and BLE communication, and domain-specific background for V2V, CGM, and Smart Lock systems.

- Chapter 3 surveys existing research on IoT security requirements, publish-subscribe security, messaging protocol comparisons, MQTT/MQTT-SN vulnerabilities, and domain-specific security studies. It identifies gaps that motivate the unified framework proposed in this thesis.

- Chapter 4 introduces a modified SREP/SQUARE methodology tailored for IoT ecosystems. It defines assets, threats, and security requirements for three use cases—V2V, Smart Lock, and CGM—using a structured threat-modelling and risk-assessment framework.

- Chapter 5 proposes a complete security architecture for V2S communication. It includes onboarding, key establishment, message protection, and access control mechanisms, along with formal verification and performance evaluation using the defined security requirements, as mentioned in Chapter 4.

- Chapter 6 presents a secure MQTT-SN-based architecture for CGM systems. It includes onboarding flows, secure data transmission, and threat-specific mitigations, supported by formal analysis and timing/energy measurements on constrained hardware. using the defined security requirements, as mentioned in Chapter 4.

- Chapter 7 introduces a secure Smart Lock architecture using MQTT-SN over ZigBee/BLE. It details trust establishment, access control, secure command execution, and audit logging, along with formal verification and comparison with commercial products.

- Chapter 8 evaluates the performance of all three proposed architectures. It includes cryptographic timing, energy consumption, end-to-end latency, formal verification results, and comparative analyses against existing technologies and commercial systems.

- Chapter 9 summarizes the thesis contributions and discusses future research directions, including standardization opportunities, lightweight cryptography, and broader applicability of the proposed MQTT-SN security framework.

1.7 List of Publications

Conferences Publication:

- H. Gupta and A. Nayak. (2023, October). Use of MQTT-SN in sending distress signals in vehicular communication. In International Symposium on Networks, Computers and Communications (ISNCC) (pp. 1-6). IEEE.

- H. Gupta and A. Nayak. (2023, October). Addressing IoT security challenges: A framework for determining security requirements of smart locks leveraging MQTT-SN. In International Symposium on Networks, Computers and Communications (ISNCC) (pp. 1-7). IEEE. [Part of Chapter 4]

- H. Gupta and A. Nayak. (2024, October). Security Architecture for Vehicle to Emergency Services (V2S) Communicating Over MQTT-SN. In IEEE 100th Vehicular Technology Conference (VTC2024-Fall) (pp. 1-6). [Part of Chapter 5]

- H. Gupta and A. Nayak. (2024, October). Access Control in Vehicle to Emergency Services (V2S) Communicating Over MQTT-SN, International Symposium on Networks, Computers and Communications (ISNCC), Washington DC, USA, 2024, pp. 1-6, doi: 10.1109/ISNCC62547.2024.10759043.

- H. Gupta and A. Nayak. (2025, August). Security Vulnerabilities and Countermeasures in MQTT-SN Communication. In International Conference on Advances in Computing and Data Sciences, pp. 117-131. Cham: Springer Nature Switzerland, 2025.

Journal Publications:

- H. Gupta and A. Nayak, "Publish Subscribe System Security Requirement: A Case Study for V2V Communication," in IEEE Open Journal of the Computer Society, vol. 5, pp. 389-405, 2024, doi: 10.1109/OJCS.2024.3442921.

- H. Gupta, B. B. Gupta and A. Nayak, "Security Architecture for Continuous Glucose Monitoring (CGM) Communicating Over MQTT-SN," submitted to Peer-to-Peer Networking and Applications.

Chapter 2

Background

2.1 Introduction

In the chapter, we discuss the detailed background related to publish/subscribe system and MQTT protocol.

2.2 Constrained IoT Devices

Borman [8] classified IoT devices into three classes based on the memory size (i.e., RAM and Flash), ranging from 10KB to 250 KB, power and energy consumption. Hemant et al. [1] extended the Borman's classification; they classified the IoT device processors into five classes.

Our thesis will focus on end-devices with low memory (128 Bytes to 1024 Kilobytes), less computational capabilities, no Wi-Fi capabilities, and battery-operated (i.e., Class III and Class IV). We will also work with Class I and Class II devices, which use batteries for power supply, they are constrained as we need to choose algorithms so that these IoT devices can run for a long time (i.e., at least 3-5 years or more). Refer to Table 2.1 for

detailed classification.

Due to all these constraints on devices, as mentioned earlier, we need a security solution to fulfill the security requirements of constrained consumer IoT devices.

Table 2.1: Extended Taxonomy of IoT Device Processors [1]

Class/ Features	**Class I**	**Class II**	**Class III**	**Class IV**	**Class V**
Bus Size (in bits)	32 or 64	32	16	8	4 and 8
RAM Size	MBs to GB	KBs to MBs	10 KB to 1024 KB	128 B to KBs	128 B to 8 KB
Wi-Fi Supported	Yes	Yes	No	No	No
Clock Frequency	250 MHz - 400 MHz	80 MHz - 180 MHz	4 MHz - 80 MHz	128 Khz - 16 MHz	32 kHz - 8 MHz
Power Usage	10 mWatts to Watts	uWatts	$<$ 1 uWatt	$<$ 1 uWatt	unknown
OS Supported	Linux, Windows, noOS	Contiki, eCos, nuttX, mbedOS, embOS, noOS	Contiki, eCos, nuttX, TinyOS, embOS, noOS	Contiki, nanoRX, nuttX, TinyOS, embOS, no OS	NA
Asymmetric Crypto Supported	Yes	Yes	Yes (EC- a few curves)	Yes (EC-a few curves)	NA
Programming Language Supported	C, C++, Python, JavaScript GO	C, GO, C++, Python, JavaScript	C, C++, Assembly	C preferred, C++, Assembly	Assembly
Example hardware (processor)	Raspberry Pi 3B+ (ARMv8), Beagle Bone	Arm Cortex M3, ARM7	AVR16, PIC24F	AVR8, AT89C51, tinyAVR	AMD Am2900, Atmel MARC4

2.3 Publish-Subscribe Systems

The are many distributed IoT systems, which require more flexible communication systems. Synchronous and point-to-point systems are inflexible and motivate a more flexible and loosely coupled communication system; therefore, publish-subscribe systems are becoming more popular.

Publish-subscribe is a communication architecture where senders of the messages are not aware of the receivers of the messages, and messages are categorized into classes. Publish-Subscribe system [2] is based on the subject/topic or content (i.e., keyword) matching or location-based, which we have explained in later sections. The primary advantage of the publish-subscribe system is the support for asynchronous processing. The message sending node (i.e., publisher) does not need to know about the nodes receiving the messages. Similarly, the message receiving node (i.e., subscriber) does not need to know about the nodes sending the messages. The Subscribers register the subject/topic or pattern for which they want to get the information to the broker. The publishers publish information on subject/topic or pattern to the broker. If the topic or pattern matches, the message is forwarded to the subscriber by the broker. See Figure 2.1.

The publish-subscribe systems are classified primarily using two schemes, i.e., the subscription and architectural models [3].

2.3.1 Subscription Model

A subscriber can define which published messages they want to receive (i.e., subscription). The broker checks if the published message matches the subscription, then the message is forwarded by the broker to the subscriber. There are three main formats of subscriptions:

1. **Topic-based Subscription Model:** In this model, the publish-subscribe system uses the topics used by the publisher to publish messages for filtering and distribution of messages to subscribers. A subscriber can show interest in a set of topics, and the subscriber will then receive matched messages. A subscriber can receive the list of topics either from its manufacturer, publish-subscribe system, or directly from the publisher. The model can be fine-grained by using sub-topics; this results in a hierarchical model.

2. **Content-based Subscription Model:** In this model, the publish-subscribe system uses the content of the messages to match them to the filters (i.e., keywords) defined by the subscriptions. The filters can be predefined or can be customized. The model takes more computational resources than topic-based, but on the other hand, it is more flexible. Please refer to Figure 2.2 as an example where we use binary pattern for matching.

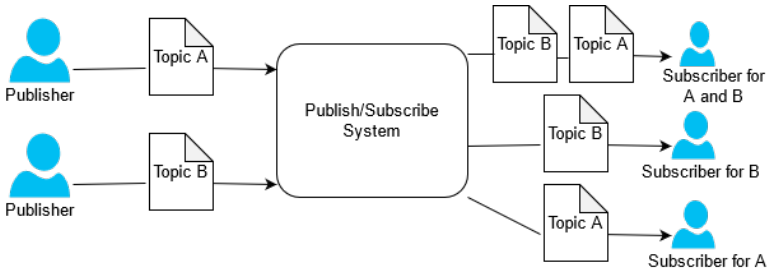


Figure 2.1: Topic-based Subscription Model [2]

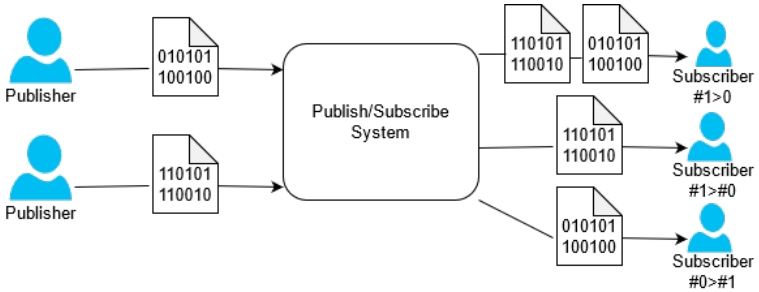


Figure 2.2: Content-based Subscription Model [2]

3. **Location-based Subscription Model:** In this model, the publish-subscribe system uses the geographic location of the publisher and subscriber for sending and receiving messages. The geographic area can be based on locality, cities, provinces, countries or continent. The model is used when a message is published for subscribers in a particular area. All nodes in that area will receive the message.

2.3.2 Architectural Model

The communication architecture is an essential part of the publish-subscribe system as it is also related to the subscription model. The architecture defines how the nodes are organized in the system and communicate with each other. There are two main architectures as defined in [3]:

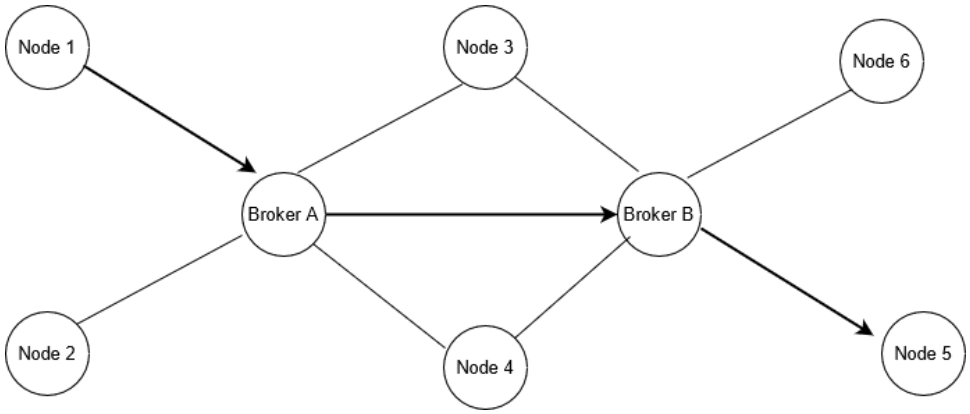


Figure 2.3: Broker Architecture [3]

1. **Broker Architecture:** This architecture is also known as client-server. The management nodes which handle subscribers and communication are called brokers (i.e. servers). Each broker receives subscription requests from the subscribers, messages from publishers. The broker stores the messages and sends them to the respective subscribers. Furthermore, the broker forwards them to other brokers. Thus, the brokers communicate with each other to ensure scalability and message forwarding

to all subscribers. See Figure 2.3. The line with the arrow represents the path taken by message to reach Node 1 to Node 5 through different brokers. The remaining lines represent the connections in the network topology. In Figure 2.3, broker A did not find any subscribers for the message published by Node 1 in its connection, and then broker A forwards the message to broker B. Broker B finds the subscriber for node 1 message and sends that message to node 5.

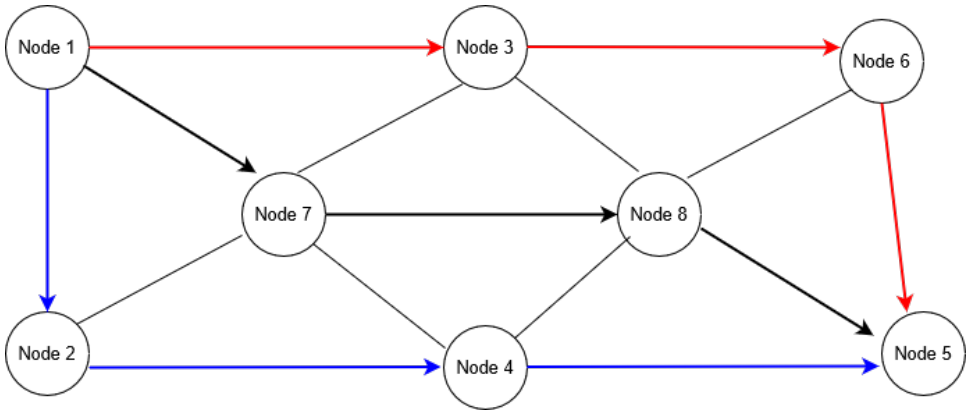


Figure 2.4: Peer-to-Peer Architecture [3]

2. **Peer-to-Peer Architecture:** In peer-to-peer architecture, the nodes directly communicate with their neighbouring nodes through techniques called gossiping and flooding. Flooding involves the broadcast of the message from a particular node to the rest of the nodes. It will continue the process until the hop count is exceeded and shows the route failed to find the destination. On the other hand, to avoid this overhead of flooding and reduce the number of transactions, gossiping broadcasts the message to its neighbour and then selects one of the neighbours and then again it will then broadcast and repeat the process till it reaches the destination.

   There is no broker involved. The nodes can form a structured network where they know network addresses of all the nodes or an unstructured network where they

only know network addresses of some node. In this architecture, each node has to take care of most server functionality. See Figure 2.4; here, we have shown different colour paths that messages can take to reach node 1 to node 5.

2.4 Security Requirements

2.4.1 Security Requirements

A security requirement is a statement that specifies a condition or constraint that must be satisfied by a system or software application to ensure its security. These requirements are crucial for identifying, mitigating, and managing security risks throughout the development life cycle of a system or application. Security requirements encompass various aspects of security, including confidentiality, integrity, availability, authentication, authorization, and non-repudiation. Security requirements specify what objects need protecting, from whom they need protecting, and why. The background of the two security requirement-gathering methods are discussed below:

- The SREP is a crucial process that ensures security considerations are integrated into the system design from the early stages, minimizing the risk of vulnerabilities and ensuring the confidentiality, integrity, and availability of the system's assets. The SREP process is a structured and systematic process that aims to identify, analyze, and specify security requirements for a software application or system. The process involves a series of steps, including initiation, elicitation, analysis, specification, validation, verification, and maintenance. Collaboration and communication among stakeholders, including security experts, software developers, system architects, and end-users, are crucial throughout the SREP process. The SREP process is iterative and should be integrated into the overall software development lifecycle

to achieve a secure and resilient system. By following the SREP process, organizations can ensure that their systems and applications are secure and can meet the ever-changing security threats, regulations, or organizational needs.

- The SQUARE is a structured methodology for incorporating security considerations into the software development process. It aims to ensure that the final product meets the desired security goals while reducing the risk of security breaches and vulnerabilities. The SQUARE method is iterative and can be integrated into existing software development processes, such as the waterfall or agile methodologies. It involves seven phases, including problem definition, elicitation, classification, analysis, specification, validation, verification, and documentation. SQUARE emphasizes the need for collaboration and communication among stakeholders, including security experts, developers, architects, and end-users, to ensure that security is adequately addressed from the early stages of software development. By following the SQUARE method, organizations can enhance the security of their software systems and reduce the risk of security breaches and vulnerabilities.

The security requirements mentioned below combine the security requirements for publish-subscribe protocols proposed in [20] and our preliminary ideas. The security requirements mentioned below are the list of security requirements that may be used to determine the individual IoT use case security requirements, which might differ from each other. Security requirements can be approached in two ways:

1. Point-to-Point: The approach is concerned with the link-level security between two end-points (i.e., peers) of a communication.

2. End-to-End: The approach is concerned with source to destination authentication. It means that the destination should be able to verify the identity of the source.

Now, we briefly discuss security requirements and their sub-divisions:

1. **Authentication:** We have created three sub-divisions of authentication to provide more clarity based on different subjects involved in communication and their security expectations:

   (a) Data Origin Authentication: This property provides supporting evidence that the source of data is as claimed. For example, a subscriber should be able to verify the publisher (i.e., sender) of the message.

   (b) Device Authentication: This property provides confidence in the identity of the device involved in communication is as claimed. For example, in pub-sub, a requirement is that a broker can uniquely identify the device and authenticate it, such that no other device can spoof another device's authentication credentials.

   (c) End-Device Owner/User Authentication: This property provides confidence in the identity of the user/owner of the end-device is as claimed. In pub-sub, the broker should be able to authenticate the owner of the publisher/subscriber end device to avoid issues related to ownership transfer of the device [28].

2. **Confidentiality:** This property makes sure that non-authorized entities are unable to access plaintext messages.

   (a) Information Confidentiality: This property makes sure that information is not in plaintext format at the broker and gateway if the gateway and broker can not be trusted. For example, the property may need an out-of-band agreement between publisher and subscriber.

   (b) Subscription Confidentiality: According to this property, a subscriber subscribes to a topic without informing the broker what the topic is. Thus, the

attacker can deduce the publisher/subscriber system a user is associated with; however, they will not find how a user uses the service (i.e., subscribing to what).

(c) Publication Confidentiality: According to this property, the data published by the publisher is encrypted. It can be of two types. The first, point to point, data transfer between publisher/subscriber to a gateway and from a gateway to a broker is encrypted, but data is available in plaintext at the publisher, gateway, subscriber and broker. The second, end-to-end, data is encrypted until it reaches the final destination (i.e., subscribers). The data remains encrypted at the broker. The end-to-end publication confidentiality may use an out-of-band group key distribution between publisher and subscribers. The out-of-band distribution of keys converts to a multicast model.

3. **Integrity:** Any modification done by unauthorized parties will be detected. Data origin authentication implies both source authentication and message integrity. Although, message integrity can be achieved without data origin authentication.

   (a) Information Integrity: According to this property, the message content is not altered once it has left the sender (i.e., publisher) until it reaches the final receiver (i.e., subscribers).

   (b) Subscription Integrity: According to this property, the topic subscription is protected from unauthorized modifications. For example, if a subscriber subscribes to topic A in the pub-sub system, the subscriber should receive a message published under topic A. No one should be able to alter the records of topics a given subscriber is interested in.

   (c) Broker Integrity: According to this property, the broker is not able to alter

the information published or subscribed to. Therefore, it will put the whole system at risk if the trusted broker is compromised. The trusted broker here implies that all data sent by the publisher for the subscribers are available in plaintext at the broker (i.e., point-to-point confidentiality).

4. **Accountability:** According to this property, a system is able identify the entity responsible for past actions.

5. **Availability:** According to this property, resources and services remain accessible for authorized use.

6. **Authorization:** This is the process of specifying which entities are allowed (e.g., by policy) to access specified computing resources. Authorization is enforced through access control.

7. **Anonymity:** Publisher and subscriber should remain anonymous to each other. However, this property is an optional requirement based on different use cases [20].

2.5 Message Queuing Telemetry Transport (MQTT) Protocol

MQTT [9] [15] is one of the lightweight publish-subscribe machine-to-machine messaging protocols designed for constrained network communication, which IBM introduced in 1999. It is adopted by OASIS (Organization for the Advancement of Structured Information Standards) in 2013 as a standard. There is a new version of MQTT released in 2019 MQTT v5 [1]. MQTT communicates over TCP using port 1883. MQTT also uses

[1]MQTT protocol in CONNECT Control Packet for v3.1 is defined as 3 and v3.1.1 is 4. Therefore latest protocol level is 5

port 8883 to communicate over TLS (Transport Layer Security) to provide security.

It has two different components. The first is the MQTT broker, also known as a "server". The role of a broker is to collect and distribute messages. It also accepts network connections from clients, subscribes and unsubscribe requests from clients, and closes the client's network connection. The second is an MQTT client (device or user application), represented in two different roles: publisher and subscriber. A publisher can publish (i.e., send a message) to the broker, and a subscriber can subscribe (i.e., request a message) from the broker in IoT environment for a topic (label attached to an Application Message). The published information (i.e., message) can be a status message or a command. Please refer to Figure 2.5 for MQTT/MQTT-SN system architecture.

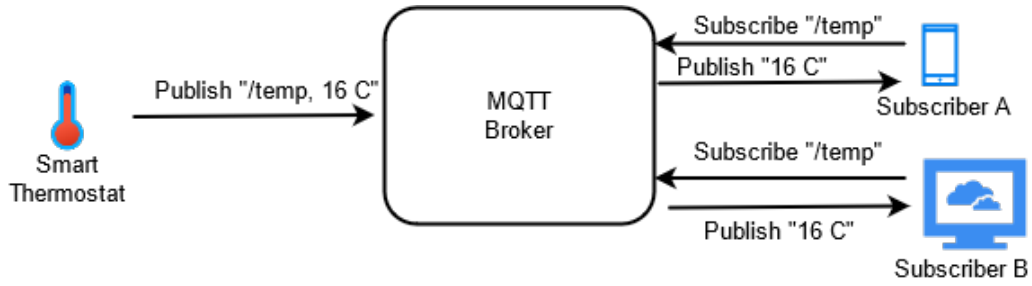


Figure 2.5: MQTT Architecture

To understand how MQTT works, we can use an analogy of on-demand video messages. A video is loaded onto the server by the source, and anyone who wants to watch the video or similar videos can request it from the server by subscribing. Thus, there is no direct relationship between the source and the people viewing the video. MQTT clients establish a connection with the broker using CONNECT message. Then, the client sends a CONNECT message to the server to establish a session with the server, to publish or subscribe to messages. In MQTT, a publisher publishes messages/data on a particular topic using PUBLISH, and a subscriber subscribes to that topic to get that information using SUBSCRIBE. The MQTT broker's primary task is to filter messages

based on topic and then distribute them to the appropriate subscribers. In MQTT, there is no direct connection between a publisher and a subscriber. All clients can publish and subscribe.

MQTT control packet structure consists of three parts: fixed header, variable header and payload (i.e., MQTT message/data). MQTT is a binary protocol with a fixed header of 2-bytes and a message payload of maximum size up to 256 MB. See Figure 2.6 for MQTT fixed header.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

Figure 2.6: MQTT Fixed Header [3]

The main parameters in the CONNECT, PUBLISH and SUBSCRIBE control packets are:

1. **ClientID:** This is a unique identifier for each client. It is a software entity that can be generated by the client, broker, or manufacturer (hardcoded in the device). The server uses it to identify the client and their state relating to an ongoing MQTT session.

2. **Clean Session:** This is a flag in the CONNECT packet. If set to false, a client wants to have a persistent session (keeps track of all the session's information if the devices go offline or disconnect). The stored session information will help the client if it connects with the same ClientID. In addition, the broker stores all the subscriptions and missed messages for a client subscribed with quality of service (QoS) level 1 or 2. If set to true, then the broker deletes all information about the session once the connection terminates.

3. **Username/Password:** This is an optional parameter. The client (device and user) can have both a username and password for authentication to the broker or only a username. If TLS is not used, then the information is sent in plaintext. In the case of username/password, if the device does not have any visual input/output interface (screen or keyboard), it is then configured in the IoT device with the help of another device such as a smartphone or laptop.

4. **Will Topic/Will Message:** This parameter is used to send information to subscribers of a topic if the connection between a publisher and a broker is terminated non-gracefully. The Will message is the application message or control commands that are published to the Will Topic. The publisher can register with the broker a special Will message for a topic. If the publishing client does not disconnect gracefully (i.e., without sending a DISCONNECT message to the broker), the broker will publish this Will message to all the subscribed clients of the topic, allowing them to take corresponding actions.

5. **Topics:** This parameter in MQTT is a string that the broker uses to filter messages for each connected client. An MQTT publisher publishes the message under a topic, which must be at least one character. Topics can be predefined by the manufacturers and brokers and may also be defined by the publishers. Two or more publishers can publish under the same topic. However, it is highly discouraged as an industry practice to identify the source of the message uniquely. MQTT Topics have one or more levels, separated by a forward-slash (/). A subscriber can subscribe to an exact topic used by the publisher, or a subscriber can subscribe to the wildcard. There are two types of wildcards:

    - Single level wildcard (+): Replaces one topic level. We take an example:

/Myhouse/kitchen/temperature, /Myhouse/room1/temperature. These both can be represented as */Myhouse/+/temperature*

- Multi-level wildcard (#): Replaces multiple topic levels. With the help of multi-level wildcard, a client receives all messages of a topic that begins with the pattern before the wildcard character. Using the above example, and it can be written as */Myhouse/# or /#.*

6. **Quality of Service (QoS):** MQTT offers three level of quality of service as part of publish message.

   (a) QoS 0: This is also referred to as fire and forgets (i.e., at most once). It works as the fastest method but is unreliable. The publisher publishes the message only once and then deletes it from the queue. Thus, there is no possibility of duplicate messages.

   (b) QoS 1: This guarantees that message will be delivered at least once or more than once to the broker. It is also referred to as acknowledgement delivery. The publisher publishes the message and waits for acknowledgement from the broker. If the acknowledgement is not received or delayed, it resends the same packet with the *DUP flag set (i.e., Duplicate flag).* The publisher will continue sending packets until it receives an acknowledgement from the broker. The broker might receive multiple copies of the same packet.

   (c) QoS 2: This is also known as assured delivery (i.e., exactly once), as used to avoid duplicate delivery of messages (which is the possibility with QoS 1) and packet loss (which is the possibility with QoS 0). The publisher publishes the message and waits for acknowledgement from the broker. It uses PUBLISH, PUBREC, PUBREL and PUBCOMP message types to ensure exactly-once

delivery of the message, and it requires two-step acknowledgement.

7. **Retain Flag:** If a publisher publishes a message on a topic and there is no available subscriber, the broker usually discards that message. However, the broker keeps the publisher's last message on a topic in memory when Retain flag bit is set in a PUBLISH control packet.

2.5.1 Comparison of MQTT v3.1.1 and MQTT v5

There are a few new features introduced as part of MQTT v5 [15], which were missing in MQTT v3.1.1 [9].

- **ClientID:** In MQTT v3.1.1, a Client may send a ClientID of length zero bytes. In such a unique case, the server assigns a unique ClientID to that client. In the case of MQTT v3.1.1, ClientID is not sent back to the client as part of the connection acknowledgement message. However, in MQTT v5, the ClientID is now sent to the client as well.

- **Reason Code:** MQTT v5 specification provides reason codes, which indicates that a pre-defined protocol error occurred.

- **Session Management (Clean Start Flag):** In MQTT v3.1.1, there is a clean session flag. In MQTT v5, there is a Clean Start flag in the CONNECT message instead of the clean session flag. Using this flag, the broker discards any previous session information, and the client starts with a new session. If the Clean Start flag is not set, the session information would not be deleted automatically after the TCP connection was closed between client and server. For the deletion of the session after the client disconnected, a new header field "Session Expiry Interval" must be set to 0. The deletion of session information means all information (i.e.,

list of subscribing topics, list of publishing topics, Will message and others) of the client stored at the broker with respect to the ClientID is deleted permanently from the broker after session termination.

- **Authentication (AUTH Packet):** In MQTT v5, different authentication schemes can be used, such as SCRAM-SHA-1 for SCRAM (Salted Challenge Response Authentication Mechanism) [29] with SHA-1 or GS2-KRB5 [30] for Kerberos. The AUTH packet chooses and describes a way of authentication that the client (i.e., device) and server have agreed.

- **DISCONNECT Packet:** In MQTT v3.1.1, only clients are allowed to send DISCONNECT messages. In the new version, the server can also send a DISCONNECT message with proper reason codes, and clients can take appropriate actions based on it.

- **QoS 1 or 2:** With MQTT v5, publishers do not retransmit to avoid overload as TCP is a reliable transfer protocol. However, the brokers and clients must re-send unacknowledged packets when the TCP connection is closed in the case of QoS 2, similar to MQTT v3.1.1.

- **Topics:** MQTT v5 supports topic aliases. Topic Aliases are integer values as an alternative for topic names. A publisher publishes under Topic Alias value in the PUBLISH message, which contains a topic name. The broker then processes the message and keeps a mapping between the Integer (Topic Alias) and String (Topic Name). After this, any PUBLISH message for the same topic name can be sent only using Topic Alias and an empty topic name.

2.5.2 MQTT for Sensor Network (MQTT-SN)

MQTT-SN [14] was designed in 2013 specifically for wireless sensor networks (WSN), which cannot communicate over TCP. It communicates over UDP, ZigBee and BLE (Bluetooth Low Energy). An MQTT-SN message consists of two parts: a 2- or 4- octet long header and an optional variable part. There are three components of MQTT-SN:

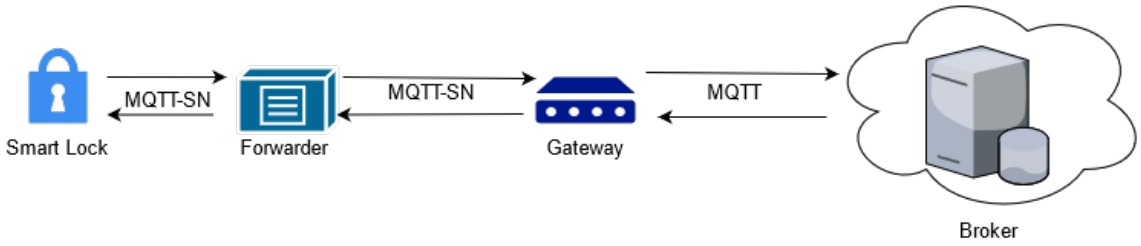


Figure 2.7: MQTT-SN Architecture [3]

1. **MQTT-SN Clients:** These are typically battery-operated constrained devices. These can be a publisher or a subscriber that wants to send/receive a message to/from the broker.

2. **MQTT-SN Gateway:** This is a bridge between MQTT-SN clients and the MQTT broker. It might be integrated with the MQTT broker as well. There are two types of MQTT Gateway:

   (a) Transparent Gateway: This will set up and maintain an MQTT connection between MQTT-SN clients and the server. There will be the same number of connections between gateway and server as MQTT-SN clients connected to the gateway.

   (b) Aggregating Gateway: This will have only one connection between the gateway and the server. All messages coming from different MQTT-SN clients

stop at the gateway, and then the gateway decides which information will be forwarded to the server.

3. **MQTT-SN Forwarder:** An MQTT-SN client can also access the gateway through the forwarder if the gateway is not directly attached to the WSN. The forwarder encapsulates the MQTT-SN client message it receives from the wireless side and forwards it to the gateway without modification; similarly, it decapsulates the message received from the gateway and forwards it to MQTT-SN clients. In the process of encapsulation, the forwarder adds broadcast radius and wireless node id (i.e., a node which has sent or should receive the encapsulated MQTT-SN message).

There are several differences between MQTT and MQTT-SN:

1. In MQTT-SN, CONNECT message is divided into three messages. The additional ones are for Will Topic and Will message and are optional.

2. In MQTT-SN, "pre-defined" topic Ids (i.e., Topic Alias) and short topic names are introduced, for which no registration is needed.

3. A discovery process is introduced to help MQTT-SN clients to discover the gateway node.

4. The Will Topic and messages can be modified during the session for MQTT-SN.

5. MQTT-SN supports offline keep alive procedure for sleeping clients.

6. MQTT publishers and subscribers use IP addresses to communicate with brokers, but many constrained devices only communicate over link layer protocols, such as ZigBee and BLE, which MQTT does not support. However, MQTT-SN also supports communication over link layer protocols. Please refer to Table 2.2.

7. DTLS (Datagram Transport Layer Security) [31] may provide transport layer security to the MQTT-SN over UDP, while MQTT specifications suggest the usage of TLS to provide security.

Table 2.2: TCP/IP Layers Protocols Supported by MQTT and MQTT-SN

TCP/IP Layers	Protocols		
Appplication	MQTT	MQTT-SN over UDP	MQTT-SN
Transport	TCP	UDP	
Network	IPv4, IPv6, 6LoWPAN	IPv4, IPv6, 6LoWPAN	
Data Link	Ethernet, IEEE 802.11, IEEE 802.15.4, IEEE 802.15.4e	Ethernet, IEEE 802.11, IEEE 802.15.4, IEEE 802.15.4e	ZigBee, BLE

2.6 Other Publish/Subscribe Protocols

In this section, we briefly discuss a few other publish/subscribe messaging protocols.

2.6.1 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) [10] [32] [33] is another lightweight machine-to-machine web-based application layer messaging protocol. It is used for constrained devices, networks, and web applications. Its specification is being developed under the IETF CoRE (Constrained Restful Environments) Working Group. It was standardized in the year 2014. It supports both Client/Server (i.e., request/response) and client/broker (i.e., publish/subscribe). CoAP uses Universal Resource Identifier (URI) instead of topics. CoAP is also a binary protocol with a 4-byte fixed header and a payload size

dependent on the webserver. Originally, it is communicated over UDP using port 5683. CoAP also uses port 5684 to communicate over DTLS to provide security.

CoAP over UDP [10] is fast and efficient due to the low overhead of the DTLS protocol as compared to the TLS. CoAP over UDP is preferred for lightweight machine-to-machine communication instead of CoAP over TCP by the developers for constrained devices. However, CoAP over UDP lacks the reliability and service guarantees offered by TCP. It also has network overhead due to many keep-alive messages. CoAP over TCP [34] reduces network overhead by sending fewer keep-alive messages, has better congestion control, and better flow control than CoAP over UDP. The use of CoAP over TCP leads to larger code size (due to support of handshake, reliability, error control and flow control), high network traffic (due to message flow), increased RAM requirements, and larger packet sizes. Therefore, developers need to check the device capabilities before using CoAP over TCP for IoT devices.

We now briefly mention a few known security issues with CoAP. At runtime, CoAP protocol requires the support of a complex runtime parser, which is an easy gateway for introducing vulnerabilities [10] that may cause remote crashes of a node or provide an adversary remote access for executing code. A CoAP response packet size might be larger than the request packet, which can be used by an attacker to convert a small attack packet into a larger attack packet and can cause a denial of service attack. CoAP using UDP is also susceptible to IP address spoofing attacks due to the lack of a handshake. CoAP using web sockets [35] is susceptible to SQL injection attacks. As per our knowledge, several research papers have been published to address CoAP vulnerabilities [36] [37].

2.6.2 Advanced Message Queuing Protocol (AMQP)

AMQP [12] is also a lightweight machine-to-machine application layer messaging protocol for asynchronous message queuing. It was developed in 2003 and standardized by OASIS in 2012. It achieved formal approval as an international standard from ISO and IEC in 2014. It is an open standard for enterprise messaging, which works like instant messaging.

It supports both Client/Server (i.e., request/response) and client/broker (i.e., publish/subscribe) like CoAP [33]. It supports a topic-based publish-subscribe subscription model. AMQP is also a binary protocol with a fixed header size of 8 bytes and has no limit for payload size. AMQP communicates over TCP using port 5672. AMQP also uses port 5671 to communicate over TLS to provide security. Due to communication over TCP, AMQP provides two QoS levels: Unsettle Format (i.e., not reliable) and Settle Format (i.e., reliable). AMQP supports both point-to-point and store-and-forward message delivery.

Like other publish-subscribe systems, the AMQP has three components: producer (i.e., publisher), consumer (i.e., subscriber), and broker. In AMQP, the broker has two parts: exchanges and queues [38] [12]. The publisher or consumer creates the exchange at the start with a given name and then makes that name public. First, the queue has to be attached to the given exchange. A consumer creates a queue and attaches it to a named exchange at the same time. Messages received by the exchange have to be matched to the queue, which is called "binding." The publisher sends messages to the named exchange, and the consumer pulls messages from a queue, or the queue pushes them to the consumer depending on the configuration. AMQP is designed to provide reliability, security, provisioning, interoperability, and extensibility. AMQP uses SASL (Simple Authentication and Security Layer) for client security and TLS for communication channel security.

2.6.3 Extensible Messaging and Presence Protocol (XMPP)

XMPP [11] is the original messaging protocol used in Jabber and Google talk services. It was developed in 1998 for Jabber and adapted by IETF and published as RFC 3920 and RFC 3921 in 2004. It is an open standard for instant messaging, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data [39].

It supports both Client/Server (i.e., request/response) and client/broker (i.e., publish/subscribe) like CoAP. It supports a topic-based publish-subscribe subscription model. XMPP communicates over TCP using port 5222, and port 5223 supports communication over TLS to provide security. Its operation is based on XML data and is designed for real-time applications. One of the XMPP extensions is PubSub. PubSub is a protocol extension for the generic publish-subscribe functionality mentioned in XEP-0060 [40]. The protocol enables XMPP entities to create nodes (topics) at a PubSub service and publish information at those nodes, and event notification is then broadcasted to all entities that have subscribed to the node.

2.6.4 Quick UDP Internet Connections (QUIC)

QUIC is one of the most important developments in the design of modern Internet transport. QUIC is a protocol developed by Google and later standardized by the IETF. It combines transport and cryptographic functions over UDP to offer faster connection establishment, improved loss recovery, and enhanced security compared to traditional TCP/TLS stacks [41, 42]. QUIC is designed to solve long-standing problems with TCP such as head of line blocking, and slow handshake performance, making it a good fit for latency sensitive applications such as video streaming, real-time communication and mobile networking.

QUIC's adoption across major platforms—including Chrome, Firefox, Edge, and large-scale services such as YouTube and Cloudflare—demonstrates its practical impact on Internet performance. QUIC provides a strong foundation for next generation web services and distributed systems through the use of multiplexed streams, encrypted transport headers and connection migration [43]. There is also recent work investigating the applicability of QUIC to non-traditional web traffic, such as IoT and constrained devices and publish-subscribe systems, demonstrating its versatility and future potential [44, 45].

QUIC was designed to address some of the architectural limitations of TCP when used in conjunction with TLS for safe communication. TCP offers in-order delivery , which causes head-of-line blocking when multiplexing several streams over one connection . QUIC solves this by allowing multiple independent transport layer streams (multiplexed streams). This means that loss in one stream does not block other streams [41]. QUIC also embeds TLS 1.3 directly into its handshake, allowing for 0-RTT and 1-RTT secure connection establishment to improve latency on subsequent connections.

The standardization of QUIC by the IETF (published as RFC 9000) formalizes its packet formats, state machine, congestion control, and loss recovery mechanisms [42]. QUIC's encrypted transport headers limit middlebox interference. The explicit packet numbers and ACK ranges in QUIC allow for better loss detection than the sequence based model in TCP. QUIC has been shown to outperform TCP/TLS in heterogeneous networks such as cellular and Wi-Fi where mobility and packet loss are common [46].

QUIC has also gained wider adoption in emerging domains beyond web applications. Matrawy et al. [44] [45] analyze in detail the behavior of QUIC on constrained devices and evaluate its suitability for IoT and MQTT based publish-subscribe systems. These studies demonstrate the promising transport protocol of QUIC for next-generation distributed and embedded systems due to its lower handshake overhead, resilience to in-

termittent connectivity and strong security properties.

2.7 MQTT-SN over ZigBee

In this section, we discuss ZigBee and its security features. We will also discuss how developers might use MQTT-SN over ZigBee. Many developers use ZigBee security to provide a secure communication link between the two end-points of a link. When developers use ZigBee for security, the MQTT-SN message is the payload, and a link key encrypts that, as shown in Figure 2.8. However, research on the vulnerabilities of ZigBee security features shows that relying only on ZigBee features for security is not sufficient [47] [48]. We now discuss the ZigBee protocol in more detail.

2.7.1 ZigBee

ZigBee is a wireless technology standard that provides short-range communication for very low-cost implementation of low-power devices with a low data rate [4] [22]. It is currently used in many IoT applications such as smart homes, medical data collection, industrial control systems and others. Table 2.3 below shows the difference among ZigBee, BLE, Bluetooth and Wi-Fi.

Table 2.3 suggests that ZigBee is suitable for constrained devices communicating over battery. Now we discuss the ZigBee stack and different functions provided by the stack as shown in Figure 2.9. ZigBee stack consists of five layers as follows:

- **Physical Layer:** This layer operates at two separate frequency ranges: 868/915 MHz and 2.4 GHz. The physical layer is responsible for packet generation, packet reception, data transparency, and power management.

- **MAC Layer:** This layer's responsibilities include controlling access to the radio channel, transmitting beacon frames, synchronization, and providing a reliable transmission mechanism. The MAC layer security is based on the IEEE 802.15.4 standard added with AES-CCM only to provide encryption and integrity capabilities.

- **Network Layer:** This layer uses certain fields of its own address space to send messages to different network nodes within the same network (refer to Table 2.3 for network size) [49]. The network layer helps the ZigBee device map the local ZigBee network based on ZigBee's address structure, which is different from the

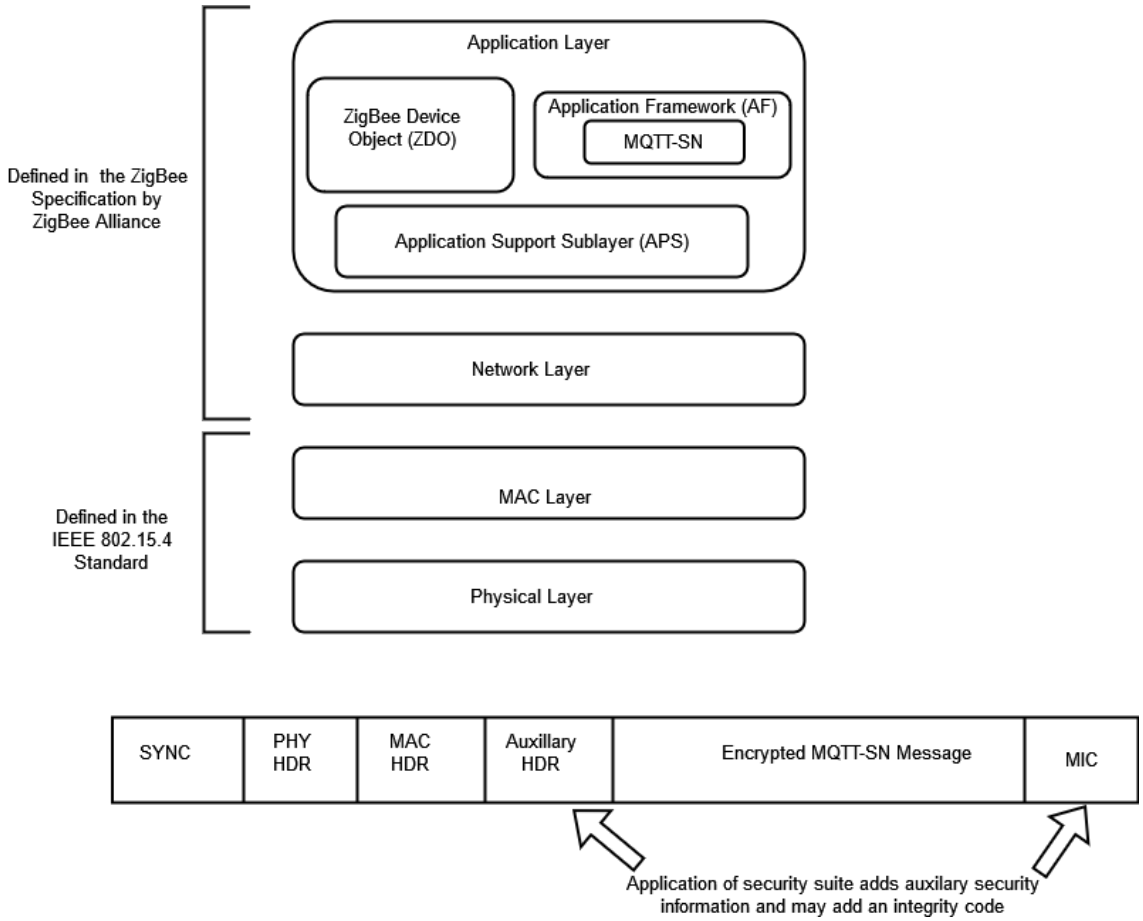


Figure 2.8: MQTT-SN over ZigBee

Table 2.3: Comparison of Link Layer Protocols

Specification	**ZigBee**	**BLE**	**Bluetooth**	**WiFi**
Operating Frequency	868 Mhz, 902-928 Mhz, 2.4 Ghz	2.4 Ghz	2.4 Ghz	2.4 Ghz, 5 Ghz
Power Consumption	Low	Very Low	High	High
Data Transfer Speed	20-250 Kbps	125 Kbps - 2 Mbps	1 Mbps - 3 Mbps	300 Mbps
Network Range	75-100 m (indoor), upto 300 m (outdoor)	1 to 100 m	1 to 10+ m	1 to 30+ m
Network Size	64000 nodes	32000 nodes	7 nodes	250 nodes

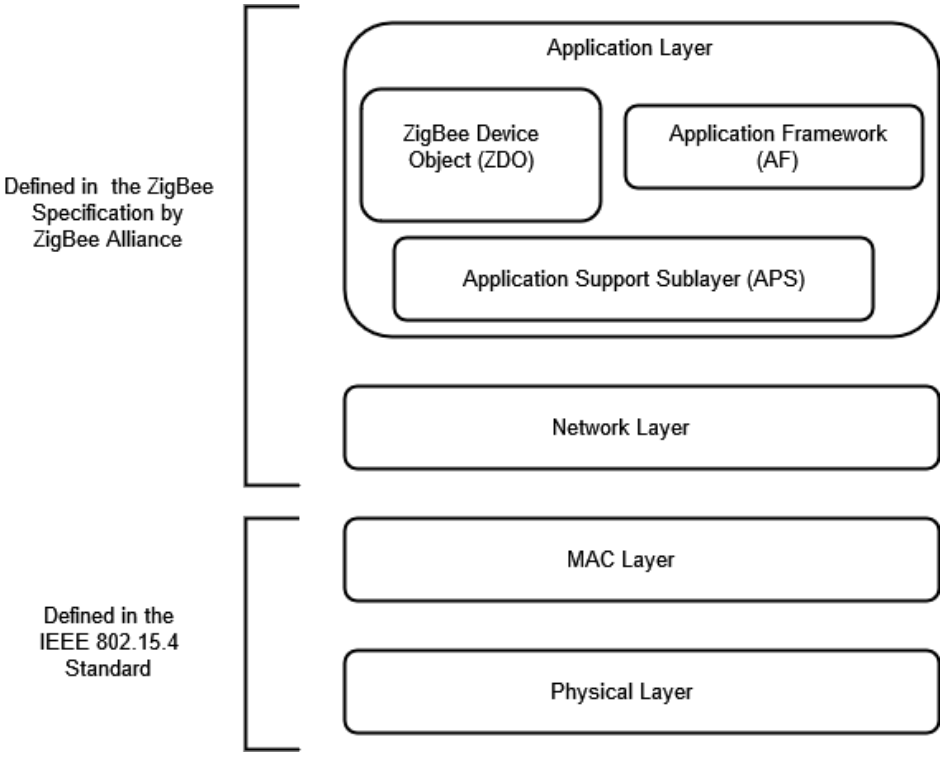


Figure 2.9: ZigBee Stack [4]

standard IP address. In addition, this layer adds routing capabilities that allow radio frequency data packets to traverse multiple devices (multiple hops) to route data from source to destination (peer to peer).

- **Application Support Sublayer (APS):** This layer defines various addressing objects including profiles, clusters, and endpoints. The APS layer allows frame security to be based on link keys or the network key. The APS layer is responsible for the processing steps needed to securely transmit outgoing frames, securely receive incoming frames, and securely establish and manage cryptographic keys. The ZDO controls the management of cryptographic keys by issuing instructions (i.e., primitives) to the APS layer.

- **Application Framework (AF):** AF is the environment or block where applications objects are hosted.

- **ZigBee Device Object (ZDO):** The ZDO is part of the application layer that provides device and service discovery features and advanced network management capabilities. It provides an interface between the application objects, the device profile, and the APS. In addition, the ZDO is responsible for initializing the APS, network, and security, which is described under the next heading below.

The components involved in the ZigBee network are as follows:

- **ZigBee End Device:** is usually a sensor node device that monitors and collects environment data. End devices are low powered or battery operated unlike routers or coordinators (discussed below).

- **ZigBee Trust Center:** It is an application that runs on the device trusted by other devices within the ZigBee network to distribute keys for network and end-

to-end application configuration management. The Trust center could be the coordinator device or a device designated as the Trust center by the ZigBee Network coordinator.

- **ZigBee Coordinator:** is a device responsible for establishing, executing, and managing the overall ZigBee network. In addition, it is responsible for configuring the security level of the network and configuring the address of the Trust Center (the default value of this address is the ZigBee coordinator's address. Otherwise, the ZigBee coordinator may designate an alternate Trust Center).

2.7.1.1 ZigBee Security

There are three different types of keys involved in ZigBee security to provide device authentication, data integrity, data confidentiality from link to link communication. These all three are symmetric keys (each of length 128-bit) used in the ZigBee standard [50] [51].

- **Master Key:** This is the key from which link keys are established. It keeps the link key exchanged between two nodes in the Symmetric-Key Key Establishment protocol (SKKE) confidential. It must be obtained by secure means (pre-installation or key transport or user-entered data such as PIN or password).

- **Network Key:** This is the key used at the network level and known by all ZigBee nodes within a local network. The network key is used in groups of more than two nodes within a network. The trust center generates the network key and distributes it to all the devices within a local network (refer to Table 2.3 for network size). A device on the network acquires a network key via key-transport (Master key is used to protect transported network keys) or pre-installation.

- **Link Key:** The Application level manages these keys. This key encrypts point-to-point communication at the application level. This key is only shared between two ZigBee network nodes and is different for each pair of nodes. A device acquires link keys either via key-transport, key-establishment (based on the "master" key and other network parameters), or pre-installation (for example, during factory installation). The link keys related to the Trust Center are pre-configured using an out-of-band method, for instance, QR code in the packaging, whereas the link keys between nodes are generated by the Trust Center and encrypted with the network key before sending it to the node.

2.7.1.2 Key Management

There are three different types of key management schemes [50] [51]:

- **Pre-Installation:** The manufacturer installs the key into the device itself before deployment. Then, the user can select one of the installed keys in the device (in devices where more than one key is preinstalled).

- **Key Transport:** The end device requests the Trust Centre for a key to be sent to it. The method is valid for requesting any of the three types of key (as explained above) in commercial mode, whereas, in residential mode, the Trust Centre holds only the network key.

- **Key Establishment:** This is a method of generating link keys based on the master key. It is based on the SKKE (Symmetric-Key Key Establishment) protocol [52]. The devices involved in communication must have the master key, which may have been obtained through pre-installation or key transport or user input.

2.8 MQTT-SN over BLE

MQTT-SN also communicates over BLE, and developers rely on BLE security features for establishing a secure communication channel at the link-layer, such as session key establishment. Therefore, we need to become familiar with the BLE security features. Here in this section, we discuss BLE and its pairing methods.

2.8.1 Bluetooth Low Energy (BLE)

BLE [53][54] is one of the most commonly used wireless standards for IoT devices. General Access Profile (GAP) is the layer of the BLE stack used to find the network topology of the BLE system. In BLE, when two devices are connected, they start a pairing process during which they share some information to establish a secure encrypted connection. BLE devices come with 4.0, 4.1, and 4.2 versions of Bluetooth. BLE uses the AES-CCM [55] algorithm for encrypting data. The method used by BLE devices for sharing the symmetric key is known as pairing or association. Information received during the pairing process is stored in the device so that the process does not have to repeat whenever devices try to connect next [56] [57].

In BLE, the devices exchange a Temporary Key (TK), which generates a Short-Term Key (STK) used to encrypt the connection. There are four well-known pairing methods used in BLE for sharing a temporary key between devices [58]:

1. Just Works: For BLE 4.0 and 4.1 devices, the temporary key (TK), which is a hard-coded key (size 16 bit) across all BLE devices of a particular type, is used to generate a short-term key (STK). Therefore, it is easy for an attacker to use brute force to get the STK. In addition, there is no method for validating devices participating in the process. For devices that support BLE 4.2, ECDH is used for

key establishment and to perform the key confirmation using a session key. This process is still vulnerable to a man in the middle (MITM) attack.

2. Out of Band (OOB) Pairing: The TK is shared between two devices using an out of band channel like wireless technology, such as NFC. The security of this method depends on the out-of-band channel.

3. Passkey: During this method, the TK is a random 6-digit number shared between the two devices by the user. Sharing TK is performed by reading the LCD screen on one device that displays the number and entering the same number on another device using a keypad. For BLE 4.2 devices, the passkey (6-digit number) is entered in each device by the user. Both devices exchange public keys and use them with the passkey and a 128-bit nonce to validate the Bluetooth connection.

4. Numeric Comparison: This method is similar to Just Works but has an extra step. Both devices generate a six-digit random value using both nonces, generated using the AES-CMAC [59] function. Both devices display it for the user to verify and give approval to the connection in case of a match. This method is only supported on BLE 4.2 devices.

The OOB, passkey, and numeric comparison methods require extra resources like the screen, keypad, and NFC. Just Works is simple but susceptible to MITM attacks. BLE 4.2 is used with 32-bit processors. Class IV and V devices (e.g., August and Kevo smartlock) use Bluetooth 4.0 for communication.

2.9 Security Vulnerabilities in MQTT-SN

2.9.1 Lack of Authentication and Weak Access Control

Client Authentication: By default, MQTT-SN does not enforce authentication of clients or brokers beyond an optional user/password in the CONNECT message (similar to MQTT). If no credentials or only default credentials are used – a common scenario in IoT – unauthorized devices can connect to the broker and publish/subscribe freely. Even with password authentication, a determined attacker who compromises a legitimate node (e.g. a malware-infected sensor) can bypass this control. MQTT-SN's client identifier (ClientId), which must be unique per client, is another weak point: the spec allows ClientIds up to 23 characters, but if an attacker can guess or learn a valid ClientId, they can impersonate that client. The protocol specifies that if a new client connects with an existing ClientId, it will forcefully disconnect the original client. This behavior, also present in MQTT, becomes a vulnerability when misused: an attacker can intentionally register as known clients to kick them offline (a "disconnect wave" attack). In resource-constrained IoT deployments where ClientIds might be derived from device serials or assigned sequentially, guessing valid IDs is feasible. Compared to standard MQTT, the impact of ClientId hijacking in MQTT-SN is similar (loss of device connectivity), but the likelihood can be higher due to simpler ID patterns and lack of TLS client certificates that some MQTT systems employ [60] [61] [62].

Authorization and Topic Access: MQTT-SN uses the same topic-based publish/subscribe model as MQTT, and it supports MQTT's wildcard subscriptions (e.g. # to subscribe to all topics). If brokers are misconfigured to allow anonymous or over-privileged subscriptions, an attacker can simply subscribe to # or to a high-level wildcard

(like house #) and receive all messages on the broker. This is effectively an information leakage vulnerability: sensitive data (medical readings, home sensor data, machine telemetry, etc.) can be siphoned off without needing a man-in-the-middle position. In MQTT, best practices restrict such wildcards to authorized roles, but IoT deployments often lack proper access control. MQTT-SN adds another wrinkle: topics may be referred to by their 2-byte ID or by a short name (two-character string). Attackers can exploit this by iterating through possible Topic IDs or short names to find valid topics. This brute-force scanning is easier in MQTT-SN than guessing arbitrary MQTT topic strings. If successful, the attacker subscribes and eavesdrops on all traffic (no need to perform complex network MITM). In summary, MQTT-SN's lack of strong authentication and topic-based access control makes unauthorized access a serious concern, especially in sensitive domains like healthcare where data confidentiality is paramount.

2.9.2 Absence of Built-in Encryption and Integrity Protection

MQTT-SN does not provide encryption or message integrity by default, leaving it to underlying layers or external solutions. In practice, many MQTT-SN networks operate entirely in plaintext. This exposes two classic vulnerabilities:

Eavesdropping: Attackers within radio or network range can capture MQTT-SN packets and read their contents (patient vital signs, home sensor readings, etc.). In healthcare IoT, this is a direct privacy violation, potentially breaching regulations if personal health information is transmitted unencrypted. Similarly, eavesdropping on smart home or vehicle telemetry could reveal sensitive information. As reported in one study, MQTT-SN communications are often unencrypted by default, enabling attackers to easily sniff data. The study demonstrated a sniffing attack where an adversary first compromises the rout-

ing layer (e.g. launching a rank attack on an IPv6 RPL mesh to position themselves in between nodes) and then intercepts all MQTT-SN messages, which were readable in cleartext. Without confidentiality, the system's secrecy is entirely lost.

Message Tampering: Lack of cryptographic integrity (no signatures or message authentication codes) means attackers can not only read but also alter MQTT-SN messages in transit if they can inject packets. A common scenario is a man-in-the-middle (MITM) attack at the network layer (as above, achieved by compromising routing or link layer in IoT). Once positioned between a device and the broker, an attacker can modify sensor readings or commands. For example, in a medical IoT setting, an attacker could falsify a patient's vital data or send unauthorized actuator commands (e.g., to a drug infusion pump) with potentially dangerous consequences. The MQTT-SN "sniffing" MITM attack mentioned earlier allows exactly this – it compromises integrity and even availability by altering or dropping messages. Notably, performing a MITM on MQTT-SN often requires exploiting another layer (since MQTT-SN by itself is at application layer), but given many IoT stacks (6LoWPAN, mesh routing, etc.) have known weaknesses, this is a realistic threat. In contrast, standard MQTT often runs over TLS, which provides built-in encryption/integrity; MQTT-SN deployments lacking an equivalent protection are much more vulnerable.

Comparing to MQTT: The core issue of optional security is present in both MQTT and MQTT-SN – neither mandates encryption or authentication. However, MQTT's typical usage on TCP/IP has allowed the widespread adoption of TLS as a solution (brokers commonly support MQTT over TLS with client certs or username/password). MQTT-SN's UDP foundation and target of very constrained devices have delayed similar

widespread security adoption. TLS/DTLS can be too heavy for battery-powered sensors, so many MQTT-SN systems forego it, resulting in plaintext communication. Researchers have noted that securing MQTT-SN is even more challenging because "when there are constrained devices, it is more difficult to implement an encryption algorithm to protect the communication". Thus, MQTT-SN often lags behind MQTT in deploying encryption, making confidentiality and integrity vulnerabilities more acute.

2.9.3 Data Injection and Publication Hijacking

Closely related to integrity issues is the threat of malicious data injection [63]. An attacker who gains publish access (via weak authentication or by spoofing packets) can inject false or malformed data into MQTT-SN topics. This "trash injection" attack was experimentally demonstrated, where an attacker publishes bogus data (e.g., non-numeric strings when an application expects numeric sensor readings) to disrupt the application's logic. In an IoT system, such injected messages can lead to incorrect decisions or system malfunctions – for instance, injecting fake high temperature readings into an industrial sensor network might trigger unnecessary cooling, or false alerts in a smart home security system could cause alarm flooding. If the attacker understands the application semantics, they could craft data to cause specific harmful outcomes. Even without that knowledge, spamming random "trash" data on important topics can overwhelm or confuse client applications. MQTT-SN's design makes injection easier in some respects. Because it operates over UDP, an attacker can spoof the source IP or pretend to be another client more easily than with MQTT/TCP (which requires a handshake). In other words, MQTT-SN's lack of session state means an attacker can forge a single UDP packet that looks like a legitimate PUBLISH from a client and the broker may accept it if not otherwise secured. One study explicitly notes that injecting malicious

packets is "even easier... against MQTT-SN than against MQTT, because the use of UDP allows an IP spoofing attack to be carried out easily.". Standard MQTT is not immune to injection (an attacker that steals credentials or session tokens could publish false data too), but the barrier is higher due to TCP connection management and TLS. In MQTT-SN, if no authentication is enforced, injection can be as simple as knowing or guessing a topic ID and sending a UDP packet with that ID and fake data. Another form of publication hijacking peculiar to MQTT-SN arises from its client session behavior. As described in the "spoofing via ClientId" attack, an attacker can exploit a client that does not use CleanSession. Suppose a legitimate client relies on the broker to remember its subscriptions (as is common for sensors that disconnect and reconnect). An attacker can connect with that client's ID and set CleanSession=true – causing the broker to wipe the stored subscriptions for that ID – then disconnect. When the real device comes back or remains connected, it finds its subscriptions gone or replaced. The attacker could even resubscribe that ID to topics of the attacker's choosing before disconnecting, effectively misrouting the victim's data (the victim might start receiving irrelevant or malicious data, or might stop receiving critical updates). This is a complex attack, but it highlights how MQTT-SN's session management can be abused to silently alter data flows. MQTT has similar session options, but this kind of attack is more likely on MQTT-SN where devices routinely sleep and rely on stored subscriptions. A misconfiguration or lack of authentication allows an attacker to "poison" the broker's session state for a client [63].

2.9.4 Denial-of-Service (DoS) and Resource Exhaustion

DoS attacks are a major threat in IoT contexts, and both MQTT and MQTT-SN have known vectors [64]. MQTT- SN, due to its broadcast/multicast-friendly design and UDP

transport, introduces some unique amplification opportunities:

Reflection/Amplification Attack: Researchers discovered MQTT-SN can be exploited for Distributed Reflection DoS (DRDoS). In a reflection attack, an attacker sends requests with a spoofed source IP (the victim's address) to many MQTT-SN brokers, causing them to send large responses to the victim, overwhelming it. MQTT-SN's publish/subscribe topology can amplify traffic: a single PUBLISH from the attacker (with victim's IP as the apparent source) could cause the broker to forward that message to many subscribers. Sochor et al. [64] demonstrated using MQTT-SN brokers as reflectors by taking advantage of the fact that a broker will send a publish to all clients subscribed to a topic. By carefully crafting packets and exploiting UDP's statelessness, an attacker could induce MQTT-SN networks to flood a target with traffic. Standard MQTT, running on TCP, is less prone to this specific reflection scenario because TCP makes spoofing harder (though application-layer amplification via MQTT is possible in other ways). The MQTT-SN DRDoS vector is a serious concern for public or large-scale broker deployments.

Client Exhaustion ("Disconnect Wave"): The earlier-described ClientId hijack not only disconnects legitimate devices but can be used repeatedly to churn the network. An attacker can iterate through known or guessed ClientIds, continually kicking devices offline (they may reconnect, only to be kicked again). This "disconnect wave" effectively denies service to a large number of nodes in succession. The attack was shown to generate a flood of CONNECT packets and CONNACK responses (no normal data flow), disrupting normal operations. In MQTT over TCP, a similar attack would require actually establishing many connections (which is harder and more detectable), whereas in MQTT-SN these are lightweight UDP interactions.

Publish Flooding (Congestion attack): A malicious MQTT-SN client can flood a topic with an extreme volume of messages (or very large messages, up to the protocol's limit) to overwhelm the broker and the network. Because the broker forwards each publish to all subscribers, the load is multiplied by the number of subscribers on that topic. For example, an attacker in an IIoT deployment could target a commonly-used topic (e.g., a sensor data feed) and publish incessantly without pauses ("messages without delay with the maximum allowed size"), causing the broker's buffers to overflow and legitimate messages to be dropped. Experimentally, such an attack caused the broker to stop sending updates for other topics, effectively starving parts of the system. The overall traffic volume and device battery usage also spiked significantly. While MQTT over TCP also allows flooding, the use of small, constrained networks in MQTT-SN means even a modest flood can cause congestion and high packet loss in a low-power mesh. The broker in MQTT-SN might also be running on limited hardware (e.g., an edge gateway), making it easier to exhaust.

Sleep Disruption ("Wake-Up" attack): MQTT-SN introduces a special vulnerability through its sleep mode mechanism. Sensors often disconnect (sleep) for long durations to save energy; MQTT-SN brokers queue messages for them and the device periodically wakes (sending a PINGREQ) to receive pending messages. An attacker can abuse this by continually sending fake wake-up signals for sleeping clients. In the documented 'Wake Up Wave' attack, the attacker iterated on all client IDs, sending PINGREQ packets to the broker as if from those clients, or otherwise causing the broker to think clients are awake. This prevents devices from staying in sleep mode: The broker keeps them 'awake' by sending messages or expecting acknowledgments. The result is that battery-operated

devices are forced to expend energy and the wireless medium is occupied with unnecessary traffic. In a healthcare scenario, for example, a wearable health monitor might be prematurely drained of battery by such an attack. This MQTT-SN-specific DoS has no equivalent in MQTT (since MQTT doesn't have a standardized sleep mode). It highlights how a feature introduced for efficiency can turn into a liability if not safeguarded.

Overall, MQTT-SN is susceptible to various DoS vectors ranging from brute-force resource exhaustion to clever misuse of protocol features. Many of these (reflection, spoofing, ping flooding) are facilitated by UDP's lack of handshake and MQTT-SN's broker behavior. Standard MQTT shares some DoS issues (e.g., flooding, client ID conflicts), but the barrier to exploit is often higher, and MQTT's typical reliance on TCP/TLS provides some built-in mitigation (like connection limits, TLS client puzzles, etc.). MQTT-SN deployments must therefore be designed with additional precautions to handle these threats

2.10 V2V Communication and its Security

V2V communication involves the transmission of information between vehicles in order to improve road safety, efficiency, and convenience. V2V communication facilitates the exchange of data between cars, including details like speed, position, acceleration, and direction. This enables surrounding vehicles to proactively anticipate and respond to potential dangers immediately. Although V2V communication shows potential to enhance road safety and traffic control, it is crucial to prioritize the security and privacy of these conversations to prevent malicious attacks and unauthorized access. It is primarily focused on safeguarding the integrity, confidentiality, and authenticity of the data communication. This involves guaranteeing that communications transmitted between

vehicles are impervious to interception, tampering, or unauthorized imitation, therefore upholding trust and dependability in the V2V communication system.

V2V communication commonly uses wireless communication protocols like Dedicated Short Range Communications (DSRC) or Cellular Vehicle-to-Everything (C-V2X) and satellite communication. These protocols establish the criteria and details for exchanging data between automobiles and infrastructure components. These protocols incorporate security features to protect V2V communications from different security risks. However, as far as we know, very little research is available related to the security of vehicles sending distress signals. Alongside the available research, there is no proper infrastructure or guidelines followed in determining the complete security requirements and methodology to address and determine the threats to V2V communication. Therefore, this gap shows the need for our research.

2.10.1 V2V Distress Signal Communication Methods

The purpose of the distress signal in a vehicle is to provide immediate assistance in case of an emergency. When the button is pressed, it sends a distress signal to a response center or emergency services, indicating the vehicle's location and providing other critical information. This allows for a quick response in an accident, breakdown, or other emergency situation. Below are different methods of sending emergency distress signals from vehicle [65]:

1. Save Our Souls (SOS): The purpose of the SOS button in a vehicle is to provide immediate assistance in case of an emergency.

2. Onboard Emergency Call System (eCall): An in-vehicle system that automatically sends an emergency distress signal in the event of a crash or collision.

3. Personal Emergency Response System (PERS): A wearable device that can be triggered to send an emergency distress signal in case of a medical emergency.

4. Mobile Phone App: An app installed on a mobile phone that can be used to send an emergency distress signal in case of an accident or other emergency.

5. Satellite-based Tracking System: A system that uses satellite technology to send an emergency distress signal in case of an accident or breakdown.

6. Roadside Assistance System: A system that provides emergency assistance when a vehicle is involved in a breakdown, accident, or other emergency.

7. Integrated Telematics System: A system that integrates various communication and navigation technologies to provide real-time information and emergency assistance.

2.11 Continous Glucose Monitoring

CGM devices are typically utilized for people with diabetes, as advised by their healthcare professionals. These devices monitor and regulate the patient's insulin levels, automatically administering insulin as their healthcare professional prescribes to control blood sugar levels [66]. Advancements in insulin monitoring system technology can automate the insulin injection process for patients, addressing their healthcare needs and improving their daily activities. In cases of local and distant attacks, these systems may fail and create life-threatening situations for diabetic patients. Inhibiting these devices from their standard operational efficacy may hinder diabetic patient's ability to manage their blood glucose levels, potentially leading to hypoglycemia or hyperglycemia, which could culminate in a diabetic coma or, in extreme cases, mortality. Every continuous glucose

monitor (CGM) comprises three primary components: a sensor, a transmitter, and a monitor.

- Sensor: A CGM user implants a small sensor under the skin, usually on the abdomen or arm, using a slender tube to measure glucose levels in the interstitial fluid. This fluid reflects blood glucose levels. Most sensors are waterproof and secured with an adhesive patch, but they must be changed every 10 to 14 days.
- Transmitter: The sensor connects to a transmitter that wirelessly broadcasts blood glucose data to a monitor. This setup requires synchronization for accurate readings. Most systems provide near-real-time data, though some may delay 10 to 12 minutes.
- Monitor: The sensor connects to a transmitter that wirelessly broadcasts blood glucose data to a monitor, which displays readings at regular intervals. Some continuous glucose monitors (CGMs) have dedicated displays, while others work with smartphone apps. CGMs store data and can send it to physicians, enabling better collaboration in diabetes management.

2.12 Smart Lock

A smart lock is an electronic locking device that connects to the internet and can be used instead of or in addition to a regular mechanical lock. The main reason for it is to let you open a door without having to use a physical key. Smart locks, on the other hand, use digital authentication methods like mobile apps, Bluetooth, Wi-Fi, PIN codes, RFID cards, and even biometric inputs like fingerprints. How a Smart Lock Works

Smart locks use both hardware (the lock mechanism) and software (the control system) to control who can get in. This is how things work:

- **Authentication:** The smart lock checks the person's identity when they try to unlock the door with a phone, PIN, fingerprint, or proximity device. It does this through its built-in sensors or wireless communication.
- **Communication:** The lock can connect to a smartphone or home network through Bluetooth, Wi-Fi, Zigbee, or Z-Wave. This link allows you to control devices remotely, check their status, and set them up to run automatically.
- **Actuation:** After the lock has been verified, its internal motor turns the deadbolt or latch, locking or unlocking the door.
- **Monitoring and Alerts:** A lot of smart locks send alerts when the lock is opened or closed, when someone tries to open it but fails, or when the battery is running low. Some even keep a log of who has access for security purposes.

Smart locks are a key part of modern smart home ecosystems because they combine convenience with better access control.

2.13 Terminologies

We use the following terminologies in this document:

- **Security Model:** It is a model used to identify and implement security policies and their relationship to system behaviour [67]. The primary aim of the security model is to provide the necessary level of understanding for implementing essential security requirements with the help of threat modelling, which includes attackers' capabilities and different threats within the scope of the system.
- **Security Requirement:** It mentions what objects need protecting and why they need protecting. In simple terms, security requirements should also cover who can

do what, when [68]. Also, security requirements help us to state what is allowed or prohibited from whom.

- **Security Architecture:** It is a security design for system components that address the security requirements of a specific scenario or environment [69]. It contains the system components and how they securely communicate with each other to meet security requirements.

- **Mosquitto:** It is an open-source message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1 [70]. Mosquitto repository also includes a C and C++ client library and the Mosquitto publisher and Mosquitto subscriber utilities for publishing and subscribing.

- **Paho:** It provides open-source, mainly client-side, implementations of MQTT and MQTT-SN in various programming languages [71]. In addition, it provides the library and software utilities for MQTT-SN clients and the MQTT-SN gateway. We will focus on C and C++ client and gateway libraries and utilities for Paho.

- **Constraint Device:** These are devices constrained in CPU, memory, and energy, such as wearable devices and low-cost environmental sensors. We will focus on battery-powered devices with several years of battery life and communicate using a low-power wireless link-layer network, such as ZigBee or BLE. These devices are heavily resource-constrained, with kilobytes of memory and can be 8-bit, 16-bit, or 32-bit. A few examples are Philips Hue Smart Bulb communicate using ZigBee and use an 8-bit processor [22]. In Fitbit Flex and Fitbit One, the processor is 32-bit[72]. Some smart locks use 32-bit processors [73] [74].

These methods use different technologies for communication which we will use for comparison with our work:

1. Cellular: A cellular communication technology that is widely used for sending distress signals from vehicles, using either voice or text messaging. There are different types of cellular technologies, like GSM (Global System for Mobile Communications), 3G, LTE (Long-Term Evolution), SMS (Short Message Service).

2. Satellite: A satellite-based navigation system that can be used to determine the location of a vehicle and send an emergency distress signal. There are different types of satellite technologies, like two-directional (Orbcomm, Iridium, Euteltrack, etc.) and uplink one-directional (Argos).

3. Advanced Emergency Braking System (AEBS): A system that uses radar or other sensors to detect a potential collision and send an emergency distress signal to alert other vehicles and/or emergency services.

4. Dedicated Short Range Communications (DSRC): A wireless communication technology that enables direct communication between vehicles and roadside infrastructure.

Chapter 3

Related Works

3.1 Security Requirements

The related work with respect to the security requirements are discussed here:

Michael et al. [75] focused on business profile modelling to express security requirements like trust, confidentiality and integrity on an abstract level. They use a model-driven approach to define security goals and their related requirements. However, there is no proper description of threats, their actors and the risks involved for the designers to make a proper informed decision.

Zhao et al. [76] focus on the overall architecture of Solar Insecticidal Lamsps Internet of Things(SIL-IoT) and share the potential security scenarios and attacks and prospective solutions. However, they are missing a properly designed threat model to identify the security requirements more clearly and attackers and vulnerable points in all channels of communication.

Myagmar et al. [77] demonstrate the significance of threat modelling in establishing security requirements. Furthermore, they emphasize that no system can be completely safeguarded. Consequently, the designer must carefully consider all potential risks and

determine which ones to prioritize addressing, while also accepting the risk associated with some aspects of the system. Nevertheless, all the conducted case studies only focus on software tools that do not cater to IoT devices.

Firesmith [78] asserts that the majority of designers and scholars tend to focus on defining architecture and design limitations, rather than addressing genuine security issues. The author delineated many categories of security prerequisites and disseminated the guidelines and protocols to enable requirements engineers to accurately specify them without impeding the security and functionality of the system. The author further emphasizes that there are security criteria that can be reused [79] and established a framework for precisely outlining the security requirements. Nevertheless, the above actions are insufficient in addressing the risks associated with the IoT ecosystem.

Mellado et al. [68] explain the SREP is a methodical and organized procedure that seeks to identify, assess, and define security requirements for a system or software application. The Security Requirement Engineering (SRE) process incorporates security considerations into the system design from its inception, thereby reducing the likelihood of vulnerabilities and guaranteeing the confidentiality, integrity, and availability of the system's assets. The following is a background summary of the typical steps involved in the Security Requirement Engineering process: initiation, elicitation, analysis, specification, validation, verification, and maintenance. The SRE process is iterative and should be integrated into the overall software development life cycle to achieve a secure and resilient system. The SREP design here is not focused on the IoT ecosystem.

Mead et. al [80] define the SQUARE process as a methodology that specifically targets the identification and definition of security quality requirements for software systems. The software development process is enhanced by a systematic method that integrates security considerations. The objective is to guarantee that the end product

fulfills the intended security objectives. Here is a background summary of the SQUARE method: problem definition, elicitation, classification, analysis, specification, validation, verification and documentation. The SQUARE method is iterative and can be integrated into existing software development processes, such as the waterfall or agile methodologies.

The system explained by Firesmith, SQUARE and SREP, do not cover the steps and guidelines to address the security requirements of the IoT ecosystem and the risk involved with each requirement.

Huang et al. [81] construct a resilient security framework and examine the security prerequisites by scrutinizing several scenarios such as body IoT, home IoT, and hotel IoT. In addition, a compelling survey is carried out by a group of 30 adults to ascertain the hierarchy of various security criteria, namely confidentiality, integrity, availability, and access control.

Rahman et al. [82] present a comprehensive IoT security framework consisting of four layers and analyze various security concerns. Furthermore, the authors examine the security prerequisites for each component (namely, IoT sensor node, base station, network, cloud) in their cloud scenario.

Lee et al. [83] discuss the security risks and requirements associated with the IoT open platform and networked ecosystem. They emphasize the significance of ensuring security protocols are compatible with existing network infrastructure and can operate effectively in diverse network environments.

Abomhara et al. [84] examine the vision, security threats, and problems of IoT. The major security criteria chosen are authentication, secrecy, and data integrity. Alqassem et al. [85] classify security requirements into access control, data integrity, contextual integrity, intrusion detection, and non-repudiation.

Kim et al. [86] categorize the various types of IoT devices and examine the security risks associated with each type. Kim's research focuses on several essential security requirements, such as the creation of efficient protocols and encryption algorithms, ensuring secure communication, safeguarding data, enhancing physical security, establishing device identification and permission systems, and implementing device monitoring and control mechanisms.

Oh et al. [87] defined the basic security requirements by using three IoT characteristics and six elements of IoT. They provide a comparison with other researchers working on IoT security requirements. However, they have not considered the threat model and designed a proper framework for defining security requirements.

3.2 Security of Publish-Subscribe Systems

Liu et al. [3] surveyed eight different publish-subscribe systems and provided a taxonomy for comparing them based on category (i.e., content-based, subject-based, hybrid), system architecture (i.e., client-server, peer-to-peer), matching algorithm, multicasting algorithm and language support. In addition, the authors tried to define the general security goals for content-based publish-subscribe systems, but it was not used in their taxonomy. Gryphon [88] is the only mentioned publish-subscribe system that provides client authentication and access control mechanisms.

Pelz [2] introduced a new method of measuring the quality of the publish-subscribe system, i.e., quality of robustness (QoR) (i.e., number of nodes allowed to fail without data loss). The author tested the system with zero node failure to all nodes failure. The author found that management node (i.e., broker) failure was the worst case for QoR. Therefore, the author suggested that the IoT system designer always ensure that the management node never fails so that there is less damage to clients.

Wang et al. [20] tried to define the general security requirements and issues present in the content-based publish-subscribe system. The authors listed the security requirements as confidentiality, integrity and availability and sub-divided them. The authors also mentioned which security requirements after implementation would take away the benefits of the publish-subscribe system. Finally, the authors provided an overview of security architecture for the large-scale publish-subscribe system.

Lagutin et al. [89] introduced a security design for a publish-subscribe network. They used elliptic curve cryptography and certificates to address the integrity, availability and scalability in forwarding (i.e., publisher/subscriber) and rendezvous (i.e., broker) planes. In addition, they implemented filters based on security roles such as namespace owner, scope owner, publisher, and data source in the pub/sub architecture to avoid bogus subscription requests. In their design, publishers could not publish under the topic until at least one subscription request was present for that topic.

Ammar et al. [90] surveyed the security aspects of eight different IoT frameworks that would help develop industrial and consumer-based IoT applications. Only four of the IoT platforms support pub/sub messaging protocols (i.e., AWS IoT by Amazon, Kura by Eclipse, ARM Mbed and Azure IoT by Microsoft), and the security of these platforms depends on symmetric cryptography algorithms or TLS. The authors also mentioned threats, which are mitigated by the available security features.

As far as we know, no literature thoroughly discusses the process of determining security requirements and the associated risks for IoT devices and services that use a publish-subscribe architecture. The primary objective of solution providers is to identify and mitigate potential threats. However, it is widely acknowledged that achieving absolute security is an unattainable goal. Hence, it is imperative to accurately define security requirements and assess the corresponding risks to make well-informed decisions

regarding which security requirements necessitate immediate attention. The main objective of this thesis is to establish security requirements for two specific scenarios using the modified SREP and SQUARE methodology.

3.3 Comparison of Messaging Protocols

Tschofenig et al. [91] surveyed a selection of IoT security protocols standardized by the IETF and discussed the vulnerabilities. With the help of the suggestions provided, designers may be better able to design a secure IoT product.

Elhadi et al. [92] presented the operating models for different communication protocols and suggested choosing the protocol based on the requirements of IoT applications.

Keitzmann et al. [93] analyzed IoT use cases based on the single-hop and multi-hop scenario and provided a comparative analysis on NDN (Named Data Networking), CoAP, and MQTT-SN. Security is not the primary focus for the authors. Instead, they focused on performance and energy consumption. The authors concluded that CoAP and MQTT-SN are faster with less overhead in the single-hop scenario, but NDN shows better performance and robustness in multi-hop scenarios. NDN [93] is designed to network devices by naming data. Named-based architectures like NDN resolve some performance issues related to IP-based routing.

Santo et al. [94] surveyed six communication protocols (i.e., RFID, NFC, 6LoWPAN, 6TiSCH, DTLS, CoAP, and MQTT) and vulnerabilities in these protocols, and how these can be used to exploit these protocols.

Naik [33] provided a comparative analysis of four application layer messaging protocols, i.e., MQTT, CoAP, AMQP and XMPP. The author's analysis includes multiple parameters such as bandwidth, latency, message size, overhead, power consumption, security and others. This analysis helps system designers to choose a protocol based

on their requirements and suitability. Chen at al. [95] compared MQTT, CoAP, DDS (Data Distribution Service) and custom-UDP-based protocol for smart healthcare based on bandwidth, latency and packet loss. The authors concluded that TCP-based protocols like DDS and MQTT have zero packet loss under lousy network conditions; however, DDS performs better than MQTT in terms of data latency and reliability. Similarly, Herrero [96] compared MQTT-SN, CoAP and RTP (Real-Time Protocol) for real-time communication and concluded that end-to-end CoAP has less packet loss than MQTT-SN and RTP.

Imane et al. [32] discussed previous work in smart healthcare alongside their strengths and weaknesses. The authors mentioned different published research that did not consider smart healthcare's security aspect. The authors suggested that MQTT and CoAP would be a better choice based on application layer protocol requirements. They also discussed the threat model and security requirements for smart healthcare. Similarly, Amaran et al. [97] compared CoAP and MQTT-SN for robotics applications based on header size, reliability, duplication, resource locator and communication mode. The authors concluded that MQTT-SN has 30% better transmission time than CoAP. Likewise Guamán et al. [98] performed experiments using MQTT and CoAP and concluded that MQTT is better used in cloud services and real-time applications due to less delay and jitter. Thangavel et al. [99] used middleware to compare CoAP and MQTT's performance evaluation based on bandwidth, message size and concluded that MQTT has less delay and smaller message size than CoAP.

Granjal et al. [100] surveyed existing protocols used in all TCP/IP layers (i.e., Application (CoAP), Network (IPv6, ROLL RPL, 6LoWPAN), MAC (IEEE 802.15.4, IEEE 802.15.4e), PHY (IEEE 802.15.4)). The authors suggested cryptographic algorithms (e.g., AES) available to handle security requirements and protect end-to-end Internet

communication, including for sensor devices. The authors also suggested open research areas for these communication protocols.

3.4 MQTT Vulnerabilities and Mitigations

Yassein et al. [101] surveyed published research done on MQTT and mentioned their methodologies, strengths and weakness. However, most did not consider the security aspects of MQTT. On the other hand, Cheng et al. [102] reviewed existing solutions used to secure communication channels and noted MQTT protocol vulnerabilities and limitations. They suggested a Value-to-Keyed-Hash Message Authentication Code (Value-to-HMAC) mapping, which depends on the Blake cipher. It offers better performance than AES symmetric algorithm to provide confidentiality and integrity.

Jia et al. [18] performed security analysis on MQTT's four features: will message, topics, clientID, and session management; three of these are part of CONNECT message type. In Connect message, vulnerabilities are demonstrated using ClientID, cleanSession Flag, username/password, and Will Topic/Will Message. According to the authors, IoT cloud platform providers explicitly implement security to protect data. However, the authors found that the security features implementations have shortcomings that could easily be exploited in an adversarial environment. The authors defined four attacks as controlling a victim device, inferring user information, denying service, and modifying the message.

Kim et al. [103] investigated the challenges faced in the formal security analysis tools in IoT and proposed solutions. The authors show the formal analysis of the DoS (Denial of Service) attack on the majority of the IoT protocols and proposed the countermeasures for IoT protocols that are easy to implement. MQTT is one of the messaging protocols that is vulnerable to IoT DoS attacks.

OConnor et al. [19] focused on the integrity and availability of smart home IoT devices. The authors studied 24 devices and found different design flaws in using IoT telemetry messaging protocols chosen by the vendors. The authors used sensor binding and state confusion attacks against these smart home devices. With these attacks, authors were able to disrupt the functioning (i.e., on-demand messaging) of the IoT devices and at the same time not impact the keep-alive (i.e., always responsive) messages. The authors also suggested some countermeasures for these design flaws.

JEDI [104] is an end-to-end identity-based encryption and certification chain method to provide confidentiality and integrity between decoupled devices. The key sharing is done based on URI (i.e., topic in MQTT) by the trusted key server. The authors tried to address known issues with certification, such as revocation and reuse of keys. According to these authors, based on the data attributes, JEDI can easily be implemented on the publish-subscribe model. The authors mentioned that the JEDI many-to-many end-to-end encryption method could be implemented on all types of IoT devices, including ultra low-power deeply embedded sensors severely constrained in CPU, memory, and energy consumption with careful design and implementation.

Buschsieweke et al. [105] extended their work of Lightweight Capability-Based Access Control (LCap) to provide application layer security for constrained IoT devices using CoAP. The authors suggested that software can provide security for highly constrained devices. The authors have used classic HMAC (keyed-hash message authentication code) in two variations named HMAC1 (for request) and HMAC2 (for response) to provide integrity and authenticity for request and response messages. In addition, for payload encryption, the authors used AES-128 for encrypting the payload of the CoAP message referred to as Crypt Option in encryption mode. They used two variations of the Crypt option, which are Crypt1 (for request) and Crypt2 (for response).

Shin et al. [106] proposed a security framework for MQTT with AugPAKE (Augmented Password Authenticated Key Exchange) to establish a secure session between publish/subscriber and broker because using TLS for session key establishment is very costly in communication and computational overhead.

Malina et al. [107] made a proposal to secure the MQTT protocol for IoT. The framework suggests the use of light cryptographic algorithms, such as ECICS (Elliptic Curve Integrated Encryption Scheme), the Rabin encryption and AES for constrained devices. The authors compared the solution suggested with other cryptographic solutions and showed that the solution suggested takes fewer mathematical operations. The solution also provides secure message publishing and privacy-enhancing data publishing and subscribing.

Anantharaman et al. [108] demonstrated a solution to provide authentication, integrity and confidentiality for industrial-based IoT applications using key management and communication based on macaroons [109] in MQTT messaging protocol. Macaroons are based on nested, chained MACs (e.g., HMAC), which may also be used for authorization in decentralized, distributed systems. They used two macaroons; the first is long-term, installed during the device setup and specifies all channels the device needs to access. The second is channel-specific macaroons, which are short-term and often expire. The message's integrity is provided using the HMACs.

Sadio [23] suggested a method to improve security for MQTT/MQTT-SN using ChaCha20-Poly1305 AEAD (Authenticated Encryption with Associated Data). The solution provides end-to-end encryption using ChaCha20 and Poly1305 lightweight stream cipher. The secret is shared between publishers and subscribers through some out-of-band medium; therefore, the MQTT-SN gateway and broker cannot read the message. The authors have tested the system using 8-bit controllers.

Nuratch [110] suggested that for wireless sensor and actuator nodes, the MQTT protocol is well suited for many IoT applications, such as smart home, smart healthcare, and other smart systems that use wireless communication technologies such as Bluetooth, ZigBee, BLE. The author successfully tested MQTT client application on an 8-bit controller using a WiFi module.

Park et al. [17] proposed a means to provide end-to-end security in an MQTT-SN architecture; however, it does not consider the gateway device. They proposed a security architecture for MQTT-SN over UDP that involves certificates and extensive certificate management. The provides performance and security analysis. The security architecture proposed considers a threat model which focuses on the broker (i.e., malicious broker or curious broker).

Kim et al. [111] provided a mechanism for device management through IoT Home gateway device. According to the authors, the mechanism helps to remove the heterogeneity issues of various IoT devices and provides dynamic device discovery and auto-configuration. Upadhyay et al. [112] implemented an MQTT ACL (access control list) for a secure home automation system to provide encryption and data monitoring. Bryce et al. [113] added the geolocation (i.e., the physical location) of the device to the MQTT protocol. The geolocation helps to design a geofence [114] based on device different physical locations. In MQTT-G, by adding geolocation, the broker will send information only to the subscribers within the publishers' geofence. Similarly Fremantle et al. [115] investigated the use of OAuth in MQTT. Gheorghe-Pop et al. [116] provided a benchmarking mechanism for the MQTT broker, using a load test for workload, an endurance test for broker's reliability and efficiency, and a stress test for broker's robustness and recovery.

Roldán-Gómez et al. [117] [118]analyzed the security aspects of the MQTT-SN protocol due to its susceptibility to various vulnerabilities that can be exploited. To exploit

vulnerabilities, they suggested several attacks with basic scenarios using Contiki and Cooja. Roldán-Gómez et al. conducted measurements to assess the impact of the attacks and put forward a threat detection system utilizing the Complex Event Processing (CEP) engine. This system achieved an impressive F1 score of 0.9963 and is capable of identifying attacks in an IoT context.

Kao et al. [119] proposed a connection authentication technique that is both simple and secure. The technique is based on MQTT-SN, AEAD (ChaCha20-Poly1305), hash function, digital signature (ECDSA), key exchange (ECDHE), and hash function. These components are used to create Safe MQTT-SN, a secure version of MQTT-SN that enables clients to encrypt and decrypt published messages. Kao et al. replaced RSA with the ChaCha20-Poly 1305 algorithm, which has the ability to authenticate published messages and has been proven to have lower power consumption than AES-GCM.

Chen et al. [120] explored the root causes of MQTT security issues and proposes various offensive and defense strategies, including machine learning-based security, replay attacks, man-in-the-middle attacks, anomaly detection, and more. With the increasing complexity of IoT, there is a greater need for effective means to deal with attacks and anomalies. The author hopes to establish a smart, safe, and reliable IoT infrastructure using technology based on data analysis. Practical attack tests in various scenarios are needed for future MQTT security research.

Asmaa [121] approached employing transmission flags, which can authenticate all aspects of data exchange across networks and gather and examine data from numerous devices. Although the payload length is short, this method has demonstrated its effectiveness in assessing the degree of security of data transfer, which necessitates encryption for tracking purposes.

Farahani [122] has shown an effort to intercept wireless communication by employing ZigBee and implementing AES-CSM. The authors have made an attempt to use the same key twice in order to achieve this, such as by deliberately causing a node reset. To execute a denial-of-service attack, they suggested modifying the content of ZigBee packets, regardless of encryption, to introduce a flaw that disrupts the intended functioning of the protocol. Husnain et al. [123] identified 81 vulnerabilities associated with MQTT using National Vulnerability Database (NVD) and Common Vulnerabilities and Exposures (CVE), which are widely recognized mechanisms for reporting vulnerabilities. The vulnerabilities arise from three primary issues: inadequate packet length, absent essential fields, and a deficiency in logical error checks, along with other miscellaneous faults that do not fit into the aforementioned categories. The authors outline a comprehensive approach for detecting and preventing vulnerabilities in an end device. This approach consists of five steps: protocol intuition, vulnerabilities assessment, protocol identification, protocol parsing, and stringent protocol validations. Additionally, rules are defined to enhance the effectiveness of the vulnerability detection and prevention process. Their technology effectively detects and prevents these vulnerabilities from being exploited on IoT devices.

Sochor et al. [124] explained the reflection attack in their study, which exploits the MQTT-SN protocol and leads to DRDoS attacks, where many servers are targeted. Only a minuscule fraction of 0.02% of the victim's bandwidth is required to initiate a reflection assault that fully saturates the target system's bandwidth. To address this issue, the authors suggested a technique that entailed incorporating an additional handshake into the protocol and restricting Quality of Service (QoS) for levels 0 and 1. A four-way handshake (consisting of the PUBLISH, PUBREC, PUBREL, and PUBCOMP messages) can be utilized to ensure QoS level 2, which guarantees that each message is received

only once. However, because this was not enough, implementing a three-way handshake would have been a better choice for the client to communicate with the broker.

Andy et al. [125] analyzed the MQTT protocol utilized in IoT devices, revealing its shortcomings in terms of data privacy, authentication, and data integrity. The paper delves into the reasons behind the insufficient security measures in many IoT systems, illustrating various attack scenarios that can compromise the protocol. For instance, one scenario entails an attacker scanning a public network and carrying out denial of service attacks or data manipulation. Another scenario involves packet data sniffing and modification within the local network. The paper suggests the implementation of security mechanisms like TLS or ECC to ensure data confidentiality and non-repudiation. Nonetheless, security measures for resource-constrained devices still require further development.

Several academics are now studying the security of MQTT and MQTT-SN, but they are not considering the gateway and forwarder. They continue to regard end-device communication on the Internet as a distinct matter.

3.5 V2V Communication Security

We focus our discussion of related work done in the field of V2V communication security:

Muslam conducted a comprehensive literature review on V2V communication [126]. The main objectives of the review were to identify existing security protocols, assess their strengths and weaknesses, investigate the integration of emerging technologies such as 5G and edge computing, analyze the interaction between V2V communication protocols and broader IoT frameworks, and evaluate the impact of existing or proposed standards on the implementation of secure V2V communication systems.

Jabeen et al. [127] conducted a comprehensive examination of the security and privacy implications of connected vehicles, as well as an assessment of cloud platforms. Through a comparative examination of multiple cloud platforms, we can choose the most effective cloudlet for obtaining improved performance.

Cao et al. [128] proposed an efficient communication framework for on-the-move electric vehicle (EV) charging application, based on the P/S mechanism and public buses to disseminate the condition information of charging stations (CSs). The CS selection decision on where to charge is made by individual EVs for privacy and scalability benefits. The authors promote the concept of Mobility as a Service (MaaS) to exploit the mobility of public transportation vehicles such as buses to bridge the information flow to EVs, given their opportunistic encounters. The authors demonstrate the advantage of introducing buses as mobile intermediaries for information dissemination based on a common EV charging management system under the Helsinki city scenario. The authors further study the feasibility and benefit of enabling EVs to send their charging reservations for CS selection logic via opportunistically encountered buses. Results show that this advanced management system improves both performances at the CS and EV sides. Another research from Cao et al. [129] proposed another efficient communication framework for on-the-move EV charging applications based on the publish/subscribe mechanism. They use push and pull mode to send and gather data from the road-side units. The evaluation results showed that the decreased number of times EVs obtain information from RSUs deteriorates the performance in terms of average waiting time as well as information freshness from the EV perspective.

Taghizadeh et al. [130] introduced a novel V2V operation that empowers electric vehicle owners with excess battery energy to transfer it to other vehicles during their daily commutes. This innovative system facilitates energy transfer between two EV on-

board chargers, allowing EV owners to exchange stored energy without impacting the grid. Additionally, an IoT platform that employs the MQTT protocol is developed to manage interactions between EV owners and chargers. By utilizing the untapped energy in EV batteries, this V2V operation can alleviate grid strain.

Papadimitratos [131] discussed the security and privacy mechanisms needed for vehicular communication systems. It emphasizes the importance of scalable credential management and convergence on policies that determine pseudonym lifetimes, rates of change, CRL update, and distribution requirements. The text also highlights the need for cryptographic primitives beyond those in standards and literature, such as the Elliptic Curve Digital Signature Algorithm (EC-DSA), and the introduction of post-quantum cryptographic primitives. It also mentions the importance of countering clogging denial of service attacks and cooperative validation, especially as network data rates and security levels increase. Finally, the text emphasizes dedicated solutions for vehicle maneuver coordination to detect insider attacks and effectively detect subtle and sophisticated attacks in real-time without misclassifying actual, benign maneuvering as misbehavior.

Thing et al. [132] organized and discussed the methods of attacking and defending autonomous vehicles (AVs), proposing a comprehensive attack and defense taxonomy. This work is expected to help technologists develop secure and dependable AVs. The use of AVs is becoming more relevant as a solution to the challenges of urban transport. However, the security aspect of these vehicles is also imperative.

Based on our knowledge of the security of V2V communication, we have discovered that the existing security architecture is still riddled with numerous areas of weakness and vulnerabilities.

The security architectures proposed in the above literature for MQTT, MQTT-SN, and vehicle communication directly mentioned the threats. However, there is no proper

framework to identify the security requirements and the risks involved with each threat. One person cannot resolve all the vulnerabilities, which shows the need for a framework to identify and prioritize these security requirements to calculate the risk involved in case we are unable to address any particular security requirement.

3.6 Continous Glucose Monitoring

Britton et al. [133] examined the data from various CGM and highlighted the privacy and security concerns associated with patients' physical safety and the role of different regulatory bodies in enhancing these security guidelines.

Paul et al. [134] discussed the risks of wireless communication in insulin therapy and additional threats to system availability and integrity. They created a trustworthy insulin pump system that enhances safety and addresses the identified security challenges.

Weng et al. [135] investigated the application of homomorphic encryption to enhance blood glucose regulation in artificial pancreas systems. They have implemented a proportional-integral-derivative (PID) controller and assessed its performance using simulations, contrasting it with a plaintext controller. The average time in range (TIR) was 85.9% for standard food intake profiles and 86.0% for severe profiles, indicating no significant difference. It resulted in a secure cloud-based diabetes management system for efficient blood glucose regulation and comfortable remote monitoring.

Cleveland et al. [136] investigated the influence of the Internet of Things (IoT) technology on diabetes management in Norway, emphasizing its beneficial effects on patient quality of life while also addressing potential privacy and cybersecurity issues. Their research encompassed the three stakeholder groups (i.e., patients, healthcare persons and industry representatives) and observed substantial enhancements in patients' situations attributable to IoT technologies such as insulin pumps. It underscores the necessity for

explicit data ownership and a uniform assessment of privacy legislation across healthcare jurisdictions. The restrictions encompass a limited quantity of interviews, particularly among industry representatives.

Mandava et al. [137] elucidates integrated continuous glucose monitoring (iCGM) systems, which amalgamate glucose monitoring with insulin pumps, thereby streamlining the procedure and providing real-time communication to smart devices. Nonetheless, their interconnectedness presents security vulnerabilities, including possible cyber intrusions that may lead to hazardous insulin overdoses. They clarified that while Bluetooth offers fundamental security elements, the onus for improved security lies with manufacturers, rendering iCGMs susceptible. They clarified the architecture, weaknesses, and necessary security measures for iCGM systems.

Venugopalan et al. [138] introduced GlucOS, an innovative system for safe automated insulin delivery that combines algorithmic and driver security with comprehensive end-to-end verification. GlucOS exhibits safety and enhanced glucose regulation, even in adverse conditions. Their algorithmic security employed traditional ways to highlight reactive safety-critical algorithms, such as reference monitoring; they programmed pumps to manage lost commands; and we devised innovative formal methods to tackle human dynamics in glucose metabolism. These insights underscore the intricacy of developing a reliable automated insulin system, integrating theoretical security with practical, interdisciplinary knowledge. Their research does not fully discuss the lower-level security aspects and the specific security mechanisms employed. We have proposed a security architecture that uses the MQTT-SN publish-subscribe protocol, which helps users to send sensor data to the subscriber providing mitigating all security concerns mentioned in Threat Model.

3.7 Smart Lock Node

In this section, we are going to discuss the work done by different researcher and industry related to Smart Lock. Several studies have analyzed vulnerabilities and short-comings in Bluetooth-enabled smart locks. Zhang et al. [139] showed that many commercial Bluetooth-enabled locks rely on static keys or predictable pairing procedures, making them susceptible to replay and relay attacks. Also, Fernandes et al. [140] showed that Bluetooth-based IoT devices often lack proper key-management and rely mostly on cloud-side authentication, creating single points of failure and also susceptible to relay and replay attacks. Many smart-home systems rely on cloud-centric access control. Ur et al. [141] found that cloud-based IoT ecosystems often expose excessive privileges and lack detailed fine-grained access control. Similarly, Alrawi et al. [142] showed that cloud APIs for smart-home devices frequently suffer from weak authentication and over-privileged tokens. In cloud dependent solutions, cloud should be accessible always as there are no offline authorization. Capability-based access control has been explored in IoT contexts. Zhang and Chen proposed CapBAC for IoT devices, showing that capability tokens can reduce reliance on centralized ACLs [143]. However, their design assumes IP-connected devices and does not address constrained, Bluetooth-only hardware and also requires online verification. MQTT-SN has been studied for constrained IoT sensor based networks. Singh et al. highlighted that MQTT-SN brokers are often untrusted and lack built-in authentication or integrity guarantees [144]. Other works propose adding TLS-like mechanisms such as DTLS, but these are too heavy for battery-powered devices. These solutions doesnot provide end-to end integrity. Many solutions provided for Smart Locks often support only coarse-grained roles. Jang et al. showed that many smart-home systems lack time-bounded or context-aware access control, leading to privilege misuse [145].

However, in our architecture we try to address all the weaknesses mentioned as by using ephemeral capability tokens with per-operation authorization, enforces authorization locally on the lock, not in the cloud, uses nonce-based replay protection and additional symmetric encryption for BLE payloads. In our architecture every operation has time-bounded capabilities, topicID is derived cryptographically. For maintaining fine-grained access control role, time and usage constraints encoded in token which uses Cryptographic technique such as Lightweight HMAC, hash, AES and RSA-1024.

3.8 Summary of MQTT-SN Countermeasures

Securing MQTT-SN in constrained IoT environments requires a multi-layered strategy that balances strong protection with minimal overhead. Research consistently highlights lightweight authenticated encryption—particularly ChaCha20-Poly1305—as an efficient means to ensure confidentiality and integrity on low-power devices, while alternatives such as OSCORE enable end-to-end payload protection independent of the transport layer. Robust authentication is equally critical, with ECC-based handshakes (ECDSA/ECDHE) and hardware-rooted PUF mechanisms emerging as effective solutions for preventing impersonation and securing key exchange. Because many IoT nodes cannot support heavy cryptographic stacks, intrusion detection systems—both rule-based and anomaly-driven—provide an essential secondary defense against flooding, spoofing, and misuse of legitimate credentials. Finally, protocol-level enhancements such as SMQTT-SN, secure gateways, and upcoming MQTT-SN standard revisions aim to embed security natively into the protocol. Together, these countermeasures form a comprehensive security architecture that strengthens MQTT-SN against confidentiality, integrity, authentication, and availability threats while preserving its lightweight nature [146] [147] [148].

Table 3.1: Summary of MQTT-SN Security Countermeasures

Category	**Key Techniques**	**Purpose / Benefits**
Lightweight Encryption & Data Protection	ChaCha20-Poly1305 AEAD; lightweight block ciphers; OSCORE; white-box cryptography	Ensures confidentiality and integrity with low overhead; enables end-to-end protection; secures keys on physically exposed devices.
Authentication & Key Exchange	ECC (ECDSA/ECDHE); PUF-based authentication; broker authentication; random ClientIDs	Prevents impersonation; enables efficient key exchange; resists key extraction; mitigates ClientId hijacking.
Intrusion Detection & Monitoring	Rule-based IDS; ML-based anomaly detection; gateway-level monitoring	Detects DoS, flooding, spoofing, and abnormal traffic; compensates for device-side limitations.
Protocol Extensions & Secure Variants	SMQTT-SN; ABE-based access control; DTLS-enabled gateways; upcoming MQTT-SN security revisions	Integrates security into the protocol; enforces topic-level authorization; offloads heavy cryptography; standardizes secure operation.

3.8.1 Best Practices in Deployment

Beyond academic proposals, it's worth noting some practical measures that IoT system designers should implement when using MQTT-SN in the mentioned domains:

Enable MQTT-SN's built-in user/password authentication and use strong, unique credentials for each device or device type. While not foolproof, this is a basic layer (the information leak attack can be "avoided by making use of MQTT-SN's built-in authentication" as one study noted). In a smart home or hospital network, never leave the broker open without auth.

Network Layer Security: Use VPNs or encrypted tunnels for sensor traffic if end-to-end encryption at MQTT-SN level is not feasible. For example, a separate network

enclave for medical IoT devices with secure gateways can contain damage.

Regular Software Updates: Many MQTT vulnerabilities (33 CVEs were noted in MQTT in recent years) are due to broker or client software bugs. Ensuring devices and brokers run up-to-date, patched firmware helps prevent known exploits (which could otherwise lead to remote code execution or broker crashes).

Monitoring and Rate Limiting: Set sane limits on the broker (e.g., max messages per second per client, disconnect clients that overload, restrict wildcard subscriptions to admin users, etc.). In IIoT, if a sensor normally sends 1 msg/min, the broker can flag if it sends 1000/min. Vehicle systems using MQTT-SN for telemetry should enforce strict timing checks and ignore unexpected bursts that could be malicious.

Redundancy and Fail-safes: For critical systems (e.g., vehicle platooning or ICU monitors), design the system to fail safe if data is missing or suspicious. For instance, if a vehicle V2X alert message is not authenticated, it should be ignored to avoid a potential accident caused by a fake message; similarly a medical system could require two-factor confirmation for drastic actions (to prevent a single spoofed MQTT-SN command from causing harm).

By combining these best practices with the technical countermeasures from research, MQTT-SN can be significantly hardened.

Chapter 4

Security Requirements for MQTT-SN-Based IoT Use Cases

4.1 Introduction

Currently, there is a wide range of IoT devices accessible in every residence. Hence, numerous multinational enterprises worldwide are actively creating IoT devices, services, and technologies to gain a competitive edge in the market beforehand. However, they fail to regard security as a functional necessity, resulting in a decrease in the priority of security issues [149]. Hence, organizations exhibit hesitancy in adequately implementing security measures for their equipment and services. When Transmission Control Protocol/Internet Protocol (TCP/IP) was initially developed, the significance of security was not widely recognized, and the majority of individuals were unaware of the potential exploits that attackers could carry out through security vulnerabilities. Due to these factors, TCP/IP was not well built for security purposes. Due to the insecure architecture, attackers can exploit numerous vulnerabilities, resulting in security breaches and significant financial losses. Nevertheless, we possess a comprehensive understanding of the

importance of security and the precise capabilities that attackers might exploit through security vulnerabilities. Therefore, it is imperative to engage in discussions about IoT security throughout the development of IoT-related standards to prevent the recurrence of past errors as modifying systems becomes increasingly challenging as the device reaches an advanced level of development, resulting in greater effort and cost. Hence, it is imperative to prioritize security right from the inception of any product development.

Most developers and engineers prioritize attaining security objectives above delineating precise actions to be executed. For an IoT product, it is crucial to provide clear specifications (i.e., actions) together with functional requirements (desired outcome) to allow a security architect or designer to efficiently deal with any security issues. These particular characteristics are commonly known as security requirements. As far as we are aware, there is a dearth of research specifically focused on the security needs of the IoT environment. Many academics have mainly focused on threat models and targeted those threats. The process being used so far has a gap in understanding the importance of every critical and non-critical asset and the priority and risk involved in addressing the security requirements. As it has been said, no system is 100% secure. Therefore, we need a proper structural security requirement-gathering framework that helps us make an informed and risk-calculated decision.

Security requirements are defined as constraints on system functions. The requirements should explicitly specify the objects that require protection and provide a clear rationale for why such protection is necessary. Essentially, requirements should encompass the specifications regarding the individuals who have the authority to perform certain actions and the specific timeframes in which these actions can be carried out. Furthermore, they should serve the purpose of clearly defining what actions are permissible or forbidden for each individual involved. The security requirements articulate the secu-

rity objectives in operational terms that are sufficiently detailed to be communicated to the system's designer/architect (actions). During the early 2000s, numerous researchers were developing various methodologies or approaches to establish the security prerequisites for software systems [150]. Among numerous methodologies, we have identified two techniques that exhibit significant similarities: SREP [68] and SQUARE [80].

SREP and SQUARE are better choices for security requirement gathering due to their tailored focus on security, systematic approach, consideration of contextual factors, integration with existing processes, community support, and comprehensive coverage of security concerns [68] [80]. By employing these frameworks, organizations can effectively mitigate security risks and enhance the resilience of IoT deployments against evolving threats.

1. These frameworks focus explicitly on identifying and prioritizing security concerns, which can be paramount in IoT deployments.

2. These frameworks are guidelines, methodologies, and tools that assist in identifying, analyzing, and documenting security requirements across Software Development Life Cycle (SDLC) stages. Following a structured process, organizations can achieve complete coverage of all potential threats, reducing the chances of missing crucial vulnerabilities that IoT systems often conceal.

3. IoT deployments are characterized by extremely varied environments, usage scenarios, and stakeholders, all with distinct and relevant security requirements and limitations. Both SREP and SQUARE are explicit regarding the importance of context.

4. SREP and SQUARE frameworks are expertly designed to seamlessly integrate with existing software engineering processes, making them versatile and adaptable to

different development methodologies and organizational contexts. Whether organizations follow traditional waterfall models or agile approaches, these frameworks can be confidently incorporated into the requirements engineering phase. This ensures that security considerations are addressed early in the development life cycle, providing a sound foundation for secure software development.

5. SREP and SQUARE offer comprehensive coverage of security concerns, encompassing various security dimensions such as confidentiality, integrity, availability, authentication, authorization, and non-repudiation.

4.1.1 Contributions

We focus on defining security requirements for one of the main aspects of V2V communication, i.e., sending secure distress signals in an emergency, which has not been addressed with a proper framework, continuous glucose monitoring, and Smart Lock communication. To perform this, we defined the detailed modified SREP and SQUARE method [151] to design the threat model for vehicle communication using publish/subscribe protocol defined in [5], MQTT-SN (i.e., MQTT for Sensor Network) [14]. Here, we provide more details of the modified method, including reasoning and the steps involved in defining security requirements for the V2S, CGM and Smart Lock communication use cases.

4.2 Proposed Security Requirement Methodology

In this section, we first defined the modified SREP and SQUARE framework steps needed for the IoT domain and the proper threat-defining framework to properly analyze a threat.

4.2.1 Modified SREP and SQUARE

SREP and SQUARE are inadequate in tackling the IoT domain because they do not adequately address its distinct difficulties, which include varied ecosystems, scale, contextual considerations, resource restrictions, dynamic nature, and transdisciplinary requirements. Their disadvantages include a lack of scalability, flexibility, and advice specifically designed for the complexity of IoT, such as the presence of diverse devices, limited resources, and the need for real-time interactions. Specialized frameworks are necessary to adapt to the dynamic surroundings, regulatory issues, and interdisciplinary nature of IoT, in order to effectively gather security requirements. Therefore, modified SREP and SQUARE methodologies can be utilized for the advancement and implementation of contemporary Smart IoT devices. Given the escalating intricacy and security apprehensions linked to IoT devices, it is imperative to integrate solid security requirements engineering methods. The utilization of SREP can be applied to ascertain and delineate the security goals, hazards, and prerequisites that are distinct to IoT devices. SQUARE, however, is especially valuable for specifying security-related quality criteria in IoT devices. To optimize the SREP and SQUARE for modern Smart IoT devices, the following improvements should be taken into account [151]:

1. Examine the specific hazards and vulnerabilities associated with IoT: Enumerate the distinct hazards and perils linked to IoT devices, including but not limited to unauthorized entry, data breaches, device manipulation, and privacy apprehensions. Integrate these hazards into the risk assessment and analysis phases of SREP.

2. Discuss communication protocols and interfaces: IoT devices commonly utilize many protocols and interfaces for communication. Make sure that SREP and SQUARE take into consideration the security of these communication routes, en-

compassing encryption, authentication, and safe data transfer.

3. Incorporate factors related to the IoT ecosystem: IoT devices are commonly integrated into a broader ecosystem comprising cloud services, gateways, and networked devices. Broaden the extent of SREP and SQUARE to encompass the security needs of the entire ecosystem, encompassing the interactions between devices and cloud services.

4. Privacy concerns arise due to the fact that IoT devices frequently handle and communicate confidential personal information. Integrate privacy standards and considerations into the process of engineering security requirements, taking into account adherence to data protection rules, implementation of user permission processes, and utilization of data anonymization techniques.

5. Enhance the emphasis on authentication procedures and access control for the IoT devices. Integrate prerequisites for ensuring the secure verification of devices, the authorization of users, and the implementation of access control regulations that are customized to the unique requirements of IoT devices.

6. Physical security considerations: IoT devices are susceptible to physical attacks or tampering due to their physical accessibility. Specify the necessary measures for physical security, such as the inclusion of tamper detection technologies, the implementation of a secure enclosure design, and the provision of protection against physical theft or manipulation.

7. Revise and enhance the techniques used for evaluating the security of systems: Modify security testing approaches and processes to specifically cater to the distinct attributes of IoT devices. Examine vulnerability scanning, penetration testing, and fuzz testing that are specifically tailored to IoT protocols and interfaces.

8. Consistent surveillance and regular updates: IoT devices may necessitate ongoing surveillance to detect security flaws and updates to counteract evolving threats. Include provisions for continuous security monitoring, patch management, and firmware/software updates within the security requirements.

By adapting SREP and SQUARE to incorporate these IoT-specific factors, we can guarantee that the security requirements engineering process effectively tackles the problems and hazards linked to Smart IoT devices.

We have modified their steps to account for the unique challenges and complexities of IoT environments. Here is our proposed framework:

1. *Identify Stakeholders and Assets:*

    - Identify all stakeholders involved in the IoT ecosystem, including manufacturers, developers, end-users, and regulatory bodies.
    - Identify the assets within the IoT system, such as devices, sensors, actuators, data repositories, and communication channels.

2. *Define Security Quality, Goals and Objectives:*

    - Define specific security goals and objectives tailored to IoT deployments, considering factors like data privacy, device integrity, resilience to cyberattacks, and compliance with industry regulations.
    - Identify security quality attributes relevant to IoT deployments, such as confidentiality, integrity, availability, authentication, authorization, and non-repudiation.
    - Consider additional quality attributes specific to IoT, such as resilience, privacy, scalability, and energy efficiency.

3. *Define Security Requirements Baseline:*

- Define a baseline of security requirements based on industry best practices, standards, guidelines, and regulatory requirements applicable to IoT deployments.
- Consider existing security frameworks and methodologies tailored to IoT, such as the IoT Security Foundation's Best Practice Guidelines.

4. *Analyze Threats and Vulnerabilities:*

   - Conduct a thorough analysis of potential threats and vulnerabilities specific to IoT environments, including device tampering, data breaches, unauthorized access, and denial-of-service attacks.
   - Consider both technical and non-technical threats, such as physical security risks and supply chain vulnerabilities.
   - Use IoT-specific use cases and threat models to guide the elicitation process and ensure comprehensive coverage of security concerns.

5. *Elicit Security Requirements:*

   - Use various elicitation techniques, such as interviews, surveys, workshops, and brainstorming sessions, to gather security requirements from stakeholders.
   - Focus on requirements related to authentication, encryption, access control, secure firmware updates, secure bootstrapping, and secure communication protocols tailored to IoT deployments.

6. *Prioritize Security Requirements:*

   - Prioritize security requirements based on their criticality, impact on system functionality, and feasibility of implementation in IoT environments.

- Consider factors like resource constraints, interoperability, scalability, and regulatory compliance when prioritizing requirements.

7. *Document Security Requirements:*

   - Document security requirements in a clear, concise, and structured manner, using standardized formats and templates.
   - Ensure that security requirements are traceable, measurable, and verifiable, enabling effective implementation, testing, and validation.

4.2.2 Modified Threat Defining Framework for IoT

We now design the threat-defining framework after defining modified security requirement-gathering methods. As numerical values for threat probability, severity and attacker's difficulty lack credibility, most practical risk assessment are based on qualitative ratings. Therefore, we use a categorical rating (low, medium, high) to calculate the risk [67]. In designing the threat model, the following framework has been used:

1. Threat: A threat is a potential cause of an unwanted incident, which may harm a system or organization, often accompanied by a condition avoiding the threat.
2. Attack Name: A successful threat is known as an attack.
3. Attacker's Difficulty/ Threat Difficulty: It defines the knowledge and tools an attacker needs to attempt a successful attack.
   (a) Low: An attacker does not need any high-level technical knowledge of hacking tools or systems. The basic knowledge of how the system works is sufficient.

(b) Medium: An attacker needs to have basic knowledge of reverse engineering IoT devices and the structures of the protocol used in the communication.

(c) High: An attacker needs to have complete knowledge of hacking and has tools to intercept messages, modify them and send them back to the broker.

4. Threat Severity: The qualitative analysis of the threat is done based on asset and success on impact:

   (a) Low: A threat severity is low when the successful impact does allow an attacker to make the device inaccessible.

   (b) Medium: A threat severity is medium when the successful impact does allow an attacker to intercept messages and determine their contents.

   (c) High: A threat severity is high when the successful impact allows an attacker to intercept messages, determine their contents, and respond with modified data, and the recipient accepts it as legitimate.

5. Threat Probability/Threat Likelihood: Based on level of attack difficulty, the probability of an attack occurs successfully and how easily it can be repeated are defined:

   (a) Low: With the high level of attacker difficulty, it is less probable for the success of an attack and not able to repeat it often.

   (b) Medium: With a medium level of attacker difficulty, it is probable for the success of an attack and as we are only capturing data or passive eavesdropping.

   (c) High: With a low level of attacker difficulty, it is highly probable for the success of an attack, and an attacker may repeat it often.

6. Summary: It summarizes the whole threat and its objective.

7. Precondition: It defines the knowledge that an attacker should have for a successful attack.

8. Step: It defines different steps taken by a user, an adversary, and a broker to make a threat into a successful attack.

9. Postcondition: It defines the results after a successful attack.

4.2.3 Generalized Security Requirements Definition

In this section, the security requirements definitions for publish-subscribe model used for V2V communication will be formulated. All definitions of the terms we are going to use here in this document have been covered and derived from [67] and [152]:

1. Identification: The statement defines the degree to which an application must recognize its external components prior to engaging with them. The system should adhere to the privacy rules, which may necessitate the anonymity of users. It can also be explained as: who you say you are (i.e., name, user identifier etc.), what you have taken (i.e., hardware key, smart card, identification card, etc.) and who you are (i.e., bio prints or behavioural traits).

2. Authentication: It ensures that person or identity are actually who or what they claim to be. It depends on identification. In order to handle the authentication requirements, we should have answers of at least one of these questions: who you know (i.e., mother maiden name, name of pet, etc.), what you have (i.e., taken, hardware key, smart card, identification card etc.), and who you are (i.e., bio prints or behavioural traits).

   The following three sub-divisions of authentication have been created:

(a) Data Origin Authentication: It should instill confidence in the authenticity of the data source. A subscriber should possess the capability to authenticate the publisher (i.e., sender) of the message.

(b) Device Authentication: It should ensure that the equipment involved in communication is truly what it claims to be, thereby instilling confidence in its identification. Within the pub-sub model, a broker possesses the exceptional capability to accurately identify and authenticate a device, hence preventing any other device from falsifying the authentication credentials of another device.

(c) Device Owner/User Authentication: It should instill assurance that the user/owner of the device is indeed who they claim to be. Within the pub-sub framework, it is imperative for the broker to possess the capability to authenticate the device owner in order to mitigate any complications arising from the transfer of device ownership.

3. Confidentiality: It is a property where non-authorized entities are not allowed to access plaintext messages.

(a) Information Confidentiality: It signifies that data is safeguarded in the event that the broker's reliability is questionable. Feasible, yet necessitates an agreement outside the usual communication channels between the publisher and subscriber. The out-of-band agreement negates the major advantages of the pub/sub architecture [67]. The pub/sub system is designed to resemble point-to-point models rather than many-to-many ones. There are two possible types. The initial data transfer between the publisher or subscriber and the broker is encrypted using point-to-point encryption. However, the data is accessible in plain text form at the publisher, subscriber, and broker. The second

data is encrypted throughout its entire journey until it reaches the intended recipients, ensuring its confidentiality. The data remains securely encrypted at the broker. To achieve complete information confidentiality, it is necessary to establish a group key distribution between the publisher and subscribers through an out-of-band method. The dissemination of keys beyond the main communication channel eliminates the advantages of the fundamental publish/subscribe model and transitions to a multicast model.

(b) Message Header Confidentiality: It means sending message header securely from one point to another.

4. Integrity: Any creation, modification or deletion done by unauthorized parties in the data and communication will be detected.

   (a) Payload Integrity: The content of the message should remain unchanged after it has been sent by the sender. In the pub-sub system, the message transmitted by the publisher must remain unchanged until it is received by the subscribers.

   (b) Header Integrity: The Header must be safeguarded against unauthorized alterations. The assumption is that the publisher/subscribers have confidence in the broker. For example, when a user subscribes to topic A in the pub-sub system, they will get a message that has been published under topic A. It is imperative that the records of topics that a certain subscriber is interested in remain unmodifiable by anyone.

   (c) Broker Integrity: Compromising the trusted broker jeopardizes the integrity of the entire system. The term “trusted broker” refers to a situation where the broker has access to the data sent by the publisher to the subscribers in plaintext, ensuring point-to-point confidentiality.

5. Availability: It is the property according to which resources and services remain accessible for authorized use. It limits the frequency and size of publication. Customized publication control allows subscribers to specify which publisher can publish information. Subscribers can use a secret, challenge-response system. It will help get rid of the bogus notifications.

6. Authorization: It is the property of computing resources being accessible by authorized entities. Authorization is achieved through access control. A broker, publisher or subscriber may control which subscribers or publishers may access the information.

7. Physical Protection: It specifies the extent to which an application or center shall protect itself from physical assault. Publisher and subscriber should protect their physical device or application from tampering.

4.3 Security Requirements on V2V Communication based on MQTT-SN

The main security objectives based on the need for the V2V communication use case are identified below.

1. Authentication: Verify the legitimacy of the end user and the source of the data.

2. Confidentiality: Ensure the prevention of unauthorized disclosure of information.

3. Integrity: Ensure that information cannot be modified without authorization.

4. Accessibility: Ensure that both the information and the device are easily accessible.

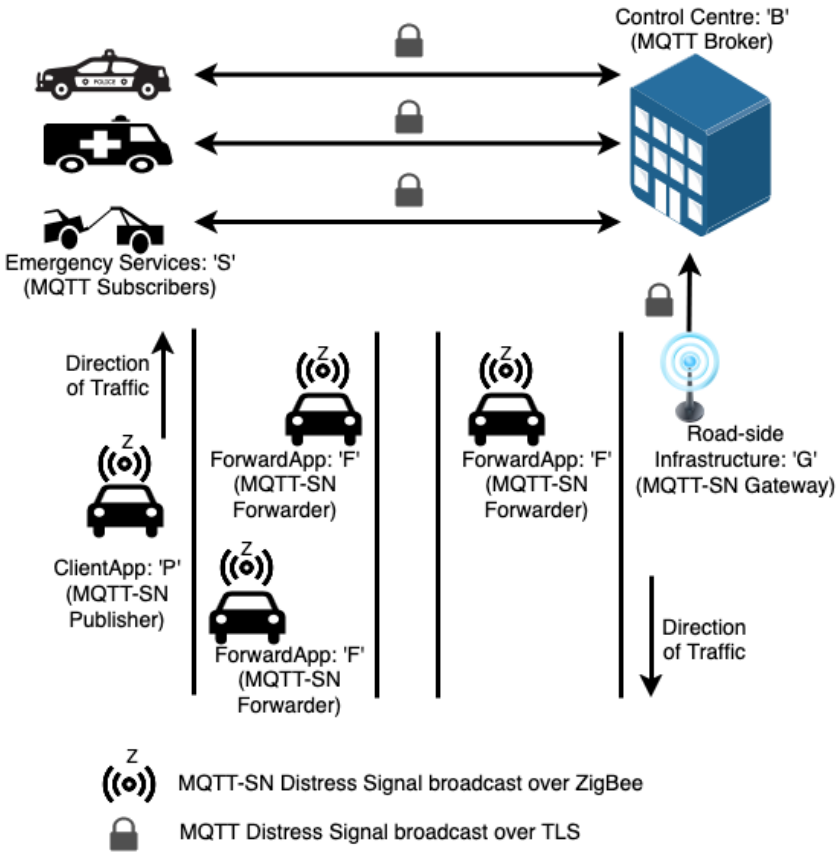


Figure 4.1: Vehicle-to-Vehicle Communication using MQTT-SN over Zigbee [5]

5. Authorization: Guarantee that information and devices can only be accessed by individuals who have been granted permission.

4.3.1 V2V Use Case Scenario

V2V communication is a rising research area. It is used for many reasons, such as informing about road congestion, traffic blockage, and accidents and sending distress signals to ask for help in case of potential and immediate danger exists. We have seen that many methods have been applied; however, the only concern with those solutions is that

they need cellular or satellite networks to communicate. What happens when no such network is available or end-users cannot afford such upgrades in their vehicles? To tackle this problem, we have proposed the MQTT-SN communication model, which uses the publish-subscribe method to send a distress signal from a vehicle to emergency services. A distress signal is sent using MQTT-SN over ZigBee [5]. Figure 4.1 shows our design architecture for distress signals moving from a vehicle to emergency services. The design architecture enables distress signals to move from a vehicle to emergency services using a ClientApp or vehicle on-board unit. The distress signal data contains relative location information and vehicle information, which is sent using VIN (Vehicle Identification Number) as ClientID. The distress message is broadcasted to the Control Centre using the MQTT-SN Forwarder and roadside infrastructure (i.e., MQTT-SN gateway). Based on topic and subscription, the signal is forwarded to the respective emergency services by the Control Centre. The message contains the VIN and the user's last known location information, which helps emergency services to dispatch the help.

In this document, when we mention the security of V2V communication and different attacks on the use case, we mainly refer to the attackers exploiting communication protocol to achieve the desired goal. When it comes to emergency services, there may and may not be human risk factors involved, and therefore, it is important to make sure these signals reach their respective locations safely and without tampering. Here, we will focus on securing how vehicles connect with the gateway (i.e., roadside infrastructure) by defining the security requirements for this particular use case and its importance has already been discussed in earlier sections. The components and setups involved in the V2V use case scenario are explained in Section 4.4.2.

4.3.2 Identification of Critical/Non-Critical Assets in a V2V Scenario

In this section, the coverage of various critical and non-critical assets in V2V communication architecture and functional requirements will be discussed:

1. Assets: The assets can be physical components and non-physical components (like messages).

   (a) MQTT-SN Publisher (ClientApp): An MQTT-SN publisher is an application that utilizes a ZigBee transceiver module to transmit distress signals to subscribers. This application might potentially be integrated into the vehicle's onboard automotive system. This application utilizes a ZigBee module transmitter that is operating in broadcast-sending mode. Every application possesses a distinct identification known as the ClientID, which may be derived from the vehicle. In this case, we can utilize the VIN number of the vehicles as the identifier. The ClientApp is developed using the React Native framework, which operates on the Samsung Galaxy Note 2 device equipped with 2 GB of RAM and 32 GB of storage. Additionally, it is connected to a plug-and-play ZigBee module for broadcasting messages. In our nomenclature, the ClientApp is represented by the subscript 'P'.

   (b) MQTT-SN Forwarder (ForwardApp): An MQTT-SN forwarder is a program that receives MQTT-SN messages from clients and transmits them to the gateway using a ZigBee transceiver module. This application can easily be integrated into the onboard Android automotive system. Under typical conditions, the ZigBee module employed by this application operates in a state of readiness to receive signals. Upon receiving any message, it promptly dis-

seminates it to the closest gateway or another MQTT-SN forwarder. The ForwardApp is developed using the React Native framework, which operates on a Samsung Galaxy S21 device equipped with 2 GB of RAM and 64 GB of storage. It is connected to a plug-and-play ZigBee module for broadcasting messages. The ForwardApp is denoted by subscript 'F' in our notation.

(c) MQTT-SN Gateway (Roadside Infrastructure): An MQTT-SN gateway is a type of roadside infrastructure that accepts a message from a MQTT-SN client or forwarder over ZigBee and then forwards it to the MQTT broker. The gateway translates the MQTT-SN message into MQTT. The message is transmitted via the MQTT protocol using the TCP channel. The ZigBee module is in a state of actively monitoring roadside infrastructures to receive data from the MQTT-SN forwarder. The Roadside infrastructure is configured with the destination address of the MQTT broker, which serves as the control centre. The implementation utilizes a Raspberry Pi 3B+ module, namely the Broadcom BCM2837B0 quad-core A53 (ARMv8) processor. This module is equipped with a 32-bit board and includes both ZigBee and Wi-Fi capabilities. The roadside infrastructure is denoted by subscript 'G' in our notation. There are already roadside infrastructures available in multiple countries. These can easily be programmed to support this architecture to make it less expensive.

(d) MQTT Broker (Control Centre): The MQTT distress message received by the broker over the TCP channel from the MQTT-SN gateway. The control centre's task is to distribute messages based on the subscriptions. Control centre runs on a laptop with specifications: M2 Pro Processor, 16 GB RAM and 512 GB hard disk. The Control centre is denoted by subscript 'B' in our notation.

(e) MQTT Subscriber (Emergency Services): The MQTT subscribers subscribe to various distress signal themes. The emergency services encompass law enforcement agencies, medical facilities, and vehicle recovery services. The emergency services operate on a laptop equipped with the following specifications: an 8th generation i7 processor, 16 gigabytes of RAM, and a 512 gigabyte hard disc. The emergency services are denoted by subscript 'S' in our notation.

(f) Information/ Data:

- Vehicle Status Messages: It represents the current location of vehicle and other information necessary for emergency services.
- ClientID: Unique vehicle information used to maintain sessions.
- Topics: It represents the current state of the vehicle (i.e., vehicle not working, accident with or without injury) sent from the vehicle to respective emergency services.

2. Functional Requirements: There is one of the main functional requirements for V2V communication:

(a) Vehicle should send a status message to the end-user through the broker when there is an emergency.

4.3.3 Threat Model for a V2S Scenario

In this section, threats to V2S communication use case are defined.

V2S-Threat 1

- Threat: The broker authorizes the adversary as if the adversary had a valid ClientID to send malicious status messages. [Authorization]
- Attack Name: Spoofing Attack
- Level of Attacker Difficulty: Medium
- Threat Severity: High
- Threat Probability: Medium
- Summary: The external attacker type guesses the unique ClientID (i.e. VIN) of a vehicle and publishes malicious status messages for the Subscriber (i.e., emergency services) through the broker. The objective of the attacker waste the resources of emergency services.
- Preconditions:
  - The attacker has access to a vehicle, which the attacker uses to deduce ClientID by guessing or reverse engineering based on the format used by manufacturers [18].
  - The attacker has an authorized account with the broker.
  - The attacker has an explicit knowledge of the structure and meaning of the ClientID and topics.
- Steps:
  1. Emergency Services: The emergency services subscribe to the topics published by the ClientApp.

2. Adversary: The external attacker publishes all status messages under the topics linked Client ID of the ClientApp.
3. Control Centre: The Control Centre forwards the information related to the topics linked to the ClientID of the vehicle to the emergency services subscribing to the topics.

- Postcondition: The attacker has successfully able to waste the resources of emergency services.

V2S-Threat 2

- Threat: Unauthorized individuals or entities may attempt to intercept and eavesdrop on V2V communications to obtain sensitive information, such as location, speed, or other private data. [Confidentiality]
- Attack Name: Passive Eavesdropping
- Level of Attacker Difficulty: Easy
- Threat Severity: Low
- Threat Probability: High
- Summary: The external attacker subscribes to the wildcard subscription (i.e., #) and gains access to all the messages published by the vehicle for the emergency services through the broker. The objective of the attacker is to know the sensitive information.
- Preconditions:
  - The attacker has physical access to a vehicle, which is running ForwardApp.

  - The attacker has an authorized account with the Control Centre.
  - The attacker has explicit knowledge of the structure and meaning of the ClientID.

- Steps:

  1. Adversary: The external attacker subscribes to all the information linked to Client ID using a wildcard subscription. The attacker may or may not use guessed ClientID. The attacker may also listen to the ForwardApp messages.
  2. ClientApp: The ClientApp publishes the status message.
  3. Control Centre: The broker sends/publishes all the information it has related to all the topics to the adversary.

- Postcondition: The attacker can use the information to understand the vehicle details and end-user current information.

V2S-Threat 3

- Threat: Attackers can record and replay legitimate V2V messages, which can have negative consequences, especially when safety-critical information is involved. [Integrity]
- Attack Name: Replay
- Level of Attacker Difficulty: Low
- Threat Severity: Medium
- Threat Probability: Medium

- Summary: The external attacker can copy the packet published by the ClientApp and replay it back to the emergency services through a Control Centre, which will give false information to the emergency services. Thus, the attacker's objective is to misguide and waste the resources of emergency services.
- Preconditions:
  - The attacker has an authorized account with the Control Centre.
  - The attacker has the means to intercept and capture a message from the ClientApp to the Control Centre.
- Steps:
  1. ClientApp: The ClientApp publishes a control/status message to the Control Centre.
  2. Adversary: The external attacker intercepts the message and saves it. The external attacker publishes the saved message.
  3. Control Centre: The Control Centre sends the old message to all the emergency services subscribing to that topic.
- Postconditions:
  - The Control Centre does not recognize the old message.
  - The attacker can replay the status messages to exhaust the resources of emergency services.

V2S-Threat 4

- Threat: The attacker can intercept and modify messages to misguide the emergency services. [Integrity]

- Attack Name: Injection Attack
- Level of Attacker Difficulty: High
- Threat Severity: High
- Threat Probability: Low
- Summary: The external attacker can copy the message published by the ClientApp, modify it, and forward the modified message to the Control Centre for the emergency services, which will give false information to the emergency services in case of a status message. Thus, the attacker's objective is to misguide the emergency services.
- Preconditions:
  - The attacker has physical access to a ForwardApp.
  - The attacker has an authorized account with the Control Centre.
  - The attacker can intercept, capture, and modify a message from the ClientApp to the Control Centre.
- Steps:
  1. ClientApp: The ClientApp publishes a status message to the Control Centre.
  2. Adversary: The external attacker intercepts the message and modifies it. The external attacker publishes the modify message.
  3. Broker: The Control Centre sends/publishes the corrupted message to all the emergency services subscribing to that topic.

- Postconditions:
  - The Control Centre does not recognize the modified message.
  - The attacker can send the status message to misguide the emergency services.
  - The attacker can send the wrong status of ClientApp.

V2S-Threat 5

- **Threat:** The attacker can exploit the long lifespan of IoT devices (5–15 years) to break a static public–private key pair using increasing computational power over time. [Confidentiality, Integrity, Authentication]
- **Attack Name:** Long-Term Key Exposure / Cryptographic Aging Attack
- **Level of Attacker Difficulty:** Medium
- **Threat Severity:** High
- **Threat Probability:** Medium
- **Summary:** IoT devices often operate for 5–15 years without hardware refresh or key rotation. As stated, "The lifespan of IoT device is between 5–15 years. Therefore, with current computational capabilities at the disposal of an attacker, a single public-private key pair for the whole lifespan can easily be attacked." Over such long periods, attackers can accumulate ciphertext, perform offline cryptanalysis, or leverage future computational advancements (e.g., cloud clusters, specialized hardware, or quantum-assisted methods) to recover long-term keys. Once the key is compromised, all past and future communications can be decrypted or forged.

- **Preconditions:**
  - Device uses a single static public–private key pair for its entire operational lifespan.
  - No periodic key rotation, re-keying, or forward-secure mechanism is implemented.
  - Attacker can collect encrypted traffic over months or years.
- **Steps:**
  1. Adversary: Continuously captures encrypted MQTT-SN traffic from the device over an extended period.
  2. Adversary: Uses increasing computational resources (e.g., GPU clusters, ASICs, cloud compute) to perform offline attacks on the long-term key pair.
  3. Adversary: Eventually derives the private key or exploits weakened cryptographic strength due to aging algorithms.
- **Post Conditions:**
  - Attacker can decrypt all historical MQTT-SN messages (loss of confidentiality).
  - Attacker can forge valid signatures or authentication messages (loss of authentication and integrity).
  - Attacker can impersonate the device or inject malicious commands into the system.
  - Entire system trust collapses because the compromised key cannot be revoked on constrained devices lacking update mechanisms.

V2S-Threat 6

- Threat: Attackers may use jamming devices to disrupt the radio frequencies used by V2V communication, preventing legitimate vehicles from communicating with each other. [Availability]
- Attack Name: Jamming
- Level of Attacker Difficulty: High
- Threat Severity: High
- Threat Probability: Medium
- Summary: An attacker may cause a signal-jamming attack and stop ClientApp from sending any message to the emergency services.
- Preconditions:
  - The attacker should be present in the communication range of the ClientApp and ForwardApp.
  - The attacker may have access to the signal jammer device.
- Step: The attacker uses a signal jammer to stop the communication between ClientApp and road-side infrastructure, or Gateway and Control Centre.
- Postcondition: The attacker tries to make sure the emergency services are inaccessible to the end-user.

V2S-Threat 7

- Threat: Attackers may try to subscribe to the messages sent to control centre. [Authorization]

- Attack Name: Physical Attack
- Level of Attacker Difficulty: Low
- Threat Severity: High
- Threat Probability: High
- Summary: An attacker may subscribe to the messages sent by ClientApp to control centre for the emergency services.
- Precondition: The attacker should know the topics of the ClientApp publishing information.
- Step: The attacker sends a subscribe message to the control center with the known topics to receive the messages sent by the ClientApp.
- Postcondition: The attacker tries to provide the physical harm after getting all the information from control centre.

V2S-Threat 8

- Threat: Attackers may tamper the ClientApp. [Authorization]
- Attack Name: Denial of Service
- Level of Attacker Difficulty: High
- Threat Severity: High
- Threat Probability: Low
- Summary: An attacker may tamper with the ClientApp such that in case of emergencies it should not be able to send emergency messages to control centre.

- Precondition: The attacker should have physical access to the ClientApp.
- Step: The attacker may tamper with the ClientApp.
- Postcondition: The attacker tampers in such a way that ClientApp is unable to send any message.

4.3.4 Security Requirements for V2V Use Case Scenario

We have identified the above security objectives and threats based on the need for the V2V communication use case. Here, we try to define security requirements based on security objectives and the threat model defined earlier:

SR1 - Data Origin Authentication: The broker transmits the information to the subscriber. A smartphone application, also known as a subscriber, should have the capability to offer the end-user a method, such as a digital signature or Message Authentication Code (MAC), to authenticate the origin of information. In the context of V2V communication, emergency services need to be able to authenticate the source of a status message received from a ClientApp.

SR2 - Identification: The broker must verify the identity of an publisher and subscriber. In case of V2V communication, emergency services subscribing to the topics at control centre and also the check identity of the publishers (i.e., ClientApp). We can use an unique identity which is only known to each vehicle (i.e., ClientApp)

SR3 - Information Confidentiality: The publisher will employ cryptography, namely cryptographic techniques and key sizes, to ensure the confidentiality of the payload data throughout the whole communication process. The payload data must undergo encryption before reaching the destination, specifically the subscribers.

Subsequently, the data remains encrypted at the Control Centre until it is transmitted to the emergency services.

SR4 - Access Control: The broker should employ an access control mechanism so that any unauthorized user should not have access to message which they are not entitled for. In the case of V2V communication, this requirement is applicable to control centre. The control center should make sure proper emergency service should subscribe to right topic.

SR5 - Information Integrity: The publisher transmits the information to the subscriber. A smartphone application should have the capability to offer the end-user the necessary tools (such as digital signature, MAC, and hashing) to identify any abnormalities in the MQTT-SN payload, including modification, deletion, insertion, replay, and other integrity issues, from the beginning to the end. In the context of V2V communication, emergency services should also have the measure to check the integrity of the status message received from the ClientApp.

SR6 - Message Header Integrity: The publisher transmits the information to the gateway. A smartphone application should have the capability to offer the end-user the necessary tools (such as digital signature, MAC, and hashing) to identify any abnormalities in the MQTT-SN header, including modification, deletion, insertion, replay, and other integrity issues, that may occur during transmission between two points. Likewise, the message header transmitted by the publisher (ClientApp) must remain unchanged until it reaches the subscribers (Emergency Services). Publishers/subscribers can rely on the broker. Regarding V2V communication, emergency services can check the integrity of the status message header received from the ClientApp.

SR7 - Limited Publication: The broker incorporates a security measure called "Handling Duplicate ClientID" to guarantee the continuous availability of services. Additionally, the gateway device includes functionality that checks the size of response messages to further ensure service availability. With respect to V2V communication, emergency services can eliminate duplicate status signals.

SR8 - Physical Security: The end-devices should be protected in such a way that it should not be tampered with. In case of V2V communication, ClientApp should be protected against physical tampering.

4.3.5 Analysis of Security Requirements for a V2V Scenario

Our security requirements based on security goals have been categorized in this section. They have also been assigned priorities based on the threats addressed by different security requirements and their severity, probability and attacker's difficulty. We quantify the various threat severity, threat likelihood, and attacker difficulty metrics in order to determine the risk based on attacker difficulty and give priority. In practical risk assessments, numerical figures for threat probability, severity, and attacker's difficulty are often not trusted enough. Therefore, a qualitative approach is commonly used to rate risks. As a result, a categorical rating system (low, medium, high) is utilized to determine the level of risk, as per the citation [67]. The formula below is taken from Chapter 1 of [67]. We have modified it instead of vulnerability; we have involved the attacker's difficulty in calculating the risk involved.

$$\textit{Risk (R)} = \textit{Threat Severity (C)} \times \textit{Threat Probability (T)} \times \textit{Attacker's Difficulty (D)}$$

The three essential characteristics that are used to calculate risk have numerical values are shown in Table 4.1. The higher numerical value of a parameter value signifies greater

risk than a lower numerical value. For example, in the case of an attacker's difficulty, if the attack is more straightforward for an adversary, the value is assigned three (i.e., more risk). On the other hand, if the attack is difficult for the attacker to execute, the value is assigned one (i.e., less risk). Similarly, for threat severity, if the threat is severe, it can cause more damage, and then the value is assigned three (i.e., more risk). Despite this, if the threat is less severe, which means minor damage to the system, then the value assigned is one (i.e., less risk). Furthermore, for threat probability, if the threat is more likely to occur in the attack, then the value is assigned three (i.e., more risk). However, if the threat is less likely to occur in the attack, the value assigned is one (i.e., less risk). In Table 4.2 and Figure 4.2, all three parameters will help us calculate the risk value for each threat mentioned in Section 4.4.3 for V2V communication.

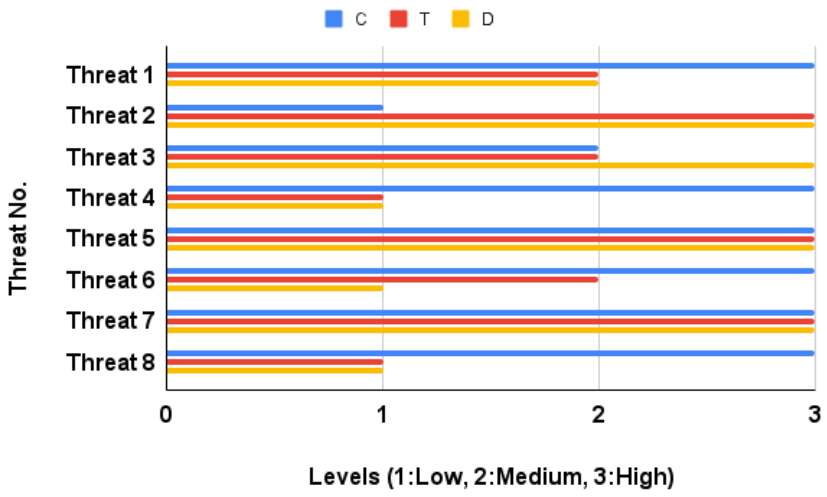


Figure 4.2: Threat vs C, T, and D for V2V Scenario

Table 4.3 illustrates which risks are being addressed by the security standards. We determine the overall risk that a security requirement addresses and then prioritize the requirements.

Table 4.1: Quantification of Different Threat Model Parameters

Parameter	**Classification**	**Value**
Attacker's Difficulty	Low (easier for attacker)	3
	Medium	2
	High (harder for attacker)	1
Threat Severity	Low (not too severe)	1
	Medium	2
	High (Severe)	3
Threat Probability	Low (less likely)	1
	Medium	2
	High (more likely)	3

Table 4.2: Risk Assessment of all the Threats for V2V Communication

Threat No.	**Threat Severity (C)**	**Threat Probability (T)**	**Attacker's Difficulty (D)**	**Risk (R) = C x D x T**
V2S-Threat 1	3	2	2	12
V2S-Threat 2	1	3	3	9
V2S-Threat 3	2	2	3	12
V2S-Threat 4	3	1	1	3
V2S-Threat 5	3	3	3	27
V2S-Threat 6	3	2	1	6
V2S-Threat 7	3	3	3	27
V2S-Threat 8	3	1	1	3

4.4 Security Requirements for Smart Lock Communication Model based on MQTT-SN

4.4.1 Smart Lock Use Case Scenario

Smart Lock [153] is a Wi-Fi, or Bluetooth-enabled IoT device, that provides keyless access to the home and locks/unlocks when instructions from an authorized device are received.

Table 4.3: Prioritizing Security Requirements of V2V Communication based on Risk Assessment

Security Goal	**Security Requirements**	**[Threat, Risk]**	**Total Risk = Sum (Ri)**	**Priority**
Identification	SR2: Identification	[V2S-Threat 1, 12]	12	6
Authentication	SR1: Data Origin Authentication	[V2S-Threat 1, 12], [V2S-Threat 3, 12]	24	2
Confidentiality	SR3: Information Confidentiality	[V2S-Threat 2, 9] , [V2S-Threat 1, 12]	21	3
Integrity	SR5: Information Integrity (Payload)	[V2S-Threat 4, 3], [V2S-Threat 3, 12],	15	4
	SR6: Message Header Integrity	[V2S-Threat 3, 12], [V2S-Threat 4, 3]	15	5
Availability	SR7: Limited Publication	[V2S-Threat 6, 6]	6	7
Authorization	SR4: Access Control	[V2S-Threat 7, 27]	27	1
Physical Security	SR8: Physical Protection	[V2S-Threat 8, 3]	3	8

Smart Locks use two different kinds of architecture: indirect (Device-Gateway-Cloud) and direct (Device-Cloud). Our focus is on indirect architecture, where the term *broker* represents cloud. In the indirect architecture, shown in Figure 4.3, the Smart Locks uses a gateway device to communicate to the MQTT broker. The Smart Lock needs a communication protocol to send status and receive commands from the end-user, i.e., MQTT-SN (as shown in Figure 4.3). The communication between the Smart Lock and the gateway device is MQTT-SN over BLE/Zigbee. The communication between the gateway device and the broker is MQTT over TLS. The communication between the broker and the subscriber's end gateway device is also MQTT over TLS. These are two different TLS channels, as shown in Figure 4.3. The access point at the subscriber end can be trusted or untrusted based on location, whether it is a home router or public Wi-

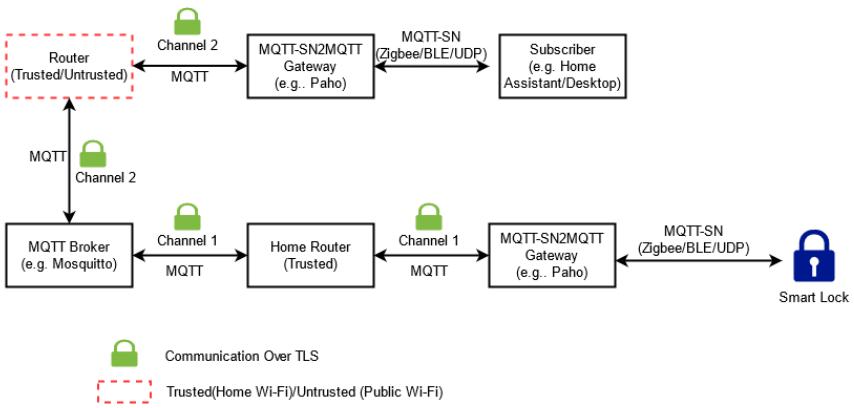


Figure 4.3: Smart Lock Node Communicating using MQTT-SN over Zigbee/BLE

Fi access point. This DGC (Device-Gateway-Cloud) architecture supports constrained IoT devices.

In this document, when we mention the security of Smart Locks, and different attacks on Smart Locks, we mainly refer to the attackers exploiting communication protocol to achieve the desired goal. Here, we will focus on securing how Smart Lock connects with the gateway, securing information Smart Lock sends and receives from end-users. In a nutshell, we will focus on the security of Smart Locks by securing the MQTT-SN communication protocol.

4.4.2 Identify Critical/Non-critical Assets in Smart Lock Scenario

In this step, we will cover different critical and non-critical assets in Smart Lock communication architecture and functional requirements:

1. Assets: The assets can be physical components and non-physical components (like messages).

   (a) Smart Lock: A Class IV device [154] with no OS installed, no user in-

put/output interface, and no visible port for configuration. Smart Lock communicates using MQTT-SN over BLE/Zigbee.

(b) Gateway Device: It is a Class I device [154] with capabilities to handle multiple smart home devices communicating over MQTT-SN. In addition, it serves as a communication portal for IoT devices, which do not have the capabilities to communicate over the Internet. The Gateway device communicates with IoT devices over BLE and the MQTT Broker over the Internet through a router (i.e., access point). There are no user input/output interfaces except one physical port, which is used to configure it with the help of a laptop or desktop by the user. The Gateway device converts MQTT-SN messages received from Smart Lock to MQTT messages and sends them to the broker.

(c) Access Point (Router): An access point is a router that helps the gateway device connect to the broker.

(d) Smartphone Application (SmartApp): This is used on a user's smartphone to subscribe to the status of Smart Lock and publish control commands (i.e., Lock/Unlock) to the Smart Lock.

(e) Broker: MQTT broker resides over the cloud. The role of a broker is to collect and distribute messages. It also accepts network connections from clients, subscribes and unsubscribes requests from clients, and closes the client's network connection.

(f) Information/ Data:

- Smart Lock Status Messages: It represents the current state of Smart Lock (i.e., Locked/Unlocked) sent from Smart Lock to the end user.
- Smart Lock Control Commands: Represents commands sent by the user to the Smart Lock.

- User Credentials: Login information that user uses to authenticate themselves to server.
- ClientID: Unique end-device information used to maintain sessions.
- Topics: Important parameter under which an end device publishes/subscribes information.

2. Functional Requirements: There are two main functional requirements for Smart Lock:

   (a) Smart Lock should send a status message to the end-user through the broker periodically or when requested.

   (b) Smart Lock should lock and unlock based on the command is received from an authorized end-user.

4.4.3 Threat Model for Smart Lock Scenario

After defining definitions and critical and non-critical assets, we design the threat model for a Smart Lock.

SLN-Threat 1

- Threat: The broker allows the adversary as if the adversary had a valid ClientID to send malicious control/data messages. [Authorization]
- Attack Name: Spoofing Attack
- Level of Attacker Difficulty: Medium
- Threat Severity: High
- Threat Probability: Medium

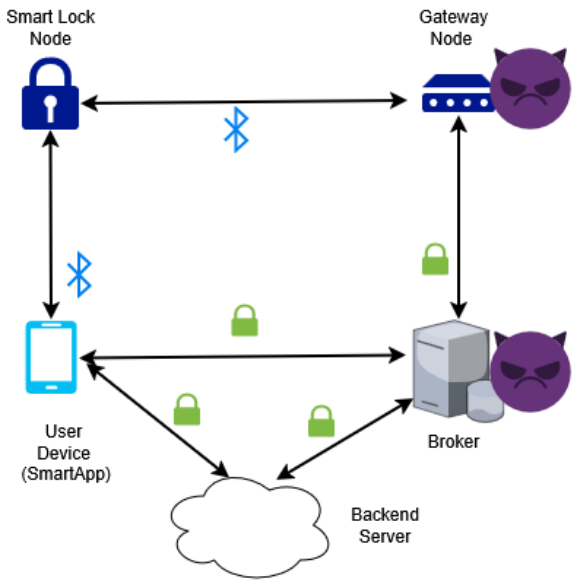


Figure 4.4: Threat Model for Proposed Smart Lock Node Architecture

- Summary: The external attacker type guesses the unique ClientID of a Smartphone application and publishes malicious control commands messages for the Subscriber (i.e., Smart Lock) through the broker. The objective of the attacker is to get access to the end-user home.

- Preconditions:

  - Attacker has access to a Smartphone application, which the attacker uses to deduce ClientID by guessing or reverse engineering based on the format used by manufacturers [18].

  - Attacker has an authorized account with the broker.

  - Attacker has an explicit knowledge of the structure and meaning of the ClientID and topics.

- Steps:

    1. Smart Lock: The Smart Lock subscribes to the topics published by the Smartphone application.
    2. Adversary: The external attacker publishes to all control commands under the topics linked ClientID of the Smartphone application.
    3. Broker: The broker publishes the information related to the topics linked to the ClientID of the Smartphone application to the Smart Lock subscribing to the topics.

- Post Conditions:

    – Attacker has successfully unlocked the end-user home.

SLN-Threat 2

- Threat: The attacker gain access to unauthorized resources. [Confidentiality]
- Attack Name: Passive Eavesdropping
- Level of Attacker Difficulty: Low
- Threat Severity: Medium
- Threat Probability: High
- Summary: The external attacker subscribes to the wildcard subscription (i.e., #) and gains access to all the messages published by the Publisher for the Subscriber through the broker. The objective of the attacker is to know the end-user routine of accessing Smart Lock.

Note: The external attacker can subscribe to all the information from the broker by subscribing to a wildcard topic of a guessed ClientID. The attacker has physical access to a similar Smart Lock, which the attacker uses to deduce ClientID by guessing or reverse engineering based on the format used by manufacturers [18].

- Preconditions:
  - Attacker has physical access to a similar Smart Lock.
  - Attacker has an authorized account with the broker.
  - Attacker has explicit knowledge of the structure and meaning of the ClientID.
- Steps:
  1. Adversary: The external attacker subscribes to all the information linked to ClientID using a wildcard subscription. The attacker may or may not use guessed ClientID.
  2. Smartphone Application: The SmartApp publishes the control commands.
  3. Smart Lock: The Smart Lock publishes the status messages.
  4. Broker: The broker sends/publishes all the information it has related to all the topics to the adversary.
- Post Conditions:
  - Attackers can use the information to understand the end-user routine to infiltrate the end-user's home.

SLN-Threat 3

- Threat: The attacker can replay old messages to misguide the end-user/Smart Lock. [Integrity]

- Attack Name: Replay
- Level of Attacker Difficulty: High
- Threat Severity: Medium
- Threat Probability: Low
- Summary: The external attacker can copy the packet published by the Smart Lock or end-user and replay it back to the subscriber through a broker, which will give false information to the end-user in case of status message or may be used to open Smart Lock in case of the control command message. Thus, the attacker's objective is to misguide the end-user or Smart Lock device to access the home.
- Preconditions:
  - Attacker has an authorized account with the broker.
  - Attacker has the means to intercept and capture a message from the publisher to the broker.
- Steps:
  1. Smartphone Application/ Smart Lock: The publisher publishes a control/status message to the broker.
  2. Adversary: The external attacker intercepts the message and saves it. The external attacker publishes the saved message.
  3. Broker: The broker sends/publishes the old message to all the subscribers subscribing to that topic.

- Post Conditions:
  - Broker does not recognize the old message.
  - Attacker can replay the command to unlock the Smart Lock to infiltrate the end-user's home.

SLN-Threat 4

- Threat: The attacker can intercept and modifies messages to misguide the end-user/Smart Lock. [Integrity]
- Attack Name: Injection Attack
- Level of Attacker Difficulty: High
- Threat Severity: High
- Threat Probability: Low
- Summary: The external attacker can copy the packet published by the Smart Lock or end-user and modifies it, and forwards the modified message to the broker for the subscribers, which will give false information to the end-user in case of status message or may be used to open Smart Lock in case of the control command message. Thus, the attacker's objective is to misguide the end-user or Smart Lock device to access the home.
- Preconditions:
  - Attacker has physical access to a similar Smart Lock.
  - Attacker has an authorized account with the broker.

  - Attacker can intercept, capture, and modify a message from the publisher to the broker.

- Steps:

  1. Smartphone Application/ Smart Lock: The publisher publishes a control/status message to the broker.
  2. Adversary: The external attacker intercepts the message and modifies it. The external attacker publishes the modify message.
  3. Broker: The broker sends/publishes the corrupted message to all the subscribers (i.e., Smart Lock/ SmartApp) subscribing to that topic.

- Post Conditions:

  - Broker does not recognize the old message.
  - Attacker can send the command to unlock the Smart Lock to infiltrate the end-user home.
  - Attacker can send the wrong status of Smart Lock to the end-user.

SLN-Threat 5

- Threat: The attacker can set up a malicious gateway to intercept and modifies messages to misguide the end-user/Smart Lock. [Integrity]
- Attack Name: Man in the Middle
- Level of Attacker Difficulty: Medium
- Threat Severity: High
- Threat Probability: Medium

- Summary: The external attacker may set up a malicious Gateway device during which the gateway discovery/onboarding process of the Smart Lock. The objective of the attacker is to misguide the end-user or Smart Lock device to access the home.
- Preconditions:
  - Attacker intercepts the discovery message for the gateway and the first connection message to the broker.
- Steps:
  1. Smart Lock: The Smart Lock broadcasts discovery messages for the search of gateway/broker.
  2. Adversary: The external attacker intercepts the message and replies with its own credentials as a gateway and also advertises its own gateway credentials.
- Post Conditions:
  - Broker does not recognize the corrupted message coming from a malicious gateway device.
  - Attacker can send the wrong status of Smart Lock to the end-user.
  - Attacker can send the command to unlock the Smart Lock to infiltrate the end-user's home through the malicious gateway.

SLN-Threat 6

- Threat: The rogue broker can modify messages to misguide the end-user/Smart Lock. [Integrity]
- Attack Name: Man in the Middle

- Level of Attacker Difficulty: High
- Threat Severity: High
- Threat Probability: Low
- Summary: The rogue broker can modify the message to the subscribers, which will give false information to the end-user in case of status message or may be used to open Smart Lock in case of the control command message. The objective of the rogue broker is to misguide the end-user or Smart Lock device to access the home.
- Preconditions:
  - Rogue broker has access to all the information from the publisher and subscriber.
- Steps:
  1. Smart Lock/ Smartphone Application: The publisher publishes a message to the broker.
  2. Adversary/Broker: The rogue broker intercepts the message and modifies it, and sends corrupt messages to the subscribers.
- Post Conditions:
  - Rogue broker can send the command to unlock the Smart Lock to infiltrate the end-user home.
  - Rogue broker can send the wrong status of Smart Lock to the end-user.

4.4.4 Security Requirements for Smart Lock Usecase Scenario

We have identified the above security objectives and threats based on the need for the Smart Lock use case. Here, we try to define security requirements based on security objectives and the threat model defined earlier:

SR1 - Data Origin Authentication: A broker sends the information to the subscriber. A smartphone application (i.e., subscriber) should be able to provide the means [assignment: digital signature/MAC (Message Authentication Code)] to the end-user to verify the source of information. Using similar means, a Smart Lock (i.e., subscriber) can verify the smartphone application (i.e., publisher) from which it is receiving control commands.

SR2 - Device Authentication: The broker should authenticate an end-user device using credentials [assignment: username/password, challenge-response technique, public-key certificate].

SR3 - Information Confidentiality: The publisher shall use cryptography [assignment: cryptographic algorithms and key sizes] to protect the confidentiality of the payload data from end to end. The payload data should be encrypted until it reaches the destination (i.e., subscribers). Then, the data remains encrypted at the broker to protect from adversaries knowing Smart Lock's status or control command for Smart Lock.

SR4 - Message Header Confidentiality: The publisher shall use cryptography [assignment: cryptographic algorithms and key sizes] to protect the confidentiality of the MQTT-SN/MQTT header from point to point. The header will be available in plaintext at the gateway device and broker to perform conversion and filtering, re-

spectively. MQTT Header from a publisher or subscriber to the broker is encrypted, but data is available in plaintext at the publisher, subscriber, and broker.

SR5 - Information Integrity: A publisher sends the information to the subscriber. A smartphone application (i.e., subscriber) should be able to provide the means [assignment: digital signature/MAC (Message Authentication Code)/ Hashing] to the end-user to detect the [selection: modification, deletion, insertion, replay and other integrity] anomalies in the MQTT-SN payload from end to end. Similarly, the control message sent by the publisher (i.e., SmartApp) should not be altered till it reaches the subscribers (i.e., Smart Lock).

SR6 - Message Header Integrity: A publisher sends the information to the gateway. A smartphone application (i.e., subscriber) should be able to provide the means [assignment: digital signature/MAC (Message Authentication Code)/ Hashing] to the end-user to detect the [selection: modification, deletion, insertion, replay and other integrity] anomalies in the MQTT-SN header from point to point. Similarly, the message header sent by the publisher (i.e., SmartApp) should not be altered till it reaches the subscribers (i.e., Smart Lock). It is assumed that publishers/subscribers can trust the broker. If a subscriber subscribes to the topic "lock/status" or "lock/cmd", the subscriber should receive a message published under that topic, similar to publishers.

SR7 - Publisher/Subscriber (i.e., End Device) Access Control: The Smart Lock should decide using some means [assignment: access control mechanism] which device it wants to send and receive information from.

4.4.5 Analysis of Security Requirements for a Smart Lock Scenario

In this section, we have categorized our security requirements based on security goals. We have also assigned them priority based on the threats addressed by different security requirements and their severity, probability and attacker's difficulty. To assign priority, we quantify the different values of the attacker's difficulty, threat severity and threat probability to calculate the risk [67, Chapter 1]. We have modified the formula; instead of vulnerabilities, we have used the attacker's difficulty.

Risk (R) = Threat Severity (C) × Threat Probability (T) × Attacker's Difficulty (D)

In Table 4.1, we have assigned a numerical value to the three crucial parameters used to calculate risk. The higher numerical value of a parameter value signifies greater risk than a lower numerical value. For example, in the case of an attacker's difficulty, if the attack is more straightforward for an adversary, the value is assigned three (i.e., more risk). On the other hand, if the attack is difficult to execute for the attacker, the value is assigned one (i.e., less risk). Similarly, for threat severity, if the threat is severe, it can cause more damage, then the value is assigned three (i.e., more risk). Also, if the threat is less severe, which means minor damage to the system, then the value assigned is one (i.e., less risk). Furthermore, for threat probability, if the threat is more likely to occur in the attack, then the value is assigned three (i.e., more risk). However, if the threat is less likely to occur in the attack, the value assigned is one (i.e., less risk). Therefore, all these three parameters will help us calculate the risk value for each threat, as shown in Table 4.4.

Here, we have shown that the security requirements are addressing previously mentioned threats. We calculate the total risk addressed by a security requirement and assign priority based on it as shown in Table 4.5.

Table 4.4: Risk Assessment of all the Threats for Smart Lock Scenario

Threat No.	**Threat Severity (C)**	**Threat Probability (T)**	**Attacker's Difficulty (D)**	**Risk (R) = C x D x T**
SLN-Threat 1	3	2	2	12
SLN-Threat 2	2	3	3	18
SLN-Threat 3	2	1	1	2
SLN-Threat 4	3	1	1	3
SLN-Threat 5	3	2	2	12
SLN-Threat 6	3	1	1	3

Table 4.5: Prioritizing Security Requirements based on Risk Assessment for Smart Lock Scenario

Security Goal	**Security Requirements**	**[Threat, Risk]**	**Total Risk = Sum (Ri)**	**Priority**
Authentication	SR1: Data Origin Authentication	[SLN-Threat 1, 12], [SLN-Threat 6, 3]	15	5
	SR2: User Authentication	[SLN-Threat 1, 12]	12	6
Confidentiality	SR3: Information Confidentiality (Payload)	[SLN-Threat 1, 12], [SLN-Threat 2, 18]	30	1
	SR4: Message Header Confidentiality	[SLN-Threat 2, 18]	18	3
Integrity	SR5: Information Integrity (Payload)	[SLN-Threat 3, 2], [SLN-Threat 4, 3], [SLN-Threat 5, 12], [SLN-Threat 6, 3], [SLN-Threat 1]	21	2
	SR6: Message Header Integrity	[SLN-Threat 3, 2], [SLN-Threat 4, 3], [SLN-Threat 5, 12]	17	4
Authorization	SR7: Publisher/ Subscriber Access Control	[SLN-Threat 1, 12]	12	7

Through the utilization of the threat model and security requirements, we have determined that the end-to-end security concept in the publish-subscribe protocol is more complex compared to the client-server protocol. The MQTT-SN packet was partitioned into two distinct elements: the payload containing the information and the message header. Given that each intermediate node needs only the MQTT-SN header to carry out its functions, we have proposed a hybrid security model for a publish-subscribe protocol that ensures both confidentiality and integrity. Regarding the MQTT-SN payload, we propose implementing end-to-end security. It is hard to provide end-to-end security where the publisher and subscriber do not know each other. As for the MQTT-SN header, we advise implementing point-to-point security.

4.5 Security Requirements for CGM Communication Model based on MQTT-SN

4.5.1 CGM Use Case Scenario

A general CGM working model shows how glucose data moves from the body to the user or clinician in a loop that never stops. A small sensor placed under the skin measures glucose levels in interstitial fluid every few minutes and converts these biochemical readings into electrical signals (Figure 4.5). A small transmitter on the sensor sends it wirelessly, usually over BLE, to a paired device, such as a smartphone, insulin pump, or dedicated receiver. The receiver receives incoming values, calibrates them if necessary, and displays real-time glucose trends, alerts, and predictive warnings. Many systems also send readings to a cloud platform, where they are stored, analysed, and, optionally, shared with caregivers or healthcare providers for remote monitoring. This cycle of sensing, sending, and displaying data lets users track changes in their glucose levels throughout the

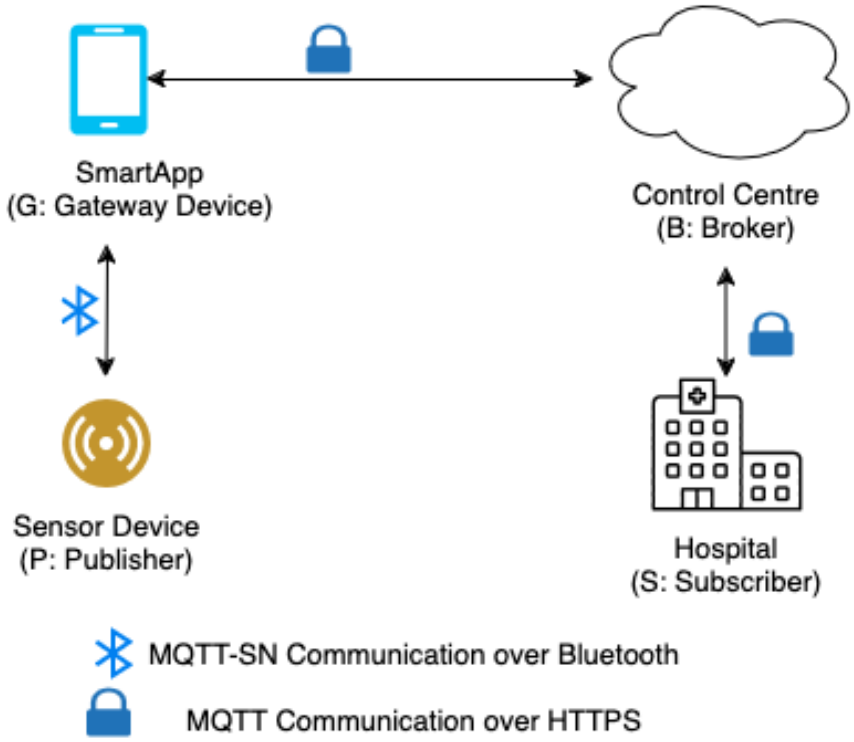


Figure 4.5: Communication Model of Our CGM System

day and night, helping them make better choices about their diet, exercise, and insulin management. As you work on your architecture, it might help to see how this general workflow aligns with the MQTT-SN communication model. We also cover the threats that affect the CGM communication model and suggest a security architecture.

4.5.2 Identify Critical/Non-critical Assets in CGM Scenario

In this step, we will cover different critical and non-critical assets in CGM communication architecture and functional requirements:

1. Assets: The assets can be physical components and non-physical components (like messages).

   (a) CGM: A Sendor device [154] with no OS installed, no user input/output inter-

face, and no visible port for configuration, which communicates using MQTT-SN over BLE.

(b) Gateway Device (SmartApp): It is a Class I device [154] with capabilities to handle multiple smart home devices communicating over MQTT-SN. In addition, it serves as a communication portal for IoT devices, which do not have the capabilities to communicate over the Internet. The Gateway device converts MQTT-SN messages received from CGM device to MQTT messages and sends them to the broker.

(c) Broker: MQTT broker resides over the cloud. The role of a broker is to collect and distribute messages. It also accepts network connections from clients, subscribes and unsubscribes requests from clients, and closes the client's network connection.

(d) Information/ Data:

- CGM Data Messages: It represents the data of the CGM value to the end user,i.e., hospital, doctors.
- User Credentials: Login information that user uses to authenticate themselves to server.
- ClientID: Unique end-device information used to maintain sessions.
- Topics: Important parameter under which an end device publishes/subscribes information.

2. Functional Requirements: There are two main functional requirements for CGM:

(a) CGM device should send a monitored value to the end-user through the broker periodically or when requested.

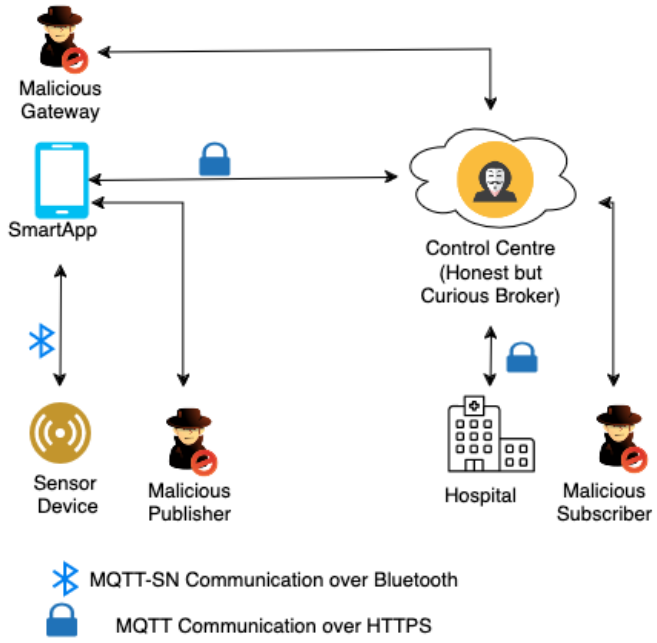


Figure 4.6: Threat Model of CGM System

4.5.3 Threat Model for a CGM Communication Model

CGM threats below are used to derive the security requirements of CGM communication. We defined them based on our earlier research [24]. Figure 4.6 shows the malicious components of the mentioned architecture in the CGM system.

In this section, threats to the CGM communication system are defined.

CGM-Threat 1

- Threat: Unauthorized individuals or entities may attempt to intercept and eavesdrop on sensor data to obtain sensitive information to manipulate insulin intake. [Confidentiality]
- Attack Name: Passive Eavesdropping
- Level of Attacker Difficulty: Easy

- Threat Severity: Low
- Threat Probability: High
- Summary: The external attacker subscribes to the wildcard subscription (i.e., #) and gains access to all the messages published by the sensor device to the hospital services through the broker. The objective of the attacker is to know the sensitive information.
- Preconditions:
  - The attacker has an authorized account with the Control Centre.
  - The attacker has explicit knowledge of the structure and meaning of the ClientID.
- Steps:
  1. Adversary: The external attacker subscribes to all the information linked to Client ID using a wildcard subscription. The attacker may or may not use guessed ClientID. The attacker may also listen to the SmartApp messages coming from the Sensor device to SmartApp.
  2. ClientApp: The ClientApp publishes the status message.
  3. Control Centre: The broker sends/publishes all the information related to all the topics to the adversary.
- Postcondition: The attacker can use the information to understand the patient's health data and information.

CGM-Threat 2

- Threat: The Control centre authorizes the adversary as if the adversary had a valid ClientID to send or receive malicious patient information messages. [Authorization]
- Attack Name: Spoofing Attack
- Level of Attacker Difficulty: Medium
- Threat Severity: High
- Threat Probability: Medium
- Summary: The malicious publisher type guesses a unique ClientID (i.e. serial number) of the sensor device and publishes malicious messages for the Subscriber (i.e., hospital services) through the broker. Similarly, a malicious subscriber can subscribe for the patient's data. The objective of the attacker is to manipulate insulin monitoring services.
- Preconditions:
  - The attacker has access to a sensor device, which the attacker uses to deduce ClientID by guessing or reverse engineering based on the format used by manufacturers [155].
  - The attacker has an authorized account with the control centre.
  - The attacker has an explicit knowledge of the structure and meaning of the ClientID and topics.
- Steps:
  1. Hospital: The emergency services subscribe to the topics published by the Sensor device.

2. Adversary: The external attacker publishes or subscribes to all messages under the topics linked to the Client ID of the ClientApp.
3. Control Centre: The Control Centre forwards the information related to the topics linked to the ClientID of the sensor device to the hospital services subscribing to the topics.

- Postcondition: The attacker has successfully been able to send the wrong information about the patient to the hospital services.

CGM-Threat 3

- Threat: The attacker can intercept and modify messages to misguide the hospital services. [Integrity]
- Attack Name: Injection Attack
- Level of Attacker Difficulty: High
- Threat Severity: High
- Threat Probability: Low
- Summary: The external attacker can copy the message published by the Sensor device, modify it, and forward the modified message to the Control Centre for the hospital, which will give false information to the hospital in case a patient's glucose data. Thus, the attacker's objective is to misguide the hospital.
- Preconditions:
  - The attacker has physical access to a Sensor device.
  - The attacker has an authorized account with the Control Centre.

  - The attacker can intercept, capture, and modify a message from the Sensor device to the Control Centre.

- Steps:

  1. ClientApp: The Sensor device publishes the patient data to the Control Centre.
  2. Adversary: The external attacker intercepts the message and modifies it. The external attacker publishes the modified message.
  3. Broker: The Control Centre sends/publishes the corrupted message to the hospital services subscribing to that topic.

- Postconditions:

  - The Control Centre does not recognize the modified message.
  - The attacker can send the status message to misguide the hospital services.
  - The attacker can send the patient's wrong data, which impacts the patient's insulin amount.

CGM-Threat 4

- Threat: An attacker intercepts communication between the CGM device and a paired device (like a smartphone or cloud service). If the data is not adequately encrypted, the attacker can read, alter, or block data transmission, which could lead to incorrect data being displayed to the user or healthcare provider.
- Attack Name: Man-in-the-middle
- Level of Attacker Difficulty: High

- Threat Severity: High
- Threat Probability: Medium
- Summary: An attacker can cause a man-in-the-middle attack by capturing a sensor device message, either by placing itself between the sensor device and SmartApp or SmartApp and Control Centre. An attacker sends a malicious message to the Control Centre or SmartApp.
- Preconditions:
  - The attacker has information about the ClientId of the Sensor device.
  - The attacker may have valid authentication with the Control Centre.
- Steps:
  1. ClientApp: The ClientApp is connected with the broker under a Client ID.
  2. Adversary: The attacker used the ClientID to connect with the same Control Centre.
  3. Control Centre: The Control Centre accepts the connection from the adversary and sends malicious messages to the hospital services.
- Postcondition: The attacker tries to make sure that the malicious message has the wrong glucose data to impact the wrong insulin information.

CGM-Threat 5

- Threat: Attackers can record and replay legitimate Sensor device messages, which can have negative consequences, especially when safety-critical information is involved. [Integrity]

- Attack Name: Replay
- Level of Attacker Difficulty: Low
- Threat Severity: High
- Threat Probability: Medium
- Summary: The external attacker can copy the packet published by the Sensor device and replay it back to the hospital through a Control Centre, which will give false information. Thus, the attacker's objective is to misguide the hospital services.
- Preconditions:
  - The attacker has an authorized account with the Control Centre.
  - The attacker has the means to intercept and capture a message from the Sensor device to the Control Centre.
- Steps:
  1. ClientApp: The ClientApp publishes a control/status message to the Control Centre.
  2. Adversary: The external attacker intercepts the message and saves it. The external attacker publishes the saved message.
  3. Control Centre: The Control Centre sends the old message to all the emergency services subscribing to that topic.
- Postconditions:
  - The Control Centre does not recognize the old message.

– The attacker can replay the patient data to hospital services to cause the wrong insulin value.

4.5.4 Security Requirements for CGM Usecase Scenario

We have identified the above security objectives and threats based on the need for the CGM use case. Here, we try to define security requirements based on security objectives and the threat model defined earlier:

SR1 - Data Origin Authentication: A broker sends the information to the subscriber. A smartphone application (i.e., subscriber) should be able to provide the means [assignment: MAC (Message Authentication Code)] to the end-user to verify the source of information, i.e., CGM device.

SR2 - Device Authentication: The broker should authenticate an CGM device using credentials [assignment: username/password, challenge-response technique, public-key certificate].

SR3 - Information Confidentiality: The publisher shall use cryptography [assignment: cryptographic algorithms and key sizes] to protect the confidentiality of the payload data from end to end. The payload data should be encrypted until it reaches the destination (i.e., subscribers). Then, the data remains encrypted at the broker to protect from adversaries knowing CGM device data.

SR4 - Message Header Confidentiality: The publisher shall use cryptography [assignment: cryptographic algorithms and key sizes] to protect the confidentiality of the MQTT-SN/MQTT header from point to point. The header will be available in plaintext at the gateway device and broker to perform conversion and filtering, re-

spectively. MQTT Header from a publisher or subscriber to the broker is encrypted, but data is available in plaintext at the publisher, subscriber, and broker.

SR5 - Information Integrity: A publisher (i.e., CGM device) sends the information to the subscriber. A CGM device should be able to provide the means [MAC (Message Authentication Code)/ Hashing] to the end-user to detect the [selection: modification, deletion, insertion, replay and other integrity] anomalies in the MQTT-SN payload from end to end.

SR6 - Message Header Integrity: A publisher (i.e., CGM device) sends the information to the gateway (i.e., a smartphone application) should be able to provide the means [MAC (Message Authentication Code)/ Hashing] to the end-user to detect the [selection: modification, deletion, insertion, replay and other integrity] anomalies in the MQTT-SN header from point to point. Similarly, the message header sent by the publisher (via SmartApp) should not be altered till it reaches the subscribers. It is assumed that publishers/subscribers cannot trust the broker.

4.5.5 Analysis of Security Requirements for a CGM Scenario

In this section, using Table 4.1, we have assigned a numerical value to the three crucial parameters used to calculate risk. The higher numerical value of a parameter value signifies greater risk than a lower numerical value. Therefore, all these three parameters will help us calculate the risk value for each threat, as shown in Table 4.6.

Here, we have shown that the security requirements are addressing previously mentioned threats. We calculate the total risk addressed by a security requirement and assign priority based on it as shown in Table 4.7.

CGM communication involves multiple intermediaries—such as the wearable sensor, the transmitter, the smartphone, and the cloud platform—each of which performs

Table 4.6: Risk Assessment of all the Threats for CGM Scenario

Threat No.	**Threat Severity (C)**	**Threat Probability (T)**	**Attacker's Difficulty (D)**	**Risk (R) = C x D x T**
CGM-Threat 1	1	3	3	9
CGM-Threat 2	3	2	2	12
CGM-Threat 3	3	1	1	3
CGM-Threat 4	3	2	1	6
CGM-Threat 5	3	2	3	18

Table 4.7: Prioritizing Security Requirements based on Risk Assessment for CGM Scenario

Security Goal	**Security Requirements**	**[Threat, Risk]**	**Total Risk = Sum (Ri)**	**Priority**
Authentication	SR1: Data Origin Authentication	[CGM-Threat 1, 9]	9	4
	SR2: User Authentication	[CGM-Threat 1, 9]	9	4
Confidentiality	SR3: Information Confidentiality (Payload)	[CGM-Threat 1, 9], [CGM-Threat 4, 6]	15	2
	SR4: Message Header Confidentiality	[CGM-Threat 2, 12]	12	3
Integrity	SR5: Information Integrity (Payload)	[CGM-Threat 3, 3], [CGM-Threat 4, 6]	9	4
	SR6: Message Header Integrity	[CGM-Threat 5, 18]	18	1
Authorization	SR7: Publisher/ Subscriber Access Control	[CGM-Threat 2, 12]	12	3

specific processing tasks on the transmitted data.For the CGM data payload, we recommend implementing true end-to-end security between the CGM sensor (publisher) and

the cloud analytics platform or authorized clinician (subscriber). This is challenging because the sensor and final data consumer typically do not share a direct trust relationship or pre-established keys. Nonetheless, protecting medical data requires that only the intended endpoint can decrypt the glucose measurements.For the communication header, we recommend point-to-point security between each hop (sensor → transmitter → smartphone → cloud). This allows intermediate devices to perform routing, validation, and protocol-specific operations without exposing sensitive medical data. This hybrid approach balances the strict confidentiality requirements of medical data with the operational needs of multi-hop CGM communication.

4.6 Conclusion

Through the application of the threat model and the defined security requirements, we determined that achieving end-to-end security in a V2S, CGM and SLN ecosystem is more complex than in a traditional client-server architecture. By utilizing the priorities outlined in Tables 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7 we can develop a security framework that effectively addresses the requirements of each use case according to their respective priorities.

Chapter 5

Security Architecture for Vehicle to Emergency Services (V2S) Communicating Over MQTT-SN

5.1 Introduction

Vehicle-to-emergency services (V2S) communication is an essential component of vehicle-to-vehicle (V2V) communication. The significance of vehicle-to-emergency services communication lies in its ability to guarantee the safety and welfare of vehicle occupants, as well as expedite prompt emergency aid when necessary. Vehicles are able to communicate with each other using V2V technology. This technology enables automobiles to share information such as speed, position, and potential hazards over wireless connections like DSRC or cellular networks. V2V communication serves multiple purposes, including alerting drivers to potential hazards to prevent accidents, optimizing travel routes to reduce traffic congestion, and facilitating rapid emergency response via transmitting distress signals. The distress signal mechanism is crucial for ensuring the safety

and well-being of both occupants and other individuals on the road.

The V2S communication systems necessitate meticulous deliberation over security protocols to thwart illegal entry, guarantee the accuracy of data, and uphold the confidentiality of sensitive information. The C-V2X communication protocol is widely used in V2V communication. It depends on cellular network infrastructure, which might pose issues in terms of network availability, coverage, and reliability, especially in remote or rural locations. This highlights the research needs we aim to address. Implementing security architecture in V2S communication presents problems due to the distinct features of vehicular networks and the requirement for secure and effective communication between vehicles and emergency services. This chapter delves into the intricacies of security architecture in V2S communication over MQTT-SN, addressing the challenges posed by the unique characteristics of emergency scenarios and the need for secure and reliable data exchange. Ensuring the integrity and security of the V2S system using publish-subscribe systems presents problems related to end-to-end security. Based on our knowledge, there is no security standard designed for MQTT-SN, and very little research has been done in the field of MQTT-SN security [23] [17]. However, due to recent advancements, MQTT-SN is of great interest because of its lightweight and communication over layer 2 protocols. With the increased usage, protecting and securing communication is also necessary.

5.1.1 Contributions

We have focused on the weakest link of the MQTT-SN communication protocol, i.e., the link between the client and the gateway device via the MQTT-SN forwarder, which many researchers did not address for the MQTT-SN protocol. We have designed and implemented the end-to-end security architecture for the V2S prototype to send distress

signals using MQTT-SN in rural areas where any network is unavailable [5]. Also, compared our security architecture designed for the MQTT-SN publish-subscribe protocol with the currently available in the market and explored the viability of implementing public-key cryptography on the 32-bit microcontroller and performed timing analysis.

5.2 Vehicle to Vehicle Communication

V2V communication is an emerging field of study. It is utilized for several purposes, including providing information on road congestion, traffic obstruction, and accidents, as well as sending distress signals to request assistance in the event of possible and immediate peril. We have introduced the MQTT-SN communication paradigm, which employs the publish-subscribe approach to transmit a distress signal from a vehicle to emergency services. An emergency signal is transmitted utilizing MQTT-SN protocol over ZigBee network [5]. Figure 5.1 depicts the design framework we have developed for transmitting distress signals from a vehicle to emergency services. The distress signal data includes information about the vehicle's relative location as well as its model, kind, and year. At this location, we utilize the Vehicle Identification Number (VIN) as the ClientID. We utilize subject aliases, which are pre-established and encoded within the ClientApp or the vehicle's on-board unit. Figure 5.1 depicts the fundamental communication framework employed for transmitting distress alerts.

5.3 Threat Model

The main security requirement is to provide an end-to-end secure architecture for the V2S system. communication.

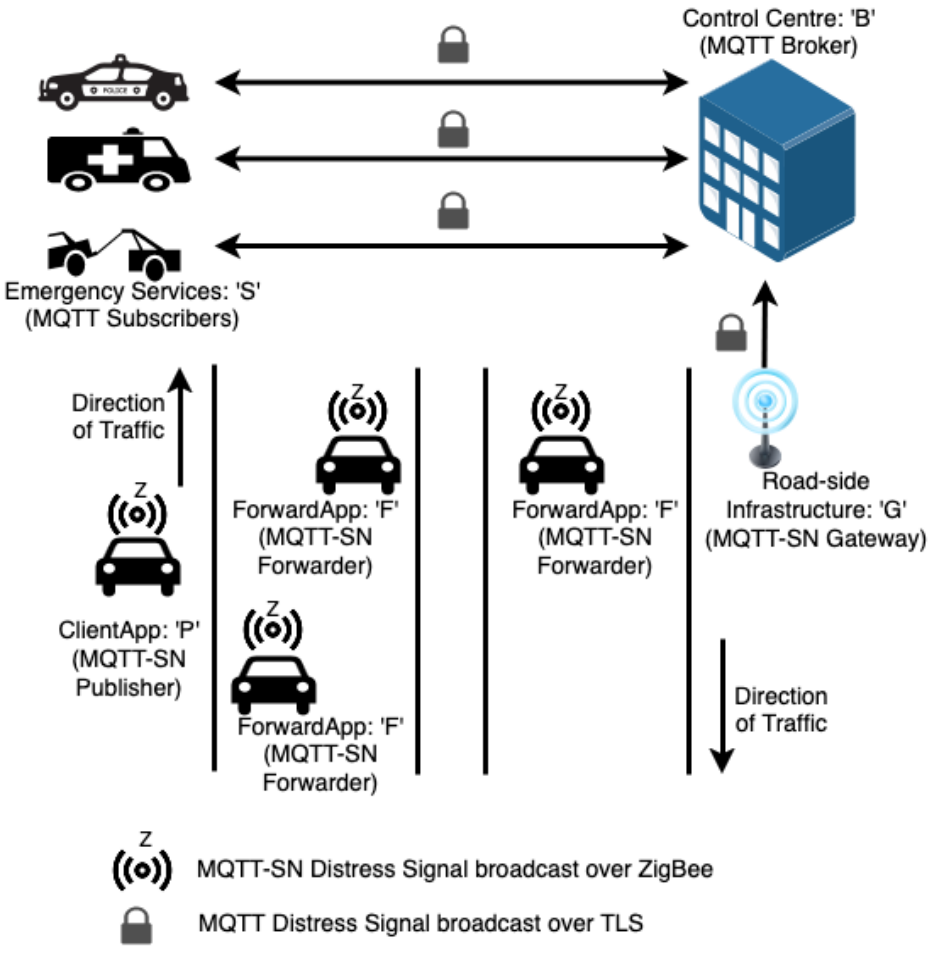


Figure 5.1: A Distress Signal using MQTT-SN over ZigBee [5]

Attacker's Goal and Capabilities: The attacker aims to read the payload information and harm the end-users. Here, we have considered four different attackers:

1. a malicious subscriber who wants to get all information from the control centre
2. a malicious FowardApp, which tries to stop the message from forwarding or also tries to tamper with the message
3. a *honest but curious* control centre, which tries to read the message data
4. a malicious publisher who wants to waste the resources of emergency services

All four attackers have various capabilities in our model. Attackers have information about the VIN and topics data published under it as these are plaintext information and are publicly available. Attackers also can tamper with messages and format new messages.

We have selected V2S-Threats T2-T6 below are used to derive security requirements of V2S communication [151]. Figure 5.2 shows the malicious components of the mentioned architecture in V2S.

- V2S-Threat 2 (V2S-T2): An adversary may attempt to eavesdrop on V2S communications to obtain sensitive information, such as location, speed, or other private data. [Confidentiality]
- V2S-Threat 3 (V2S-T3): An adversary may record and replay the old message to exhaust the emergency services resources. [Availability]
- V2S-Threat 4 (V2S-T4): An adversary may intercept and modify messages to misguide the emergency services. [Integrity].

- V2S-Threat 5 (V2S-T5): An adversary may send fake emergency message that can have negative consequences, especially when safety-critical information is involved. [Identification].

- V2S-Threat 6 (V2S-T6): Ensure that the message reaches the Gateway (road-side infrastructure) and emergency services. Even when An adversary may use jamming devices to disrupt the radio frequencies used by V2V communication, preventing legitimate vehicles from communicating with each other. [Availability]

Assumptions: We make the following assumptions for the prototype:

1. Each type of emergency service generates an RSA public-private key pair (e_S, d_S) during initialization, the public keys stored in a secured ROM (Read Only Memory) of the vehicle by the manufacturer.

2. At least one of the FowardApp is non-malicious.

3. FowardApp is always in receiving mode on running messages till it receives any message to broadcast.

4. Communicating details of the control centre is correctly programmed in the road-side infrastructure (i.e., Gateway Device), which can not be modified.

5. Our system does not support wildcard topics.

5.4 Design and Implementation

A primary goal of our prototype is to send a distress signal using MQTT-SN over ZigBee mentioned in [5]. This chapter focuses on one of a few security requirements to protect the design, i.e., identification, confidentiality, integrity, availability and accountability.The

encryption and decryption process is accomplished by utilizing RSA (Rivest-Shamir-Adleman) algorithm with a 2048-bit key. Additionally, the message header is verified using SHA-256. An essential criterion in this application is that publishers and subscribers remain unaware of each other. Thus, in order to preserve anonymity, we have implemented the publish-subscribe protocol, specifically MQTT-SN. The rationale for choosing MQTT-SN for this application has been extensively examined in [5]. The encrypted data from vehicles to emergency services is transmitted using RSA encryption with a key-size of 2048 bits. We have selected RSA because it has undergone thorough examination and analysis over an extended period of time, and its security is comprehensively understood and widely implemented. For ensuring security, we prioritize RSA over ECC (Elliptic Curve Cryptography) because of its extensive testing and widespread use. RSA is often favoured in contexts where backward compatibility is essential, as even older systems may depend on it for compatibility purposes [156] [157].

5.4.1 Components

We now discuss the types and capabilities of different software and hardware components used in system design based on Figure 5.2.

MQTT-SN Publisher (ClientApp): An MQTT-SN publisher is an application that utilizes a ZigBee transceiver module to transmit distress signals to subscribers. This application could potentially be integrated into the onboard automotive system. The ClientApp is denoted by subscript 'P' in our notation.

MQTT-SN Forwarder (FowardApp): A MQTT-SN forwarder is an application which is used to receive the MQTT-SN message from the client and forward it to the gateway with the help of a ZigBee transceiver module. The FowardApp is denoted by subscript 'F' in our notation.

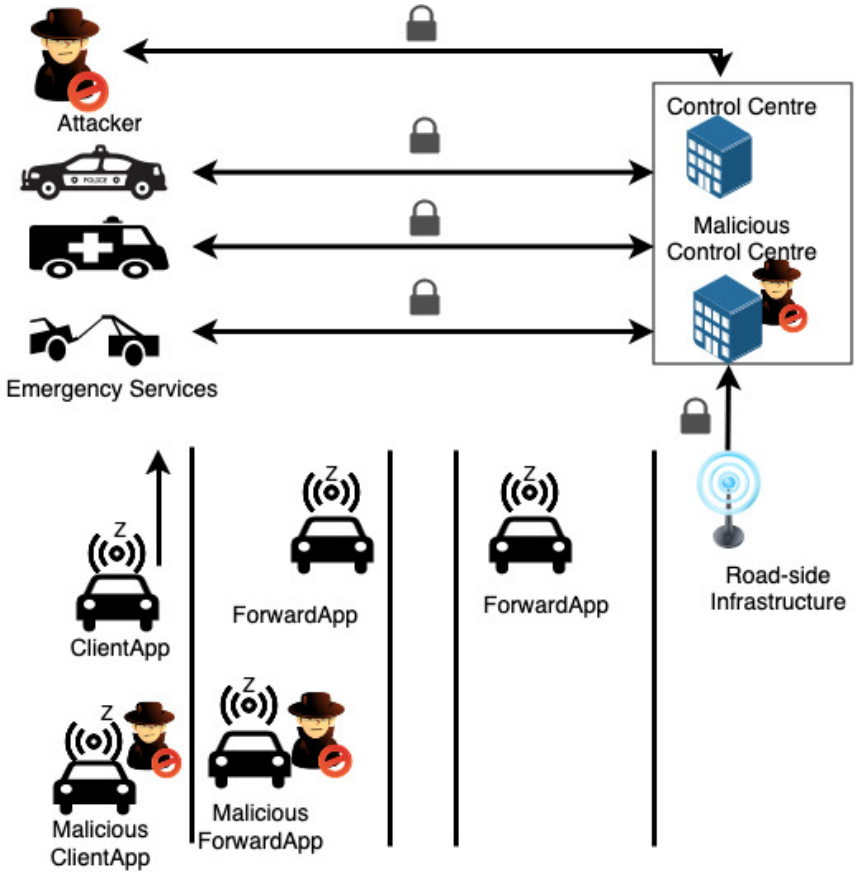


Figure 5.2: Threat Model for V2S

MQTT-SN Gateway (Road-side Infrastructure): An MQTT-SN gateway is a piece of infrastructure located on the side of the road that accepts a message from an MQTT-SN client or forwarder via ZigBee and then sends it to the MQTT broker. The road-side infrastructure is denoted by subscript 'G' in our notation.

MQTT Broker (Control Centre): The primary function of the control centre is to disseminate communications in accordance with the subscribed preferences. The control centre is denoted by subscript 'B' in our notation.

MQTT Subscriber (Emergency Services): The MQTT subscribers subscribe to various distress signal themes. The emergency services encompass law enforcement agencies, medical facilities, and vehicle recovery services. The emergency services are denoted by subscript 'S' in our notation.

5.4.2 Implementation

We now discuss the implementation of the prototype for security architecture and how a distress signal is sent securely using MQTT-SN over ZigBee. In this architecture, we use a unique random number to generate the vehicle manufacture as an identification parameter, which is secured in the secure ROM (Read Only Memory) of the vehicle. We want to provide end-to-end security; therefore, we do not trust any middle components like ForwardApp, road-side infrastructure, and control centre, which has not been done by many researchers when implementing security for the publish-subscribe systems, and it will also open the gate for single-point failure. We will provide more details in security analysis.

Figure 5.3 depicts the comprehensive message flow of the security architecture for distress signals transmitted from a vehicle to emergency services. At this location, we utilize the VIN as the ClientID despite its intended status as confidential information.

Table 5.1: Symbols and Terminologies Used in Design and Implementation

Notations	**Variables or Symbol Meaning**
ClientID	Unique identifier for vehicles
TID	Topic id 01- Vehicle not starting 02- Accident with no injury 03- Accident with injury
Data	Combination of vehicle status and Last known location and VIN and RND_C and Timestamp
V_F	Destination address for the vehicle (i.e., ClientApp) to Broadcast, i.e.,(0xFFFFFFFF)
F_G	Destination address for the vehicle (i.e., FowardApp) to Broadcast, i.e.,(0xFFFFFFFF)
P_{S_1}, P_{S_2}	RSA Public & Private Key of Connect Device with key size of 2048 bits
SHA_{256}	Hash algorithm used for message header
H_P, H_S	Hash output of SHA-256 on message header generated by ClientApp & emergency services, respectively
e_S, d_S	RSA encryption & description algorithm
M	Manufacturers
MH	MQTT-SN Message Header
VIN_M	VIN Sent by Manufacturer to Emergency Services
VIN_C	VIN Sent by ClientApp to Emergency Services
RND_M	Random number corresponding to VIN Sent by Manufacturer to Emergency Services
RND_C	Random number corresponding to VIN Sent by ClientApp to Emergency Services
Message	MH + Payload

Nevertheless, the public record can be accessed by anybody through online platforms, and mechanics have the ability to view the VIN when vehicles are brought to them. We utilize pre-established and programmed topic aliases within the ClientApp or vehicle on-board device. End-users are unable to change the information. Users have the option to provide additional information via the car dashboard interface or smartphone application prior to transmitting the distress signal. The communication between road-side infrastructure and emergency services is conducted using HTTPS, which stands for HTTP over TLS.

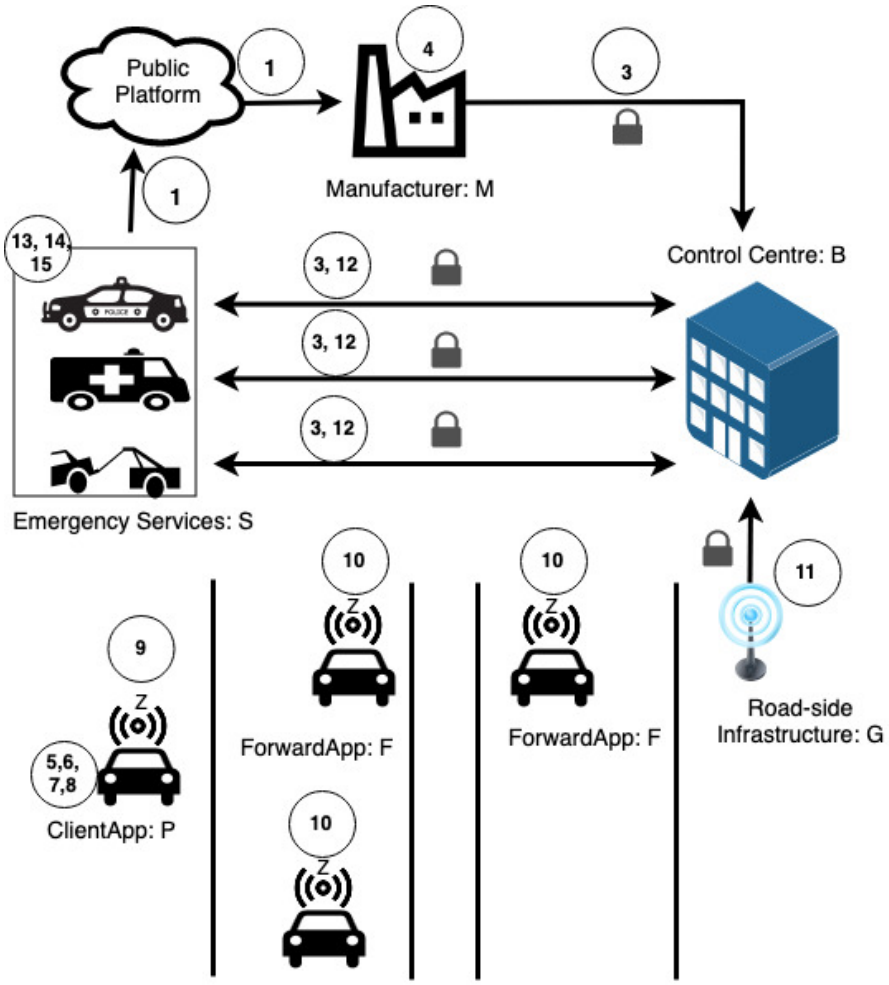
Here are the steps involved in the process:

1. Each emergency service generates its RSA public-private key pair and shares the public key $PS_1$with the vehicle manufacturers through a public platform like the cloud or website.

2. Each manufacturer sends the list of VIN numbers and their corresponding secure random number for identification to the control centre through secure medium.

3. Each emergency services subscribe to the corresponding topics to the control centre after proper authentication based on their certification, address and government-approved details. On successful verification, the control centre shares the list of VIN numbers and their corresponding secure random number for identification received from manufacturers to the emergency services. The emergency services keep the data received secured.

4. Vehicle manufacturers embed the PS_1 into the secure ROM of the vehicle along with RSA encryption and the SHA-256 algorithm. In case vehicles are old, each end-user can download the software application (i.e., ClientApp) provided by their vehicle manufacturer, which includes all the mentioned details.



1. S—-> M: PS_1
2. M—-> B: [VIN_M + RND_M]
3. S—-> B: [Auth Details] and B—->S: [[VIN_M + RND_M]]
4. M—->P : PS_1 + e_S + SHA_{256} + RND_C
5. Data: (Vehicle Status + Last Known Location + User Details + VIN_C + RND_C)
6. MH: (ClientId + TID + V_F)
7. H_P: $SHA_{256(}$MH)
8. Payload: $e_S(PS_1)$[Data + H_P]
9. Message: MH + Payload and P—->F : Message
10. F—->G: Message
11. G—->B: Message
12. B—->S: Message
13. $d_S(PS_2)$[Payload]
14. H_S: SHA_{256}(MH)
15. VIN_M + RND_M == VIN_C + RND_C

Figure 5.3: Data flow in V2S Security Architecture

5. In case of emergency, the end-user selects the type of emergency, adds details if needed and prepares the data, which includes vehicle status, last known location, VIN and RND_C and current timestamp.

6. ClientApp now constructs the message header using ClientID, TID, and destination address (i.e., MH).

7. ClientApp collects the message header and generates its hash (i.e., H_C) using the SHA_{256} algorithm.

8. ClientApp joins the Data and H_C and encrypts them using the RSA public key (i.e., PS_1) of the corresponding emergency service the end-user selects.

9. ClientApp broadcasts the message to be received by the FowardApp.

10. FowardApp adds its identity in front of it and again broadcasts the message (i.e., F_D). The message might be received by another FowardApp, which repeats the same process, or it will be received by road-side infrastructure.

11. Once the message is received by road-side infrastructure, it will forward it to the control centre over TLS using HTTP.

12. Based on the topic subscription, the control centre filters the message and forwards the message to the respective emergency service. The control centre is unable to read any details of the payload.

13. The emergency service decrypts the payload using the RSA private key (i.e., PS_1).

14. Emergency service then generates the hash of the message header and compares it with the hash available in the decrypted payload to check for any tampering.

15. After verifying the hash of the message header, the emergency service compares the VIN_C and RND_C available in the payload to the VIN_M and RND_M provided by the manufacturer.

16. After successful verification, emergency services deploy the assets to help the end user based on location and user-shared information.

We also keep modifying the RSA public-private key pair of emergency services to avoid any post-quantum attacks. The new RSA public keys are generated every month and will be shared with the manufacturers via a secure public platform. The manufacturers will download these keys and provide them to all the vehicles in terms of monthly updates. The vehicle, upon receiving new keys, will save it in the secure ROM. Using the mentioned mechanism, we will maintain backward compatibility and can easily be able to work with legacy systems of emergency services.

5.5 Limitations

Although the proposed V2S security architecture provides strong end-to-end protection for MQTT-SN communication, several limitations remain due to the assumptions and design constraints outlined in Sections 5.3.

- The architecture assumes that each vehicle's secure ROM is correctly provisioned with public keys, random identifiers, and cryptographic functions. Any compromise or misconfiguration at manufacturing time cannot be detected by the system.

- The threat model requires at least one non-malicious ForwardApp. If all forwarders in a region are compromised or jammed, distress messages may fail to reach the gateway.

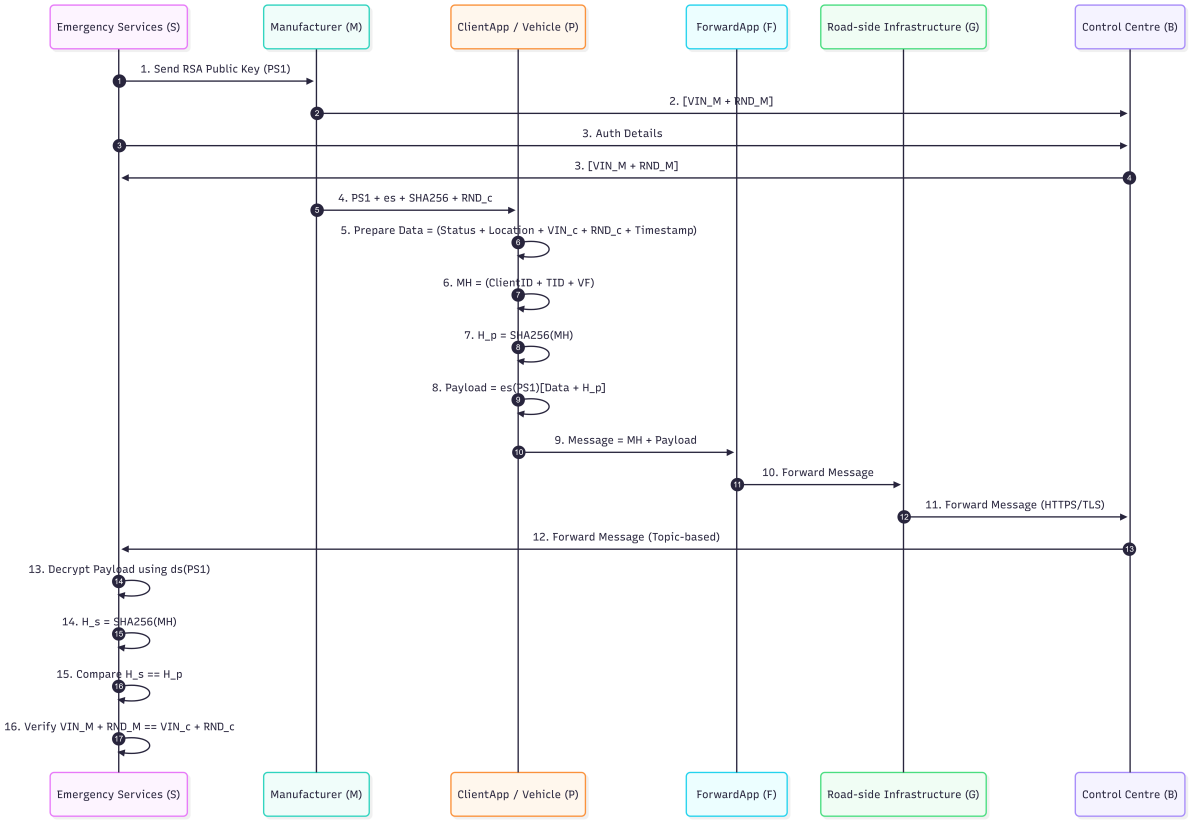


Figure 5.4: V2S Architecture: Sequence Diagram

- Intermediate nodes (forwarders, gateway, broker) cannot authenticate the publisher. A malicious vehicle can still inject encrypted but bogus distress messages, potentially exhausting emergency service resources.

- While feasible, RSA-2048 introduces non-negligible computational overhead on constrained devices. The architecture does not explore more efficient alternatives such as ECC or hybrid encryption.

- The system does not mitigate jamming, interference, or physical tampering with roadside infrastructure. Availability remains dependent on ZigBee channel conditions.

- Replay detection relies on timestamps and per-vehicle random values. Clock drift or inaccurate time sources on constrained devices may affect reliability.
- Although treated as "honest but curious," the control centre remains a single point of message filtering. Misconfiguration or compromise could delay or drop legitimate distress messages.
- The system disables wildcard topics for security reasons, limiting scalability for multi-agency emergency deployments.
- Monthly RSA key updates require manufacturers to distribute updates and vehicles to install them. Legacy or offline vehicles may operate with outdated keys.

Chapter 6

Security Architecture for Smart Lock Communicating Over MQTT-SN

6.1 Introduction

Smart locks are now a key part of the new smart homes and smart building ecosystems. Smart locks now come with built-in features that use microcontrollers, wireless communication modules, and cloud-connected services, enabling keyless entry, remote control, automated scheduling, and access management for multiple users. Now we can also manage temporary guest access, activity logs, and work with IoT frameworks such as Amazon, Apple HomeKit, Google Home, and Alexa. The use of Smart Locks has grown quickly in homes, rental properties, offices, and hotels [158] [140].

Bluetooth and other wireless communication technologies enable smart locks to communicate with each other. Low Energy (BLE), Wi-Fi, Zigbee, Z-Wave, NFC, or RF protocols that are unique to the company. These wireless channels enable low-power

operation and flexible deployment, but they also introduce new security risks. Academic research has consistently shown that smart locks are vulnerable to eavesdropping, replay, relay, and unauthorized key-cloning attacks, insecure firmware updates, and cloud-side compromise [159] [160]. BLE-based locks, for instance, have demonstrated susceptibility to passive sniffing, and key-recovery attacks can happen when old pairing modes are used [161]. Weak cloud authentication or insecure mobile APIs have made Wi-Fi-enabled locks less safe [140]. Even commercial locks advertised as secure have been broken by replaying encrypted packets or exploiting weak key-exchange protocols [158].

One common problem we found is that smart-lock makers often prioritise usability, low cost, and battery life over strong cryptographic design. Many devices rely on static keys, weak pairing methods, or proprietary security systems that lack formal verification [160]. Some people rely heavily on cloud services, which means that a breach of the backend or mobile app can lead to unauthorized physical access [162]. Also, smart locks often lack secure firmware update mechanisms, which makes them easy to exploit for a long time after a vulnerability is discovered [159].

Another big problem is that there is no fine-grained, multi-user access control put directly on the lock. Many commercial systems support temporary or guest access. However, not all of them support access; academic evaluations indicate that these mechanisms are frequently implemented at the cloud layer rather than enforced on the device itself. This makes it necessary to have a permanent internet connection and makes the system vulnerable to attacks from the cloud or to API abuse [140]. Most smart locks do not support cryptographically verifiable authorisation tokens for each operation, which makes them vulnerable to replay or privilege-escalation attacks when communication channels are intercepted [160]. From a systems point of view, smart locks operate in resource-constrained environments—limited CPU, memory, and battery capacity—which makes

it hard to use TLS or DTLS, both heavyweight security protocols. This has spurred research into lightweight cryptographic schemes, capability-based access control, and safe communication protocols for limited IoT devices [163] [164]. However, many of these solutions are still just ideas or have not been used very often in products sold.

The flaws found in current smart lock systems show that we need a communication and authorization architecture that is:

- light enough for limited hardware
- safe from attacks that happen again, relay, or on the cloud side
- able to enforce access control based on roles for multiple users locally
- not dependent on being connected to the internet all the time
- works with wireless networks that use little power

MQTT-SN (MQTT for Sensor Networks) is a publish-subscribe protocol made just for IoT devices with limited resources that work over low-bandwidth, low-power wireless networks [165]. MQTT-SN can work over UDP or even non-IP transports, unlike MQTT, which needs TCP. This makes it good for battery-powered embedded devices. It uses numeric topic IDs instead of long string topics, which makes packets smaller and processing faster [166] [167]. These features make MQTT-SN a good choice for smart lock communication, especially when used with a capability-based authorization model. The system can enforce per-operation authorization directly on the lock by linking each lock/unlock action to a one-time topic ID that was derived from cryptography. This works even when the lock is offline or can only be reached via Bluetooth. This thesis looks at an architecture that uses MQTT-SN not as the main way for the lock to communicate, but as a safe way for the user's smartphone to audit and sync. The lock can

only connect to Bluetooth, but the phone can connect to both Bluetooth and MQTT-SN. This hybrid model makes it possible to:

- checking capability tokens on the lock locally
- MQTT-SN lets you see things from far away
- cryptographically enforced access control for multiple users
- lightweight operation that works well on limited hardware

This method directly fixes the problems that were found in academic research and offers a new, useful way to build smart lock systems that are safe and can grow.

6.1.1 Contribution

The major contributions of this work are:

- We design a new Capability-Token authorization Model for Smart Locks, which uses Topic-Bound Ephermal Capability Tokens (TBECT).
- We design a hybrid communication architecture where Bluetooth is used for local control, and MQTT-SN is for remote audit and synchronisation.
- We extend the capability-token model to support multiple user roles.
- We design a complete end-to-end protocol for Secure Smart-Lock Operation.
- We have performed the formal security analysis using the Scyther tool.

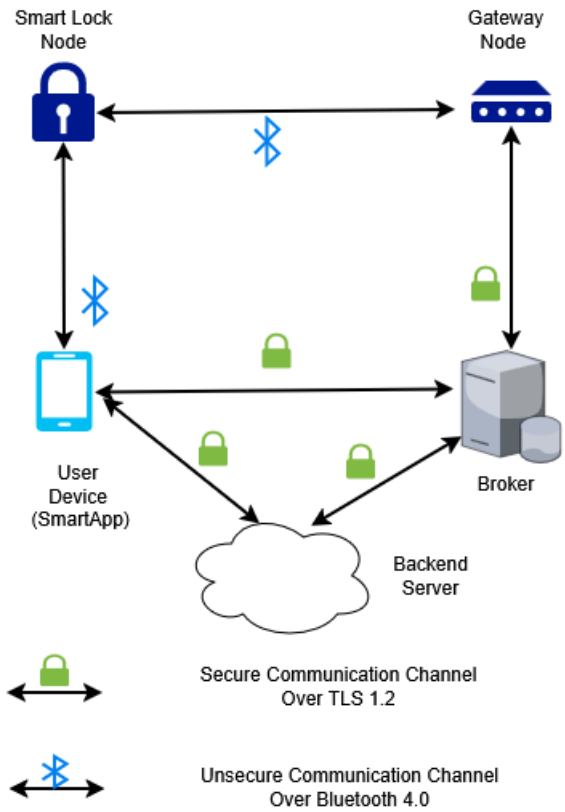


Figure 6.1: Proposed System-Design for Smart Lock

6.2 Smart Lock

Smart Lock [168] is a Wi-Fi, or Bluetooth-enabled IoT device, that provides keyless access to the home and locks/unlocks when instructions from an authorized device are received. Smart Locks use two different kinds of architecture: indirect (Device-Gateway-Cloud) and direct (Device-Cloud). Our focus is on indirect architecture, where the term *broker* represents cloud. In the indirect architecture, shown in Figure 6.1, the Smart Locks uses a gateway device to communicate to the MQTT broker. The Smart Lock needs a communication protocol to send status and receive commands from the end-user, i.e., MQTT-SN (as shown in Figure 6.1). The communication between the Smart Lock and the gateway device is MQTT-SN over BLE/Zigbee. The communication between the

gateway device and the broker is MQTT over TLS. The communication between the broker and the subscriber's end gateway device is also MQTT over TLS. These are two different TLS channels, as shown in Figure 6.1. The access point at the subscriber end can be trusted or untrusted based on location, whether it is a home router or public Wi-Fi access point. This DGC (Device-Gateway-Cloud) architecture supports constrained IoT devices.

6.3 Threat-Model

In this section, we aim to provide the threats related to Smart Lock of by identifying the attacker goals and its capabilities. We also discuss the different attacks possible on working of Smart Lock in Figure 6.1.

- **Attacker Goal and Capabilities**: The goal of an attacker in a Smart Lock scenario is to access the home without proper authorisation. We take into account two potential attackers (i.e. an attacker who wants to access the smart lock without proper authorisation and an attacker who stores the old message and replay the message to lock/unlock). The attacker may attempt to act as a man-in-the-middle, and when the device sends a lock/unlock message to the Smart Lock, the attacker stores the message and attempts to reverse-engineer it to obtain the message contents. The second potential attacker is a broker gone rogue, which has all the information related to Smart Lock, and can use that information for sharing with malicious agents.

We consider the following threats which are used to derived the security requirements for the smart lock [169]:

- SLN-Threat 1 (SLN-T1): **Spoofing Attack:** The broker authorizes the adversary as if the adversary had a valid ClientID to send malicious control/data messages.
- SLN-Threat 2 (SLN-T2): **Passive Eavsdropping:** The attacker gain access to unauthorized resources.
- SLN-Threat 3 (SLN-T3): **Replay Attack:** The attacker can replay old messages to misguide the end-user/Smart Lock.
- SLN-Threat 4 (SLN-T4): **Injection Attack:** The attacker can intercept and modifies messages to misguide the end-user/Smart Lock.
- SLN-Threat 5 (SLN-T5): **Non-Ephermal Keys-** The lifespan of IoT device is between 5-15 years. Therefore, with current computational capabilities at the disposal of an attacker, a single public-private key pair for the whole lifespan can easily be attacked.

We make the following assumptions:

A1: Physical tampering with device is out of scope.

A2: Long Term Key are embed in Smart Lock Node (SLN) duing manufacturing and is stored in backend server in reference to LockID.

A3: New TopicId and session key is generated after every 7 days for admin and family.

A4: The initial imprinting phase occurs in a trusted physical environment where no adversary can intercept BLE messages or replace the Gateway during first-time setup Between SLN and Gateway device.

A5: The SmartApp runs on a smartphone assumed to be free from malware that could steal capability tokens, or session keys.

A6: The backend server is assumed to be uncompromised and correctly enforces authentication, role assignment, and capability-token generation.

A7: The attacker can passively sniff BLE traffic but cannot inject packets during the imprinting phase without being detected.

A8: The SLN and SmartApp maintain loosely synchronized clocks so that StartTime/EndTime enforcement is meaningful.

A9: All long-term key, RSA private key and session keys are stored in a secure location in SLN, SmartApp, Gateway and Backend Server.

6.4 Design and Implementation

The primary goal is to design and implementation of a secure, lightweight authorization for smart-lock systems based on ephermal capability tokens and a hybrid Bluetooth + MQTT-SN communication model. the architecture is tailored for constrained Bluetooth-only locks and supports fine-grained, role-based access control, per-operation authorization, and tamper-evident auditability.

A key enhancement in this design is the explicit separation of the MQTT-SN Gateway and the MQTT-SN Broker into two distinct entities. The gateway perform protocol translation and network bridging, while the broker handles publish-subscribe routing. Both are treated as untrusted or semi-trusted components; all security is enforced end-to-end between the lock, user's device (smartphone) and the backend.

In our architecture, the lock independently verifies whether an operation is permitted , ensuring secure operation even when offline or when cloud services are unavailable. The lock uses only symmetric primitives HMAC, hash functions and AES encryption -making our design suitable for low-power microcontrollers.

6.4.1 Components

In this section, we are going to discuss the types and capabilities of different software and hardware components used in the system design based on Figure 6.1.

- Smart Lock Node (SLN); An IoT device with no operating system installed on it. It is implemented on Arduino Uno board which is 8-bit micro-controller (ATmega328P) with 16 Mhz of clock frequency capable of communicating only over Bluwtooth. It is capable of running symmetric encryption/decryption and verification of capability tokens. The Bluetooth module (i.e., HC-05) is in listening mode. It has no user input/output interface.

- Gateway node (Connect device): It is implemented using Raspberry Pi 3B+ module which is programmed in nodejs. Its a 32-bit board with an inbuilt Bluetooth and WiFi module. It serves as a gateway node for the SLN which does not have the capabilities to communicate the internet by itself. It communicates with SLN over BLE and with the backend server using WiFi through a router. It has no user input/output interface. It is configure with the help of laptop/desktop manually.

- Android Application (SmartApp): An Android application programmed using React platform running over smartphone (Samsung Galaxy Note 2 with 2GB RAM and 32 GB memory) provided by the service provider for the User to communicate with the device locally via BLE, and remotely over the internet through the Gateway device. With the help of the service provider mobile app, i.e., SmartApp, the owner can communicate with the smart lock locally over BLE for locking and unlocking and remotely to assign different tokens to legitimate users requiring access through backend server.

- Backend Server (Server): Its an software module and programmed in nodejs and uses mongoDB as a database. It runs over laptop with specification i7-8th Gen processor, 16 GB RAM and 512GB hard disk. It is the cloud database managed by service providers which stores the device information with respect to the unique user details. The service provider server autheticates users, issues capability tokens,enforces role based access control, subscribes to audit topics and store audit tolics.

- MQTT-SN Broker: Its is a software module. It is Mosquitto broker. It routes publish/subscribe messages, stores no secrets and treated as untrusted. It is also running over laptop with specification i7-8th Gen processor, 16 GB RAM and 512GB hard disk.

In this design for providing the secure smart lock architecture we need to establish trust between all the components. Our both physical devices, i.e., IoT device and Gateway device do not have any input/output interface and there is no pre-shared secret between them. We use the trusted network for establishing trust between both the devices.

6.4.2 Implementation

According to our design assumption, our IoT device (i.e. Smart lock) and Gateway device do not have any input and output interface for user interaction. Therefore, we need a out of band medium for device pairing. SLN supports Bluetooth 4.0 protocol which is not secure; therefore we need a method to secure the communication channel between the IoT device and Gateway device and SLN and SmartApp. Communication between the SmartApp and Backend Server and the Gateway device is over https using TLS.

We are now going to discuss the authenticated key management using Figure 6.1 and Figure 6.2.

1. User download and install the android application of SLN for the smartphone from a trusted source.

2. The SmartApp scans the SLN secret (DS_L) and public-key (P_{L_1}) using the QR-code of the booklet comes inside the sealed box of the SLN.

3. When powered on for the first time or after a factory reset, the SLN enters the `UNIMPRINTED` state. It broadcasts a BLE advertisement indicating its availability for imprinting:

$$\texttt{SLN\_STATE} = \texttt{UNIMPRINTED}.$$

   The SLN accepts exactly one imprinting session and ignores all other communication attempts.

4. During installation, the Gateway is connected to laptop or SmartApp and initiates the imprinting procedure by sending the following message over BLE:

$$\text{ImprintReq} = \{\text{GID},\ P_{G_1},\ \text{Nonce}_G\}.$$

   This message is accepted only when the SLN is in close physical proximity (e.g., physical trigger such as a button press).

5. Upon receiving the imprint request, the SLN generates a fresh local nonce $Nonce_L$ and combine it with device secret(DS_L).

6. The SLN derives a long-term shared key using HKDF:

$$K_{LG} = \mathrm{HKDF}\big(DS_L,\ \mathrm{Nonce}_G \parallel \mathrm{Nonce}_L \parallel \mathrm{GID}\big).$$

7. The SLN stores:

$$\{\mathrm{GID},\ K_{LG},\ P_{G_1}\},$$

sets `Imprinted = true`, and transitions to the `IMPRINTED` state.

8. The SLN sends the LockID and local nonce $Nonce_L$, encrypted using the Gateway public key PG_1

$$E(P_{G_1})(\mathrm{LockID},\ \mathrm{Nonce}_L)$$

9. The Gateway decrypts using private key P_{G_2} stores the SLN's identity and the derived the long-term key:

$$D(P_{G_2})[E(P_{G_1})(\mathrm{LockID},\ \mathrm{Nonce}_L)]$$

$$K_{LG} = \mathrm{HKDF}\big(DS_L,\ \mathrm{Nonce}_G \parallel \mathrm{Nonce}_L \parallel \mathrm{GID}\big).$$

All subsequent SLN–Gateway communication is encrypted and authenticated using K_{LG}. The SLN rejects any message from a device that cannot prove knowledge of K_{LG}.

10. User creates his account on the Backend server using signup on SmartApp. After successful account creation, SmartApp sends the LockID, UserID, Role, different operations performed by user (i.e., OpType), StartTime and EndTime over HTTPS.

11. The Backend server generated the capability token using the randomly generated Nonce.

    $Cap = Concat(LockID||UserID||Role||OpType$
    $||StartTime||EndTime||Nonce)$

12. After Cap generation, backend server calculated the MAC using the Long-Term Secret key K_{LS}. This binds the token to the lock's secret key, preventing forgery.

$$\text{MAC} = \text{HMAC}_{K_{LS}}(\text{Cap})$$

13. The Backend server derived the session key (AES-128) using Long-Term key K_{LS} and Nonce.

$$K_S = H\big(\text{Concat}(K_{LS} \,\|\, \text{Nonce})\big)$$

14. The Backend server sends the Cap, MAC, K_S to the SmartApp over HTTPS, i.e., Capability Token, MAC, Session Key.

15. The SmartApp encrypts the Nonce with RSA public-key of the Smart Lock Node (P_{L_1}) and remaining data, i.e., Cap, MAC, device secret (i.e., DS_L) using session key and sends to the Smart Lock Node over Bluetooth.

$$\text{Data} = E(P_{L_1})(\text{Nonce}) \,\|\, S(K_S)\big(\text{Cap, MAC, } DS_L\big)$$

16. The SLN upon receiving the encrypted data, recover Nonce by decrypt using the private-key of the Smart-Lock Node (P_{L_2}) and collect the Nonce and stores it.

$$D(P_{L_2})[E(P_{L_1})(\text{Nonce})]$$

17. After receiving the Nonce, SLN calculates the session key using already stored long-term key.

$$K_S = H\big(\text{Concat}(K_{LS} \,\|\, \text{Nonce})\big)$$

18. The SLN decrypts the MAC using the derived session key and validate the device secret, and stores the UserID, StartTime, EndTime, Nonce, and Role, OpType and starts the timer.

$$S(K_S)[S(K_S)\big(\text{Cap, MAC, } DS_L\big)]$$

19. The SmartApp sends the LockId, UserID, Opcode, Timestamp encrypted using the session key.

$$S(K_S)\big(\text{LockID, UserID, OpCode, TimeStamp}\big)]$$

20. The SLN decrypts the message and validates the UserID against the role and timestamp against the time. If within the timeline, lock performs the requested operation.

$$S(K_S)[S(K_S)\big(\text{LockID, UserID, OpCode, TimeStamp}\big)]$$

21. The SLN publishes an audit record back to the Gateway device encrypted using session key on TopicID .

$$\text{TopicID} = \text{Truncate}_K\big(H\big(\text{Concat}(\text{Cap} \,\|\, \text{MAC})\big)\big)$$

22. The SLN encrypts the audit log and TopicID with K_{LG} and K_S, publishes it through Gateway device.

$$S(K_{LG})\big(\text{TopicID},\ S(K_S)(\text{AuditLog})\big)$$

23. The Gateway receives the message from SLN and decrypts the audit message and forwards it to the Broker.

$$S(K_{LG})[S(K_{LG})\big(\text{TopicID},\ S(K_S)(\text{AuditLog})\big)]$$

24. The Broker routes the message to the backend server as only valid capability holders can compute the TopicID. The Gateway and broker cannot alter audit logs. Audit entries are cryptographically bound to specific operations. The backend server stores the audit log.

$$S(K_S)\big[S(K_S)(\text{AuditLog})\big]$$

25. The SmartApp can subscribe to AuditLog based on the TopicID.

The user with admin role registers other users into the backend server using UserID, so that whenever family or guest user signup to the backend server using the UserID shared by admin, it sends them capabity token based on the role. The Long-term-keys between SLN and Gateway device (i.e., K_{LG}) is generated every seven days using a new $Nonce_L$, which is shared to gateway device after encrypting with RSA public-key of Gateway device. The Session key between SLN and Backend Server (i.e., K_S) is generated by Backend Server and shared along with capability token, which are modified after every 7 days using a new nonce to maintain the uniqueness.

6.5 Limitations

L1: The system does not defend against physical tampering of the SLN, Gateway, or smartphone.

L2: If an attacker compromises the first-time imprinting session, they can permanently impersonate the Gateway because the SLN accepts only one imprinting session. Otherwise we have to reset SLN.

L3: If the SmartApp or smartphone OS is compromised, an attacker can steal capability tokens, DS_L, or session keys, enabling unauthorized unlock operations.

L4: We have limited cryptographic agility due to 8-bit hardware.

L5: Although the broker cannot decrypt messages, denial-of-service attacks on the broker or Gateway can prevent audit logs from reaching the backend.

Table 6.1: Data Structures and Symbols used in Design and Implementation

Data Structure/ Symbols	**Variables or Symbol Meaning**
LockID	Unique Smart Lock Identification
UserID	Unique Username
Role	Admin Family Guest
OpType	Operation Type Lock, Unlock
Nonce	Unique Random Number Generated by Backend Server
$Nonce_L$	Unique Random Number Generated by SLN
$Nonce_G$	Unique Random Number Generated by Gateway device
GID	Gateway Unique ID
DS_L	SLN Secret 32-bit
StartTime (ST)	Start Duration of the Lock Access
EndTime (ET)	End Duration of the Lock Access
Cap	Capability Token
MAC	HMAC of the Capability Token using Long Term Key
TopicId	Topic Alias of the MQTT-SN
K_{LS}	Symmetric Long Term Key (AES-256) of SLN and Backend Server
K_{LG}	Symmetric Long Term Key (AES-256) of SLN and Gateway device
P_{L_1}	RSA Public Key of SLN Device with key size of 1024 bits
P_{L_2}	RSA Private Key of SLN Device with key size of 1024 bits
P_{G_1}	RSA Public Key of Gateway Device with key size of 1024 bits
P_{G_2}	RSA Private Key of Gateway Device with key size of 1024 bits
S_G	Symmetric Session Key (AES-128) generated by Gateway Device for IoT device
$a\|\|b$	Concatenation of a and b
H	Secure Cryptographic Hash Function, i.e., SHA-256
LTK_D(x)	Encryption or Decryption of x using LTK_D AES-256 key
S(x)	Encryption or Decryption of x using Session Key S_C AES-128 key
E(x)	Encryption of x using RSA Public Key with key size of 1024 bits
D(x)	Decryption of x using RSA Private Key with key size of 1024 bits
HKDF	HMAC-based Key Derivation Function

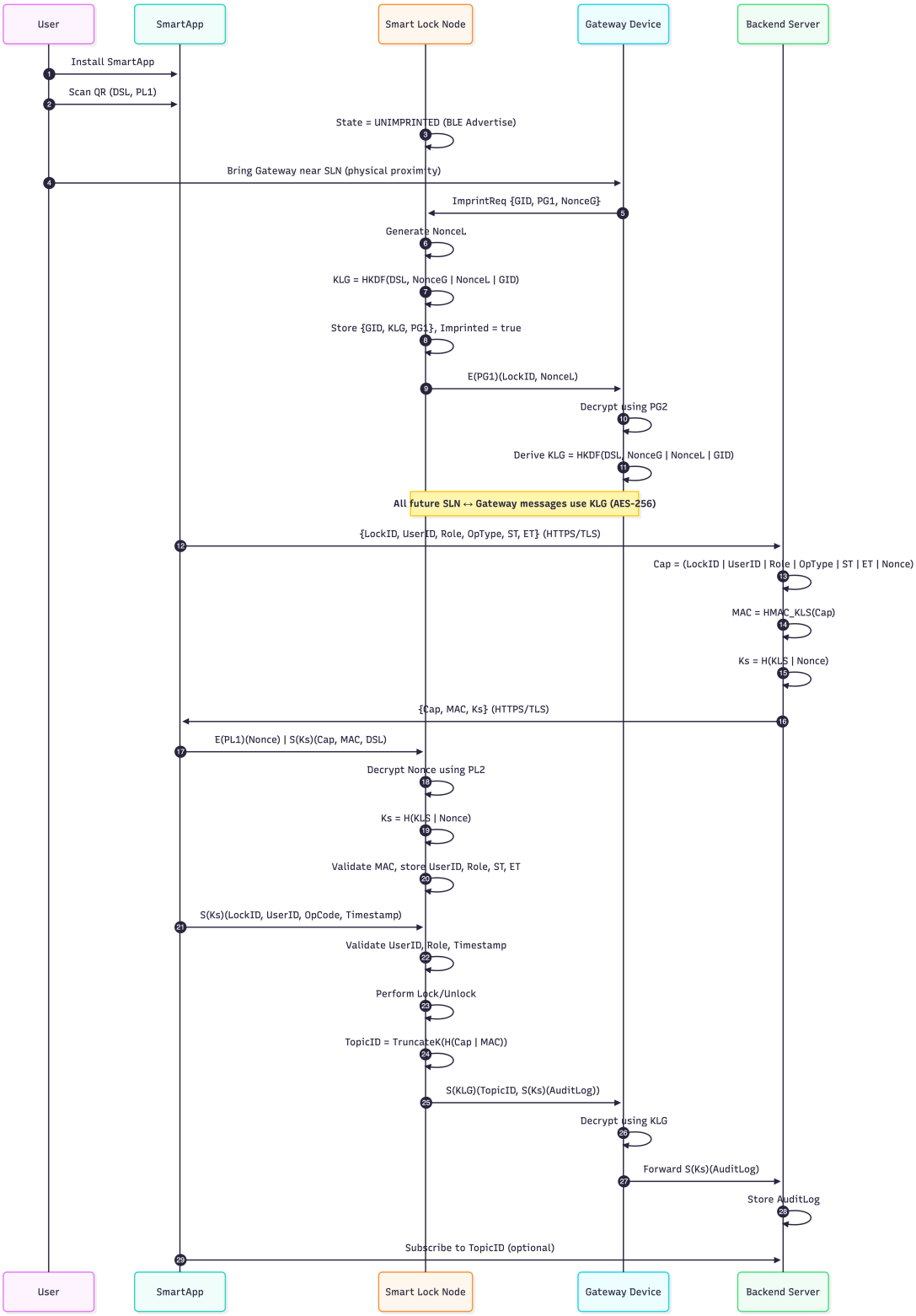


Figure 6.2: Smart Lock: Sequence Diagram

Chapter 7

Security Architecture for Continuous Glucose Monitoring (CGM) Communicating Over MQTT-SN

7.1 Introduction

The evolution of internet communications infrastructure underscores the growing need for effective communication mechanisms for restricted Internet of Things (IoT) devices. These devices, utilized in sectors such as smart healthcare (e.g., remote patient monitoring), frequently encounter constraints in power, processing capabilities, and memory capacity. To address these difficulties, restricted IoT devices employ lightweight message protocols designed for minimal power consumption and bandwidth usage. MQTT-SN, a variation of the MQTT protocol tailored for sensor networks, facilitates publish-subscribe

communications and is optimized for battery-powered devices.

Statistics from the World Health Organization indicate that roughly 430 million individuals globally are affected by diabetes [49]. The annual increase of this number results in significant repercussions for both health and public spending. To maintain physical well-being, a diabetic individual must consistently monitor their glycemia and administer appropriate interventions, such as insulin injections. Historically, to assess blood glucose levels, the majority of diabetic patients would puncture a finger to obtain a blood sample for a specialized instrument known as a blood glucometer [16]. Currently, technology in this domain has markedly advanced. Continuous Glucose Monitoring Systems (CGMs) are extensively utilized to assess blood glucose levels and enhance diabetes management wirelessly. A continuous glucose monitor (CGM) comprises a disposable sensor implanted subcutaneously, a transmitter linked to the sensor, and a receiver that acquires and displays the readings. The sensor is operable for some days before necessitating replacement [72]. It offers periodic real-time measurements, eliminating the necessity for fingerstick glucose testing. CGM solutions utilize common transmission frequencies, such as the Industrial, Scientific, and Medical (ISM) band at around 2.4 GHz. This article demonstrates widespread systems fail to implement adequate cryptographic safeguards to protect transferred data.

Multiple CGM devices are available in the market, such as Dexcom G6 [170], Abbott Freestyle Libre 3 [171], Sensonics [172], Medtronic Guardian Connect [173], and many more [174]. Nevertheless, numerous issues arise when using these CGMs [175]. Embedded systems are susceptible to multiple issues, such as hacking, due to their constrained computational capacity and resources. The existing procedures are insufficient for securing the CGM. Consequently, a novel protocol is required to ensure the security of this IMS. In our proposed system, we send the CGM sensor readings to the doctors with the

help of a secure architecture using the MQTT-SN protocol, as shown in Figure 4.5.

In smart healthcare, MQTT-SN can serve as a significant application-layer protocol for Continuous Glucose Monitoring (CGM) devices, which monitor glucose levels and relay data to distant monitoring apps. CGM systems have stringent communication protocols requirements as they are battery-powered, resource-constrained devices, and because of the safety-critical nature of glucose data. MQTT-SN is a well suited protocol for such a system, as it is designed for sensor networks which are deployed in constrained environments. MQTT-SN has a lightweight architecture. MQTT-SN introduces TopicID compression, i.e., 2-bytes, which reduces parsing overhead and good for communication as CGM publish frequently. Also, The reduced packet size minimizes bandwidth usage, lowers collision probability in the 2.4 GHz ISM band, and decreases processing overhead on constrained devices. CGM sensors spends most of the time asleep and MQTT-SN protocol supports asynchronous wake-up, queued messages and offline buffering at at the gateway. The gateway-centric design also supports scalability and ensures that intermediate components—such as the control center—can remain untrusted without compromising end-to-end security. CGM data is time-sensitive, MQTT-SN provides deterministic latency characteristics that are essential for real-time glucose monitoring. By avoiding TCP's head-of-line blocking and retransmission delays, MQTT-SN ensures timely delivery of glucose readings and alerts, even in noisy RF environments. MQTT-SN's security-layer independence complements the end-to-end security architecture proposed in our work. Our design ensures that sensitive data remains protected even if intermediate components are compromised, directly mitigating the threats outlined in the system's threat model. Collectively, MQTT-SN is not only compatible with the operational constraints of CGM devices but also enhances the overall security, efficiency, and reliability of the communication architecture. MQTT-SN lightweight design, energy

efficiency, and integration with end-to-end cryptographic protections make it the optimal protocol for CGM communication in modern smart healthcare environments.

7.1.1 Contribution

- We have designed and implemented the end-to-end security architecture for the CGM data transfer using MQTT-SN in IoT environment.
- We compared our security architecture, designed for the MQTT-SN publish-subscribe protocol, with those currently available on the market and perform formal security analysis.
- We have explored the viability of implementing public-key cryptography on the 8-bit microcontroller and performed timing analysis.

7.2 Continuous Glucose Monitoring (CGM)

People with diabetes can use Continuous Glucose Monitoring (CGM) systems to keep an eye on their glucose levels in real time. CGMs give you a constant view of glucose trends, which lets you take action quickly and better control your blood sugar levels. Traditional finger-prick glucometers only give you measurements every now and then. A CGM system usually has three parts: a sensor, a transmitter, and a receiver. The sensor is put under the skin and checks the interstitial fluid for glucose every few minutes. The transmitter sends these readings wirelessly to a receiver, like a smartphone, a dedicated monitor, or an app that connects to the cloud [176].

CGM devices are embedded systems that run on batteries and have limited resources, which makes them more likely to have security and communication problems. Because they are so important for safety, any change, manipulation, or delay in glucose data can

directly affect insulin dosing decisions and the health of the patient. As more people use CGM sensors, gateway devices, and healthcare providers, it is important to make sure that communication between them is safe, reliable, and has low latency. This necessitates the development of lightweight, end-to-end secure communication architectures specifically designed for constrained IoT environments.

7.3 Threat Model

The main security requirement is providing an end-to-end architecture for the CGM system. Figure 4.6 shows our architecture's threat model.

Attacker's Goal and Capabilities: The attacker aims to read the payload information and harm the end-users. Here, we have considered four different attackers:

1. a malicious subscriber who wants to get all information from the control center and harm the end user

2. a malicious gateway device which tries to stop the message from forwarding and tries to connect with the sensor device before the authentic gateway device or also tries to tamper with the message

3. a malicious publisher (i.e., sensor device) who wants to send a malicious message to the subscriber.

4. an honest but curious control center who wants to read the data

In our model, all three attackers have various capabilities. Attackers have information about the sensor device's serial number and topics data published under it, which is plaintext and publicly available. Attackers can also tamper with messages and format new messages.

CGM-Threats T1-T5 below are used to derive the security requirements of CGM communication. We defined them based on our earlier research [151]. Figure 4.6 shows the malicious components of the mentioned architecture in the CGM system.

- CGM-Threat 1 (CGM-T1): Unauthorized individuals or entities may attempt to intercept and eavesdrop on sensor data to obtain sensitive information to manipulate insulin intake. [Confidentiality]
- CGM-Threat 2 (CGM-T2): The Control Center authorizes the adversary as if the adversary had a valid ClientID to send or receive malicious patient information messages. [Authorization]
- CGM-Threat 3 (CGM-T3): The attacker can intercept and modify messages to misguide the hospital services. [Integrity]
- CGM-Threat 4 (CGM-T4): An attacker intercepts communication between the CGM device and a paired device (like a smartphone or cloud service). If the data is not adequately encrypted, the attacker can read, alter, or block data transmission, which could lead to incorrect data being displayed to the user or healthcare provider. [Man-in-the-middle]
- CGM-Threat 5 (CGM-T5): Attackers can record and replay legitimate Sensor device messages, which can have negative consequences, especially when safety-critical information is involved. [Replay Attack]

We make the following assumptions for the prototype:

1. Each type of sensor device generates an Elliptic curve public-private key pair ($e1_D$, $d1_D$) during initialization and every day; the public keys are stored in a secured ROM (Read Only Memory) of the sensor by the manufacturer.

2. Each type of gateway device generates an Elliptic curve public-private key pair ($e1_G$, $d1_G$) during initialization and every day.

3. Gateway device is always in receiving mode on running messages till it receives any message to broadcast.

4. Communicating details of the control center is pre-programmed in the Gateway device (i.e., SmartphoneApp) during initialization, which cannot be modified.

5. Our system does not support wildcard topics.

6. The unique password (w_P), is shared with subscribers using a secure out-of-band method.

7.4 Design and Implementation

A primary goal of our prototype is to send glucose level information using MQTT-SN over Bluetooth to the respective doctors and registered end users. This chapter focuses on one of a few security requirements to protect the design, i.e., authentication, confidentiality, integrity, availability and accountability. The encryption and decryption process is accomplished by utilizing the ECDH (Elliptic Curve Diffie Hellman) algorithm with a 256-bit key and RSA (Rivest-Shamir-Adleman) with a 1024-bit key. Additionally, the payload and message header are verified using SHA-256. As there can be one sender and multiple receivers, or multiple senders and multiple receivers, or multiple senders and multiple receivers, we have chosen to implement the publish-subscribe protocol, specifically MQTT-SN. The MQTT-SN is well suited for sensor devices, has low power consumption, is lightweight, and supports sleep mode. ECDH generates a symmetric key, which helps send secure encrypted data from the CGM sensor to the gateway device

(i.e., Smartphone or Monitor). The reason for using ECDH is its key size is small, has less computational overhead and has faster operations, which is useful for sensor devices. The shared secret key from the sensor device to the subscriber gateway is transmitted using RSA encryption with a key size of 1024 bits. We have selected RSA because it has undergone thorough examination and analysis over time, and its security is comprehensively understood and widely implemented. RSA is often favoured in contexts where backward compatibility is essential, as even older systems may depend on it for compatibility purposes [59]. As the life of the CGM sensor is 10-14 days, the shared secret key is regenerated with a new sensor device.

7.4.1 Components

Based on Figure 4.5, we now discuss the types and capabilities of different software and hardware components used in system design.

MQTT-SN Publisher (CGM Sensor): An MQTT-SN publisher is a 4-bit or 8-bit sensor device that utilizes a Bluetooth transceiver module (i.e., HM-05) to transmit glucose levels to subscribers. Our architecture used Arduino Uno (8-bit) and a glucose sensor module. The ClientApp is denoted by subscript 'P' in our notation.

MQTT-SN Gateway (Monitor or SmartApp): An MQTT-SN gateway is a piece of hardware device or can be a software application downloaded based on the CGM device that accepts a message from an MQTT-SN client via Bluetooth and then sends it to the MQTT broker (i.e., control center). Our architecture used Rasberry Pi B+ as a gateway module running an Android application in a simulator. The Gateway device is denoted by subscript 'G' in our notation.

MQTT Broker (Control Center): The primary function of the control center is to disseminate communications by the subscribed preferences. Our architecture used a

Macbook as a control center running a modified mosquitto broker. The control center is denoted by subscript 'B' in our notation.

MQTT Subscriber: The MQTT subscribers subscribe to the CGM device based on their unique serial number. The subscribers can be hospital doctors, parents, and guardians. Our architecture used a modified PahoMQTT client application as a subscriber module. In our notation, the subscribers are denoted by subscript 'S' .

7.4.2 Implementation

We now discuss the implementation of the prototype for security architecture and how a glucose sensor reading is sent securely using MQTT-SN over Bluetooth. In this architecture, we use a unique secret password as part of QR code, which is only visible to the product buyer as located inside the booklet as an identification parameter. We want to provide end-to-end security; therefore, we do not trust any middle components like a control center, which has not been done by many researchers when implementing security for the publish-subscribe systems, and it will also open the gate for single-point failure. We will provide more details in security analysis.

Figure 7.1, Figure 7.2, and Figure 7.3 depicts the comprehensive message flow of the security architecture for data transmitted from a CGM sensor to subscribers. At this location, we utilize the sensor device's serial number as the ClientID despite its intended status as confidential information. End-users are unable to change the information. The communication between the SmartApp and the subscribers is conducted using HTTPS, which stands for HTTP over TLS. There are two steps involved in the implementation:

1. Onboarding

2. CGM Working

1. Onboarding: It is a process of connecting, configuring and securely integrating all IoT devices in a network or system. Here are the steps involved in the process of onboarding:

1. Each sensor device has a Unique Password (w_P), serial number (N_P), and RSA public-private key pair (P_{D_1}, P_{D_2}) are programmed into the device's ROM during manufacturing. The unique password, serial number, and RSA public key are in a sealed booklet or sticker (in QR code format) with the device.

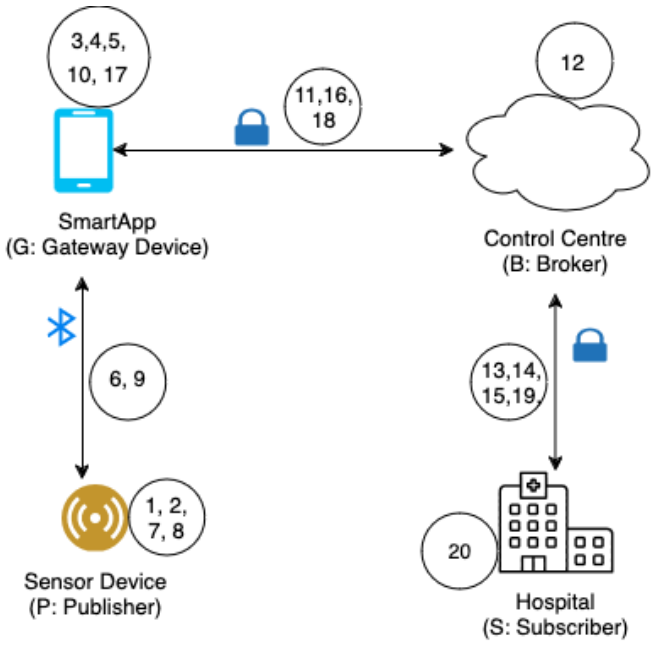
2. Each sensor device generates an elliptic curve (EC) public-private key pair ($e1_D, d1_D$) during initial initialization and on any reset, which is stored in EEPROM (Electronically Erasable Programmable Read Only Memory).

3. User downloads and installs the Smartphone application (i.e., SmartApp) of the CGM sensor from a trusted source, which is a gateway device.

4. The User initializes the SmartApp and it is pre-configured with the control center's URL.

5. The user scans the QR bar code from the device-sealed booklet for the unique password (w_P), Bluetooth MAC address (B_D), and RSA public key of the CGM sensor (P_{D_1}) with the SmartApp from the Smartphone camera during its configuration and saves it in secure ROM of the smartphone. Alongside, the SmartApp generates an RSA (P_{G_1}, P_{G_2}) and EC key pair ($e1_G, d1_G$) for ECDH.

6. The SmartApp encrypts the unique password (w_P), its RSA public key (P_{G_1}), and its own EC public key ($e1_G$) with the RSA public key of the sensor device (P_{D_1}) and sends it to the CGM sensor device using Bluetooth MAC address (B_D).



1. P: w_D, N_D, P_{D1}, P_{D2}
2. P: $e1_D$, $d1_D$
3. User: G
4. G: URL
5. G: w_D, N_D, P_{D1}, B_D, P_{G1}, P_{G2}, $e1_G$, $d1_G$
6. G ----> P: Enc(P_{D1}) [w_D + P_{G1} + $e1_G$]
7. P: Dec(P_{D2}) [Enc(P_{D1}) [w_D + P_{G1}, $e1_G$]] & w_D == $w_{D'}$
8. K": ECDH ($d1_D$, $e1_G$)
9. P ----> G: Enc(P_{G1}) [w_D + $e1_D$]
10. G: Dec(P_{G2}) [Enc(P_{G1}) [w_D + $e1_D$]] & w_D' == w_D
    & K': ECDH ($d1_G$, e1D) & K' == K" == K
11. G ----> B: N_D [A_G, P_{G1}]
12. B: N_D, A_G, P_{G1}
13. S ----> B: N_D, Request (P_{G1})
14. B ----> G: P_{G1}
15. S ----> B: N_D, Enc(P_{G1}) [$w_{D"}$ + P_{S1}]
16. B -----> G: Enc(P_{G1}) [$w_{D"}$ + P_{S1}]
17. G: Dec(P_{G2}) [Enc(P_{G1}) [w_D" + P_{S1}]] & $w_{D'}$ == $w_{D"}$
18. G ----> B: ND, Ack, [Enc(P_{S1}) [w_D' + K]]
19. B -----> S: Enc(P_{S1}) [w_D' + K]
20. S: Dec(P_{S2}) [Enc(P_{S1}) [w_D' + K]] & $w_{D"}$ == $w_{D'}$

Figure 7.1: CGM Onboarding Message Flow

Table 7.1: Symbols and Terminologies Used in Design and Implementation of CGM Security Architecture

Notations	**Variables or Symbol Meaning**
ClientID	Unique identifier
TID	Topic id
w_P	Unique Secret Password of Sensor device
w'_P	Unique Secret Password of Sensor device stored in SmartApp
w_P"	Unique Secret Password of Sensor device stored in Subscriber
N_P	Serial Number of the Sensor device
P_{D_1}, P_{D_2}	RSA Public & Private Key of Sensor Device with key size of 1024 bits
P_{G_1}, P_{G_2}	RSA Public & Private Key of SmartApp with key size of 1024 bits
P_{S_1}, P_{S_2}	RSA Public & Private Key of Subscriber with key size of 1024 bits
$e1_D$, $d1_D$	Elliptic Curve Public & Private Key of Sensor Device with key size of 256 bits
$e1_G$, $d1_G$	Elliptic Curve Public & Private Key of SmartApp with key size of 256 bits
Data	Combination of CGM status and Timestamp
K	Shared Secret Key (i.e., AES-256)
A_G	Destination address for the SmartApp device
B_D	Bluetooth address of the Sensor device
SHA_{256}	Hash algorithm used for message header
H_P, H_S	Hash output of SHA-256 on message header generated by Sensor device & Subscriber, respectively
Enc, Dec	RSA encryption & description algorithm
e, d	Symmetric encryption & description algorithm
M	Manufacturers
MH	MQTT-SN Message Header
Message	MH + Payload

7. The sensor device uses its RSA private key to decrypt the data (P_{D_2})and compare the unique password to check if it's communicating with authorized SmartApp.

8. The sensor device uses the SmartApp's EC public key ($e1_G$) and its own EC private key ($d1_D$) to compute the shared secret (K").

9. The sensor device sends its own EC public key ($e1_D$) and unique password (w_P) to the SmartApp by encrypting it with the SmartApp's RSA public key (P_{G_1}).

10. The SmartApp decrypts, verifies the password and on successful verification, uses the sensor's EC public key ($e1_D$) and its own EC private key ($d1_G$) to generate shared secret (K'). Due to the properties of elliptic curves, both the sensor device and SmartApp arrive at the same shared secret independently, without ever transmitting the private keys or the shared secret ($K" = K' = K$). This shared secret can then be used as a key in symmetric encryption algorithms, such as AES, to encrypt and decrypt further communication between the two parties.

11. The SmartApp publishes its address (A_G), its RSA public key (P_{G_1}) and the serial number of the sensor device (N_P) (i.e., topic) to the control center to register.

12. The control center stores this information on its registry.

13. The subscribers who want to read the CGM sensor data send the subscription request with the topic (i.e., serial number of the device) and request the SmartApp's public key.

14. The control center sends the SmartApp's RSA public key to the subscriber.

15. The subscriber encrypts the unique password (w_P) and their public key (P_{S_1}) with the SmartApp's public key (P_{G_1}) and sends it to the control center using the serial number as the topic.

16. The control center forwards that information to the SmartApp based on the serial number.

17. The SmartApp decrypts the message and compares the unique secret password of the sensor device.

18. On verification, it sends an acknowledgement to the control center along with the

unique secret password (w_P), shared secret (K) by encrypting using the RSA public key of the subscriber (P_{S_1}). Otherwise, it will reject the subscriber request.

19. The control center stores the subscriber information for that particular device and forwards the encrypted message to the subscriber.

20. The subscriber decrypts and verifies the unique password and then securely stores the shared secret key (K).

With the last step, complete onboarding is done. ECDH symmetric key generation takes place every day between the sensor device and SmartApp. That shared symmetric key is sent to all registered subscriber by encrypting using their public key, which is stored at the SmartApp (i.e., gateway device). The SmartApp also tries to re-establish the whole connection if it does not receive any reading for one hour and will alert the SmartApp user. All these steps must be performed every 15 days when the user uses a new CGM sensor, as every sensor has its own RSA and EC keys. Therefore, we get the unique symmetric key (i.e., K) every day, which helps maintain long-term security.

2. CGM Working: Here are the steps involved in the process of data transfer. Figure 7.3 shows the message flow in the CGM working:

1. After the onboarding, the sensor device generates the hash of the sensor reading, secret password (w_P), and serial number of the device (N_P) along with a timestamp every 15 min.

2. The sensor device later encrypts the sensor reading, timestamp and hash generated above with the shared secret key (K), i.e. payload.

3. The sensor device uses its serial number as the client ID and SmartApp address (i.e., Message header) and combines it with payload.

4. The sensor device publishes the message.

5. The SmartApp is pre-programmed with the control center address. The SmartApp forwards the information to the control center using HTTP over TLS.

6. The control center checks the topic of the message and, based on the subscription list, forwards it to all the registered subscribers.

7. The subscriber decrypts the data using the shared secret key (K) and generates the hash of the sensor reading, timestamp, serial number (N_P) and secret password (w_P) it has received from the SmartApp during the onboarding of the device.

8. The subscriber compares the hash generated with the received within the payload. If it matches, then the data and the topic are verified, and it will take appropriate action based on the sensor reading and record the reading for the registered patient.

We also modify the RSA public-private key pair of the subscriber and SmartApp to avoid post-quantum attacks. The new RSA public keys are generated monthly and shared with the subscriber via the above onboarding process.

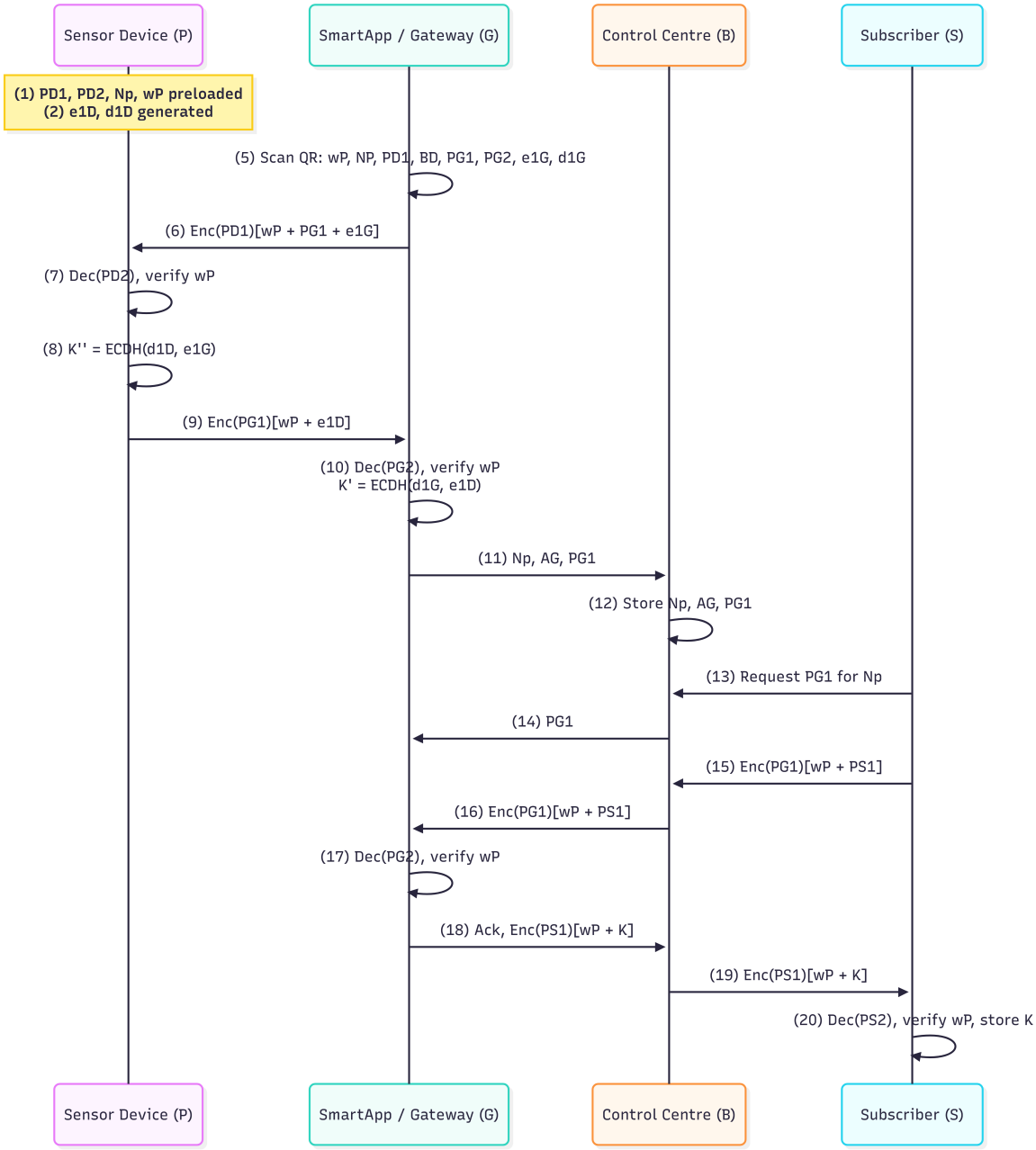


Figure 7.2: CGM Onboarding: Sequence Diagram

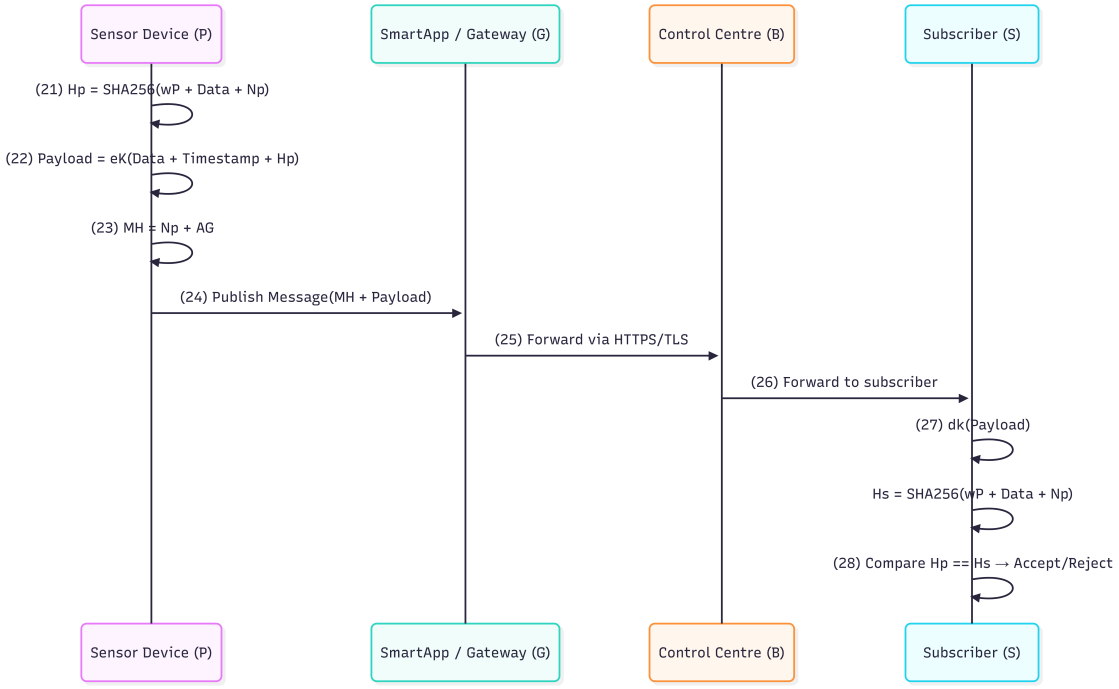


Figure 7.3: CGM Working: Sequence Diagram

7.5 Limitations

Although the proposed CGM security architecture provides strong end-to-end protection for MQTT-SN communication, several limitations arise from the assumptions and design constraints in Sections 7.3 and 7.4.

- The system assumes that each sensor device is correctly provisioned with its unique password, serial number, and cryptographic keys stored in secure ROM. Any compromise during manufacturing or onboarding cannot be detected or mitigated by the architecture.
- The threat model requires that the legitimate gateway (SmartApp) connects to the sensor before any adversarial device. If a malicious gateway reaches the sensor first, onboarding may fail or leak metadata.

- The control Center and gateway do not authenticate publishers at the network layer. A malicious sensor or subscriber with valid credentials can still inject encrypted but harmful messages before end-to-end verification occurs.

- While ECDH and RSA-1024 are feasible, they introduce computational and energy overhead on constrained microcontrollers. Frequent glucose reporting or repeated onboarding may increase latency or reduce battery life.

- Bluetooth communication remains vulnerable to jamming, interference, and proximity-based attacks. The architecture does not mitigate physical-layer disruptions, leaving availability dependent on RF conditions.

- Replay detection relies on timestamps and session-specific values. Clock drift or unsynchronized devices may lead to incorrect acceptance or rejection of messages.

- Although treated as "honest but curious," the control center remains a single point of message filtering. Misconfiguration or compromise could delay, drop, or misroute CGM readings.

- Wildcard subscriptions are disabled for security reasons, limiting scalability for multi-subscriber healthcare environments.

Chapter 8

Performance Analysis

8.1 Introduction

This chapter provides a thorough security assessment of the three proposed MQTT-SN based architectures developed in this thesis: the Vehicle-to-Emergency-Services (V2S) system, the Smart Lock system, and the Continuous Glucose Monitoring (CGM) system. In previous chapters, we talked about how security architecture of each was designed, built, and used. In this chapter, we look at their security features using timing analysis, formal verification, and informal analysis. The goal is to demonstrate that the proposed architectures not only satisfy the security requirements defined in Chapter 4 but also remain practical for constrained IoT environments.

This chapter's analysis is done in three parts that work together. Timing analysis looks at the cost of cryptographic operations on limited hardware, the time it takes for data to travel from one end to the other, and whether real-time communication is possible. Second, the Scyther protocol verification tool is used to do a formal security analysis. It checks secrecy, authentication, integrity, and replay-protection properties very carefully using the Dolev–Yao adversarial model. Finally, informal analysis looks at

how each architecture deals with the threats that are listed in the threat models. This gives you a general idea of how strong the system is.

The Scyther tool is used for formal verification because it is made to look at security protocols using symbolic modelling, automated attack discovery, and trace-based reasoning. Scyther lets us model untrusted intermediaries, like forwarders, gateways, and brokers, in the exact way that MQTT-SN communication patterns need. Scyther allows us to model untrusted intermediaries—such as forwarders, gateways, and brokers—exactly as required by MQTT-SN communication patterns. Unlike general-purpose model checkers, Scyther supports unbounded protocol runs, multiple roles, and adversarial message manipulation, making it well-suited for evaluating end-to-end security in publish-subscribe systems. Its ability to verify secrecy, agreement, synchronization, and reachability properties provides strong evidence that the proposed architectures achieve the intended security guarantees even when intermediate components are compromised.

Overall, this chapter demonstrates that the proposed MQTT-SN security architectures are both secure and practical. By combining empirical timing results with formal and informal security analysis, we provide a rigorous validation of the end-to-end protections required for safety-critical IoT applications across vehicular, smart home, and healthcare domains.

8.2 V2S Security Architecture Analysis

8.2.1 Timing Analysis

We have performed a timing analysis of the operations mentioned in Table 8.1.

- RSA-2048 Encryption is performed on a 32-bit microcontroller, the 7 ms delay demonstrates that RSA-2048 remains feasible for resource-constrained vehicular

hardware. This cost is incurred once per distress message.

- RSA-2048 Decryption is performed by the emergency service (S) using its private key. The slightly higher cost compared to encryption is expected due to RSA's asymmetric computational profile. A 33 ms delay confirms that emergency services can process incoming distress messages in real time, even under high load.

- All Components in a Room (1.23 s) value represents the best-case latency of the system and confirms that the architecture supports rapid emergency response.

- Different Location for B and S (1.78 s) scenario evaluates the system under realistic deployment conditions where the broker (control centre) and the emergency service are geographically separated.

In summary, this timing analysis shows that the cryptographic overhead is minimal and compatible with low-power vehicular hardware, end-to-end latency remains under 2 seconds, even in geographically distributed deployments. This shows that the architecture is practical for real-world V2S emergency scenarios, including rural areas with limited infrastructure.

Table 8.1: Timing Evaluation of Secure Distress Signal

Scenario	**Time Taken**
RSA-2048 Encryption	7 milli-seconds
RSA-2048 Decryption	33 milli-seconds
All Components in a Room	1.23 seconds
Different Location for 'B' and 'S'	1.78 seconds

8.2.2 Formal Analysis

The proposed V2S security architecture was formally analyzed using the Scyther protocol verification tool [177] [178] [179] [180] against the threats identified in Section 5.3. The

analysis focuses on secrecy, integrity, authentication properties across all protocol roles: Publisher (P), Forwarder (F), Gateway (G), Broker (B), and Emergency Services (S). The Scyther results confirm that the protocol achieves end-to-end security between the vehicle (P) and emergency services (S), while correctly modeling the untrusted nature of intermediate entities (F, G, B).

- **A. Secrecy Guarantees:** The security principle confidentiality requirement is that the payload (i.e., VIN, random identifier (RND), location, timestamp, and vehicle status -must remain hidden from the adversaries and untrusted intermediaries. The Scyther results confirm:
  - **Publisher (P):** All secrecy claims for vin, rnd, loc, and ts are verified. The only failing secrecy claim is status, which is expected because status is transmitted in plaintext as part of the MQTT-SN header and is not intended to be confidential.
  - **Emergency Services (S):** All secrecy claims are verified, including secrecy of both the received identifiers and the locally generated fresh values. This confirms that the encrypted payload is only accessible to S, satisfying V2S-T2.

  Forwarder, Gateway, and Broker do not possess the private key skS and are not intended to guarantee confidentiality. Their model correctly reflects their untrusted status.

- **B. Integrity and Replay Protection:** Integrity protection (V2S-T3) is achieved by hashing the MQTT-SN message header using SHA-256 and embedding the hash inside the encrypted payload. Scyther's analysis confirms that the emergency service can recompute the hash and detect any modification by a malicious forwarder

or broker. Because the hash is encrypted under pkS, no intermediate node can forge or alter it.

Replay protection (V2S-T5) is provided by including a timestamp and a per-vehicle random number (RNDc) inside the encrypted payload. Scyther verifies that these values remain secret and bound to the session, preventing an attacker from reusing old messages.

- **C. Authentication, Identification, and Agreement:** Using the Scyther tool, we confirms that:
  - **Strong agreement (Niagree) and synchronization (Nisynch)** between P and S are verified.
  - **Weak agreement** The protocol does not authenticate the publisher; it only identifies it via VIN and RNDc after decryption at S.

  The successful verification of Commit(P, vin, rnd, ts) and Nisynch for S (Figure 8.1) demonstrates that the emergency service receives a message that is cryptographically bound to the publisher's identity and session parameters, satisfying V2S-T2 and V2S-T3.

- **D. Reachability and Protocol Executability:** The final Scyther results confirm that both P and S roles are reachable and executable once the decryption and data-extraction steps are correctly modeled. This ensures that the protocol has at least one valid execution trace and that no deadlocks or impossible message patterns exist.

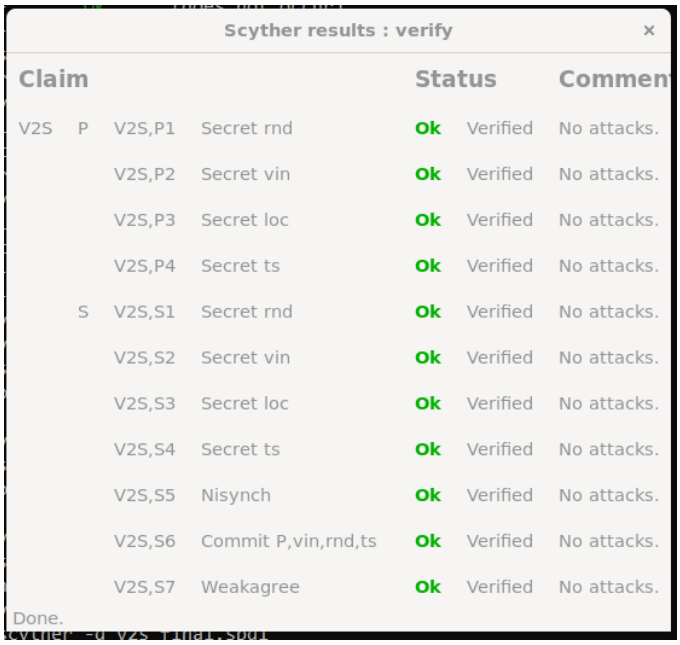


Figure 8.1: Scyther Successful Results: V2S Architecture

8.2.3 Informal Security Analysis

In this section, we provide theoretical assessments that show our system can address all the threats mentioned in Section 5.3.

- **V2S-Threat 2 (V2S-T2): Confidentiality** An adversary must not learn sensitive fields loc, vin, rnd, or ts from V2S communication. The only place where these values appear on the network is inside the ciphertext

$$Data = \mathsf{Concat}(\mathsf{status}, \mathsf{loc}, \mathsf{vin}, \mathsf{rnd}, \mathsf{ts})$$

$$C_S = \{\, H_p, \mathsf{Data} \,\}_{pk_S} = \{\, \mathsf{SHA256}(\mathsf{MH}), Data \,\}_{pk_S}.$$

Under the Dolev–Yao model [181] [182], RSA is treated as a perfect encryption

scheme: for any adversary term A,

$$A \not\equiv sk_S \;\Rightarrow\; A \text{ cannot derive } (H_p, \mathsf{Data}) \text{ from } C_S.$$

In particular, the adversary cannot derive loc, vin, rnd, or ts from C_S without sk_S. Intermediate entities F, G, and B do not possess sk_S and therefore cannot decrypt C_S. Formally, let $\mathcal{K}_{\mathsf{adv}}$ be the set of keys known to the adversary. By construction,

$$sk_S \notin \mathcal{K}_{\mathsf{adv}} \quad \Rightarrow \quad (\mathsf{loc}, \mathsf{vin}, \mathsf{rnd}, \mathsf{ts}) \notin \mathsf{closure}(\mathcal{K}_{\mathsf{adv}} \cup \mathcal{M}),$$

where $\mathcal{M}$ is the set of all messages on the network and $\mathsf{closure}(\cdot)$ is the Dolev–Yao derivation closure. Hence, confidentiality of the payload fields is preserved, satisfying V2S-T2.

In simple terms to handle the V2S-T2, we have used the RSA encryption algorithm to encrypt the payload, and encrypted information can only be decrypted by the corresponding emergency services. Due to the encrypted payload, neither the malicious FowardApp nor the *honest but curious* control centre can read the payload. In addition to this, we also continuously update the emergency services public key, which vehicle manufacturers collect from the emergency services and send to the vehicles in terms of monthly updates.

- **V2S-Threat 3 (V2S-T3): Replay Attacks** In order to prevent an adversary from exhausting emergency services by replaying old messages. Each payload includes a timestamp ts and a per-vehicle random value rnd inside the encrypted Data:

$$\mathsf{Data} = \mathsf{Concat}(\mathsf{status}, \mathsf{loc}, \mathsf{vin}, \mathsf{rnd}, \mathsf{ts}).$$

At S, after decryption, the emergency service checks:

1. whether ts is within an acceptable freshness window,
2. whether $(\mathsf{vin}, \mathsf{rnd}, \mathsf{ts})$ has already been processed.

Let $\mathcal{H}$ be the set of previously seen tuples. The emergency service enforces:

$$(\mathsf{vin}, \mathsf{rnd}, \mathsf{ts}) \in \mathcal{H} \;\Rightarrow\; \text{discard message.}$$

An attacker can replay an old ciphertext C_S, but cannot change rnd or ts inside C_S without breaking RSA. Therefore, any replayed message will either:

- have an outdated ts and be rejected by freshness checks, or
- match an existing entry in $\mathcal{H}$ and be discarded as a duplicate.

Thus, replay attacks cannot cause unbounded resource exhaustion at emergency services, satisfying V2S-T3.

In simple terms to handle the V2S-T3, we have added a timestamp in the payload, which helps the emergency services discard the replay messages from an attacker.

- **V2S-Threat 4 (V2S-T4): Integrity of Messages** An adversary must not be able to undetectably modify messages to misguide emergency services. The integrity mechanism is:

  1. At P, compute $H_p = \mathsf{SHA256}(\mathsf{MH})$.
  2. Encrypt (H_p, Data) under pk_S to obtain C_S.
  3. At S, decrypt C_S with sk_S to obtain (H_p, Data).
  4. Recompute $H_s = \mathsf{SHA256}(\mathsf{MH})$ from the received header.

5. Accept the message only if $H_s = H_p$.

Suppose an adversary modifies the header from MH to $\mathsf{MH}' \neq \mathsf{MH}$ in transit. At S, the decrypted hash is still $H_p = \mathsf{SHA256}(\mathsf{MH})$, while the recomputed hash is $H_s = \mathsf{SHA256}(\mathsf{MH}')$. If SHA-256 is collision-resistant, then

$$\mathsf{MH}' \neq \mathsf{MH} \;\Rightarrow\; \mathsf{SHA256}(\mathsf{MH}') \neq \mathsf{SHA256}(\mathsf{MH}),$$

so $H_s \neq H_p$ and the message is rejected. Similarly, the adversary cannot modify Data inside C_S without either:

- breaking RSA (producing a valid ciphertext under pk_S for a different plaintext), or
- forging a collision in SHA-256 to keep H_p consistent.

Both are excluded by our cryptographic assumptions. Therefore, any tampering with routing information (header) or payload is detected at S, satisfying V2S-T4.

In simple terms to handle the V2S-T4, we have implemented the SHA-256 technique to compute the hash of the MQTT-SN message header. Within the publish-subscribe system, the message header serves the purpose of guiding the message from the publisher to the subscriber. Consequently, it is not possible to encrypt the message header in order to ensure end-to-end security. Consequently, in order to preserve its authenticity and safeguard against harmful ForwardApp, we employ SHA-256 to compute the hash value of the message header and then encrypt it together with the Data. At the receiving end (namely, emergency services), it computes the hash value of the message header and compares it to the decrypted hash value, which is included in the payload. If the verification is successful, the

system will proceed with the request; otherwise, it will deny it.

- **V2S-Threat 5 (V2S-T5): Fake Emergency Messages and Identification** An adversary may send fake emergency messages, but the system must be able to identify the malicious source (vehicle). Each vehicle is associated with a pair $(\mathsf{VIN}_M, \mathsf{RND}_M)$ provisioned by the manufacturer and stored securely at the emergency services. At runtime, the ClientApp sends

$$\mathsf{Data} = \mathsf{Concat}(\mathsf{status}, \mathsf{loc}, \mathsf{VIN}_c, \mathsf{RND}_c, \mathsf{ts}),$$

and the emergency service S checks, after decryption:

$$(\mathsf{VIN}_M, \mathsf{RND}_M) = (\mathsf{VIN}_c, \mathsf{RND}_c).$$

Consider an adversary that forges a ciphertext C_S^* containing some pair $(\mathsf{VIN}^*, \mathsf{RND}^*)$ in the data field. There are two cases:

1. $(\mathsf{VIN}^*, \mathsf{RND}^*)$ does not match any entry in the manufacturer database. Then S rejects the message as unauthentic.
2. $(\mathsf{VIN}^*, \mathsf{RND}^*)$ matches a valid pair $(\mathsf{VIN}_M, \mathsf{RND}_M)$. In this case, the forged message is attributed to that specific vehicle identity.

Thus, while the adversary may inject fake messages, any accepted message is bound to a unique vehicle identity via $(\mathsf{VIN}_M, \mathsf{RND}_M)$. This provides identification and accountability: any fake emergency message that passes the check can be linked to a concrete vehicle, satisfying the identification aspect of V2S-T5 and enabling legal or administrative action.

Therefore, while fake messages cannot be fully prevented (as in fake 911 calls), the architecture ensures that any such message is attributable to a specific vehicle identity, satisfying the identification requirement of V2S-T5.

In simple terms to handle the V2S-T5, we use a VIN number and a unique secure random number corresponding to each VIN, which is available to the emergency services securely. That way we are able to identify the malicious publisher or vehicle used to send fake emergency messages. For example, the government cannot control fake 911 calls, only charge them with crimes. Similarly, we cannot control the fake emergency messages, however, we can charge the person as an criminal offence in case anyone attempt to do that.

- **V2S-Threat 6 (V2S-T6): Availability and Delivery** In order to ensure that the distress message reaches the gateway and emergency services even in the presence of malicious forwarders and network interference. The architecture uses:
  - broadcast at the MQTT-SN layer from P to all reachable ForwardApps,
  - multiple ForwardApps $F_1, F_2, \ldots,$
  - a fixed, non-modifiable mapping from Gateway G to Broker B.

  Let $\mathcal{F}$ be the set of ForwardApps, and assume (Assumption 2) that at least one $F^* \in \mathcal{F}$ is honest. When P broadcasts a message m, every $F \in \mathcal{F}$ receives m. A malicious forwarder may drop or alter m, but the honest F^* will forward the correct m to G. In the Dolev–Yao model, this can be expressed as:

$$P \xrightarrow{\text{broadcast}} \mathcal{F} \ \wedge \ \exists F^* \in \mathcal{F} \text{ honest } \Rightarrow \ G$$

  eventually receives m unmodified.

Since $G \rightarrow B$ uses TLS and $B \rightarrow S$ is topic-based forwarding, the message is eventually delivered to the correct emergency service. Under the stated assumptions, the probability that no correct copy of m reaches S is zero in the symbolic model, satisfying the availability requirement of V2S-T6.

In simple terms to handle the V2S-T6, we have implemented MQTT-SN, in which the publisher (referred to as the ClientApp of the vehicle) lacks knowledge about the control centres. The gateway, which serves as the road-side infrastructure, contains all the programmed information of the control centres. No vehicle is granted physical access or authorization to modify this information. We have also used MQTT-SN, in which we broadcast the ClientApp message; even if one of the FowardApp is malicious, there will be at least one, which will help the ClientApp message reach the road-side infrastructure.

8.2.4 Comparison with Different technologies

The effectiveness of the dedicated emergency button in sending emergency distress signals depends on various factors, such as the availability of the onboard communication system, the reliability of the button, and the response time of the emergency contacts and/or emergency services. Technologies used to send distress signals have many limitations [183] [184]. Here, we show that our design has overcome those limitations. Another benefit of using our prototype is that it's easy to maintain, as the software application can be easily updated. MQTT-SN is a standard protocol that overcomes one issue of lack of standardization.

The following are some of the problems faced by vehicles in sending distress signals and how our approach has addressed them:

1. Bandwidth Limitation: In rural areas, the coverage of mobile networks and GPS systems are often weak, making it difficult for vehicles to send out distress signals. However, even though ZigBee has a bandwidth limitation, by leveraging and adapting different optimizations of MQTT-SN features like message compression, topic-based filtering, QoS levels, sleep modes, and smaller packet overhead, it is possible to mitigate bandwidth limitations in V2V networks. These techniques enable efficient use of available bandwidth and ensure reliable communication while accommodating the constrained nature of ZigBee devices and networks. Alongside, DSRC is one of the widely accepted V2V communication protocols. DSRC (works on a bandwidth of 75 MHz in the 5.9 GHz band) and cellular communication share the same frequency bands used by mobile phones and other electronic devices. This can cause network congestion, dropouts, and other reliability issues. On the other hand, ZigBee (works on a bandwidth of 2 MHz in the 2.4 GHz band) uses its unique frequency band, meaning it can operate with less interference and congestion. Alongside ZigBee helps us reduce packet loss through mesh topology.

2. Power Consumption: MQTT-SN over ZigBee's low power consumption can significantly increase the lifespan of embedded devices, such as sensors and controllers. We use sleep modes to conserve power when the system is not actively transmitting or receiving data. MQTT-SN can be integrated with the power management mechanisms of ZigBee devices, allowing them to wake up efficiently and establish connections only when necessary. This helps to optimize energy usage. This feature is not supported by AEBS, DSRC, cellular and satellite.

3. Cost: The infrastructure needed for our design is already present in current vehicles and vehicles without that infrastructure can also use their smartphone for the same because the end-user (i.e., publisher) only needs to have a ClientApp, which can be

Table 8.2: Comparison of Technologies with Our Work Used in Distress Signal

Technology/ Features	**AEBS**	**DSRC**	**Cellular**	**Satellite**	**Proposed Solution**
Bandwidth Limitation	Yes	Yes	Yes	Yes	Yes
Power Consumption	Watts to 10 Watts	mWatts to Watts	10 mWatts to Watts	mWatts to Watts	mWatts to Watts
Cost	$500 to $1,500	$500 to $2,000	$100 to $500	$200 to $1,000	$30-$70
Maintenance	High	High	-	-	Medium
Range Limitation	152 meters (Yes)	1000-5000 meters (Yes)	No	No	300 meters (Yes) (Use MQTT-SN to compensate)
Security Concerns	Yes (Eavesdrop)	Yes (Eavesdrop)	Yes (Eavesdrop)	Yes (Jamming & Spoofing)	Yes (8.3)

installed on the vehicle during manufacturing or old-vehicle owners can install them over their Smartphone. The end-user only needs to purchase ZigBee modules, which cost around $20-$30. Many countries already have built roadside infrastructures, and we need to modify them to support MQTT-SN over ZigBee.

4. Maintenance: Maintenance of the various systems and technologies used to send distress signals can be difficult, especially in areas with limited access to trained technicians. Our design is easy to maintain as the software application can be easily updated. However, roadside infrastructure might need regular maintenance based on location and usage.

5. Interoperability Issues: Different technologies and protocols used by different systems may not be fully compatible, making it difficult to send distress signals across different networks. All the technologies used for sending distress signals lack a stan-

dard standardization. However, our design uses the MQTT-SN protocol, an OASIS standard, and ZigBee's support for mesh network topologies provides a reliable and scalable communication infrastructure for MQTT-SN messages.

6. Range Limitation: Cellular and Satellite technologies have no range limitation. However, AEBS has a range of 152 meters. On the other hand, DSRC has a range of 1000 to 5000 meters. Similarly, MQTT-SN over ZigBee has a range of 300 meters. So a car travelling 100 Kilometers per hour on a highway (i.e., ForwardApp) and a car not working in a remote area (i.e., ClientApp). Therefore with the delays involved in opening and closing a connection on the order of 0.02 seconds, in DSRC has 36 seconds to send the message, and AEBS has 5.2 seconds. While in the case of MQTT-SN over ZigBee, the ForwardApp has 10.8 seconds to receive broadcast messages from ClientApp. With 50 bytes of payload (within one packet), ZigBee maintains a latency below 140 milliseconds out to 6 hops. In addition, our design leverages the MQTT-SN protocol and the MQTT-SN forwarder, which acts as a repeater to forward messages from one vehicle to another when the network is unavailable. This approach can substantially improve communication reliability and extend the range of functionality for transportation systems operating in remote and rural environments.

Table 8.2 compares each technology used by different methods to send a distress signal from a vehicle.

8.2.5 Comparative Analysis: Security

There are a few V2V communication protocols used and implemented by vehicle manufacturers to send the message from one vehicle to another or vehicle to road-side infrastructure. However, the security of the information transferred by the vehicles is

also important. AEBS (Automatic Emergency Braking System) does add measures to add the secure transfer of data; its main focus is to provide safety features to prevent or mitigate collisions by autonomously applying the brakes in critical situations [185]. On the other hand, DSRC (Dedicated Short-Range Communications) does not provide confidentiality; however, it will provide integrity, authentication, identification and accountability through the digital certificates, digital signatures and unique identity of the vehicles [186]. Cellular and Sattelite communication uses the AES algorithm to provide confidentiality and digital signatures, and MACs (Message authentication codes) provide integrity, source authentication, and accountability [187] [188]. Sattelite communication is available in remote areas compared to cellular communication; however, it is costly to implement and does not support old legacy vehicles. On the other hand, our proposed solution provides confidentiality using RSA public-key cryptography and integrity using the SHA-256 hash algorithm. Our proposed solution does not focus on authentication. However, identification is an important aspect of our security architecture. The solution proposed also helps in providing accountability, and with the method of broadcasting messages through ClientApp and ForwardApp, we ensure the availability of the services. Our proposed security architecture for MQTT-SN, a publish-subscribe protocol, protects us from a malicious publisher, malicious middle entity and even curious control centre and even malicious subscribers.

8.3 Smart Lock Security Analysis

This section discuss the evaluation and limitation of the model and additionally, we have perform security analysis.

Table 8.3: Comparison of Technologies with our work used in Distress Signal

Technology/ Security Objective	**AEBS**	**DSRC**	**Cellular**	**Satellite**	**Proposed Solution**
Confidentiality	No	No	Yes	Yes	Yes
Integrity	No	Yes	Yes	Yes	Yes
Authentication	No	Yes	Yes	Yes	No
Availability	Yes	Yes	Yes	Yes	Yes
Accountability	No	Yes	Yes	Yes	Yes
Identification	No	Yes	Yes	Yes	Yes

8.3.1 Timing Evaluation

This subsection evaluates the computational cost, latency, and energy consumption of the proposed smart-lock architecture. All measurements are based on the Smart Lock Node (SLN), implemented on an 8-bit ATmega328P microcontroller running at 16 MHz. Cryptographic timings are derived from the Arduino Cryptography Library (rweather), which provides optimized AES, SHA-256, HMAC, and HKDF implementations for AVR-class devices.

A. Cryptographic Cost on SLN:

We calculated the size of message (i.e.,Concat(Cap, MAC, DS_L) is around 64 bytes (i.e., 4 AES blocks) and command message (LockID,UserID,OpCode,Timestamp) is around 32 bytes (i.e., 2 AES blocks) and audit log payload is around 64 bytes (i.e., 4 AES blocks). In section 6.4.2, steps 16–22 of the protocol represent all cryptographic operations performed on the SLN. Table 8.4 summarizes the execution time of each operation. RSA-1024 private-key decryption estimate of 200 ms based on published AVR benchmarks. The full sequence (Section 6.4.2, Steps 16–22) requires approximately 212 ms, with RSA-1024 decryption dominating the cost. This step occurs only during initial capability-token

Table 8.4: Cryptographic Cost on SLN (ATmega328P @ 16 MHz)

Operation	**Time**
RSA-1024 Decryption (Step 16)	$\approx$ 200 ms (estimated)
SHA-256 (Key Derivation, Step 17)	4.625 ms
AES-128 Decryption (4 blocks, Step 18)	2.066 ms
AES-128 Decryption (4 blocks, Step 20)	4.044 ms
SHA-256 (TopicID, Step 21)	4.896 ms
AES-256 Encryption (4 blocks, Step 22)	2.983 ms
Total (Steps 16–22) (Section 6.4.2)	$\approx$ 218.59 ms
Total (Steps 19–22) (Section 6.4.2)	$\approx$ 11.896 ms

installation. During day-to-day operation, only Steps 19–22 (Section 6.4.2) are executed, resulting in a cryptographic overhead of approximately 5.7 ms per lock/unlock operation. This demonstrates that symmetric primitives are highly efficient even on 8-bit hardware in terms of daily usage.

B. Worst-Case Lock/Unlock Latency

The total time includes BLE transmission, SLN cryptographic processing, and mechanical actuation. BLE communication typically introduces 20–50 ms of delay, while the motorized deadbolt requires 300–500 ms to complete a lock/unlock cycle. The SLN's cryptographic overhead of 11.9 ms is negligible in comparison. The worst-case end-to-end latency is therefore:

$$\text{Latency (ms)} \approx 50\,(\text{BLE}) + 11.9\,(\text{crypto}) + 400\,(\text{motor}) \approx 461.9\,\text{ms}.$$

This is consistent with commercial BLE-based smart locks, which exhibit unlock times in the 300–800 ms range.

Real-Life Time: On calculating the actual time from sending command to lock/unlock in our model comes around 115 ms, we have tried these operation 30-40 times. We

have not included the time taken by dead-bolt lock to open. However, this time include the BLE connection establishment and command processing time.

C. Energy Consumption

The ATmega328P consumes approximately 10 mA at 5 V when active, corresponding to a power draw of 50 mW. The energy consumed during a single lock/unlock operation is:

$$E = P \cdot t = 50\,\text{mW} \times 11.9 \times 10^{-3}\,\text{s} = 595\,\mu\text{J}.$$

Even the full capability-token installation sequence (dominated by RSA-1024 decryption) consumes only:

$$E_{\text{install}} = 50\,\text{mW} \times 0.218\,\text{s} = 10.9\,\text{mJ}.$$

Both values are negligible compared to the energy required for mechanical actuation, confirming that cryptographic operations do not significantly impact battery life.

8.3.2 Formal Security Analysis

In this section, we discuss the formal analysis performed using the Scyther Tool [189] [190]. We have modelled our analysis into three different part.

8.3.2.1 Trust Establishment SLN and Gateway:

We tried to show the resurrection duckling for trust establishment using Scyther tool (Figure 8.2). For final trust establishment we compare the device secret when N_L received from the SLN. All G-phase claims pass because once the SLN and Gateway share a trusted key, the rest of the protocol uses authenticated encryption and behaves securely.

Scyther results : autoverify

Claim				Status		Comment
LOCK	P	LOCK,P1	Secret BA	Ok	Verified	No attacks.
		LOCK,P2	Secret NL	Ok	Verified	No attacks.
		LOCK,P3	Secret NG	Ok	Verified	No attacks.
		LOCK,P4	Secret GID	Ok	Verified	No attacks.
		LOCK,P5	Secret DSL	Ok	Verified	No attacks.
		LOCK,P6	Alive	Ok	Verified	No attacks.
		LOCK,P7	Weakagree	Ok	Verified	No attacks.
		LOCK,P8	Niagree	Ok	Verified	No attacks.
		LOCK,P9	Nisynch	Ok	Verified	No attacks.
	G	LOCK,G1	Secret _Hidden_ 1	Ok	Verified	No attacks.
		LOCK,G2	Secret DSL	Ok	Verified	No attacks.
		LOCK,G3	Secret NL	Ok	Verified	No attacks.
		LOCK,G4	Secret GID	Ok	Verified	No attacks.
		LOCK,G5	Secret NG	Ok	Verified	No attacks.
		LOCK,G6	Secret DSL'	Ok	Verified	No attacks.
		LOCK,G7	Secret BA	Ok	Verified	No attacks.
		LOCK,G8	Alive	Ok	Verified	No attacks.
		LOCK,G9	Weakagree	Ok	Verified	No attacks.
		LOCK,G10	Niagree	Ok	Verified	No attacks.
		LOCK,G11	Nisynch	Ok	Verified	No attacks.

Done.

Figure 8.2: Scyther Result: Trust Establishment SLN and Gateway

8.3.2.2 Trust Establishment SLN, SmartApp and Backend Server

The Scyther output shows that every secrecy, authentication, and synchronization property across all three roles (P, A, and S) is successfully verified, with no attacks found within the explored bounds (Figure 8.3). This means the protocol maintains confidentiality of all sensitive values—keys, identifiers, timestamps, roles, and operation types—and ensures that each party agrees on the session state and message ordering. The presence of “Verified” or “No attacks within bounds” across all claims indicates that the modeled protocol behaves consistently and securely under an active adversary. In short, the re-

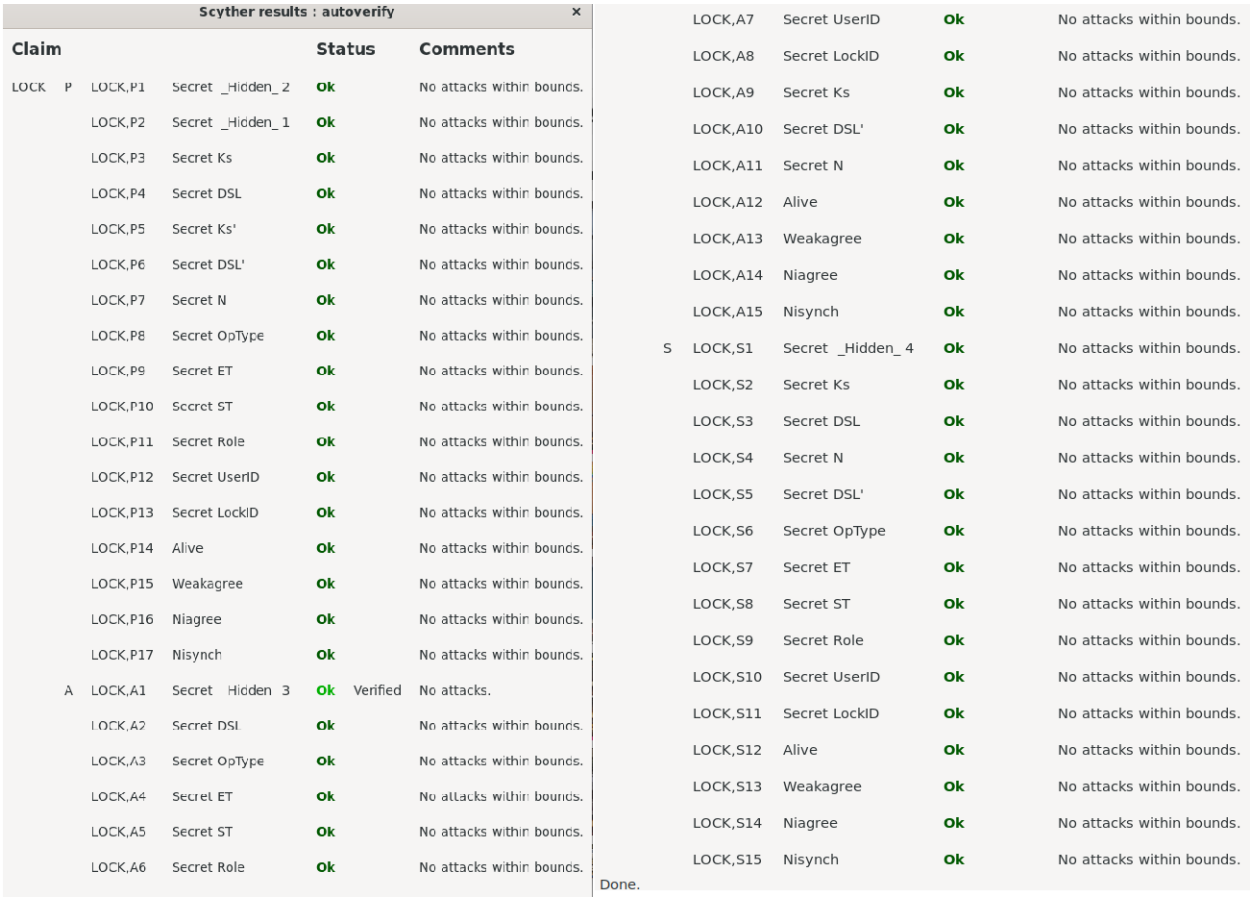


Figure 8.3: Scyther Result: Trust Establishment SLN, SmartApp and Backend Server

sults confirm that the protocol's end-to-end design, including capability tokens, session keys, and encrypted communication, satisfies all formal security goals within Scyther's analysis limits.

8.3.2.3 Trust Establishment Audit Log

Two claims for the P role, Alive and Weakagree, are wrong. The Alive violation is like a normal denial-of-service attack: the enemy can start a run of the peer (for example, by replaying or partially delivering messages) without letting P finish its own run.

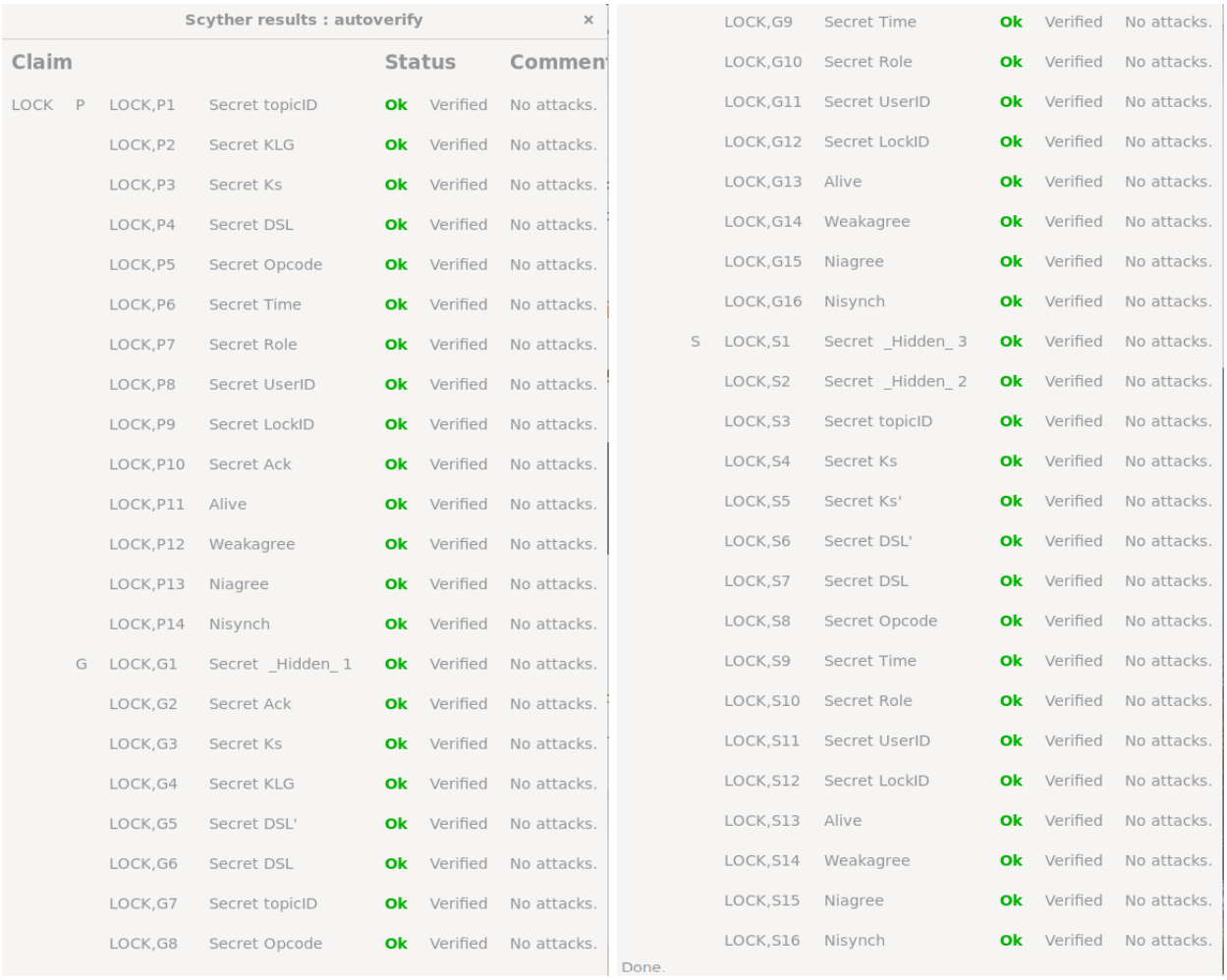


Figure 8.4: Scyther Result: Audit Log without Broker

The broker secrecy claim that is true is Secret DS_L. This proves that the device secret never leaves the protected channels between the Smart Lock Node, SmartApp, and backend. This fits with our architecture: the broker could be malicious or hacked without breaking any of the main security rules that are enforced at the endpoints. We also performed Scyther tool analysis without Broker as well (Figure 8.4).

8.3.3 Informal Security Analysis

Mitigation of SLN-Threat 1 (SLN-T1) – Spoofing Attack

The threat SLN-T1 considers an adversary is attempting to impersonate as a legitimate client. As stated in the design, "the MQTT-SN broker stores no secrets and is treated as untrusted" and all security is enforced end-to-end between the SLN, Gateway, SmartApp, and Backend Server. The backend generates a capability token

$$\mathsf{Cap} = \mathsf{Concat}(\mathsf{LockID} \| \mathsf{UserID} \| \mathsf{Role} \| \mathsf{OpType} \| \mathsf{ST} \| \mathsf{ET} \| \mathsf{Nonce}),$$

and binds it to the lock using

$$\mathsf{MAC} = \mathsf{HMAC}_{K_{LS}}(\mathsf{Cap}).$$

Since only the backend and SLN know K_{LS}, a rogue broker cannot forge valid tokens or TopicIDs. Thus, spoofing attacks are prevented.

Mitigation of SLN-Threat 2 (SLN-T2) – Passive Eavesdropping

The system assumes that BLE is insecure, and therefore all sensitive data is encrypted. As described in Section 6.4.2, the SmartApp sends

$$E_{PL1}(\mathsf{Nonce}) \parallel S_{K_s}(\mathsf{Cap}, \mathsf{MAC}, \mathsf{DSL})$$

to the SLN. All SLN–Gateway communication is encrypted and authenticated using K_{LG} derived via HKDF. Communication with the backend uses TLS 1.2. Therefore, passive eavesdroppers cannot recover keys, capability tokens, or device secrets.

Mitigation of SLN-Threat 3 (SLN-T3) – Replay Attack

Replay attacks are mitigated through the use of timestamps and nonces. The SLN verifies that the received operation request is within the interval $[\mathsf{StartTime}, \mathsf{EndTime}]$. The Capability tokens and session keys are refreshed every seven days using new nonces, as stated in the design. Thus, replayed messages either fall outside the valid time window or fail MAC verification.

Mitigation of SLN-Threat 4 (SLN-T4) – Injection Attack

All critical communication channels use authenticated encryption. Messages between SLN and Gateway are protected using K_{LG}, while SmartApp–SLN messages are protected using K_s. Any modification of ciphertexts results in MAC failure at the SLN. Audit logs are bound to specific operations using

$$\mathsf{TopicID} = \mathsf{Truncate}_K(H(\mathsf{Cap} \| \mathsf{MAC})),$$

preventing forged audit entries. Hence, message injection attacks are detected and rejected.

Mitigation of SLN-Threat 5 (SLN-T5) – Non-Ephemeral Keys

Although RSA keys are long-lived, the effective operational keys (K_{LG}, K_{LS}, K_s) are periodically refreshed using new nonces. The design explicitly states that "capability tokens are modified every 7 days using a new nonce to maintain uniqueness." Thus, the system avoids long-term reliance on static keys and reduces the impact of key compromise.

8.3.4 Comparison Evaluation

We have compared our solution with different market available Smart Locks [191] [192] [193] [194] [195]. Please refer to Table 8.5.

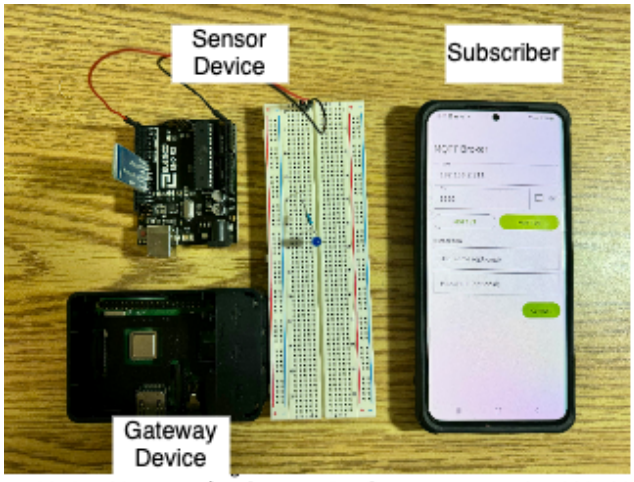
Based on the comparison, cloud-centric authorization, coarse access controls, and trusted middlemen like vendor backends and Wi-Fi bridges are all crucial components of commercial smart locks. Our proposed architecture, supports fine-grained per-operation permissions, maintains full functionality offline, and uses HMAC-verified capability tokens to enforce authorization locally on the lock. The our proposed design provides better security even on limited 8-bit hardware by treating gateways and brokers as untrusted and utilizing transparent, lightweight cryptography.

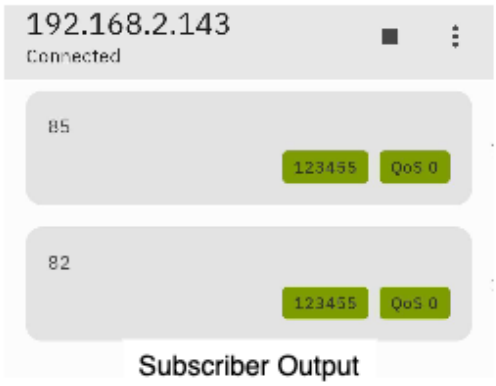
8.4 CGM Security Analysis

We have performed different analyses. Figure 8.5 shows our setup and output on the subscriber and the gateway device.

8.4.1 Timing Analysis

We have performed a timing analysis of the operations. We only considered the time taken by the 8-bit processor to perform different symmetric and asymmetric operations. Here, we have used RSA with a key size of 1024 bytes as with an 8-bit processor; we need to check the timing analysis of the operations, and based on our analysis, RSA-1024 is still one of the secure algorithms. We have compared the timing of an 8-bit microcontroller with a 32-bit processor and found a difference in timing per operation. Based on Table 8.6, we can see that the ARM process is almost 5 times faster than the 8-bit processor for 64-byte data; however, as with higher processing speed, 32-bit processors






```
1730125589: Received PUBLISH from mymqtt-124139068244151 (d0, q0, r0, m0, '123455', ... (10 b
ytes))
1730125589: Sending PUBLISH to auto-B9B926AB-FCF1-F0C5-4538-CD1BB5543583 (d0, q0, r0, m0, '12
3455', ... (10 bytes))
1730125608: Received PUBLISH from mymqtt-124139068244151 (d0, q0, r0, m0, '123455', ... (10 b
ytes))
1730125608: Sending PUBLISH to auto-B9B926AB-FCF1-F0C5-4538-CD1BB5543583 (d0, q0, r0, m0, '12
3455', ... (10 bytes))
1730125623: Received PINGREQ from mymqtt-124139068244151
1730125623: Sending PINGRESP to mymqtt-124139068244151
```

Gateway Output

Figure 8.5: CGM Secure Proposed Architecture Results

Table 8.5: Comparison Between Proposed Architecture and Real-World Smart Locks

Security Dimension	**Typical Commercial Smart Locks (August, Nuki, Yale, Schlage)**	**Proposed Architecture (This Work)**
Primary Auth Model	Phone app + Cloud account; BLE/Wi-Fi; static and long-lived credentials August, Nuki, and Yale.	Ephemeral, per-operation capability token bound to long-term key K_L.
Where Authorization is enforced	Mostly in cloud/backend;lock often trusts app/clound decision	Enforced locally on the lock using HMAC-verified tokens
Connectivity Requirements	Cloud or bridge often needed for full functionality and remote control (Wi-Fi bridge, hubs).	Fully offline-capable.
Access Granularity	Time windows and "virtual keys" supported, but usually coarse grained (guest vs owner)	per-operation, per-role, time-bounded encoded in token
Audit Model	Cloud activity log tied to account/app; broker/bridge is trusted	Cryptographically bound audit via TopicID derived from capabiliy token; gateway/broker untrusted
Crypto on Lock	BLE pairing + vendor-specific schemes; often no explicit per-operation MAC	Lightweight HMAC + hash, AES for BLE payloads, new keys after 7 days. All primitives fit on constrained 8-bit MCUs.
Failure Mode	Cloud outage, account lockout,, or bridge failure can block access.	Lock continues to operate with previously issued capabilities; no hard dependency on cloud at unlock time.

take significantly less time than 8-bit processors; however, when it comes to healthcare products, we also need to check their useful life. Strong processors use more power, leading to the battery's early drainage, which is not a good sign for mainly healthcare devices. We have used the following libraries to test the value of our system: [196] and [197]. The CGM sensor node is implemented on an ATmega328P microcontroller

operating at 16 MHz. To evaluate the feasibility of the proposed end-to-end security architecture, we measured the execution time of all cryptographic operations using the Arduino Cryptography Library (rweather), supplemented with established benchmarks for RSA-1024, ECDH-P256, AES-256, and SHA-256 on AVR-class devices. Table 8.7 summarizes the timing results.

Table 8.6: Timing Evaluation

Metric	**8-bit MCU (ATmega328P)**	**32-bit MCU (ARM Cortex-M4)**
Clock Frequency	16 MHz	48–120 MHz
AES-256 Encryption (64 bytes)	3.0–5.0 ms	0.03–0.06 ms
SHA-256 Hashing (64 bytes)	5.6–8.0 ms	0.12–0.25 ms
ECDH-P256 Shared Secret	1.2–1.6 ms	4–7 ms
RSA-1024 Decryption	180–220 ms	8–12 ms
Total Per-Reading Crypto Cost	8.6–13.0 ms	0.15–0.35 ms
Energy Consumption (Active Mode)	0.48 mJ (12–15 mA, 3.3 V))	0.006mJ (6–10 mA, 3.3V)
Typical CGM Publish Interval	5 minutes	5 minutes

8.4.1.1 Cryptographic Cost on the CGM Sensor

The onboarding phase incurs the highest computational cost, as the sensor performs RSA and ECDH operations to establish trust and derive the shared secret key. During normal CGM operation, only symmetric primitives are used, resulting in significantly lower overhead.

Table 8.7: Cryptographic Timing on ATmega328P (16 MHz)

Operation	**Description**	**Time**
RSA-1024 Decryption	Decrypts {Wp, PG1, e1G}	180–220ms
ECDH-P256 KeyGen	Generate EC key pair	1.2–1.6s
ECDH-P256 Shared Secret	Compute $K'' = \text{ECDH}(d_{1p}, e_{1G})$	1.2–1.6s
SHA-256 Hashing	64-byte block	5.6–8.0ms

Onboarding Phase (One-Time per Sensor Reset) The total onboarding cost is approximately:

$$2.59\text{–}2.63 \text{ s},$$

which is acceptable since onboarding occurs only once every 10–14 days when the CGM sensor is replaced or reset.

Normal CGM Operation (Per Reading Transmission) During routine glucose reporting, the sensor performs lightweight symmetric operations.

Table 8.8: Per-Reading Cryptographic Overhead

Operation	**Description**	**Time (ms)**
AES-256 Encryption	Encrypt 32–64 byte payload	1.5–5.0 ms
SHA-256 Hashing	Compute integrity tag	4.4–8ms
AES-256 Decryption	For acknowledgments	3.14–6.2ms

The total per-reading overhead is:

$$9.2\text{–}20.0 \text{ ms},$$

which is negligible given that CGM sensors typically publish every 5 minutes.

8.4.1.2 End-to-End Latency

End-to-end latency includes Bluetooth transmission, symmetric cryptography, MQTT-SN encoding, and gateway forwarding:

$$\text{Latency} \approx 30\text{–}50 \text{ ms},$$

excluding Internet delay. This satisfies the real-time requirements of CGM systems.

8.4.1.3 Energy Impact

The ATmega328P consumes approximately 12–15 mA in active mode. A 6 ms cryptographic cycle consumes less than 0.1 mJ, and since the sensor sleeps for more than 99.9% of its lifetime, the energy overhead of security operations is negligible compared to Bluetooth radio usage.

8.4.1.4 Summary

The onboarding phase incurs a one-time cost of approximately 0.5 s, while normal operation requires only 3–6 ms per reading. Combined with MQTT-SN's lightweight design, the architecture meets the real-time, low-power, and safety-critical requirements of CGM systems.

8.4.2 Formal Security Analysis

We have used the scyther tool for formal analysis [177] [178] [179] [180]. Scyther auto-verifies all the claims and confirms that the CGM security architecture satisfies core properties across all roles. Please refer to Figure 8.6, Figure 8.7 , Figure 8.8.

Confidentiality Claims (Secret) All Secret claims across roles were verified with no attacks, confirming that:

- Shared keys (K) are never leaked.
- Passwords (W_p and W_p') remain confidential.
- Serial numbers (N_p and N_p') are protected.
- Session identifiers and timestamps (T_s and T_s') are not exposed.

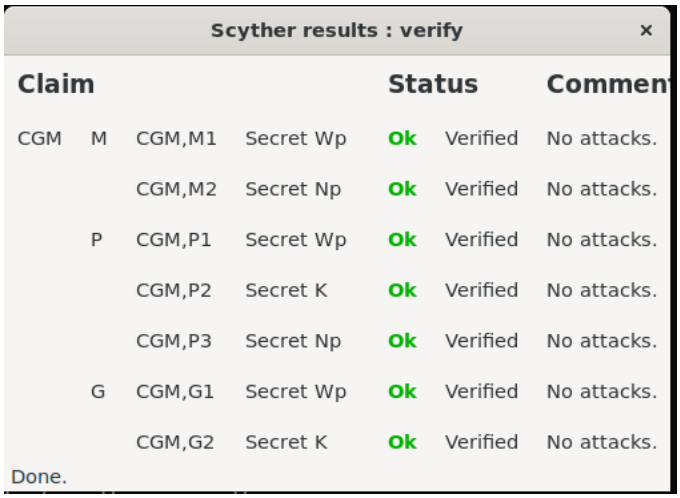


Figure 8.6: Scyther Result: CGM Publisher Gateway Onboarding

- Application-level secrets (Data) are secure.

This confirms that our use of RSA, ECDH, and AES-256 achieves end-to-end confidentiality across all communication paths.

Authentication and Agreement Claims All roles (P, G, B, S) successfully verified:

- Alive: Each party completes its protocol run without impersonation.
- Weakagree: Each party agrees the other was involved.
- Niagree: Each party confirms the origin of messages.
- Nisynch: Message order and freshness are preserved.

These results confirm that:

- Mutual authentication between sensor and SmartApp is robust.
- Subscriber onboarding and key exchange are securely bound to the correct identity.

- Message integrity and freshness are preserved throughout CGM data transmission.

Our CGM protocol passes formal verification for all critical security properties. The architecture's use of ECDH, RSA, SHA-256, and AES-256 is validated by Scyther's symbolic analysis, confirming that the protocol is resistant to spoofing, replay, MITM, and eavesdropping attacks.

8.4.3 Informal Security Analysis

- **CGM-Threat 1 (CGM-T1)-Passive eavesdropping (confidentiality):** The attacker observes MQTT-SN traffic and tries to recover plaintext glucose data. Payloads are encrypted under a symmetric key K derived via ECDH and protected by RSA during onboarding. Without private keys, the best generic attack is brute force on K:

$$P_{\text{succ}}^{(CGM-T1)}(q) \leq q \cdot 2^{-|K|} = q \cdot 2^{-256}.$$

  for q decryption attempts. For any realistic q, this is negligible, so CGM-T1 is effectively mitigated.

  In more detail, we have implemented a robust security measure by utilizing the Elliptic Curve Diffie-Hellman (ECDH) protocol to generate a unique symmetric encryption key, referenced here as AES-256. This key serves as the backbone for encrypting the sensitive payload associated with the transaction. The encryption process ensures that the information contained in the payload is secured in such a manner that only the intended subscriber, who possesses the corresponding decryption key, is able to decrypt and access the original content. The underlying principle of this encryption strategy is to address the concerns posed by an *honest but curious* control center. In this context, while the control center is acting in good

Scyther results : autoverify

Claim				Status		Comments
CGM	B	CGM,B2	Secret _Hidden_ 3	Ok	Verified	No attacks.
		CGM,B3	Secret _Hidden_ 2	Ok	Verified	No attacks.
		CGM,B4	Secret _Hidden_ 1	Ok	Verified	No attacks.
		CGM,B5	Secret K	Ok	Verified	No attacks.
		CGM,B6	Secret Wp	Ok	Verified	No attacks.
		CGM,B7	Secret Wp'	Ok	Verified	No attacks.
		CGM,B8	Secret Np'	Ok	Verified	No attacks.
		CGM,B9	Secret Np	Ok	Verified	No attacks.
		CGM,B10	Alive	Ok	Verified	No attacks.
		CGM,B11	Weakagree	Ok	Verified	No attacks.
		CGM,B12	Niagree	Ok	Verified	No attacks.
		CGM,B13	Nisynch	Ok	Verified	No attacks.
	S	CGM,S2	Secret _Hidden_ 4	Ok	Verified	No attacks.
		CGM,S3	Secret K	Ok	Verified	No attacks.
		CGM,S4	Secret Np'	Ok		No attacks within bounds.
		CGM,S5	Secret Wp	Ok	Verified	No attacks.
		CGM,S6	Secret Wp'	Ok	Verified	No attacks.
		CGM,S7	Alive	Ok	Verified	No attacks.
		CGM,S8	Weakagree	Ok	Verified	No attacks.
		CGM,S9	Niagree	Ok	Verified	No attacks.
		CGM,S10	Nisynch	Ok	Verified	No attacks.
	G	CGM,G2	Secret _Hidden_ 5	Ok	Verified	No attacks.
		CGM,G3	Secret K	Ok	Verified	No attacks.
		CGM,G4	Secret Np	Ok	Verified	No attacks.
		CGM,G5	Secret Wp	Ok	Verified	No attacks.
		CGM,G6	Secret Wp'	Ok	Verified	No attacks.
		CGM,G7	Alive	Ok	Verified	No attacks.
		CGM,G8	Weakagree	Ok	Verified	No attacks.
		CGM,G9	Niagree	Ok	Verified	No attacks.
		CGM,G10	Nisynch	Ok	Verified	No attacks.

Done.

Figure 8.7: Scyther Result: CGM Gateway Subscriber Onboarding

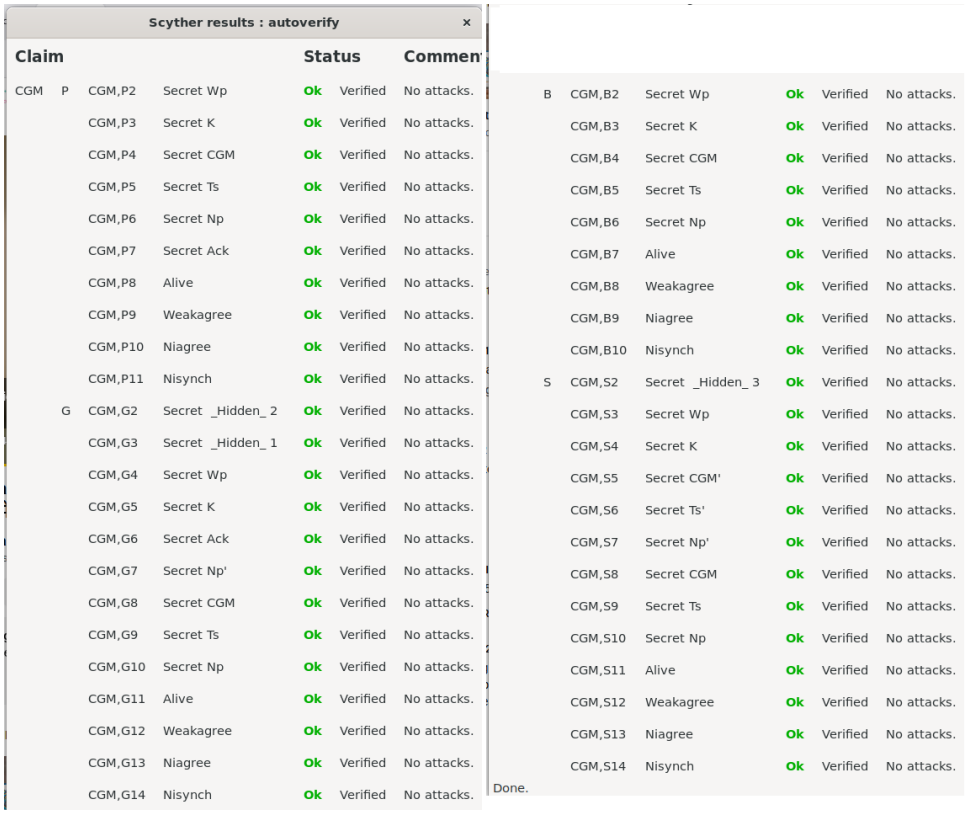


Figure 8.8: Scyther Result: CGM Working

faith, it may still seek to gain insights into the data being transmitted. However, due to the implementation of strong encryption, the control center is unable to read the contents of the encrypted payload, thus safeguarding the privacy of the subscriber's information. To further enhance security, we ensure that a new symmetric key is generated and shared on a daily basis through our SmartApp. This dynamic key management system not only reinforces the security of each transaction but also minimizes the risk of potential key compromise over time. By regularly rotating the encryption keys, we are able to maintain a high level of confidentiality and integrity in our communications, ensuring that even if a key were to be intercepted, it would only remain valid for a very limited time frame. In summary, the combination of ECDH-derived symmetric encryption, an *honest but curious* design framework, and a daily key exchange protocol through the SmartApp collectively strengthens the security of our transaction handling process, ultimately protecting our users' sensitive information from any unauthorized access.

- **CGM-Threat 2 (CGM-T2)- Spoofing (authentication, authorization):** The attacker guesses a valid ClientID (serial number) and attempts to impersonate a sensor or subscriber. Even if a valid N_D is guessed with probability

$$P_{\text{ID}} = 2^{-|N_P|},$$

the attacker must still produce:

  - a valid ciphertext under AES-256 (without K), and
  - a valid hash header H_P or H_S bound to W_P, N_P, and Data.

Without K and W_P, forging a valid hash is equivalent to a successful preim-

age/collision on SHA-256:

$$P_{\text{hash}} = 2^{-|H|} = 2^{-256}.$$

Overall spoofing success probability:

$$P_{\text{succ}}^{(CGM-T2)} \leq 2^{-|N_P|} \cdot 2^{-|H|} = 2^{-(|N_P|+256)}.$$

which is negligible for realistic N_P.

In full detail, we implemented a robust security protocol centered around a unique secret password. This password is communicated securely to our subscribers through an out-of-band channel, ensuring that only authorized users have access to it. This method minimizes the risk of interception or unauthorized access, creating a secure environment for effective communication. When a subscriber initiates a request, it is transmitted to SmartApp via the control center. At this point, the control center undertakes a crucial role in the authentication process. It verifies the integrity of the unique password associated with the subscriber's account. If the authentication is successful, the control center proceeds to approve the subscriber's request, facilitating a reliable and secure interaction. Similarly, the process flows in reverse when the control center receives messages from publishers. The control center rigorously checks the unique password to authenticate the publisher, ensuring that only legitimate sources can send messages. This two-fold authentication mechanism not only protects against unauthorized access but also maintains the integrity of the data being exchanged. Importantly, every CGM (Continuous Glucose Monitoring) sensor is equipped with its own unique secret password. This individual password enhances security by preventing cross-sensor data breaches and ensuring that each

device operates within its secure environment. By leveraging these unique passwords across our system, we cultivate a highly secure network that safeguards both subscriber and publisher communications.

- **CGM-Threat 3 (CGM-T3)– Injection (integrity):** The attacker intercepts a valid message, modifies it, and forwards a corrupted version. Each message carries a hash:

$$H_P = SHA256(W_P, Data, N_D)$$

At the subscriber, the receiver recomputes

$$H_S = SHA256(W_P, Data, N_D)$$

and checks H_S=H_P. Any modification without knowledge of W_p yields a random hash output. The probability that a modified message still satisfies H_S=H_P is:

$$P_{\text{succ}}^{(CGM-T3)}(q) \leq q \cdot 2^{-|H|} = q \cdot 2^{-256},$$

for q forgery attempts, which is negligible.

In more detail, to handle the threat CGM-T3 messages effectively, we have implemented the SHA-256 technique for calculating the hash of both the MQTT-SN message header and its payload. In the context of a publish-subscribe messaging system, the message header plays a crucial role in routing the message from the publisher to the intended subscriber. Its importance in the overall communication process means that encrypting the message header is not feasible if we aim to maintain end-to-end security across the network. Given this limitation, we have adopted a dual approach to ensure the integrity and authenticity of our messages.

Instead of encrypting the message header—which would hinder its visibility during transit—we instead use the SHA-256 technique to generate a hash value that encompasses both the message header and the payload. This calculated hash value is an essential component in preserving the integrity of the message while protecting it from potential threats posed by malicious middle devices. Once the hash value is computed, we then proceed to encrypt both this hash value and the actual data payload together. This encryption ensures that, while the header remains unencrypted for routing purposes, the critical components of the message are still protected from unauthorized access and manipulation. Upon reaching the subscriber, a verification process is initiated to uphold the security framework. The subscriber computes the hash value of the received message header anew and then compares it to the decrypted hash value that was transmitted alongside the payload. This verification step serves as a critical checkpoint: if the computed hash value matches the decrypted hash, it signifies that the message has remained intact and unaltered during transit, allowing the system to proceed with processing the request. Conversely, if the hash values do not align, this discrepancy indicates potential tampering or interception of the message. In such a case, the system will deny the request, thereby safeguarding the integrity of the communication and protecting against unauthorized access or malicious actions. By leveraging this method, we not only enhance the security of the MQTT-SN messaging protocol but also ensure that our system remains resilient against attacks, thereby maintaining the reliability and trustworthiness of the communication between publishers and subscribers. This approach highlights a balanced trade-off between usability and security in message handling.

- **CGM-Threat 4 (CGM-T4)- Man-in-the-middle (confidentiality, integrity, authentication):** The attacker attempts to sit between sensor and SmartApp or SmartApp and control center, altering keys or messages. The shared secret K is computed via ECDH:

$$K = ECDH(d_{1D}, e_{1G}) = ECDH(d_{1G}, e_{1D}),$$

with public keys exchanged inside RSA-encrypted onboarding messages and bound to W_pP. A successful MITM must either:

  - break RSA-1024,
  - solve the elliptic curve discrete logarithm problem, or
  - guess W_P to forge valid onboarding messages.

Under standard assumptions:

$$P_{RSA} \approx 0, \quad P_{ECDH} \approx 0, \quad P_{guessW_P} = 2^{-|W_P|}$$

Hence

$$P_{\text{succ}}^{(CGM-T4)} \approx 2^{-|W_P|}.$$

which is negligible for sufficiently strong W_P.

In more detail to effectively manage the threat CGM-T4, we have implemented a robust system that includes the integration of public keys for the sensor device. These keys are meticulously stored within a sealed booklet, ensuring that only the device owner has access to them. This design not only enhances security but also allows the owner to securely scan and manage their keys without fear of interception

or unauthorized access. During the initialization phase, both the SmartApp and the sensor devices collaboratively generate ECDH (Elliptic Curve Diffie-Hellman) keys. This process is vital for establishing a secure communication channel. When it comes to pairing the devices, the SmartApp utilizes the RSA public key of the sensor device. This enables the SmartApp to send its own public key along with a unique secret password. This password acts as a crucial element for the verification process carried out by the sensor device, ensuring that only authorized connections are established. Crucially, the framework we have developed ensures that no data is transmitted in plaintext between any two components. This approach significantly mitigates the risks of data breaches and eavesdropping, as all communications are encrypted. The entire system has been designed with a strong emphasis on maintaining data integrity and confidentiality, thereby fostering trust in the use of the sensor devices and the SmartApp. By adhering to these security measures, we aim to create a safe and reliable environment for users to interact with their devices.

- **CGM-Threat 5 (CGM-T5)– Replay (integrity, freshness):** The attacker records a valid message and replays it later. Messages include time-varying data (e.g., timestamp) and are hashed as:

$$H_P = SHA256(W_P, Data, N_P)$$

. At the subscriber, freshness checks (timestamp window, sequence numbers, or replay cache) reject old or duplicate messages. A pure replay of an old message:

  – fails freshness checks with probability close to 1, or

  – is detected as a duplicate if a replay cache is maintained.

If the attacker modifies the timestamp to appear fresh, they must recompute a valid H_P without W_P, giving

$$P_{\text{succ}}^{(CGM-T5)}(q) \leq q \cdot 2^{-|H|} = q \cdot 2^{-256}.$$

Thus, replay attacks are effectively mitigated.

In more detail to handle threat CGM-T5, we have implemented a timestamp feature within the payload. This addition plays a crucial role in enabling the subscriber to effectively identify and discard any replay messages that an attacker may send. Replay attacks occur when an unauthorized entity intercepts and then retransmits valid data or messages to deceive the recipient. By incorporating a timestamp into each message, we provide a mechanism for the subscriber to ascertain the freshness of the incoming data. When a message is received, the subscriber can quickly check the timestamp against its own current time. If the received timestamp is older than a predefined threshold—indicating that it has been sent at an earlier time than acceptable—the subscriber can safely disregard it, thus preventing any potential exploitation of sensitive information or commands. This strategy not only strengthens our overall security posture but also enhances the reliability of our messaging system by ensuring that only real-time, legitimate communications are acted upon. In summary, the addition of a timestamp in the payload is a vital step towards safeguarding our applications against various forms of attack, particularly replay attacks.

In overall security, for all CGM-T1 to CGM-T5, under standard cryptographic assumptions and sufficient entropy in K, W_p, and N_D, the architecture informally satisfies

confidentiality, integrity, authentication, authorization, and freshness.

$$P_{\text{succ}}^{(Ti)} \ll 2^{-256}, \qquad i \in \{1, 2, 3, 4, 5\},$$

indicating negligible success probability.

8.4.4 Comparative Analysis

As we mentioned earlier, multiple CGM devices are available in the market. However, limited research is available on the security of these CGM devices based on our knowledge. We compare our proposed solution with these products based on the resources available [198], [133], [199]. The functionality of CGM devices such as Dexcom G6, Abbott FreeStyle Libre 3, Medtronic Guardian Connect and Senseonics is the same. Therefore, this product uses BLE security (Sensor to Monitor) and HTTPS (Monitor to Cloud) to send data. Even on the cloud, data is encrypted. However, they also share common vulnerabilities, such as weak Bluetooth security during data transfer between sensor devices and mobile applications, reliance on cloud storage or third-party applications, which risk data leaks, and weak device pairing methods that can result in MITM and replay attacks.

On the other hand, in our security architecture, we use RSA and ECDH to set up the shared secret for data encryption alongside BLE security. This shared secret is generated daily and shared with the subscriber via SmartApp, as it has the RSA public key of the authorized subscriber. We have also used SHA-256 to maintain the integrity of the message sent from the sensor device to the subscriber. We do not store the data on the cloud. All data is stored at the authorized subscriber end to avoid single-point failure; therefore, a publish-subscribe system is the best solution. We also check the connection between the sensor device and the SmartApp to prevent any denial of service attack. We

can provide a better security architecture for CGM devices using all these mechanisms. Table 8.9 shows the comparative analysis discussed above.

Table 8.9: Comparison of Security of CGM Products and Proposed Solution

Technology/ Security Objective	**Dexcom G6**	**Abbott Libre 3**	**Medtronic Guardian Connect**	**Proposed MQTT-SN CGM Architecture**
Confidentiality	BLE encryption + HTTPS; no public E2E design	BLE encryption; cloud TLS	BLE + app/cloud TLS	Full end-to-end encryption (ECDH + AES-256 + RSA); broker and gateway cannot decrypt
Integrity	Cloud-side validation; no public message-level integrity	BLE pairing integrity; no published payload integrity	App/cloud integrity checks	SHA-256 integrity over header + payload; tamper-evident at subscriber
Authentication	Device/app pairing; cloud account login	BLE pairing; app login	BLE pairing + cloud account	Mutual authentication using RSA keys + unique password (Wp)
Availability	Dependent on BLE + cloud uptime	BLE + cloud	BLE + cloud	Lightweight MQTT-SN over BLE; supports sleeping devices; resilient to DoS on intermediaries, Keep-Alive Messages
Authorization	Cloud controls access; assumes trusted backend	Cloud-based authorization	Cloud-based authorization	Authorization enforced at endpoints; subscribers must possess Wp + RSA keys

Chapter 9

Conclusion and Future Work

9.1 Summary of Contributions

Here, we have summarized the contributions of our research:

- **A Cohesive Security Requirements Engineering Framework Tailored for IoT:** This thesis makes a big contribution by combining SREP and SQUARE into a single security requirements engineering method that is made just for IoT ecosystems with limited resources. Conventional requirements engineering methodologies presuppose robust devices, reliable connectivity, and sophisticated user interfaces—presumptions that are not applicable to MQTT-SN implementations. Our new framework adds IoT-specific threat categories, asset classifications, and risk-scoring systems that take into account the fact that devices are different, batteries have limits, clients sleep, and gateways are the main focus. Using this framework in three different areas—vehicle communication, smart locks, and continuous glucose monitoring—shows that it is general and strong. The threat models and prioritized security requirements that come out of this work give a systematic basis for designing secure MQTT-SN systems in many different fields.

- **Security architectures for systems like V2S, Smart Lock, and CGM:** The thesis presents three comprehensive, end-to-end security architectures for MQTT-SN-based systems, each customized to the specific constraints and risks of its domain:
  - V2S Architecture: Secure onboarding, setting up temporary keys, sending distress signals that are verified, and message flows that can't be replayed.
  - Smart Lock Architecture: Lightweight mutual authentication, secure command delivery, audit logging, and protection against spoofing and unlocking without permission.
  - CGM Architecture: This protects sensor data all the time with energy-efficient cryptography, makes sure that sensors and gateways can only connect with each other, and keeps medical telemetry private.
- **Evaluation of systems like V2S, Smart Lock, and CGM:** Formal verification with Scyther, timing and energy analysis on limited microcontrollers, and comparisons with commercial systems are used to check each architecture. The results show that MQTT-SN can be designed with strong security guarantees like authentication, confidentiality, integrity, authorization, and replay protection without going over the limits on processing power or energy of limited IoT devices.

Security Blueprint for MQTT-SN That Works Across Domains and Can Be Used Again: The thesis creates the first cross-domain, reusable security blueprint for MQTT-SN by designing and testing security architectures in three different IoT domains. This plan shows that:

- We don't have to use link-layer encryption to protect MQTT-SN at the application layer.

- A consistent set of rules—secure onboarding, lightweight key exchange, authenticated publish-subscribe flows, and cryptography with little overhead—can be used in healthcare, smart homes, and vehicles.
- The fact that MQTT-SN is lightweight does not mean that it can't be secure when supported by well-designed protocols.

These contributions establishes MQTT-SN as a feasible and secure communication framework for forthcoming IoT implementations, laying the groundwork for standardization initiatives and industry acceptance.

9.2 Best Practices for Deploying Secure MQTT-SN Architecture

1. **Treat the Gateway as Untrusted (Zero-Trust Principle)**

   Across all three architectures, the gateway is the most vulnerable component.

   - Never rely on ZigBee/BLE link-layer security alone.
   - Always enforce application-layer authentication and encryption.
   - Gateways must not store long-term secrets.
   - Gateways must only forward AEAD-protected frames.

   All architectures assume gateway compromise and still maintain confidentiality, integrity, and replay protection.

2. **Use a Standardized Secure Onboarding Procedure**

   Onboarding must be mandatory, lightweight, and uniform across all use cases.

- Use OOB or QR-based bootstrap for identity binding.
- Establish a Temporary Key (TK) and upgrade to a Long-Term Key (LTK).
- Bind device identity (DID) to Topic IDs during onboarding.
- Ensure onboarding is formally verified.

3. **Use Application-Layer Key Establishment (Not Link-Layer)**

   MQTT-SN must not depend on ZigBee/BLE encryption.

   - Use ECDH/ECDHE when possible.
   - Use SKKE for ultra-constrained devices.
   - Derive SK, IK, and replay counters from the same handshake.
   - Refresh session keys periodically or after N messages.

4. **Protect Every MQTT-SN Message with AEAD**

   This is the strongest cross-domain pattern in the thesis.

   - Use AES-CCM or AES-GCM for confidentiality and integrity.
   - Bind Topic ID to the AEAD associated data.
   - Include replay counter in every message.
   - Use per-session nonces or counters.

5. **Enforce Replay Protection Using Counters or Nonces**

   Replay attacks appear in all threat models.

   - Use monotonic counters per session or per topic.
   - Reject messages with stale counters.

- Reset counters only after secure re-keying.
- Counters must be part of AEAD associated data.

6. **Use Topic-ID–Based Authorization (Not Topic Strings)**

   Topic IDs are lightweight and ideal for constrained devices.

   - Bind Topic ID to device identity during onboarding.
   - Enforce authorization at both gateway and broker.
   - Use capability-based access (publish, subscribe, control).
   - Never allow wildcard Topic IDs for constrained devices.

7. **Use Formal Verification for Every Protocol Phase**

   Use formal verification tools such as Scyther and Tamarin Prover.

   - Verify onboarding, key establishment, messaging, and replay protection.
   - Validate secrecy, authentication, agreement, and reachability.
   - Use formal models before hardware deployment.

8. **Benchmark Cryptographic Cost on Actual Constrained Hardware**

   Measure timing and energy cost of each operation on constrained devices.

   - Measure ECDH/SKKE timing.
   - Measure AEAD encryption/decryption timing.
   - Measure energy consumption per message.
   - Ensure latency fits domain requirements (e.g., < 200 ms for safety systems).

9. **Use a Modular Architecture That Scales Across Domains**

   The same architecture works for vehicles, smart locks, healthcare, and can extend to Smart Grid, Industrial IoT, Wearables, and Smart Cities.

   Best practices:

   - Keep onboarding, keying, messaging, and authorization modular.
   - Avoid domain-specific assumptions in protocol design.
   - Use the same AEAD + counter + Topic-ID model everywhere.

10. **Compare Against Existing Technologies Before Deployment**

    Best practices:

    - Validate that MQTT-SN + AEAD is lighter than TLS.
    - Validate that MQTT-SN is more natural for publish-subscribe IoT.
    - Validate that application-layer security survives link-layer compromise.

These ten best practices form clean, reusable, and scalable guidelines suitable for future MQTT-SN standardization.

9.3 Limitations

Generalized Limitations of Deploying Secure MQTT-SN Architectures:

- MQTT-SN targets devices with very low RAM, CPU, and battery, which restricts use of heavy cryptographic primitives, large key sizes, complex handshake protocols, and frequent re-keying or high-rate AEAD operations.

- No native security support in MQTT-SN specification.
- Even with zero-trust design, gateways introduce unavoidable limitations such as single points of failure, can be physically accessed or tampered with, used for traffic analysis even if messages are encrypted, and add latency and processing overhead.
- Link layer protocol (such as, ZigBee/BLE) security is not sufficient, but MQTT-SN depends on these layers for transport. If link-layer encryption fails, MQTT-SN has no fallback.
- Secure onboarding cannot be fully automated for constrained devices. It needs human or infrastructure support such as OOB channels (QR, NFC, physical pairing), trusted initial environment.
- Cannot scale easily to thousands of devices without automation infrastructure. Physical onboarding is slow for large deployments.
- Key establishment is expensive for ultra-constrained devices. Even lightweight ECDH/SKKE operations introduce noticeable latency, energy consumption spikes, increased onboarding time and reduced battery life for medical or wearable devices.
- Replay counters introduce limitations, need persistant counters: counters must be stored in non-volatile memory; power loss can desynchronize counters; counter resets require secure re-keying; multi-gateway environments complicate counter synchronization.
- Topic-ID–based access control is lightweight but hard to manage at scale.
- MQTT-SN deployments with multiple gateways might face key synchronization issues, replay counter conflicts, increased attack surface.

9.4 Conclusion

The need for safe, portable, and compatible communication methods for limited devices has increased due to the IoT explosive growth. In order to overcome this difficulty, this thesis concentrated on MQTT-SN, the only publish-subscribe protocol specifically created for devices operating under severe computational, memory, and energy constraints and lacking IP connectivity. "MQTT-SN is used in IoT environments for constrained devices, which do not have IP-addressing capabilities... it improves efficiency in data transmission as it has a small message size compared to other messaging protocols," according to the thesis. Because of its distinct positioning, MQTT-SN is both promising and under-secured, making it a prime candidate for thorough research.

This thesis showed that the current MQTT-SN specification leaves significant gaps in authentication, confidentiality, integrity, authorization, and trust establishment across three representative IoT domains: continuous glucose monitoring, smart home access control, and vehicle communication. Despite known flaws, current deployments usually rely on link-layer security (ZigBee, BLE). "When developers rely on encryption provided by link layer protocols like ZigBee, if link-layer encryption fails...the developers are left in a situation where what they rely on does not deliver what they hope," the thesis states. The thesis's main contribution—a unified, application-layer security architecture for MQTT-SN that is resilient even in the event that lower layers are compromised—was inspired by this observation.

9.5 Future Work

Based on on the limitations identified in the V2S, CGM, and Smart Lock security architectures, a few directions for future research are:

- **Adding support for large-scale, multi-gateway deployments to the Security Architecture:** The existing architectures are based on the idea that there will only be a few gateways and tiny networks. In future research investigate scalable key management, distributed trust anchors, and hierarchical authorization frameworks appropriate for dense IoT ecosystems, including smart cities or hospital networks.

- **Enhancing Mobility Support for V2S and V2V Scenarios:** The V2S architecture assumes intermittent connectivity and short-range ZigBee links. Future research can explore seamless handover, roaming trust establishment, and cross-domain authentication for high-mobility vehicular environments.

- **Making Energy Optimization Better:** The CGM and Smart Lock tests reveal that cryptographic processes can shorten battery life. Future studies could provide adaptive security modes that change the strength of cryptography based on the battery level, the situation, or the severity of threat.

- **Real-World Deployment Studies and Long-Term Reliability Testing:** The architectures were validated through prototypes and simulations. Future work should include long-term field deployments in homes, vehicles, and clinical environments to study reliability, latency under load, and user-centric factors.

- **Standardization Contributions to the Evolving MQTT-SN Specification:** As MQTT-SN is undergoing active revision, future work can translate the thesis findings into contributions for authentication, onboarding, and message-protection extensions in the upcoming MQTT-SN standard.

References

[1] H. Gupta and P. C. van Oorschot. Onboarding and software update architecture for IoT devices. In *2019 17th International Conference on Privacy, Security and Trust (PST)*, pages 1–11. IEEE, 2019.

[2] K. Pelz. Robustness of Publish/Subscribe Systems: A Survey. *Department of Telecommunication Systems, Technical University Berlin*, 2020.

[3] Y. Liu and B. Plale. Survey of publish subscribe event systems. *Computer Science Dept, Indian University*, 16, 2003.

[4] N. A. Somani and Y. Patel. ZigBee: A low power wireless technology for industrial applications. *International Journal of Control Theory and Computer Modelling (IJCTCM)*, 2(3):27–33, 2012.

[5] H. Gupta and A. Nayak. Use of MQTT-SN in Sending Distress Signals in Vehicular Communication. In *International Symposium on Networks, Computers and Communications (ISNCC)*, doi: 10.1109/ISNCC58260.2023.10323828, 2023.

[6] C. Bellman and P. C Van Oorschot. Analysis, Implications, and Challenges of an Evolving Consumer IoT Security Landscape. In *17th International Conference on Privacy, Security and Trust (PST)*. IEEE, 2019.

[7] Canadian Institute for Knowledge Development. An Introduction to Internet of Things (IoT). https://cscitconf.com/an-introduction-to-internet-of-things-iot/, 2019.

[8] C. Bormann, M. Ersue, and A. Keränen. Terminology for Constrained-Node Networks. RFC 7228, May 2014.

[9] A. Banks and R. Gupta. MQTT Version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, Oct 2014.

[10] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[11] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120, 2011.

[12] R. Jeyaraman and A. Telfer. OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 0: Overview. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html, Oct 2011.

[13] U. Hunkeler, H. L. Truong, and A. J. Stanford-Clark. MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks. In *Third International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE)*. IEEE, 2008.

[14] A. Stanford-Clark and H. L. Truong. MQTT for Sensor Networks (MQTT-SN) Protocol Specification Version 1.2, Nov 2013.

[15] A. Banks, R. Gupta, E. Briggs, and K. Borgendale. MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, Mar 2019.

[16] S. Johnson. MQTT SN Subcommittee Public Documents. https://www.oasis-open.org/committees/documents.php?wg_abbrev=mqtt-sn&show_descriptions=yes, Nov 2021.

[17] C. S. Park and H. M. Nam. Security Architecture and Protocols for Secure MQTT-SN. *IEEE Access*, 8:226422–226436, 2020.

[18] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang. Burglars' IoT paradise: Understanding and mitigating security risks of general messaging protocols on IoT clouds. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 465–481. IEEE, 2020.

[19] TJ OConnor, W. Enck, and B. Reaves. Blinded and confused: uncovering systemic flaws in device telemetry for smart-home Internet of Things. In *12th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019.

[20] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *35th Annual Hawaii International Conference on System Sciences*. IEEE, 2002.

[21] M. Ryan. Bluetooth: With Low Energy Comes Low Security. In *Workshop on Offensive Technologies*. USENIX, 2013.

[22] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. *IEEE Security Privacy*, 16(1):54–62, Jan 2018.

[23] O. Sadio, I. Ngom, and C. Lishou. Lightweight Security Scheme for MQTT/MQTT-SN Protocol. In *Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 2019.

[24] MH Amaran, MS Rohmad, LH Adnan, NN Mohamed, and H Hashim. Lightweight security for MQTT-SN. *International Journal of Engineering & Technology*, 7(4.11):223–226, 2018.

[25] C. Gotz. Trends to watch in 2021 for MQTT and HiveMQ. https://www.hivemq.com/blog/trends-to-watch-in-2021-for-hivemq-and-mqtt/, 2021.

[26] uBlox. MQTT Anywhere. https://www.u-blox.com/en/product/mqtt-anywhere, 2020.

[27] OASIS-OPEN. MQTT SN Subcommittee Public Documents. https://www.oasis-open.org/committees/documents.php?wg_abbrev=mqtt-sn&show_descriptions=yes, 2021.

[28] M. Khan. Enhancing Privacy in IoT Devices through Automated Handling of Ownership Change. http://urn.fi/URN:NBN:fi:aalto-201709046805, 2017-08-28.

[29] A. Menon-Sen, A. Melnikov, N. Williams, and C. Newman. Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms. RFC 5802, July 2010.

[30] S. Josefsson and N. Williams. Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family. RFC 5801, July 2010.

[31] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, January 2012.

[32] S. Imane, M. Tomader, and H. Nabil. Comparison Between CoAP and MQTT in

Smart Healthcare and Some Threats. In *International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*, 2018.

[33] N. Naik. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In *IEEE International Systems Engineering Symposium (ISSE)*, 2017.

[34] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor. CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets. RFC 8323, February 2018.

[35] A. Melnikov and I. Fette. The WebSocket Protocol. RFC 6455, December 2011.

[36] R. A. Rahman and B. Shah. Security analysis of IoT protocols: A focus in CoAP. In *3rd MEC international conference on big data and smart city (ICBDSC)*. IEEE, 2016.

[37] M. Brachmann, O. Garcia-Morchon, and M. Kirsche. Security for practical CoAP applications: Issues and solution approaches. *GI/ITG KuVS Fachgesprch Sensornetze (FGSN). Universitt Stuttgart*, 2011.

[38] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni. A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. In *12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015.

[39] N. Nikolov. Research of MQTT, CoAP, HTTP and XMPP IoT Communication protocols for Embedded Systems. In *XXIX International Scientific Conference Electronics (ET)*, 2020.

[40] P. Millard, P. Saint-Andre, and R. Meijer. XEP-0060: publish-subscribe. *XMPP Standards Foundation*, 2010.

[41] Adam Langley et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the ACM SIGCOMM*, 2017.

[42] Jana Iyengar and Martin Thomson. Quic: A udp-based multiplexed and secure transport. RFC 9000, 2021.

[43] Ian Cook et al. Quic loss detection and congestion control. *IETF Draft*, 2020.

[44] Ashraf Matrawy et al. Demystifying quic for iot: Performance evaluation on constrained devices. In *IEEE ICC*, 2022.

[45] Ashraf Matrawy et al. Mqtt over quic: A secure and efficient transport for iot. In *IEEE Globecom*, 2023.

[46] Arash Molavi Kakhki et al. End-to-end performance of quic in cellular networks. In *Passive and Active Measurement Conference*, 2017.

[47] S. Khanji, F. Iqbal, and P. Hung. ZigBee security vulnerabilities: Exploration and evaluating. In *10th International Conference on Information and Communication Systems (ICICS)*. IEEE, 2019.

[48] N. Vidgren, K. Haataja, J. L. Patino-Andres, J. J. Ramirez-Sanchis, and P. Toivanen. Security threats in ZigBee-enabled systems: vulnerability evaluation, practical experiments, countermeasures, and lessons learned. In *46th Hawaii International Conference on System Sciences*. IEEE, 2013.

[49] ZigBee Alliance. ZigBee Specification. https:

//zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf, 2015.

[50] G. Dini and M. Tiloca. Considerations on security in ZigBee networks. In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*. IEEE, 2010.

[51] H. Li, Z. Jia, and X. Xue. Application and analysis of ZigBee security services specification. In *Second International Conference on Networks Security, Wireless Communications and Trusted Computing*. IEEE, 2010.

[52] H. Kim, J. Chung, and C. H. Kim. Secured communication protocol for internetworking ZigBee cluster networks. *Computer Communications*, 32(13-14):1531–1540, 2009.

[53] T. Willingham, C. Henderson, B. Kiel, Md S. Haque, and T. Atkison. Testing Vulnerabilities in Bluetooth Low Energy. In *ACMSE*, 2018.

[54] C. Julien, C. Liu, A. L. Murphy, and G. P. Picco. BLEnd: Practical Continuous Neighbor Discovery for Bluetooth Low Energy. In *ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2017.

[55] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610, September 2003.

[56] M. Ryan. Bluetooth: With Low Energy Comes Low Security. In *Workshop on Offensive Technologies*. USENIX, 2013.

[57] K. Fawaz, K. Kim, and K. G. Shin. Protecting Privacy of BLE Device Users. In *USENIX Security Symposium*, 2016.

[58] M. Bon. A Basic Introduction to BLE Security. *digikey.com*, Oct 2016.

[59] T. Iwata, J. Song, J. Lee, and R. Poovendran. The AES-CMAC Algorithm. RFC 4493, June 2006.

[60] Bihuan Zhao, Shouling Ji, Wen-Han Lee, Chia-Mu Lin, Hongwei Weng, Jinjun Wu, Ping Zhou, Lei Fang, and Raheem Beyah. A Large-Scale Empirical Study on the Vulnerability of Deployed IoT Devices. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1826–1840, 2022.

[61] Muhammad Husnain, Khizar Hayat, Enrico Cambiaso, Ubaid Ullah Fayyaz, Maurizio Mongelli, Habiba Akram, Syed Ghazanfar Abbas, and Ghalib A. Shah. Preventing MQTT Vulnerabilities Using IoT-Enabled Intrusion Detection System. *Sensors*, 22(2):567, 2022.

[62] S. Andy, B. Rahardjo, and B. Hanindhito. Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT Systems. In *Proceedings of the 2017 Int'l Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 1–6, Yogyakarta, Indonesia, 2017.

[63] José Roldán-Gómez, Javier Carrillo-Mondéjar, Juan Manuel Castelo Gómez, and Sergio Ruiz-Villafranca. Security Analysis of the MQTT-SN Protocol for the Internet of Things. *Applied Sciences*, 12(21):10991, 2022.

[64] Hannes Sochor, Flavio Ferrarotti, and Rudolf Ramler. Exploiting MQTT-SN for Distributed Reflection Denial-of-Service Attacks. In *Proceedings of the Int'l Conference on Database and Expert Systems Applications (DEXA)*, volume 1285 of *Communications in Computer and Information Science*, pages 74–81, Bratislava, Slovakia, 2020. Springer, Cham.

[65] Gilbert Held. *Inter-and intra-vehicle communications*. CRC Press, 2007.

[66] David C Klonoff. Continuous glucose monitoring: roadmap for 21st century diabetes therapy. *Diabetes care*, 28(5):1231–1239, 2005.

[67] P. C. van Oorschot. *Computer Security and the Internet Security: Tools and Jewels*. Springer Nature, 2019.

[68] D. Mellado, E. Fernández-Medina, and M. Piattini. A common criteria based security requirements engineering process for the development of secure information systems. *Computer standards & interfaces*, 29(2):244–253, 2007.

[69] M. Gasser. *Building a secure computer system*. Van Nostrand Reinhold Company, 1988.

[70] Eclipse. Eclipse Mosquitto: An open source MQTT broker. https://mosquitto.org/, 2021.

[71] Eclipse. Paho. https://www.eclipse.org/paho/, 2021.

[72] H. Fereidooni, J. Classen, T. Spink, P. Patras, M. Miettinen, A. Sadeghi, M. Hollick, and M. Conti. Breaking fitness records without moving: Reverse engineering and spoofing fitbit. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 48–69. Springer, 2017.

[73] H. Tao, S. Zhang, and C. Chen. A design of wsn based locking system. *Acta Informatica Malaysia*, 2(1):04–06, 2018.

[74] S. Raghavan and G. S Tewolde. Cloud based low-cost home monitoring and automation system. In *ASEE north central section conference*, 2015.

[75] M. Menzel, I. Thomas, and C. Meinel. Security Requirements Specification in Service-Oriented Business Process Management. In *International Conference on Availability, Reliability and Security*, doi: 10.1109/ARES.2009.90, 2009.

[76] Q. Zhao, L. Shu, K. Li, M. A. Ferrag, X. Liu, and Y. Li. Security and Privacy in Solar Insecticidal Lamps Internet of Things: Requirements and Challenges. *IEEE/CAA Journal of Automatica Sinica*, doi: 10.1109/JAS.2023.123870, 2024.

[77] S. Myagmar, A. J Lee, and W. Yurcik. Threat modeling as a basis for security requirements. Technical report, 2005.

[78] D. Firesmith. Engineering security requirements. *J. Object Technol.*, doi: 10.5381/jot.2003.2.1.c6, 2003.

[79] D. Firesmith. Specifying reusable security requirements. *J. Object Technol.*, doi:10.5381/jot.2004.3.1.c6, 2004.

[80] N. R Mead. How to compare the Security Quality Requirements Engineering (SQUARE) method with other methods. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2007.

[81] X. Huang, P. Craig, H. Lin, and Z. Yan. SecIoT: a security framework for the Internet of Things. *Security and communication networks*, doi: 10.1002/sec.1259, 2016.

[82] A. F. A. Rahman, M. Daud, and M. Z. Mohamad. Securing sensor to cloud ecosystem using internet of things (iot) security framework. In *Proceedings of the International Conference on Internet of things and Cloud Computing*, doi: 10.1145/2896387.2906198, 2016.

[83] I. Alqassem. Privacy and security requirements framework for the internet of things (IoT). In *Proceedings of the 36th International Conference on Software Engineering*, doi: 10.1145/2591062.2591201, 2014.

[84] M. Abomhara and G. M Køien. Security and privacy in the Internet of Things: Current status and open issues. In *International conference on privacy and security in mobile systems (PRISMS)*, doi: 10.1109/PRISMS.2014.6970594, 2014.

[85] I. Alqassem and D. Svetinovic. A taxonomy of security and privacy requirements for the Internet of Things (IoT). In *IEEE International Conference on Industrial Engineering and Engineering Management*, doi: 10.1109/IEEM.2014.7058837, 2014.

[86] H. Kim, H. Chang, J. Suh, and T. Shon. A study on device security in IoT convergence. In *International Conference on Industrial Engineering, Management Science and Application (ICIMSA)*, doi: 10.1109/ICIMSA.2016.7503989, 2016.

[87] S. Oh and Y. Kim. Security requirements analysis for the IoT. In *International Conference on Platform Technology and Service (PlatCon)*, doi: 10.1109/PlatCon.2017.7883727, 2017.

[88] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering. *arXiv preprint cs/9810019*, 1998.

[89] D. Lagutin, K. Visala, A. Zahemszky, T. Burbridge, and G. F. Marias. Roles and security in a publish/subscribe network architecture. In *The IEEE symposium on Computers and Communications*, 2010.

[90] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. Internet of things: A

survey on the security of iot frameworks. *Journal of Information Security and Applications*, 38:8–27, doi: https://doi.org/10.1016/j.jisa.2017.11.002, 2018.

[91] H. Tschofenig and E. Baccelli. Cyberphysical security for the masses: A survey of the internet protocol suite for internet of things security. *IEEE Security & Privacy*, 17(5):47–57, 2019.

[92] S. Elhadi, A. Marzak, and N. Sael. Operating models of application protocols. In *4th International Conference on Smart City Applications*, 2019.

[93] C. Gündoğran, P. Kietzmann, M. Lenders, H. Petersen, T. C Schmidt, and M . Wählisch. NDN, CoAP, and MQTT: a comparative measurement study in the IoT. In *5th ACM Conference on Information-Centric Networking*, 2018.

[94] W. E. Santo, R. JP de B. Salgueiro, R. Santos, D. Souza, A. Ribeiro, and E. Moreno. Internet of things: A survey on communication protocol security. In *Euro American Conference on Telematics and Information Systems*, pages 1–5, 2018.

[95] Y. Chen and T. Kunz. Performance evaluation of IoT protocols under a constrained wireless access network. In *International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)*, 2016.

[96] R. Herrero. MQTT-SN, CoAP, and RTP in wireless IoT real-time communications. *Multimedia Systems*, 26(6):643–654, 2020.

[97] M. H. Amaran, N. A. Mohd Noh, M. S. Rohmad, and H. Hashim. A Comparison of Lightweight Communication Protocols in Robotic Applications. *Procedia Computer Science*, 76:400–405, 2015. IEEE International Symposium on Robotics and Intelligent Sensors.

[98] Y. Guamán, G. Ninahualpa, G. Salazar, and T. Guarda. Comparative Performance Analysis between MQTT and CoAP Protocols for IoT with Raspberry PI 3 in IEEE 802.11 Environments. In *15th Iberian Conference on Information Systems and Technologies (CISTI)*, 2020.

[99] D. Thangavel, X. Ma, A. Valera, H. Tan, and C. K. Tan. Performance evaluation of MQTT and CoAP via a common middleware. In *IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014.

[100] J. Granjal, E. Monteiro, and J. Sá Silva. Security for the Internet of Things: a survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials*, 17(3):1294–1312, 2015.

[101] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi. Internet of Things: Survey and open issues of MQTT protocol. In *International Conference on Engineering MIS (ICEMIS)*, 2017.

[102] D. Dinculeană and X. Cheng. Vulnerabilities and Limitations of MQTT Protocol Used between IoT Devices. *Applied Sciences*, 9(5):848, 2019.

[103] J. Y. Kim, R. Holz, W. Hu, and S. Jha. Automated analysis of secure Internet of Things protocols. In *33rd Annual Computer Security Applications Conference*, 2017.

[104] S. Kumar, Y. Hu, M. P Andersen, R. A. Popa, and D. E. Culler. {JEDI}: Many-to-Many End-to-End Encryption and Key Delegation for IoT. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019.

[105] M. Buschsieweke and M. Güneş. Application Layer Security for the IoT. *INFORMATIK*, 2021.

[106] S. Shin, K. Kobara, and and. A security framework for MQTT. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 432–436, Oct 2016.

[107] L . Malina, G. Srivastava, P. Dzurenda, J. Hajny, and R. Fujdiak. A secure publish/subscribe protocol for Internet of Things. In *14th International Conference on Availability, Reliability and Security*, 2019.

[108] P. Anantharaman, K. Palani, and S. Smith. Scalable Identity and Key Management for Publish-Subscribe Protocols in the Internet-of-Things. In *9th International Conference on the Internet of Things*, 2019.

[109] A. Birgisson, J. G. Politz, U. Erlingsson, A. Taly, M. Vrable, and M. Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. In *NDSS*, 2014.

[110] S. Nuratch. Applying the MQTT Protocol on Embedded System for Smart Sensors/Actuators and IoT Applications. In *15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2018.

[111] S. Kim, H. Choi, and W. Rhee. IoT home gateway for auto-configuration and management of MQTT devices. In *IEEE Conference on Wireless Sensors (ICWiSe)*, 2015.

[112] Y. Upadhyay, A. Borole, and D. Dileepan. MQTT based secured home automation system. In *Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016.

[113] R. Bryce, T. Shaw, and G. Srivastava. MQTT-G: A Publish/Subscribe Protocol with Geolocation. In *41st International Conference on Telecommunications and Signal Processing (TSP)*, 2018.

[114] D. Namiot and M. Sneps-Sneppe. Geofence and network proximity. In *Internet of Things, Smart Spaces, and Next Generation Networking*. Springer, 2013.

[115] P. Fremantle, B. Aziz, J. Kopeckỳ, and P. Scott. Federated identity and access management for the Internet of Things. In *2014 International Workshop on Secure Internet of Things*. IEEE, 2014.

[116] I. D. Gheorghe-Pop, A. Kaiser, A. Rennoch, and S. Hackel. A Performance Benchmarking Methodology for MQTT Broker Implementations. In *IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2020.

[117] J. Roldán-Gómez, J. Carrillo-Mondéjar, J. M. Castelo Gómez, and S. Ruiz-Villafranca. Security Analysis of the MQTT-SN Protocol for the Internet of Things. *Applied Sciences*, 12(21), doi: 10.3390/app122110991, 2022.

[118] José Roldán-Gómez, Javier Carrillo-Mondéjar, Juan Manuel Castelo Gómez, and José Luis Martínez Martínez. Security assessment of the mqtt-sn protocol for the internet of things. *Journal of Physics: Conference Series*, 2224(1):012079, apr 10.1088/1742-6596/2224/1/012079, 2022.

[119] T Kao, H Wang, and J Li. Safe mqtt-sn: a lightweight secure encrypted communication in iot. *Journal of Physics: Conference Series*, 2020:012044, 09 doi: 10.1088/1742-6596/2020/1/012044, 2021.

[120] Fu Chen, Yujia Huo, Jianming Zhu, and Dan Fan. A review on the study on mqtt security challenge. In *IEEE International Conference on Smart Cloud (SmartCloud)*, pages 128–133, 10.1109/SmartCloud49737.2020.00032, 2020.

[121] Asmaa Munshi. Improved mqtt secure transmission flags in smart homes. volume 22, doi: 10.3390/s22062174, 2022.

[122] S. Farahani. *ZigBee Wireless Networks and Transceivers*. Newnes Publisher, ISBN: 978-0-7506-8393-7, 2008.

[123] Muhammad Husnain, Khizar Hayat, Enrico Cambiaso, Ubaid U. Fayyaz, Maurizio Mongelli, Habiba Akram, Syed Ghazanfar Abbas, and Ghalib A. Shah. Preventing mqtt vulnerabilities using iot-enabled intrusion detection system. *Sensors*, 22(2), doi: 10.3390/s22020567, 2022.

[124] H. Sochor, F. Ferrarotti, and R. Ramler. Exploiting mqtt-sn for distributed reflection denial-of-service attacks. In *International Conference on Database and Expert Systems Applications*, doi: 10.1007/978-3-030-59028-47, 2020.

[125] Syaiful Andy, Budi Rahardjo, and Bagus Hanindhito. Attack scenarios and security analysis of mqtt communication protocol in iot system. In *4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, doi: 10.1109/EECSI.2017.8239179, 2017.

[126] N. R Mead and T. Stehney. Security quality requirements engineering (SQUARE) methodology. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

[127] S. Jabeen and S. R. Potturu. Survey on security and privacy of connected vehicles and cloud platforms for communication. In *AIP Conference Proceedings*. AIP Publishing, 2024.

[128] Yue Cao and Ning Wang. Toward Efficient Electric-Vehicle Charging Using VANET-Based Information Dissemination. *IEEE Transactions on Vehicular Technology*, 66(4):2886–2901, 10.1109/TVT.2016.2594241, 2017.

[129] Yue Cao, Ning Wang, and George Kamel. A publish/subscribe communication framework for managing electric vehicle charging. In *International Conference on Connected Vehicles and Expo (ICCVE)*, pages 318–324, 10.1109/ICCVE.2014.7297564, 2014.

[130] Seyedfoad Taghizadeh, Pouya Jamborsalamati, M. J. Hossain, and Junwei Lu. Design and implementation of an advanced vehicle-to-vehicle (v2v) power transfer operation using communications. In *IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, doi: 10.1109/EEEIC.2018.8494480, 2018.

[131] Panos Papadimitratos. Secure vehicular communication systems. In *Encyclopedia of Cryptography, Security and Privacy*, pages 1–6. Springe, 2024.

[132] Vrizlynn L.L. Thing and Jiaxi Wu. Autonomous vehicle security: A taxonomy of attacks and defences. In *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2016.

[133] Katherine E Britton and Jennifer D Britton-Colonnese. Privacy and security issues surrounding the protection of data generated by continuous glucose monitors. *Journal of diabetes science and technology*, 11(2):216–219, 2017.

[134] Nathanael Paul, Tadayoshi Kohno, and David C Klonoff. A review of the security of insulin pump infusion systems. *Journal of diabetes science and technology*, 5(6):1557–1562, 2011.

[135] Haotian Weng, Chirath Hettiarachchi, Christopher Nolan, Hanna Suominen, and Artem Lenskiy. Ensuring security of artificial pancreas device system using homomorphic encryption. *Biomedical Signal Processing and Control*, 79:104044, 2023.

[136] Signe Marie Cleveland and Moutaz Haddara. Internet of things for diabetics: Identifying adoption issues. *Internet of Things*, 22:100798, 2023.

[137] Lavanya Mandava, Husam Ghazaleh, and Guilin Zhao. Securing modern insulin pumps with icgm system: protecting patients from cyber threats in diabetes management. IET, 2024.

[138] Hari Venugopalan, Shreyas Madhav Ambattur Vijayanand, Caleb Stanford, Stephanie Crossen, and Samuel T King. Glucos: Security, correctness, and simplicity for automated insulin delivery. *arXiv preprint arXiv:2406.18262*, 2024.

[139] L. Zhang, H. Wang, and Q. Li. Security analysis of bluetooth smart locks. *IEEE Transactions on Consumer Electronics*, 65(4):412–421, 2019.

[140] E. Fernandes, J. Jung, and A. Prakash. Security implications of iot device design. In *Proceedings of the USENIX Security Symposium*, pages 1101–1118. USENIX Association, 2017.

[141] B. Ur, J. Jung, and L. Cranor. Smart home security: User behavior and system weaknesses. In *Symposium on Usable Privacy and Security (SOUPS)*, pages 55–72. USENIX Association, 2018.

[142] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. Sok: Security evaluation of home-based iot deployments. In *IEEE Symposium on Security and Privacy*, pages 1–17. IEEE, 2019.

[143] Y. Zhang and X. Chen. Capbac: Capability-based access control for iot. *IEEE Internet of Things Journal*, 5(3):1505–1514, 2018.

[144] M. Singh, R. Sharma, and D. Patel. Security challenges in mqtt-sn for iot. *Sensors*, 20(12):1–18, 2020.

[145] J. Jang, S. Kim, and H. Lee. Access control weaknesses in smart home systems. In *Annual Computer Security Applications Conference (ACSAC)*, pages 321–332. ACM, 2019.

[146] A. Banks, R. Gupta, E. Briggs, and K. Borgendale. MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, Mar 2019.

[147] C. Bormann, M. Ersue, and A. Keränen. Terminology for Constrained-Node Networks. RFC 7228, May 2014.

[148] Prashantkumar Oza and Dipesh Kamdar. A Review on Security Approaches of MQTT Protocol with Respect to Internet of Things. *Int'l Journal of Research and Analytical Reviews*, 7:1–11, 2020.

[149] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. Sok: Security evaluation of home-based iot deployments. In *IEEE symposium on security and privacy (sp)*, 2019.

[150] B. Fabian, S. Gürses, M. Heisel, T. Santen, and H. Schmidt. A comparison of security requirements engineering methods. *Requirements engineering*, 15(1):7–40, 2010.

[151] H. Gupta and A. Nayak. Addressing IoT Security Challenges: A Framework for Determining Security Requirements of Smart Locks Leveraging MQTT-SN. In *International Symposium on Networks, Computers and Communications (ISNCC)*, doi: 10.1109/ISNCC58260.2023.10323864, 2023.

[152] John R Vacca. *Computer and information security handbook*. Newnes, 2012.

[153] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *11th ACM on Asia conference on computer and communications security*, 2016.

[154] H. Gupta and P. C. van Oorschot. Onboarding and software update architecture for IoT devices. In *17th International Conference on Privacy, Security and Trust (PST)*. IEEE, 2019.

[155] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang. Burglars' IoT paradise: Understanding and mitigating security risks of general messaging protocols on IoT clouds. In *IEEE Symposium on Security and Privacy (SP)*, 2020.

[156] Dindayal Mahto and Dilip Kumar Yadav. Rsa and ecc: a comparative analysis. *International journal of applied engineering research*, 12(19):9053–9061, 2017.

[157] Elisabeth Oswald. *Topics in Cryptology–CT-RSA 2024: Cryptographers' Track at the RSA Conference 2024, San Francisco, CA, USA, May 6–9, 2024, Proceedings*. Springer Nature, 2024.

[158] L. Zhang, X. Chen, and H. Wang. Security analysis of bluetooth and wi-fi smart locks. *IEEE Internet of Things Journal*, 5(6):4821–4833, 2018.

[159] S. Das, Y. Shah, and C. Wilson. A study of vulnerabilities in bluetooth smart

locks. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 45–56. ACM, 2018.

[160] Y. Tian, X. Luo, and P. Ning. Security risks in iot smart locks: An empirical analysis. *IEEE Transactions on Dependable and Secure Computing*, 17(5):1024–1037, 2020.

[161] M. Ryan. Bluetooth low energy: The good, the bad, and the ugly. In *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, pages 1–10. USENIX Association, 2013.

[162] N. Apthorpe, D. Reisman, and N. Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. In *Proceedings of the Workshop on Data and Algorithmic Transparency*, pages 1–6. ACM, 2017.

[163] R. Hummen, H. Shafagh, S. Raza, T. Voigt, and K. Wehrle. Delegation-based authentication and authorization for the ip-based internet of things. In *Proceedings of the IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2013.

[164] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. Lithe: Lightweight secure coap for the internet of things. *IEEE Sensors Journal*, 13(10):3711–3720, 2012.

[165] A. Stanford-Clark and H. L. Truong. MQTT for Sensor Networks (MQTT-SN) Protocol Specification Version 1.2, Nov 2013.

[166] C. Park and H. Nam. Security Architecture and Protocols for Secure MQTT-SN. *IEEE Access*, 2020.

[167] K. Govindan and A. P. Azad. End-to-end service assurance in IoT MQTT-SN. In *12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015.

[168] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *11th ACM on Asia conference on computer and communications security*, 2016.

[169] Hemant Gupta and Amiya Nayak. Addressing iot security challenges: A framework for determining security requirements of smart locks leveraging mqtt-sn. In *2023 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–7. IEEE, 2023.

[170] Kristin Castorino, Sarit Polsky, Grenye O'Malley, Camilla Levister, Kristen Nelson, Christian Farfan, Scott Brackett, Sarah Puhr, and Carol J Levy. Performance of the dexcom g6 continuous glucose monitoring system in pregnant women with diabetes. *Diabetes technology & therapeutics*, 22(12):943–947, 2020.

[171] Anna Nguyen and John R White. Freestyle libre 3. *Clinical Diabetes*, 41(1):127–128, 2023.

[172] Jeffrey I Joseph. Review of the long-term implantable senseonics continuous glucose monitoring system and other continuous glucose monitoring systems. *Journal of Diabetes Science and Technology*, 15(1):167–173, 2021.

[173] Damon C McMillan. Guardian connect continuous glucose-monitoring system. *US Pharm*, 43(9):27–29, 2018.

[174] Giacomo Cappon, Martina Vettoretti, Giovanni Sparacino, and Andrea Facchinetti. Continuous glucose monitoring sensors for diabetes management: a

review of technologies and applications. *Diabetes & metabolism journal*, 43(4):383–397, 2019.

[175] David Rodbard. Continuous glucose monitoring: a review of successes, challenges, and opportunities. *Diabetes technology & therapeutics*, 18(S2):S2–3, 2016.

[176] American Diabetes Association. Standards of medical care in diabetes—2024. *Diabetes Care*, 47(Supplement 1):S1–S200, 2024. Includes clinical guidelines and background on Continuous Glucose Monitoring (CGM) systems.

[177] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of the dolev–yao model). *Journal of Cryptology*, 15(2):103–127, 2002.

[178] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols: Tool paper. In *International conference on computer aided verification*, pages 414–418. Springer, 2008.

[179] Casimier Joseph Franciscus Cremers. Scyther: Semantics and verification of security protocols. 2006.

[180] Huihui Yang, Andreas Prinz, and Vladimir Oleshchuk. Formal analysis and model checking of a group authentication protocol by scyther. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 553–557, 2016.

[181] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[182] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography

(the computational soundness of the dolev–yao model). *Journal of Cryptology*, 15(2):103–127, 2002.

[183] R. Kraemer and M. Katz. Short-range wireless communications: Emerging technologies and applications. 2009.

[184] M. A. Al-Absi, A. A. Al-Absi, and H. J. Lee. Comparison between DSRC and other Short Range Wireless Communication Technologies. In *22nd International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2020.

[185] Mochammad Ariyanto, Gunawan D Haryadi, Munadi Munadi, Rifky Ismail, and Zulfa Hendra. Development of low-cost autonomous emergency braking system (aebs) for an electric car. In *5th International Conference on Electric Vehicular Technology (ICEVT)*, pages 167–171. IEEE, 2018.

[186] Christine Laurendeau and Michel Barbeau. Threats to security in dsrc/wave. In *International Conference on Ad-Hoc Networks and Wireless*, pages 266–279. Springer, 2006.

[187] Vishal Sharma, Ilsun You, and Nadra Guizani. Security of 5g-v2x: Technologies, standardization, and research directions. *IEEE Network*, 34(5):306–314, 2020.

[188] Zhisheng Yin, Min Jia, Nan Cheng, Wei Wang, Feng Lyu, Qing Guo, and Xuemin Shen. Uav-assisted physical layer security in multi-beam satellite-enabled vehicle communications. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):2739–2751, 2021.

[189] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols: Tool paper. In *International conference on computer aided verification*, pages 414–418. Springer, 2008.

[190] Casimier Joseph Franciscus Cremers. Scyther: Semantics and verification of security protocols. 2006.

[191] Z. Berkay Celik, Earlence Fernandes, Gang Tan, and Patrick McDaniel. Security analysis of smart lock systems. In *Proceedings of the IEEE Symposium on Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018.

[192] X. Liu, J. Zhao, and K. Li. Security analysis of smart home door locks. *IEEE Access*, 7:92745–92756, 2019.

[193] A. Maiti, M. Jadliwala, and W. He. Security and privacy of smart locks: A systematic evaluation. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 1–12. ACM, 2020.

[194] Anupam Das, Nikita Borisov, and Matthew Caesar. Breaking smart locks: Security evaluation of bluetooth and wi-fi locks. In *Proceedings of the IEEE Security and Privacy Workshops (SPW)*, pages 1–6. IEEE, 2016.

[195] Omar Alrawi, Charles Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. *IEEE Symposium on Security and Privacy*, pages 1–17, 2019.

[196] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 119–132. Springer, 2004.

[197] Utku Gulen and Selcuk Baktir. Side-channel resistant 2048-bit rsa implementation

for wireless sensor networks and internet of things. *IEEE Access*, 11:39531–39543, 2023.

[198] David C Klonoff. Cybersecurity for connected diabetes devices. *Journal of diabetes science and technology*, 9(5):1143–1147, 2015.

[199] Niki O'Brien, Saira Ghafur, and Mike Durkin. Cybersecurity in health is an urgent patient safety concern: we can learn from existing patient safety improvement strategies to address it. *Journal of Patient Safety and Risk Management*, 26(1):5–10, 2021.