DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

APPLICATION OF LIGHTWEIGHT CRYPTOGRAPHY TECHNIQUES IN INTERNET OF THINGS SCENARIOS

by

Özlem ŞEKER

August, 2024

İZMİR

APPLICATION OF LIGHTWEIGHT CRYPTOGRAPHY TECHNIQUES IN INTERNET OF THINGS SCENARIOS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Department of Computer Engineering

by

Özlem ŞEKER

August, 2024
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"APPLICATION OF LIGHTWEIGHT CRYTOGRAPHY TECHNIQUES IN INTERNET OF THINGS SCENARIOS"** completed by **ÖZLEM ŞEKER** under supervision of **ASSOC.PROF.DR. GÖKHAN DALKILIÇ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Assoc.Prof.Dr.Gökhan DALKILIÇ

Supervisor

Prof. Dr. Yalçın ÇEBİ

Thesis Committee Member

Prof. Dr. M. Süleyman ÜNLÜTÜRK

Thesis Committee Member

Prof. Dr. Muhammet Gökhan ERDEM

Examining Committee Member

Assoc. Prof. Dr. Özgü CAN

Examining Committee Member

Prof. Dr. Okan FISTIKOĞLU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENT

From the beginning of my academic journey, I would like to express my gratitude to my esteemed advisor, Assoc. Prof. Dr. Gökhan DALKILIÇ, for his invaluable contributions to my studies. His extensive knowledge and experience have enriched my work, and his innovative ideas have provided me with essential guidance.

I am also deeply thankful to my thesis committee members, Prof. Dr. Yalçın ÇEBİ and Prof. Dr. Mehmet Süleyman ÜNLÜTÜRK, for their insightful ideas, time, and recommendations during the progress of my thesis.

I would like to extend my heartfelt thanks to my esteemed colleague, Umut Can ÇABUK, whose work discipline and positive energy have greatly contributed to our collaborative efforts.

Finally, my deepest appreciation goes to my biggest supporters: my beloved husband Yunus ŞEKER, my dear son Eymen Efe ŞEKER, my precious family, and my dear friends. Thank you for your belief in me, your patience, and for keeping my motivation high throughout this journey.

Özlem ŞEKER

APPLICATION OF LIGHTWEIGHT CRYPTOGRAPHY TECHNIQUES IN INTERNET OF THINGS SCENARIOS

ABSTRACT

The Internet of Things (IoT) refers to the network of devices with limited power and processing capacity, such as sensors, radio frequency identification (RFID) tags, and smart cards, that can communicate with each other without human interaction or user interface. Therefore, more efficient methods are sought to ensure security. This thesis introduces a robust framework called the securing the IoT through a multi-protocol multi-gateway scheme (SIMMS). SIMMS aims to enhance security with various security tiers, including received signal strength indicator (RSSI) based physical tracking, multi-factor authentication, role-based access authorization, data encryption, and data integrity control for closed-environment IoT applications. The scheme proposes a two-tier gateway architecture: a series of relay gateways managing communication at the lower tier and a main gateway regulating data flow at the upper tier.

Message queuing telemetry transport (MQTT) is preferred as the communication protocol between the relay gateway and the main gateway. However, the lack of valid security features beyond username/password authentication in MQTT and the inefficiency of transport layer security/secure hyper text transfer protocol (TLS/HTTPS) for constrained devices pose a challenge. To address this issue, a solution named mutual authentication and role-based authorization scheme (MARAS) is developed for lightweight IoT applications. Using MQTT, MARAS and SIMMS provide mutual authentication, authorization, message confidentiality, and message integrity security levels between edge devices, relay gateways, and main gateway through dynamic access tokens, quick response (QR) code-based secure transmissions, hash-based message authentication code (HMAC-based) one-time passwords (HOTP), advanced encryption standard (AES), elliptic curve digital signature algorithm (ECDSA), and a reliable server running open authorization (OAuth2.0). A prototype combining ESP32 modules, a Raspberry Pi, and a PC-based identity server demonstrates the practical implementation and effectiveness of the framework.

The proposed system architecture integrates modular security features, considering device limitations and IoT application security needs. The security measures are organized into tiers to accommodate diverse protection requirements. Tier 1 involves basic physical tracking with plain text data transmission. Tier 2 strengthens authentication via QR codes and enforces access control using OAuth 2.0. Tier 3 adds persistent data encryption through AES. Finally, Tier 4 incorporates digital signatures along with AES encryption to ensure data confidentiality and authenticate the data's origin, preventing repudiation.The security tiers affects both the payload size and transmission time, leading to increased communication and computation costs. An analysis of these costs has been conducted to understand the trade-offs between security enhancements and performance.

SIMMS modifies only the "publish" and "connect" messages among the 14 types of MQTT messages. For instance, it adds 32 bytes of overhead to "publish" messages and 36 bytes to "connect" messages. Conceptual evidence shows that with SIMMS, the overall data traffic remains less than twice the traffic without it because "publish" messages are the most common. The round-trip time for connection messages is delayed by less than one percentile of a millisecond. For publish messages, the delay, which depends on the size and frequency of published information, can be safely limited to 163% of the network defaults. Therefore, the scheme's overhead on the network is tolerable.

Performance evaluations validate SIMMS's message reliability and network performance while confirming its scalability and compatibility. Overall, SIMMS offers a flexible and scalable security solution, significantly enhancing the security capabilities of IoT systems.

Keywords: Authentication, authorization, AES, BLE, confidentiality, ECDSA, ESP32, HOTP, integrity, IoT, MQTT, OAuth 2.0, raspberry pi, Wi-Fi, ZigBee.

NESNELERİN İNTERNETİ SENARYOLARINDA HAFİF ŞİFRELEME TEKNİKLERİNİN UYGULANMASI

ÖZ

Nesnelerin İnterneti (IoT), insan etkileşimi veya kullanıcı arayüzü olmadan birbirleriyle iletişim kurabilen sensörler, radyo frekansı ile tanımlama (RFID) etiketleri ve akıllı kartlar gibi sınırlı güç ve işlem kapasitesine sahip cihazların ağını ifade eder. Bu nedenle, güvenliği sağlamak için daha verimli yöntemler aranmaktadır. Bu tez, nesnelerin İnterneti güvenliğini sağlamak için çok protokollü çok ağ geçitli şema (SIMMS) adlı güçlü bir çerçeve tanıtır. SIMMS, kapalı ortam IoT uygulamaları için alınan sinyal gücü göstergesi (RSSI) tabanlı fiziksel takip, çok faktörlü kimlik doğrulama, role dayalı erişim yetkilendirme, veri şifreleme ve veri bütünlüğü kontrolü gibi çeşitli güvenlik katmanlarıyla güvenliği artırmayı hedefler. Şema, alt katmanda iletişimi yöneten aktarma ağ geçitleri ve üst katmanda veri akışını düzenleyen ana ağ geçidi olmak üzere iki katmanlı bir ağ geçidi mimarisi önerir.

Aktarma ağ geçidi ile ana ağ geçidi arasında iletişim protokolü olarak mesaj kuyruk telemetri taşıma (MQTT) tercih edilir. Ancak, MQTT'de kullanıcı adı/şifre kimlik doğrulaması dışında geçerli güvenlik özelliklerinin bulunmaması ve sınırlı cihazlar için taşıma katmanı güvenliği/güvenli hiper metin transfer protokolü (TLS/HTTPS) verimsizliği bir zorluk oluşturur. Bu sorunu çözmek için, hafif IoT uygulamaları için MARAS (karşılıklı kimlik doğrulama ve role dayalı yetkilendirme şeması) adlı bir çözüm geliştirilmiştir. MQTT kullanarak, MARAS ve SIMMS, dinamik erişim belirteci, hızlı yanıt (QR) kod tabanlı güvenli iletimler, karma tabanlı mesaj doğrulama kodu (HMAC) tabanlı tek kullanımlık şifreler (HOTP), gelişmiş şifreleme standardı (AES), eliptik eğri dijital imza algoritması (ECDSA) ve OAuth2.0 çalıştıran güvenilir bir sunucu aracılığıyla uç cihazlar, aktarma ağ geçitleri ve ana ağ geçidi arasında karşılıklı kimlik doğrulama, yetkilendirme, mesaj gizliliği ve mesaj bütünlüğü güvenlik seviyeleri sağlar. ESP32 modülleri, bir Raspberry Pi ve bir PC tabanlı kimlik sunucusunu birleştiren bir prototip, çerçevenin pratik uygulamasını ve etkinliğini gösterir.

Önerilen sistem mimarisi, cihazların sınırlı kaynakları ve IoT uygulamasının güvenlik ihtiyaçlarını dikkate alarak modüler olarak tasarlanmış güvenlik özelliklerini entegre eder. Güvenlik önlemleri, çeşitli koruma gereksinimlerini karşılamak için seviyelere ayrılmıştır. Birinci seviye, düz metin veri iletimiyle temel fiziksel izlemeyi içerir. İkinci seviye, QR kodlarıyla güçlendirilmiş kimlik doğrulaması ve OAuth 2.0 kullanılarak erişim kontrolü sağlar. Üçüncü seviye, AES ile sürekli veri şifreleme ekler. Son olarak, Dördüncü seviye, AES şifrelemesine ek olarak dijital imzaları dahil eder, bu da veri gizliliğini sağlamanın yanı sıra verinin kaynağını doğrulayarak inkarı önler. Güvenlik seviyesi, hem veri boyutunu hem de iletim süresini etkileyerek artan iletişim ve hesaplama maliyetlerine yol açar. Bu maliyetlerin analizi, güvenlik iyileştirmeleri ve performans arasındaki dengeyi anlamak için yapılmıştır.

SIMMS, MQTT'nin 14 mesaj türü arasında yalnızca "publish" ve "connect" mesajlarını değiştirir. Örneğin, "publish" mesajlarına 32 bayt, "connect" mesajlarına ise 36 bayt ek yük getirir. SIMMS'in kavramsal kanıtı, genel veri trafiğinin SIMMS kullanımıyla, SIMMS kullanılmadığı duruma göre iki katından daha az arttığını göstermektedir. Bu durum, en yaygın mesaj türü olan "publish" mesajlarının etkisiyle açıklanmaktadır. Bağlantı mesajları için gidiş-dönüş süresi milisaniyenin yüzde birinden daha az gecikmiştir. "Publish" mesajları için gecikme, yayınlanan bilginin boyutu ve sıklığına bağlıdır, ancak gecikme ağ varsayılanlarının %163'ü ile sınırlandırılabilir. Bu nedenle, şemanın ağa getirdiği ek yük tolere edilebilir düzeydedir.

Performans değerlendirmeleri, SIMMS'in mesaj güvenilirliğini ve ağ performansını doğrularken, ölçeklenebilirlik ve uyumluluğunu da onaylar. Genel olarak, SIMMS, IoT sistemlerinin güvenlik yeteneklerini önemli ölçüde artıran esnek ve ölçeklenebilir bir güvenlik çözümü sunar.

Anahtar kelimeler: Kimlik doğrulama, yetkilendirme, AES, BLE, gizlilik, ECDSA, ESP32, HOTP, bütünlük, IoT, MQTT, OAuth 2.0, raspberry pi, Wi-Fi, ZigBee.

CONTENTS

Page

Ph.D. THESIS EXAMINATION RESULT FORM ii
ACKNOWLEDGMENT iii
ABSTRACT iv
ÖZ vi
LIST OF FIGURES xi
LIST OF TABLES xiii
ABBREVIATIONS xiv

CHAPTER ONE INTRODUCTION 1

1.1 Motivation 1
1.2 Problem Statement 3
1.3 Scope & Purpose 5
1.4 Contributions and Organization of Thesis 6

CHAPTER TWO BACKGROUND 9

2.1 Related Work 9
2.1.1 An Overview of Studies on MQTT Security in the Literature 9
2.1.2 Lightweight Encryption Algorithms 17
2.1.3 An Overview of Studies on Security with Multi-Protocol Gateways 19
2.2 Enabling Technology 21
2.2.1 Message Queue Telemetry Transport 22
2.2.1.1 MQTT Message Packet 23
2.2.1.2 Security Features in MQTT 24
2.2.2 OAuth 2.0 Authorization Framework 25
2.2.3 HMAC Based One Time Password 25
2.2.4 QR-based Authentication Protocol 26
2.2.5 Advanced Encryption Standard Cipher-Block Chaining 27
2.3 Wireless Communication Technology 27

2.3.1 Wireless Fidelity (Wi-Fi) 28
2.3.2 Bluetooth Low energy (BLE) 29
2.3.3 ZigBee Communication 32

CHAPTER THREE DESIGN OF SECURE MULTI-PROTOCOL GATEWAY 34

3.1 System Model, Assumptions, and Configurations 34
3.2 System Components 35
3.2.1 Multi-Protocol Multi-Gateway Architecture 35
3.2.2 Main Gateway 36
3.2.3 Relay Gateway 37
3.2.4 IoT Edge Devices 38
3.2.5 Server and Cloud Database 40
3.3 Configuration of Wireless Communication Technologies 40
3.3.1 To set up ESP32 Wi-Fi PainlessMesh Network 41
3.3.2 To set up BLE Communication 49
3.3.3 ZigBee Configuration with XCTU 56
3.4 Secure Tiers 58

CHAPTER FOUR PROTOTYPING AND VALIDATION OF MULTI-PROTOCOL GATEWAY IN THE SECURE TIERS 60

4.1 An Overview of Secure Tier 1: Physical Tracking 60
4.1.1 Implementation and Test Configuration for Wi-Fi 62
4.1.2 Selecting the Experimental Environment for BLE and Testing 67
4.1.3 Performance Discussion of Secure Tier 1 75
4.2 Outlook of Secure Tiers 2 and 3 76
4.2.1 Procedures of Security Mechanism 76
4.2.2 Registration Stage 77
4.2.3 Authentication Stage 79
4.2.3.1 End Nodes Authenticated by Relay Gateways 81
4.2.3.2 Relay Gateways Authenticated by Main Gateway 85
4.2.4 Authorization Stage 89

4.3 Outlook of Secure Tier 4: Digital Signing for Enhanced Data Integrity 92
4.3.1 Implementation of ECDSA on ESP32 92
4.3.2 Evaluating the performance of the ECDSA 94

CHAPTER FIVE PERFORMANCE EVALUATIONS 97

5.1 Test Configuration for Secure Tiers 2 and 3 97
5.2 Complexity Discussion for Secure Tiers 2 and 3 99
5.3 Comparative Analysis for Secure Tiers 2 and 3 100
5.3.1 Computational Costs 101
5.3.2 Communication Costs 102
5.4 Throughput, Message Reliability and Delay Time for Secure Tiers 2 and 3 104
5.5 Security Analysis of Multi-Protocol Multi Gateway 115
5.6 Performance Discussion of Results 118

CHAPTER SIX CONCLUSION AND FUTURE WORK 126

6.1 Conclusion 126
6.2 Future Work 128

REFERENCES 129

APPENDICES 140

Appendix 1: RSSI Measurements and Distances for Indoor and Outdoor Environments 140
Appendix 2: Message Reliability, Throughput, and Transmission Time for BLE and ZigBee 141
Appendix 3: Performance Metrics for Key Generation, Encryption, and Signing with Various Payloads Using ECDSA Algorithm on ESP32 145

LIST OF FIGURES

Page

Figure 3.1 Multi-protocol gateway: prototype components, connections, and interactions 36
Figure 3.2 Compare PainlessMesh Wi-Fi network structure with traditional and ESP-Mesh network architectures 42
Figure 3.3 The setup, message transmission, and security of the Wi-Fi mesh structure 43
Figure 3.4 Log files of setting up and new connection of PainlessMesh network..... 45
Figure 3.5 The process of sending a 16-byte message from Node1 to Node2, Node3, and NodeWRG in the mesh network..... 47
Figure 3.6 Set maximum transmission unit as 517 bytes..... 50
Figure 3.7 Data transmission from NodeB over BLE..... 51
Figure 3.8 Packet captured by Wireshark integration with Nordic nRF 52840 when NodeB advertises (a) and received handle value notification (b) 53
Figure 3.9 Creation of BLE master (NodeBRG), initiation of communication with NodeB, and transfer of NodeB data to the MG..... 54
Figure 3.10 Configuration of XBEE S2C pro modules with XCTU program..... 56
Figure 4.1 The component, relation and interaction of indoor position scheme..... 63
Figure 4.2 Sampling the location of ESP32 in station mode, and reference points... 64
Figure 4.3 Test environment, positioning of ESP32 access points, and RSSI measurement of ESP32 station mode..... 68
Figure 4.4 Indoor and outdoor RSSI distribution for all access points (ESP32). 69
Figure 4.5 Mean error comparison between indoor and outdoor measurements in meters 72
Figure 4.6 Correlation matrix of features (including label). 73
Figure 4.7 Procedure for end device registration and token acquisition..... 78
Figure 4.8 Algorithm 1: Device Registration Procedure 81
Figure 4.9 Authentication of end node by relay gateway with QR code and HOTP . 83
Figure 4.10 Algorithm 2: Authentication of end nodes by relay gateway and data transfer..... 85

Figure 4.11 Authenticate ESP32 relay gateway by gateway on Raspberry Pi..... 87
Figure 4.12 Algorithm 3: Authentication of relay gateway by main gateway..... 89
Figure 4.13 Algorithm 4: Authentication of relay gateway by main gateway..... 91
Figure 4.14 Algorithm 5: Key generation, signing and verification..... 93
Figure 5.1 BLE packet structure 105
Figure 5.2 Average throughput for 80, 100, and 150 ms connection intervals..... 107
Figure 5.3 Comparison of the one-way message transmission times per data size for BLE and ZigBee..... 108
Figure 5.4 BLE message realibility..... 109
Figure 5.5 ZigBee message realibility 110
Figure 5.6 The growth of node payload during the authentication process (Bytes) 111
Figure 5.7 The Delay time (ms) in ZigBee and BLE communication..... 114
Figure 5.8 Payload increase based on security tiers..... 124
Figure 5.9 Increase in BLE RTT based on security tiers, shown in milliseconds (ms) 125

LIST OF TABLES

Page

Table 4.1 Case1: RSSI values (dBm) for ESP32 device positioned outside the designated area, with fixed four reference points 65
Table 4.2 Case2, 3, and 4: RSSI values (dBm) for ESP32 device positioned within the designated area, with fixed two reference points 65
Table 4.3 Actual distance compared with RSSI-to-Distance 66
Table 4.4 The actual distance, measured distance in the indoor environment, and measured distance in the outdoor environment 70
Table 4.5 The comparisons of the applied techniques in terms of accuracy, precision, recall and fscore 74
Table 4.6 Approximate time estimates in ms for various cryptographic operations on ECDSA 95
Table 5.1 Approximate time estimates in ms for various cryptographic operations on SIMMS 101
Table 5.2 Computation costs comparison in ms 102
Table 5.3 Communication costs comparison in bits 103
Table 5.4 Unique message transmission rate 112
Table 5.5 The transmission rates of all messages sent in the mesh network 113
Table 5.6 Extra network traffic caused by SIMMS (in terms of message size) 120
Table 5.7 Comparative analysis of existing works and MARAS (Şeker, Dalkılıç, & Çabuk, 2023) 121
Table 5.8 Comparative analysis of existing works and SIMMS 122

ABBREVIATIONS

IoT	: Internet of Things
RFID	: Radio Frequency Identification
MQTT	: Message Queuing Telemetry Transport
HMAC	: Hash-based Message Authentication Code
HOTP	: Hash-based Message Authentication Code One-Time Password
QR Code	: Quick Response Code
OAuth	: Open Authorization
Wi-Fi	: Wireless Fidelity
BLE	: Bluetooth Low Energy
AES-CBC	: Advanced Encryption Standard Cipher-Block Chaining
RSA	: Rivest–Shamir–Adleman
SPN	: Substitution-Permutation Network
MCU	: Microcontroller Unit
ECC	: Elliptic Curve Cryptography
PKI	: Public Key Infrastructure
MAC	: Message Authentication Code
TLC	: Transport Layer Security
TCP	: Transmission Control Protocol
DoS	: Denial-of-Service
OASIS	: Organization for the Advancement of Structured Information Standart
JSON	: JavaScript Object Notation
EEPROM	: Electronically Erasable Programmable Read-Only Memory
PHY	: Physical Layer
MAC	: Medium Access control Layer
NWK	: Network Layer
APL	: Application Layer
SIG	: Bluetooth Special Interest Group
WPAN	: Wireless Personal Area Network
GAP	: Generic Access Profile
GATT	: Generic Attribute

UUID	: Universally Unique Identifier
NFC	: Near Field Communication
IEEE	: Institute of Electrical and Electronics Engineers
PWM	: Pulse-Width Modulation
VREF	: Voltage Reference
RNG	: Random Number Generator
TFT	: Thin Film Transistor
LCD	: Liquid Crystal Display
SHA	: Secure Hash Algorithm
XCTU	: XBee Configuration and Testing Utility
WSO2IS	: WSO2 Identity Server
MIPI	: Mobile Industry Processor Interface
DSI	: Display Serial Interface
CSI	: Camera Serial Interface
ACL	: Access Control List
JWT	: JavaScript Object Notation (JSON) Web Token
MITM	: Man-in-the-Middle
PDU	: Protocol Data Unit
CRC	: Cyclic Redundancy Check
TIFS	: Time Inter-Frame Space
DLE	: Data Length Extension
UART	: Universal Asynchronous Receiver Transmitter
IPS	: Indoor Positioning System
RSSI	: Received Signal Strength Indicator
FSPL	: Friis Free Space Path Loss
ECDSA	: Elliptic Curve Digital Signature Algorithm
WEP	: Wired Equivalent Privacy
WPA2	: Wi-Fi Protected Access 2

CHAPTER ONE
INTRODUCTION

1.1 Motivation

The notion of the Internet of things (IoT) traces back to Kevin Ashton's seminal radio frequency identification (RFID) presentation in 1999 (Ashton, 2009) where he envisioned a world where everyday objects could be connected to the internet, enabling them to communicate and interact seamlessly. Since then, the concept has evolved and gained momentum, permeating various sectors and revolutionizing the way we perceive and interact with technology. The concept of the IoT has gained prominence due to the increasing variety of Internet access options and the widespread adoption of Internet-enabled devices. Today, IoT has become an integral part of our digital landscape, promising unprecedented levels of connectivity and efficiency across industries. From smart homes and healthcare to agriculture and manufacturing, IoT technologies are reshaping processes, enhancing productivity, and driving innovation (Farooq et al., 2023). Essentially, IoT devices communicate with each other over the internet, collecting and storing data while autonomously managing themselves. This functionality is particularly valuable in areas where human access or physical intervention is challenging.

In the realm of computer networks, an edge device represents an end device that connects to the network infrastructure without being a critical component of it. These devices have the capability to communicate not only with other edge devices but also with cloud servers or central units that form part of the infrastructure (Qiu et al., 2020). They are frequently employed for computational tasks in scenarios requiring real-time or near-real-time processing and decision-making. By executing local processing and analysis at the edge, these devices can deliver prompt responses and lessen reliance on centralized resources. Processing data at the network's edge, closer to the data source, edge devices help to minimize latency and bandwidth demands.

IoT devices have the flexibility to utilize either wired or wireless connections, including options like Ethernet, wireless fidelity (Wi-Fi), cellular networks, and specialized protocols such as Bluetooth low energy (BLE), ZigBee, or LoRa. In the context of an IoT setup, a dedicated gateway device assumes the role of an

intermediary. This device facilitates data collection, processing, and communication among local devices, sensors, and cloud servers, utilizing the aforementioned networks or protocols.

Given the likelihood of IoT devices containing sensitive personal or commercial data, ensuring the security and privacy of IoT applications has become imperative. Monitoring device presence, establishing secure data transmission, and preventing unauthorized data transmission, possibly malicious in nature, are critical considerations. However, devising universally accepted security solutions for IoT proves challenging due to the lightweight nature of most IoT devices. These devices typically feature limited processing capability, low energy resources, and minimal data storage. The coexistence of diverse communication technologies further complicates this landscape.

At present, edge devices play a pivotal role in upholding the security and privacy of data within IoT applications. These devices implement essential security measures, including encryption, authentication, and access controls. These measures serve to protect data both at the edge and during transmission, addressing the unique challenges posed by the characteristics of IoT devices and the diversity of coexisting communication technologies.

Hence, it is crucial to implement a comprehensive security mechanism capable of accommodating multiple co-existing communication protocols within a single large-scale IoT application, which would significantly benefit the entire IoT industry. The objective of our study was to design a multi-protocol gateway capable of operating across different security levels and communication protocols, providing authentication and authorization for lightweight devices connected to the Internet. The primary motivation behind this research stems from the absence of a secure multi-protocol gateway in both commercial markets and existing literature that has proven successful in practice, particularly in addressing the diverse security requirements of BLE, Wi-Fi, and ZigBee IoT devices.

Therefore, for deployment in such a gateway unit, we propose a secure and lightweight multi-factor authentication scheme tailored for IoT applications,

leveraging three wireless communication technologies: Wi-Fi mesh, BLE, and ZigBee. This scheme enhances security and ensures encrypted communication channels between IoT devices (e.g., sensors, consumer electronics, automation devices, etc.) and the gateway. It incorporates multi-factor authentication, employing two distinct factors: a hash-based message authentication code (HMAC)-based one-time password and a quick response (QR) code. These factors offer "something you have" authentication for IoT device integration into the network via Wi-Fi mesh, BLE, and ZigBee. Moreover, our approach integrates password-based authentication as "something you know" and token-based access control management as "something you have" for interactions between IoT devices and the gateway. This fusion of authentication and access control methods aids in mitigating the risks associated with password guessing, replay attacks, and impersonation attempts.

1.2 Problem Statement

Here are the main issues outlined by the thesis, arising from the lack of a standard secure communication protocol covering various communication technologies and revealing the need for a secure IoT gateway with the increasing variety of devices in IoT applications.

When IoT devices transmit data through their network connections, it's crucial to safeguard the information from interception and unauthorized exploitation by third parties not authenticated within the network (Valea et al., 2019). Ensuring secure communication requires confidentiality, especially when transmitting data over insecure channels in plaintext. However, the MQTT protocol lacks native mechanisms for enforcing data encryption and integrity between clients and brokers.

While well-known asymmetric encryption mechanisms can be beneficial in such networks, not all devices capable of connecting to the network possess the resources to effectively execute complex encryption algorithms. Additionally, some devices may lack the capability to perform any encryption operations whatsoever (Singh, Sharma, Moon, & Park, 2017). Therefore, encryption stands out as a crucial focal point that requires attention.

Maintaining data integrity, a significant challenge in lightweight networks, guarantees that data remains unchanged during transmission from the source to the target device, with no unauthorized modifications. Detecting and rejecting any unauthorized alterations by receiving devices is essential.

Message integrity can be confirmed through the utilization of a message authentication code (MAC) (Mahmood, Chaudhry, Naqvi, Shon, & Ahmad, 2016). While powerful computers can often tackle these issues in both local and internet communications, it poses a significant challenge for lightweight devices within the IoT ecosystem. Similarly, ensuring service integrity involves securely collecting data from devices, either through device communication or direct transmission to the gateway, without distortion. This is crucial for maintaining accurate real-time data and identifying corrupted device data.

Authentication necessitates reciprocal verification of identities between end devices and gateways, essential given the multitude of IoT devices, hindering data tracing and thwarting malicious eavesdropping or tampering by third parties (Esfahani et al., 2017).

On the contrary, non-repudiation ensures that parties cannot deny their participation in data communication by necessitating the recording and digital signing of the communication. Although it may appear insignificant for IoT, it presents substantial risks for e-commerce and e-signature applications, where gateways play a vital role in addressing these vulnerabilities.

Messaging protocols set standards for data exchange among devices. Although secure hypertext transfer protocol (HTTP/HTTPS) is extensively utilized on the Internet, it's customized for sophisticated computers, incorporating intricate features unnecessary or unsupported by lightweight IoT devices. Consequently, protocols such as MQTT, constrained application protocol (CoAP), advanced message queuing protocol (AMQP), and extensible messaging and presence protocol (XMPP), crafted for these more straightforward devices, prove to be more fitting (Yerlikaya & Dalkılıç, 2017). The selection of a protocol is contingent upon the specific use case and represents a fundamental consideration within our research framework. In the context of IoT applications, where devices possess diverse attributes including bit rate,

capacity, and functionalities, inter-device communication is imperative. This necessitates the formulation of a security framework capable of dynamically accommodating the hardware characteristics inherent in these heterogeneous devices. The resolution of this challenge thus emerges as a pivotal aspect of our scholarly inquiry.

1.3 Scope & Purpose

The potential for transporting sensitive data, whether personal or commercial, through a large number of connected devices to the Internet poses a significant risk of data theft and corruption, especially considering that many of these devices may lack modern security measures. However, due to the diversity of devices and technologies, it is not possible to develop universally applicable security solutions, and generalizing threats may not be accurate. The aim of the thesis is to address this inconsistency and standardize communication methods by developing a gateway capable of implementing various security mechanisms with different security tiers, ensuring that connected devices can securely access the Internet. The lack of a widely accepted secure gateway or gateway protocol for IoT devices, despite ongoing research, is the primary motivation behind the thesis.

The following research questions are addressed:

1. How to overcome the limit on the number of devices that can communicate with each other in an IoT application (e.g., smart home environment) where all devices are connected via Wi-Fi?
2. How to design a gateway that integrates Wi-Fi Mesh network, BLE, and ZigBee communication?
3. Which IoT devices are preferable when establishing BLE, ZigBee, and Wi-Fi Mesh Network?
4. What is the preferred IoT device for serving as the gateway, and how does the increasing number of connected IoT devices impact performance?
5. Can security threats be generalized and standard security mechanisms be implemented in IoT, considering the absence of a standard architecture?

6. Is it feasible to implement existing cryptographic techniques considering devices with low processing power and energy constraints, as well as communication technologies like BLE, ZigBee, and Wi-Fi?
7. How can the increasing number of devices in a smart home environment, each requiring network access and access control, be managed securely and effectively?
8. How can security be tiered based on device processing capability, device diversity, and the security requirements of the application?

1.4 Contributions and Organization of Thesis

In the scope of the thesis, there is a design and validation of a multi-protocol gateway that includes Wi-Fi, BLE, and ZigBee wireless communication and ensures the security of these communication technologies. Our solution serves as a means for physical tracking, authentication, and authorization, providing security based on the selected security level between the end device and the gateway. Simply put, our approach can be summarized as follows: suggested design enables end-to-end secure communication of IoT devices, including end devices, relay gateway, and main gateway. Initially, a developer sets up the authorization server by registering IoT devices with their relevant topics, representing access rights and permissions of nodes communicating over Wi-Fi mesh, BLE, and ZigBee networks. The relay gateway acts as an intermediary for end devices communicating with multiple protocols. End devices generate HMAC-based OTPs to join the network and transmit them to the relay gateway via QR-based codes. Secure communication between the relay gateway and the main gateway, incorporating password and token-based authentication and access control, is achieved through the enhanced MQTT communication protocol.

The main contributions the relevant literature and industry of the thesis are six-fold as follows:

First, it provides a general overview of the recent literature in design of multi-protocol gateway by discussing which communication technologies are co-exist and which IoT devices are mostly used.

Secondly, this thesis will introduce a secure gateway architecture that unifies devices with various connection technologies. Furthermore, while existing devices and systems provide a certain level of security, there is currently no standardized device or system that offers security at the cryptographic level, which is the focus of this thesis.

Third, a new approach is proposed, integrating mutual authentication and role-based authorization functionalities between relay gateway and the main gateway using the OAuth 2.0 authorization framework.

Fourth, lower-level solutions are implemented to authenticate end devices on a Wi-Fi mesh, BLE and ZigBee network to relay gateway by employing multi-factor authentication with QR code-based and HMAC-based one-time passwords.

Fifth, the enhanced secure MQTT protocol has been adapted to facilitate communication between the main gateway and connected relay gateway.

Lastly, a method is introduced to determine the location/node where security measures, such as device authentication, mutual authentication, encryption, and decryption and integrity, are applied across the network.

This thesis consists of six chapters, with the remaining sections structured as follows:

Chapter 2 delves into recent research on multi-protocol gateways for IoT applications, focusing on three wireless communication technologies: Wi-Fi, BLE, and ZigBee. The discussion aligns with lightweight cryptographic techniques such as authentication, authorization, and availability.

Chapter 3 introduces the design of a multi-protocol gateway with multiple gateways. It also provides detailed information about the hardware and software components used and their interactions. Following this, it gives an overview of setting up the Wi-Fi mesh network, configuring the Zigbee network, and establishing BLE communication. Additionally, it explicates the security levels.

Chapter 4 extensively covers the security levels of the secure multi-protocol network supporting the wireless communication technologies mentioned in Chapter 3.

Chapter 5 visualizes the computational cost, communication cost, message reliability, and latency brought by the developed security mechanism based on communication technologies. It also discusses the percentage increase in cost when applying cryptographic techniques to IoT devices compared to advanced devices. Moreover, it delves deeply into the security threats emerging in the multi-protocol gateway with multiple gateways. In response to the discussed issues, it proposes and thoroughly analyzes a novel mutual authentication and role-based authorization scheme for IoT applications. Finally, Chapter 6 presents the conclusions of the thesis.

CHAPTER TWO
BACKGROUND

In this section, the security requirements of IoT are examined, and information about the current cryptographic algorithms used in the literature is provided. The cryptographic techniques and protocols used in the multi-protocol gateway that we designed and validated are explained in detail. Furthermore, the communication technologies supported by the developed multi-protocol gateway are identified.

2.1 Related Work

This section provides a detailed analysis of communication protocols, technologies, lightweight encryption algorithms, and cryptographic methods used in IoT applications. It categorizes research efforts into improving the security of the MQTT protocol, ensuring data confidentiality through lightweight encryption algorithms, and enhancing the security of multi-protocol accepting gateways. The current state is elucidated by categorizing studies and discussing their provided security measures (authentication, confidentiality, data integrity, authorization mechanisms, etc.), cryptographic methods and algorithms, performance metrics (throughput, message reliability, etc.), and test environment (used IoT devices, protocols, etc.). The reviewed studies mainly cover the latest developments in the field, spanning from 2014 to 2024.

2.1.1 An Overview of Studies on MQTT Security in the Literature

This part covers earlier attempts to secure the MQTT protocol by adding authentication and authorization measures. The successes of these methods are detailed, and any remaining problems are highlighted.

Fremantle, Aziz, Kopecký, & Scott (2014) incorporated the OAuth 2.0 protocol to enable access control in MQTT communications. They argue that traditional authorization and authentication systems are inadequate for IoT scenarios, as managing each device with a username-password pair and related role credentials is challenging. However, they did not offer any user interface or interaction scheme between the devices and the developer. Using the OAuth 2.0 protocol, each client acquires the access token and scopes to handle authentication and authorization when

connecting to the MQTT broker. MQTT runs over TCP, which allows for TLS to secure the connection. Nonetheless, many IoT devices have limited battery life, storage capacity, and computational power. In their study, the connection between clients and the MQTT broker was made without TLS, as they used an 8-bit Arduino for the client device. Limited storage makes refreshing the token challenging. Reusing the same token, however, poses security threats like eavesdropping, spoofing, and denial of service attacks. Additionally, exchanging the scope is not feasible because each scope is linked to an access token. Despite being an inspirational effort, a practical solution must employ dynamic tokens or passwords. Later, Singh, Rajan, Shivraj, & Balamuralidhar (2015) enhanced the security of the MQTT protocol by utilizing lightweight elliptic curve cryptography (ECC) for encrypting broadcast messages. They opted for ECC to reduce the computational complexity. Initially, the publisher creates an access list to be sent to the broker and requests an encryption key from the broker. The publisher also includes details about user access rights needed to decrypt the messages and transmits this information to the broker. If a subscriber possesses access permission for a particular message, they receive the key from the broker and use it to decrypt the message. However, the process by which the key is generated is not specified. Additionally, the publisher transmits the access tree (a hierarchical data structure representing the access policy) to the broker and receives the key from the broker over an unsecured channel. Consequently, an attacker could easily intercept this information through eavesdropping.

In a home automation system proposed by (Upadhyay, Borole, & Dileepan, 2016), an access control list (ACL) plugin is integrated into the MQTT protocol to restrict access to sensor data. In their model, an ACL is a file that contains usernames, encrypted passwords, and topic permissions. This approach has two significant implications. Firstly, in a dynamic and densely populated ecosystem, the frequent addition or removal of devices is common. This ongoing activity necessitates the continuous updating of the list with new usernames, passwords, and access rights. As a result, the list must be continually updated and redistributed among the devices, leading to significant overhead. Additionally, this method only addresses authorization without implementing specific authentication controls. Niruntasukrat et al. (2016) proposed an MQTT access control system based on the OAuth 1.0 authorization

protocol to minimize computational demands. Furthermore, an additional authentication factor was introduced, which involves distributing unique credential information over an HTTPS connection to secure the tokens. However, the security issues they addressed represent only a subset of the potential risks, as they themselves acknowledged. While the system does provide authorization, it lacks confidentiality for the shared information. Consequently, it remains vulnerable to eavesdropping, replay attacks, man-in-the-middle attacks, and denial-of-service attacks.

Zamfir, Balan, Iliescu, & Sandu (2016) performed a comparative study on the security aspects of MQTT and CoAP protocols, examining both the pre-shared key method and the utilization of raw public keys. The pre-shared key technique utilizes symmetric encryption and TLS to secure communications between endpoints. This method is straightforward with CoAP but is not inherently supported by MQTT. Furthermore, the use of TLS in this context increases computational overhead. The raw public key method faces similar limitations. Although both protocols support the use of certificates, they demand substantial computational power and energy. In contrast, our approach employs AES for payload encryption. Additionally, our method eliminates the need for extra computation and avoids using non-secure channels for key generation and distribution. Rajan et al. (2016) introduced a mechanism where the broker utilizes hierarchical inner product encryption (HIPE) to provide data privacy based on secure user profiles for data subscriptions. As a result, they succeeded in granting access exclusively to clients authorized for real-time video sharing. The study concludes that the use of HIPE encryption and decryption introduces some delays in data transmission. Furthermore, the implementation of role-based access control, which necessitates a hierarchical structure, is typically not well-suited for IoT environments.

Nagarajan, Deverajan, Chatterjee, Alnumay, & Ghosh (2021) introduced the RES-256 algorithm, which provides an access control mechanism designed to enhance the security of healthcare systems based on the Internet of Medical Things. This algorithm incorporates advanced encryption mechanisms such as RC6, secure key exchange protocols like ECDSA, and SHA-256 for hashing. Employing RC6 for encryption, ECDSA for key exchange, and SHA-256 for data integrity helps secure sensitive data

and maintain the integrity of healthcare information. These cryptographic algorithms are well-established and robust, making them suitable for enhancing the security of medical data. However, the suggested mechanism fails to address the weaknesses associated with weak or short passwords due to its reliance on SHA-256. In an alternative approach within the healthcare sector, (Alshammari, 2023) introduced a method where the broker distributes a pre-shared public key to clients, and the broker verifies usernames and passwords using a database. Fathy & Ali (2023) examined an IoT scenario focused on agricultural irrigation involving sensor and actuator devices, where it is crucial to ensure authentication between clients and brokers. They achieved confidentiality exclusively among clients by employing the expeditious cipher (X-cipher) without placing emphasis on access authorization. Furthermore, the study evaluated and compared the runtime memory usage performance of the X-cipher, AES, and PRESENT encryption algorithms.

Shilpa, Vidya, & Pattar (2022) proposed utilizing the secure reliable message communication (SEC-RMC) protocol in conjunction with Mosquitto to enhance the security of MQTT data transmission. In the study, a public-private key exchange occurs between Mosquitto and the clients, with AES encryption safeguarding data confidentiality. The broker validates the topic using the database before disseminating it to the subscriber. In contrast to this scheme, MARAS (Şeker, Dalkılıç, & Çabuk, 2023) utilizes the OAuth 2.0 protocol to grant access to topics and manage client permissions for publishing or subscribing. Similarly, (Winarno & Sari, 2022) compared the runtime performances of various lightweight encryption algorithms concerning encryption and decryption operations on MQTT payloads. The study assesses the performance of AES-128 in Galois/Counter Mode (GCM), GIFT-Combined Feedback (COFB), Romulus N1, and Tiny JAMBU. Although payload encryption ensures data transfer confidentiality, it fails to address other essential elements of secure communication, such as authentication, data integrity, and protection against replay and impersonation attacks. Solely depending on payload encryption is inadequate for achieving comprehensive security in MQTT communication. MARAS (Şeker et al., 2023) employs a holistic security approach, encompassing all aspects of secure communication to effectively reduce potential risks and vulnerabilities.

When asymmetric encryption is preferred, ECC stands as another powerful alternative. When opting for asymmetric encryption, ECC is a highly effective option. Ramyasri, Murthy, Itapu, & Krishna (2023) discuss the deployment of a public-key encryption system that uses ECC to protect data within an MQTT-based energy management system. ECC employs elliptic curves over finite fields to offer strong encryption with smaller key sizes compared to other encryption algorithms. Hence, it is the preferred choice for resource-constrained devices in IoT applications. Patel & Doshi (2023) similarly endorsed the adoption of ECC. They introduced a novel authentication method for IoT applications, referred to as Level-Dependent Authentication for Generic IoT (LDA-2IoT), which leverages ECC. The suggested method enables users at a particular level in the hierarchy to access sensor devices deployed either below or at the same hierarchical level. The research validated the LDA algorithm in a communication setting utilizing MQTT over TLS as its secure channel, encompassing users, gateways, and sensor devices. Utilizing ECC for encryption, (Saqib, Jasra, & Moon, 2022) presented a framework to tackle major security challenges in IoT-based networks. Their method integrates ECC and hash chains to develop a signature-based three-factor authentication system customized for IoT applications. Like our proposition, MARAS (Şeker et al., F2023) utilizes a dynamic session key for mutual authentication to thwart known session key attacks. Payload encryption and access authorization play a pivotal role in safeguarding the security of their MQTT communications. While the study may not have specifically underscored these facets, it is imperative to address payload encryption and access authorization when deploying a secure MQTT protocol.

In a study conducted by (Katsikeas et al., 2017), payload encryption was applied to the data transmitted via the MQTT protocol using various AES modes. Their findings indicated that AES-CBC is the optimal choice for resource-constrained IoT devices. The study establishes a robust understanding of AES-based encryption schemes, although it excludes coverage of the AES-CFB mode. However, the research primarily concentrates on confidentiality and overlooks other security considerations. Moreover, the study lacks clarification on the favored methods for key generation and distribution. Absence of an authentication scheme leaves the system vulnerable to potential threats. A malicious subscriber could intercept all published information on

any topic, while a malicious publisher might flood the system with spam messages, potentially leading to a Denial of Service (DoS) attack.

Bhawiyuga, Data, & Warda (2017) employed a JSON Web Token (JWT) authentication server to authenticate devices within the MQTT network. This process involved a direct token exchange between clients and the broker, with the broker subsequently validating the tokens by transmitting them to the authentication server. However, although mentioned briefly, their research fails to clarify how a legitimate client device can sustain or restore its connection following the expiration of its token. It seems that both the broker and the authentication server decline connection requests under such circumstances. Furthermore, although the token employs a static structure and utilizes a hash algorithm to guard against integrity attacks, it remains susceptible to interception by eavesdroppers. Consequently, malicious nodes can repeatedly reuse the token through replay attacks until its expiration. To address these vulnerabilities, a two-step authentication scheme incorporating a dynamic password mechanism should be implemented, as demonstrated in our study.

Bashir & Mir (2017) advocated the utilization of a trusted third-party server for managing key generation and distribution in MQTT. The server generates distinct client IDs and disseminates them to clients, which are subsequently employed as pre-shared keys for encrypting the transmission of a secondary key. Following this, the clients' data payloads are encrypted using the aforementioned secondary key. In their setup, the initial distribution of the IDs necessitates a TLS connection, unless they are transmitted in clear text (which poses a heightened risk). Furthermore, this method fails to address replay attacks or the inundation of large messages by malicious nodes.

As a result, attackers could readily overload the message traffic due to the absence of a proper device authentication mechanism within the broker. In their research, (Wardana & Perdana, 2018) introduced a prototype of an access control mechanism for IoT devices, centered on the utilization of tokens alongside the Redis authorization server. Additionally, to ensure data integrity and confidentiality between the cloud and the MQTT broker(s), an SSL certificate was provided. While the proposed method is robust within its defined parameters, it lacks a mechanism for clients to authenticate brokers. Another novel security proposal for MQTT was put forth by (Calabretta,

Pecori, & Veltri, 2018). They presented a novel method for enhancing the MQTT protocol, utilizing the augmented password-authenticated key agreement (AugPAKE) algorithm over the ActiveMQ middleware. Additionally, they suggested a method for generating authorization tokens for individual topics to enable topic-specific access control. Their research surpasses a previous similar study that also utilizes AugPAKE without TLS, conducted by (Shin, Kobara, Chuang, & Huang, 2016), which lacks authorization capabilities. Despite its broader scope, the study by (Calabretta, Pecori, & Veltri, 2018) overlooks the authentication of brokers by clients, which is crucial.

A study authored by (Bali, Jaafar, & Zavarasky, 2019) highlights that a perpetrator with access only to ciphertext may potentially intercept the traffic and decrypt the password immediately after the broker authenticates the initial publisher and subscriber. They introduced a chaotic lightweight algorithm as a safeguard against the aforementioned ciphertext-only attacks. This algorithm guarantees that the key is regularly updated by both the broker and the client(s). However, this consistent effort has two minor drawbacks: firstly, it relies on TLS for the initial authentication, and secondly, it maintains a record of topics and authorized clients on the broker. Sundarrajan & Narayanan (2019) implemented advanced multiple encryption systems (AMES) utilizing Hardy Wall encryption, which demands reduced bandwidth and minimal memory usage within the MQTT protocol. This approach aims to guarantee secure message communication without any loss of packets. IoT device IDs and user IDs are encrypted using Hardy Wall encryption to offer authentication to a certain extent. However, their framework assumes an unwarranted level of trust among the nodes. The paper also omits critical details concerning security vulnerabilities, potential attacks, and their consequential impacts, which are indispensable in such studies. Public key cryptography effectively addresses the authentication challenge in MQTT. However, key distribution remains a significant issue in this context. Aknin & Bentaleb (2023) put forward a framework that combines blockchain and smart contracts within an MQTT infrastructure to guarantee data immutability and automate the authentication, publishing, and subscribing procedures. In their framework, authentication in the MQTT protocol utilizes OTP. TLS is employed during the initial phase to securely transmit the keys necessary for OTP generation and message encryption.

Sanjuan, Cardiel, Cerrada, & Cerrada (2020), similar to (Patel & Doshi, 2023), suggested employing smart cards for distributing keys to clients and brokers. Their approach includes mutual authentication and payload encryption. While effective against many remote attacks, their approach remains susceptible to physical attacks such as node and smart card capture. In contrast, (Amoretti, Pecori, Protskaya, Veltri, &, Zanichelli, 2020) employed a cloud server alongside a token-based authentication scheme. They also incorporated support for multiple brokers through broker bridging. However, their approach lacks mutual authentication and payload encryption. Nevertheless, their method does not include mutual authentication and payload encryption. The challenge of securing the MQTT protocol remains relevant, highlighted by (Blazy, Conchon, Klingler, & Sauveron, 2021), who suggested a comprehensive framework to improve the protocol's security beyond its fundamental encryption scheme. Their authentication mechanism was based on attribute-based signatures. While this framework may be suitable for numerous IoT systems, it could pose computational challenges for networks dominated by lightweight sensors due to its quasi-complex computation requirements. In the study by (Azzedin & Alhazmi, 2023), the utilization of multiple MQTT brokers and their authentication using JSON web tokens was proposed. However, the research overlooks essential security aspects such as client-broker authentication, data confidentiality, and client authorization. This oversight leaves the system vulnerable to potential attacks including eavesdropping, man-in-the-middle attacks, and impersonation attacks.

The results of our detailed comparative analysis are provided later in Chapter 5. According to our literature review, the existing works within this field are either a survey/analysis of the security risks concerning the use of plain MQTT; or a (more conventional) security scheme that is inefficient (i.e., uses SSL/TLS, etc.); or a (more innovational) security scheme that is incomplete (i.e., lacking mutual authentication, vulnerable to eavesdropping, etc.). Thus, having a high-level security approach that is both efficient and complete at the same time is still a paramount concern, which explains why this study is conducted and which problems it resolves.

2.1.2 Lightweight Encryption Algorithms

Within this subsection, we delve into the discussion of lightweight encryption algorithms found in the literature. Examining various symmetric encryption algorithms used in IoT scenarios, we compare their attributes, including the number of rounds and key sizes. Furthermore, an overview of asymmetric encryption methods is provided, drawing from the research contributions of (Bhardwaj, Kumar, & Bansal, 2017) and (Singh, Sharma, Moon, & Park, 2017). This comprehensive analysis sheds light on the diverse encryption techniques employed in the realm of IoT security.

AES is a symmetric encryption algorithm that uses the same key for both encryption and decryption of messages, making it suitable for resource-constrained devices. AES operates as a block cipher with a block length of 128 bits and supports keys of varying lengths such as 128, 192, and 256 bits. Each key undergoes a different number of processing rounds, specifically 10, 12, or 14 rounds. AES is built upon a substitution-permutation network (SPN). During encryption, each round involves operations such as substituting bytes, shifting rows, mixing columns, and adding the round key. In contrast, decryption entails operations including inverse shifting rows, inverse substituting bytes, adding the round key, and inverse mixing columns (Stallings, 2006).

PRESENT is classified as one of the ultra-lightweight algorithms and is commonly chosen for ensuring secure communication. Its encryption process operates on 64-bit blocks with key sizes of either 80 or 128 bits. Similar to AES, it is structured based on the SPN. Notably, the substitution layer employs 4-bit input and output S-boxes, which are optimized for hardware implementation. While PRESENT boasts hardware efficiency, it may suffer from increased cycle consumption at the software level due to its permutation layer. To adapt PRESENT for IoT applications and mitigate its high power consumption, it has been modified to work in conjunction with the TEA algorithm. To demonstrate its enhanced efficiency compared to the original PRESENT algorithm, various tests have been conducted. The N-gram test assesses character repetition in the encrypted text, while histogram and floating frequency tests, along with the non-homogeneity test, analyze the efficiency of the modified PRESENT algorithm (Chatterjee & Chakraborty, 2020).

According to (Lee & Lim, 2014) the proposed protocol consumes less energy per data transaction by using HEIGHT encryption to address the security vulnerability of RFID systems.

Ertaul & Rajegowda (2017) presents the results of a performance analysis conducted on three generalized Feistel lightweight block crypto ciphers, namely CLEFIA, PICCOLO, and TWINE, utilizing an STM32F401RE Microcontroller (MCU). The analysis focuses on energy consumption, memory requirements, and encryption execution time across various key sizes and plaintext sizes ranging from 512 to 3072 Bytes. TWINE, with a block size of 64 bits, offers two key sizes of 80 and 128 bits and supports between 25 and 32 rounds. It is noted that TWINE may be susceptible to zero-correlation attacks when employing 23 rounds with an 80-bit key and 25 rounds with a 128-bit key. To mitigate these vulnerabilities, it is recommended to utilize 36 rounds for both key sizes. CLEFIA, on the other hand, features a block length of 128 bits and supports key lengths of 128, 192, and 256 bits. To bolster security against both differential and integral attacks, round counts of 18, 22, and 26 are recommended for key sizes of 128, 192, and 256 bits respectively.

The paper proposes enhancements to the SIMON cryptography algorithm to reduce encryption time. It compares the execution time and memory requirements of the AES and the original SIMON block cipher algorithms. The SIMON algorithm may be susceptible to differential and linear attacks depending on the number of rounds employed. Increasing the number of rounds can negatively impact performance (Alassaf, Gutub, Parah, & Ghamdi, 2019).

SPECK is recognized as a lightweight encryption algorithm for IoT applications. In this study (Ge et al., 2019), the SPECK encryption algorithm is investigated in terms of susceptibility to side-channel attacks. The SPECK algorithm is refined to bolster resistance against power attacks, and experimental results regarding additional code, RAM usage, and execution time are analyzed.

The key advantage of ECC encryption algorithm lies in its ability to provide security comparable to other public encryption systems while using smaller key sizes. ECC achieves the same level of security as the RSA encryption algorithm with a 1024-

bit key by employing a much smaller 160-bit key. This makes ECC significantly more suitable for use in wireless networks due to its ability to maintain security while utilizing smaller key sizes (Mahto, Khan, & Yadav, 2016; Sinha, Srivastava, & Gupta, 2013).

2.1.3 An Overview of Studies on Security with Multi-Protocol Gateways

Several studies in the literature have delved into addressing aspects or variations of the problem highlighted in this paper. While their contributions are significant, our study offers unique insights, as outlined in Chapter One. In this section, we delve into previous research endeavors concerning the security of IoT utilizing multi-protocol gateways, with a focus on authentication and authorization aspects, while also highlighting unresolved issues.

Beniwal & Singhrova (2022) have classified IoT-based designs and research endeavors, particularly those addressing security concerns. Among the highlighted works (Jabbar et al., 2019), one accomplished real-time monitoring of a smart home environment and the management of household appliances through sensor data linked to NodeMCU. This process was enabled by utilizing the MQTT dash mobile application and the Adafruit IO Web interface, which could be accessed via a laptop or PC. Security measures were implemented through RFID, allowing users to conveniently monitor and take action via mobile devices in case of unexpected events detected within the home environment. However, their study does not address the simultaneous operation of multiple protocols but is restricted to Wi-Fi.

Saqib & Moon (2022) employed three-factor authentication employing ECC and hash chain techniques to authenticate IoT devices via the gateway. However, the gateway exclusively supports Wi-Fi communication. Similarly, (AlQahtani, Alamleh, & Al Smadi, 2022) introduced a proximity-based authentication system for Wi-Fi-enabled ad-hoc networks, utilizing Euclidean distance calculated from the devices' SSID, basic SSID (BSSID), and received signal strength indicator (RSSI) data. This study focuses solely on Wi-Fi communication, and the potential implementation of the proposed approach with BLE and ZigBee communication based on RSSI remains an aspect for future exploration.

A gateway architecture accommodating protocol interoperability was devised by (Chaudhary, Yousuff, Meghana, Ashwin, & Guddeti, 2021), facilitating remote home management and monitoring to promote energy conservation. Smart home gadgets synchronize various parameters including time, battery life, recharge intervals, deployment, and other cost-related factors using Wi-Fi, ZigBee, and Bluetooth protocols. Nevertheless, despite these advancements, the study overlooks the security of operational data. Moreover, there exist other studies presenting gateway architectures supporting multiple protocols yet deficient in essential security attributes (Khanchuea & Siripokarpirom, 2019; Hanggoro, Rosyidi, & Sari, 2019; Amiruddin, Ratna, Harwahyu, & Sari, 2018).

Hajian, Erfani, & Kumari (2022) focus on enhancing security against various threats, encompassing gateway key compromise impersonation attacks, sensing device compromise attacks, and insider attacks involving users, sensing devices, and gateways. Their efforts involve implementing robust login procedures, authentication mechanisms, and effective key management strategies. The protocol they developed aims to minimize processing overheads and payload costs by utilizing a three-factor authentication system, comprising personal identity, password, and biometric parameters. These objectives are achieved through simple yet effective cryptographic operations like XOR, concatenation, and hashing.

Marktscheffel et al. (2016) endeavored to thwart man-in-the-middle and replay attacks through QR code-based mutual authentication. Although their concept parallels ours, our objective is to tailor the approach for authenticating low-cost IoT devices rather than sophisticated devices such as smartphones. In the references (Saputra & Raharjo, 2019), the suggested authentication mechanism entails the creation of QR codes utilizing the password-based key derivation function 2 (PBKDF2), which relies on regularly updated personnel shift schedules and officer account details. Authentication is conducted by scanning the QR codes displayed on personnel devices via a mobile application, ensuring validation at the correct time and location.

In their study, (Naveen, Teja, Kalyan, & Basha, 2021), aimed to enhance the security of bank lockers by utilizing an ESP32 to scan QR information. If the scanned

QR code matches the personal information stored in the database, the bank locker is opened. This approach mitigates risks associated with unauthorized access to bank safes, such as duplicating physical keys or obtaining and misusing passwords.

Pearson et al. (2019) conducted extensive evaluations of the cryptographic and network performance of IoT microcontroller units and modules, including CC3220SF, ESP8266, and ESP32. They advocated for the utilization of the ATECC608A processor module, which offers hardware acceleration for AES, HMAC, and ECC, along with secure key storage capabilities. This module enhances security measures for low-cost IoT devices. The researchers proposed solutions to address hardware, firmware, and network security concerns, such as disabling I/O interfaces, implementing secure boot mechanisms, and utilizing secure sockets layer/transport layer security (SSL/TLS) protocols. Zhang, Liang, & Cai, (2019) highlight security vulnerabilities present in wearable devices and propose an authentication and availability solution utilizing time-based one-time password (TOTP), AES-128 encryption, and elliptic curve digital signature algorithm (ECDSA). Their research, conducted using the nRF52832 development board and a mobile device, addresses similar concerns to our security enhancement approach. Both studies focus on mitigating the lack of input/output (I/O) capabilities in the pairing process of BLE devices. However, our system design takes into account resource-constrained BLE slave and master devices. Additionally, to mitigate synchronization issues associated with TOTP, we opted for HMAC-based one-time password (HOTP) authentication.

Gupta & Varshney, (2023) introduce a secure authentication method relying on digital signatures to counter device spoofing and man-in-the-middle (MITM) attacks during BLE pairing and authentication. The study included an assessment of the energy consumption of ESP32, Arduino Nano, and Arduino smartphones.

2.2 Enabling Technology

The subheading emphasizes the importance of a secure multi-protocol gateway capable of accommodating various communication technologies with IoT devices, along with a brief explanation of multi-factor authentication methods, including QR code-based and HMAC-based OTP used to authenticate ESP32 devices. The MQTT

protocol is an overview and its security capabilities are discussed. And also, it elaborates on the OAuth 2.0 authorization protocol, which is utilized for token-based authentication, verification, and authorization of end devices by the main gateway. Additionally, it briefly explores the utilization of the Advanced Encryption Standard cipher-block chaining mode (AES-CBC).

2.2.1 Message Queue Telemetry Transport

Developed by IBM in 1999, the MQTT protocol was later standardized by the organization for the advancement of structured information standards (OASIS) in 2013 Organization for the Advancement of Structured Information Standards [OASIS], (2014) to facilitate lightweight machine-to-machine communication. Due to its publish/subscribe-based architecture, MQTT serves as a convenient communication protocol for numerous IoT scenarios and is particularly favored for devices with constrained resources.

In an MQTT-based network, nodes may assume various roles. The clients, primarily sensor or actuator devices, can function as publishers or subscribers for any data exchange, such as temperature sensor data. Each data flow is associated with a topic label, referred to as a "topic," based on its context, such as temperature. Consequently, certain clients may subscribe to specific topics while disregarding others.

There are intermediary nodes, referred to as brokers, tasked with disseminating information from publishers to interested subscribers in an MQTT network. Brokers maintain an updated record of which clients are interested in specific topics and subsequently forward incoming data to subscribers while filtering access rights. Publishers and subscribers lack prior knowledge of each other, with all communication routed through the broker(s). Publishers and subscribers are not required to be connected to the same network simultaneously. Occasionally, a broker may also function as a client. Similarly, a publisher of certain topics may also be a subscriber of others. Additionally, other types of nodes, such as servers or databases, may optionally coexist within the ecosystem. MQTT inherently facilitates communication over TCP/IP, enabling its usage in scenarios where Wi-Fi is employed, unlike BLE and

ZigBee communication which require assistance (such as a relay) to utilize MQTT. The message to be sent first has TCP packet headers added, giving the message TCP packet characteristics. Subsequently, IP and Ethernet headers are added in sequence, and the message is then sent to the recipient (Ebleme, Bayilmis, & Cavusoglu, 2018).

2.2.1.1 MQTT Message Packet

MQTT message packets consist of three elements: a compulsory 2-byte fixed-length header, an optional variable-length header, and an optional payload. The fixed-size header comprises the control field containing a 4-bit message type field, a 4-bit control flag, and the packet length field. MQTT supports 14 unique message types: connection request (CONNECT), connection acknowledgment (CONNACK), subscription request (SUBSCRIBE), subscription acknowledgment (SUBACK), unsubscription request (UNSUBSCRIBE), unsubscription acknowledgment (UNSUBACK), publish data (PUBLISH), publish acknowledgment (PUBACK), publish received (PUBREC), publish released (PUBREL), publish complete (PUBCOMP), ping request (PINGREQ), ping response (PINGRESP), and disconnection request (DISCONNECT) (Şeker et al., 2023).

The PUBLISH message packet contains a topic name to specify the topic of interest for the sending client. This topic name, represented as a string, is delineated with a slash (e.g., "/temperature/"). The "QoS" field determines the reliability of message transmission, particularly in scenarios involving an unstable Internet connection. It offers three levels: '0', '1', or '2', each tailored to varying degrees of transmission assurance. At level '0', the protocol dispatches the message at most once, thus disregarding potential message loss. Messages sent at this level are not stored further on Mosquitto. When set to '1', messages are retained until the client acknowledges receipt with a PUBACK message, guaranteeing delivery at least once. The "PacketID" field facilitates matching of the PUBLISH message packet with the corresponding PUBACK message packet. At level '2', the highest level, the protocol ensures the most reliable transmission by delivering messages exactly once, employing a four-handshake process with MQTT message packets. Upon transmitting the PUBLISH message to the broker, the broker confirms reception by sending a PUBREC message to the client. In the event the client does not receive the PUBREC message, it continues

to resend the PUBLISH message until receiving the PUBREC message with the "DupFlag" (duplicate flag) field. Upon receiving the PUBREC message, the client proceeds to send a PUBREL message containing the packet ID. Subsequently, the broker replies with a PUBCOM message to the client, ensuring the message's availability once again. The inclusion of the "DupFlag" field is imperative for both QoS levels '1' and '2'. Similarly, the packet ID holds significance beyond QoS level '0' as it ensures message uniqueness. The "Payload" field delineates the content of the message. QoS level '2' entails increased overhead and is infrequently employed in lightweight networks.

2.2.1.2 Security Features in MQTT

Due to its lightweight design, the MQTT protocol doesn't include advanced security measures beyond basic username and password authentication. In Networks without resource limitations, the transport layer security (TLS) protocol can be used alongside MQTT for enhanced data privacy and integrity (Munshi, 2022). However, this might not be feasible for lightweight devices, as setting up a secure end-to-end channel with TLS requires complex computations. Without additional security measures, MQTT subscribers could inadvertently provide access to all messages.

Furthermore, a malicious publisher has the potential to function as an attacker and initiate a denial-of-service (DoS) attack. The utilization of access tokens has been suggested in existing literature to establish authentication and authorization mechanisms (Ragothaman, Wang, Rimal, & Lawrence, 2023) yet currently there exists no efficient security mechanism between clients and an authorization server, except for TLS-based HTTPS. Moreover, previous research has not proposed the authentication of brokers from the client side, a vital aspect for fostering mutual trust and security within the ecosystem. The study presents a two-step mutual authentication and authorization system employing a "trusted" third-party authentication server operating on OAuth 2.0. Furthermore, the management of role-based access controls can pose challenges due to the potentially extensive number of connected devices. Given MQTT's absence of a built-in security mechanism to tackle these challenges, a dynamic token scheme integrating clients' scopes (i.e., authorized topics) is being contemplated through the OAuth 2.0 protocol (Şeker et al., 2023).

2.2.2 OAuth 2.0 Authorization Framework

The utilization of IoT applications and the global count of IoT devices in operation have seen a consistent rise over time. As the quantity of devices continues to expand, distinguishing individual IoT devices uniquely has become increasingly challenging. Furthermore, one of the primary challenges lies in resolving the access authorization of these devices lacking a user interface and input unit. To address this concern, we have integrated the OAuth 2.0 authorization protocol into our multi-protocol gateway. It enables secure delegated access by permitting devices to acquire a restricted access token (Hardt, 2012). This token serves as the device's credentials and permissions within the system, efficiently controlling access without revealing device secrets. Through this methodology, our gateway possesses the capability to dynamically authenticate and authorize IoT devices, thereby ensuring secure communication and access control within the IoT ecosystem.

2.2.3 HMAC Based One Time Password

Established as a standard in 2005 (Park, 2018), HOTP is utilized for mutual authentication and the prevention of replay attacks, all achieved without the need for time synchronization (Kim & Kim, 2011; Saxena, 2008). In the underlying mechanism, HOTP depends on HMAC, which utilizes a distinctive secret (K) and a counter value (C) that need to stay synchronized with the HOTP generator. HOTP is a variable number typically consisting of 6 or 8 digits, which remains valid for a short duration of time. To calculate the OTP, both the client and the server sides necessitate a secret key and a counter. This process involves performing a hashing operation and truncating the result on both the client and server sides. An HOTP is produced by truncating a pre-computed HMAC-SHA-1 value, demonstrated below (M'Raihi, Bellare, Hoornaert, Naccache, & Ranen, 2005):

$$\text{HOTP}(K, C) = \text{Truncate}(\text{HMAC_SHA-1}(K, C)) \quad (2.1)$$

HOTP is a variable number typically consisting of 6 or 8 digits, which remains valid for a short duration of time. To calculate the OTP, both the client and the server sides

necessitate a secret key and a counter. This process involves performing a hashing operation and truncating the result on both the client and server sides.

Our multi-gateway design is utilized for both authenticating nodes by the RG and dynamically authenticating the RG by the Main gateway. Unique values manually added to IoT devices (details of their creation and integration into devices are elaborated in later sections) serve as secrets for HOTP calculations, and they determine the counter values. When the RG receives an MQTT connect message request from a device, one of the authentication steps will require HOTP verification. Hence, depending on the application's requirements, N HOTP values are generated by the MG and stored in javaScript object notation (JSON) format associated with the respective node in MongoDB Atlas cloud database. After each authentication, the counter value is updated by the Node to maintain synchronization. In case of potential internet connection interruptions or unused OTP values, the counter value is periodically checked and updated within a certain window interval. Therefore, the parameters required for the calculation are stored in the node's electronically erasable programmable read-only memory (EEPROM).

2.2.4 QR-based Authentication Protocol

The QR code, short for Quick Response code, was introduced in 1994 by Denso Wave, a subsidiary of Toyota, as a two-dimensional code designed to store a large amount of information, up to 4296 alphanumeric characters (Hung et al., 2019). Unlike traditional barcodes, which can only store data in one dimension, QR codes store data both horizontally and vertically, allowing for significantly more information to be encoded in a smaller space. The technology quickly gained popularity due to its versatility and ease of use, enabling quick access to information through a simple scan using a smartphone or QR code reader. QR codes are widely used in various applications, from product tracking and inventory management to mobile payments and website URLs. The MQTT communication protocol supports password-based device authentication. However, the static use of passwords brings security challenges such as eavesdropping and malicious use if the password is shared with others and used repeatedly. One significant advantage of QR codes is their ability to be dynamically updated; they can be refreshed or redrawn to represent changing

information, such as in the case of HOTP, enhancing security and convenience in authentication processes. This makes QR codes a valuable tool in modern digital interactions, providing both functionality and security.

2.2.5 Advanced Encryption Standard Cipher-Block Chaining

Rivest–Shamir–Adleman (RSA) and AES are two popular algorithms commonly compared in encryption applications. RSA is an asymmetric encryption algorithm typically used for key exchange or digital signature creation, while AES is a symmetric encryption algorithm primarily employed in data encryption tasks. The comparison between these two algorithms often revolves around factors such as security, speed, and resource usage.

In the realm of IoT, the utilization of these algorithms is particularly crucial, especially for resource-constrained devices. Asymmetric algorithms like RSA demand complex mathematical operations, leading to higher computational power and memory consumption, which may not align well with the limited resources of IoT devices. Conversely, symmetric algorithms like AES offer faster processing times and lower memory usage, thus providing more efficient performance in IoT devices.

Considering that IoT devices typically operate with limited memory and processing power, symmetric encryption algorithms like AES are often preferred. These algorithms facilitate fast and effective data communication between devices while ensuring security. Therefore, in the scope of thesis, the choice was made to prioritize data privacy, opting for AES symmetric encryption algorithm support on the ESP32 device, which promises better performance and lower resource consumption. AES-CBC mode stands out as the most widely utilized and highly secure encryption method for IoT devices.

2.3 Wireless Communication Technology

In this section, we discuss the widely accepted wireless communication technologies supported simultaneously by the secure multi-protocol gateway we designed, along with the existing security measures.

2.3.1 Wireless Fidelity (Wi-Fi)

Wi-Fi is a technology that enables wireless communication among devices using radio waves. Wi-Fi standards, defined by the Institute of Electrical and Electronics Engineers (IEEE), are identified by numbers such as 802.11a/b/g/n (Qureshi & Asghar, 2023). These standards vary in terms of frequency bands, data transfer rates, and other features. The widely used first Wi-Fi standard introduced in 1999, 802.11b, operated on the 2.4 GHz frequency band and offered a maximum data transfer rate of 11 Mbps, whereas the 802.11a standard, introduced at the same time, used the 5 GHz frequency band and transferred data at 54 Mbps. In 2003, the 802.11g standard was announced, providing a data transfer rate of 54 Mbps on the 2.4 GHz frequency band. The 802.11n standard, introduced in 2009, represented a significant advancement by utilizing multiple antennas to achieve a maximum data transfer rate of 600 Mbps. Subsequent standards, namely 802.11ac and 802.11ax (Wi-Fi 6), announced in 2013 and 2019 respectively, further increased the maximum data transfer rates to 6.9 Gbps and 9.6 Gbps, improving performance particularly in crowded environments.

The widespread support of IEEE 802.11 standard by almost all smartphones, laptops, tablets, etc., has increased the ability to respond rapidly to high-demand situations in real-time IoT applications, regardless of time and location. Additionally, it has contributed to energy savings for continuously operating devices. For instance, in smart cities, continuous data flow is supported through the smart and automatic management of Wi-Fi resources (Khanchuea & Siripokarpirom, 2019). Communication between users and access points, bandwidth allocation, channel settings, and data security are all managed optimally.

Wi-Fi security consists of various protocols and technologies used to protect wireless networks from unauthorized access. IEEE 802.11 security relies on the wired equivalent privacy (WEP) security method, which aims to provide privacy equivalent to wired networks. However, WEP had many security vulnerabilities and had to be replaced by stronger protocols. To address these shortcomings, the robust security network (RSN) concept was introduced, allowing only robust network security associations (RSNAs) to protect WLAN security against threats (Boland & Mousavi,

2004; Chen, Jiang, & Liu, 2005). RSN consists of three components: a wireless terminal (STA), an Access Point (AP), and an authentication server.

Among the most commonly used security protocols today are Wi-Fi protected access 2 (WPA2) and WPA3 (Valchanov, Edikyan, & Aleksieva, 2019). WPA2, introduced in 2004, provides strong protection using AES encryption. However, it can be vulnerable to certain security exploits. WPA3, introduced in 2018, offers stronger encryption algorithms and advanced security features, making it the most secure Wi-Fi protocol available today. WPA3 improves encryption key exchange with the simultaneous authentication of equals (SAE) protocol, providing more robust security. Additional security measures include SSID hiding, MAC address filtering, using strong passwords, and network segmentation. Moreover, regularly updating the firmware of routers and access points helps protect against new threats. These protocols and measures are critical for ensuring the security of Wi-Fi networks and preventing unauthorized access.

2.3.2 Bluetooth Low energy (BLE)

BLE was incorporated into the Bluetooth (BT) specification with the release of Bluetooth version 4.0 in 2009 (Cäsar, Pawelke, Steffan, & Terhorst, 2022). Since its introduction, BLE has become the prevailing protocol for low-power devices. Maintained by the (Bluetooth special interest group [Bluetooth SIG], 2014), BLE employs a wireless personal area network (WPAN) paradigm. Unlike classic Bluetooth, BLE operates at the 2.4 GHz frequency over short distances and can attain data transfer rates ranging from 1 to 2 Mbps, accommodating up to 251 bytes per unit of time (Gupta & Varshney, 2023). BLE operates on a narrower bandwidth compared to classic Bluetooth, utilizing 40 channels, each 2 MHz wide, while classic Bluetooth operates on 79 radio channels, each 1 MHz wide. Specifically, channels 37, 38, and 39 are reserved for advertising purposes in BLE (Tosi, Taffoni, Santacatterina, Sannino, & Formica, 2017). One notable feature of BLE is its energy efficiency, achieved by allowing nodes to remain in sleep mode until a connection is initiated.

The BLE standard (Woolley, 2019) introduces two fundamental structures. The first structure, known as the generic access profile (GAP), serves the purpose of enabling

devices to discover and establish connections with one another. On the other hand, the second structure, the generic attribute (GATT), facilitates the communication and interaction between connected devices. Within the BLE standard, there exist two primary roles: the GATT client and the GATT server. The GATT client, also referred to as the master or central, is responsible for initiating and controlling the communication sessions with other BLE devices. Considering the design in our study, the ESP32 node assumes the role of a GATT server (also known as slave or peripheral) while collecting sensor data, whereas the ESP32 relay gateway, which connects to this data, takes on the role of a GATT client. SIG (Bluetooth SIG, 2014), has established a collection of standard services known as GATT services. Each service is identified by a unique identifier known as the universally unique identifier (UUID). Within each service (GATT service), there exist characteristics (GATT characteristic) that define the specifics of the service. The points at which communication occurs are characteristic features. To facilitate this, the GATT server generates a GATT descriptor for each characteristic feature. These descriptors delineate the functioning of characteristics and the type of information they contain. GATT characteristics interact with the following methods outlined by a GATT descriptor:

Write: The characteristic feature is capable of being written by GATT clients, enabling the transmission of data from the GATT client to the GATT server.

Read: The characteristic can be read once by the GATT client, meaning that each READ operation retrieves data only once.

Notify: The GATT client continuously listens for updates on this characteristic feature. The GATT server regularly updates this characteristic feature and notifies the GATT client accordingly. Unlike with indications, the GATT client does not confirm the receipt of the message to the GATT server, making the process faster.

Inform: Similar to the notify definition, the GATT client monitors this characteristic. However, upon receiving this message, it notifies the GATT server of its reception. Consequently, the inform operation is slower than notify due to the transmission of an acknowledgment message.

The security level achievable during the pairing process between BLE devices is contingent upon their input-output (IO) capabilities. The BLE security framework encompasses four methods for establishing a secure connection during pairing between two BLE devices: “just works,” “numeric comparison,” “out-of-band (OOB),” and “passkey”. If both devices possess display and input capabilities, the “numeric comparison” or “passkey” methods can be employed. However, if one or both devices lack display or input capabilities, the a “just works” method is the sole option available (Cäsar, Pawelke, Steffan, & Terhorst, 2022; Gupta & Varshney, 2023).

The “Just Works” method, regarded as the most straightforward, offers a fundamental level of security by employing a randomly generated 128-bit key for devices lacking input-output capabilities. Nonetheless, it is susceptible to man-in-the-middle attacks.

The “Numeric Comparison” method is employed when both devices possess display and input capabilities. It offers an elevated level of security by necessitating user confirmation of a 6-digit passcode displayed on both devices. Compared to the “Just Works” method, it provides increased security, particularly against man-in-the-middle attacks.

The “passkey” method is employed when one device features both display and input capabilities, while the other device only has a display. It offers enhanced security compared to the “just works” method by requiring the user to input a 6-digit passkey on the device with input capabilities. This method aims to mitigate man-in-the-middle attacks and provides a higher level of security.

The “OOB” method involves the exchange of a 128-bit value between two devices utilizing a communication technology other than BLE, such as near field communication (NFC), for authentication purposes. This approach enhances security by leveraging an additional communication channel outside of BLE to facilitate authentication between the devices.

2.3.3 ZigBee Communication

ZigBee is a wireless communication technology designed to facilitate the transmission of small data for devices and applications with low power consumption needs within a distance range of 10 to 100 meters (Lata, 2022). The ZigBee protocol is built upon the IEEE 802.15.4 wireless standard, comprising two layers: the physical layer (PHY) and the medium access control layer (MAC). Based on these layers, it creates the network layer (NWK) and the application layer (APL). ZigBee technology operates on the 2.4 GHz frequency band, universally available. Moreover, it can utilize the 915 MHz frequency band in the Americas and the 868 MHz band in Europe. The speed on the 2.4 GHz band, spanning 10 channels, is 250 kbps, while on the 915 MHz band with 6 channels, it is 40 kbps, and on the 868 MHz band with a single channel, it is 20 kbps (Seker & Dalkilic, 2022).

With its low power consumption, compatibility with various network topologies (such as star, mesh, and cluster), low-latency data transmission, and data security (especially with AES), ZigBee has become increasingly sought after in IoT applications, particularly in sectors such as smart health, smart building, and smart energy.

One of the main motivations of the thesis is to emphasize the need for a gateway due to ZigBee's absence on smartphones, tablets, and computers, as well as its lack of utilization of IP addressing.

Although ZigBee security is not considered optimal, a variety of measures have been implemented to address potential vulnerabilities. In the study by Deniz, & Samet, 2019), various aspects such as encrypted communication, key sharing, and network joining vulnerability are examined. Additionally, a cloud-based solution approach is suggested, involving the sharing of a network key across all network components to facilitate encryption and decryption of communication data. Moreover, during network enrollment, the network key is transmitted encrypted alongside the link key. Two types of link keys exist: the global link key, which remains consistent across all nodes within the network. Nodes utilize either the key provided by the ZigBee Alliance to manufacturers or a key internally generated by the manufacturers. Through the ZigBee

Alliance, nodes from various manufacturers and versions can be integrated into the network. Utilizing a manufacturer-specific key allows only nodes manufactured by the same entity to join the network, with each node possessing a unique link key. In ZigBee version 3, network joining is facilitated through the Touchlink Commissioning method. The Touchlink Commissioning method facilitates the joining of nodes from various manufacturers and versions to the same network. In ZigBee 3.0 networks, wireless communication is encrypted using the network key. When a new node joins the network, the network key is transferred to it using the link key. During the network joining process, the network coordinator encrypts the network key using the link key and then transmits it securely to the new node. Since the new node possesses the identical link key, it decrypts the encrypted data to acquire the network key, facilitating its successful integration into the network. It is noteworthy that communication within ZigBee 3 networks is encrypted, ensuring secure data transmission. In the event that attackers intercept the "network join request" packet relayed by the coordinator and acquire the link key, they can extract the network key transmitted within the same packet. Subsequently, this enables them to assume control over the network. The encryption of data traffic using AES encryption, as supported by ZigBee communication, and the interception of traffic by devices joining the network through decryption are highlighted in the studies by (Rizzardi, Sicari, & Coen-Porisini, 2022; Zohourian et al., 2023). However, (Li, Zhang, Wu, Tang, & Xie, 2023) argues that encrypting network traffic using the default global link key poses security vulnerabilities in ZigBee devices during network joining.

CHAPTER THREE
DESIGN OF SECURE MULTI-PROTOCOL GATEWAY

This section covers our multi-protocol multi-gateway design, along with its assumptions and components. It provides a detailed overview of the interaction among IoT devices, servers, communication technologies, and communication protocol components within the design. And then, the ESP32 Wi-Fi PainlessMesh structure is introduced and compared with the existing network architecture. Following that, the configuration of XBee modules required for ZigBee communication is discussed, and BLE and ZigBee communication among ESP32 devices is addressed. Lastly, it offers a general perspective on security levels in line with application requirements and the capacity of components.

3.1 System Model, Assumptions, and Configurations

In this section, we introduce and justify the major design decisions of our proposed security protocol, as well as the involved parties and components. A system of interconnected devices implementing the proposed algorithm should inherently consist of a main gateway, an identity server, a multi-protocol edge IoT gateway (i.e., relay gateway), and various edge IoT devices (e.g., sensor nodes, smart home devices, etc.).

We can list the specific assumptions and limitations covered in our thesis as follows;

- The IoT devices have been positioned in an indoor environment and communicate with the local network.
- Data communication between edge devices is carried out via Wi-Fi, BLE, or ZigBee. The Wi-Fi mesh network is limited to four devices, while BLE and ZigBee facilitate peer-to-peer communication between two devices.
- The primary security focus revolves around authenticating devices upon network entry, with subsequent encryption of network data communication. Potential threats include attackers exploiting the Dolev-Yao model (Dolev & Yao, 1983).
- The security requirements and corresponding measures offered by the proposed protocol are categorized into four tiers, offering developers and operators flexibility to choose based on their application scenarios.

- It is assumed that both the devices and the network are consistently available without any flaws or interruptions.
- In the test environment, the Relay gateway is assumed to act as an MQTT client with only publishing privileges. Although it is possible to adapt it to have subscriber privileges, this aspect has been disregarded in the test environment.

3.2 System Components

This section provides a comprehensive overview of the primary components and methodologies employed in constructing a multi-protocol, multi-gateway system aimed at facilitating secure communication with authenticated and authorized lightweight IoT devices. In the subsequent subsections, we initially present the definition and conceptual design of each component, followed by discussions on implementation decisions, including the hardware selected to materialize the respective unit. The implementation decisions, such as selecting hardware, are not strict requirements, but rather our chosen approach to create a cost-effective and straightforward proof-of-concept.

3.2.1 Multi-Protocol Multi-Gateway Architecture

Scalability and applicability can be enhanced by designing a two-layered multi-protocol multi-gateway model instead of a single gateway device that integrates the functionalities of both the main gateway and relay gateway. In this approach, a relay gateway communicates with end nodes via three different protocols (BLE, ZigBee, Wi-Fi), while a main gateway manages high-level device authentication and facilitates the connection to the Internet.

The layered design enables the deployment of multiple relay gateways across different locations, enhancing scalability and reducing device complexity. The main gateway serves as an MQTT broker, while the relay gateway integrates three publisher/subscriber hybrid clients. Figure 3.1 depicts the architecture and network hierarchy.

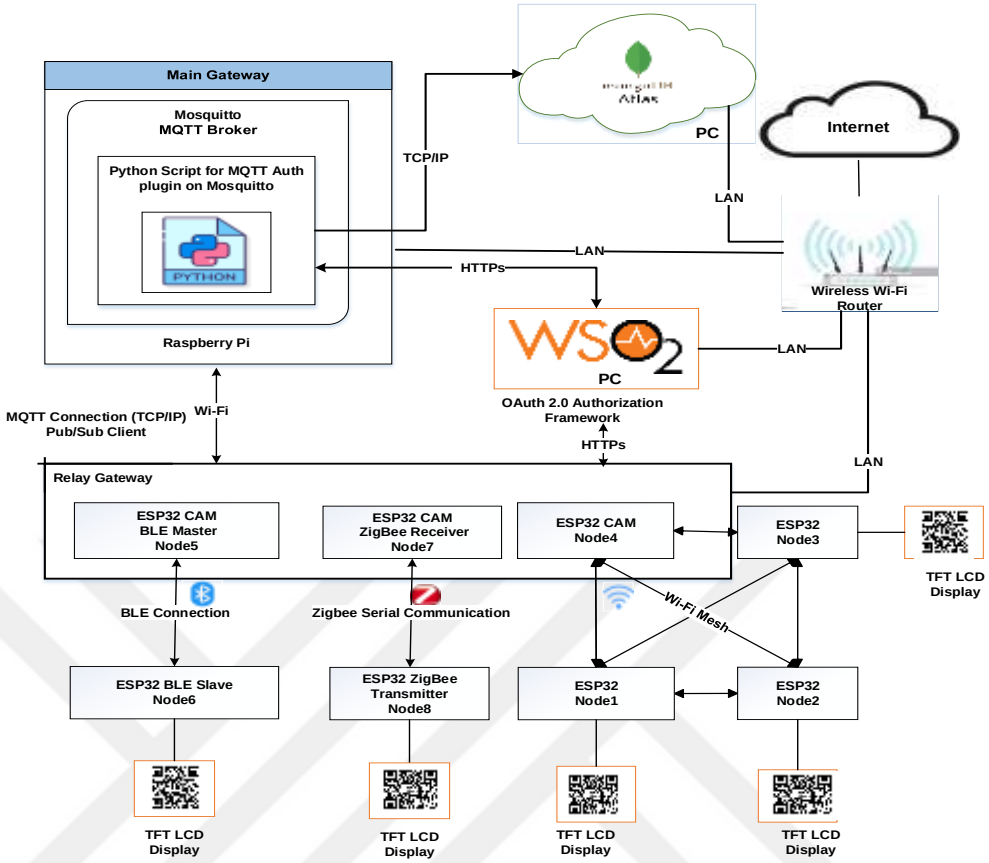


Figure 3.1 Multi-protocol gateway: prototype components, connections, and interactions

3.2.2 Main Gateway

The main gateway manages communication between the relay gateway and the entire IoT infrastructure, encompassing the identity server, log-tracking databases (such as MongoDB Atlas), and the Internet. It exclusively utilizes Wi-Fi for communication and operates the MQTT broker software to accept messages from MQTT clients, facilitated through the relay gateway. This component plays a critical role in the role-based authorization framework. We employed a Raspberry Pi 4 as the main gateway. Equipped with Wi-Fi and BLE 5.0 capabilities, it functions as a single-board computer with processor, memory, and I/O integrated onto a single circuit board (Ghael, Solanki, & Sahu, 2020). It also stands out with its H.265 decoding capability, 2 USB 2.0, 2 USB 3.0 outputs, 2 micro HDMI, 4k video, 1 mobile industry processor interface (MIPI) display serial interface (DSI) display port, 1 MIPI camera serial interface (CSI) camera port. On the Raspberry Pi, we operate Mosquitto version 2.0.9

as the broker software, supporting MQTT version 3.1. Mosquitto is tasked with forwarding messages from authenticated clients with appropriate access permissions to other clients based on the topics assigned to the devices. An identity server such as WSO2IS is required for registering end devices within applications, including the assignment of device ID, device secret, device scope, and device access token. The responsibility of Mosquitto includes interfacing with WSO2IS for device authentication and authorization. This involves verifying the device's topic and scope against the permissions granted when publish and subscribe messages are received.

3.2.3 Relay Gateway

The relay gateway, also referred to as the edge gateway, consolidates data generated by numerous IoT devices and facilitates connectivity to the broader network, including the WSO2 identity server. It serves as a relay network device positioned near the network edge, offering storage, processing, and caching services to minimize latency and relieve the main gateway during the authentication phase. It incorporates a network interface card (i.e., TCP/IP router modem) for IP connectivity and is equipped with integrated BLE, ZigBee, and Wi-Fi mesh capabilities, enabling it to adopt a multi-protocol connectivity approach.

Our implementation of the relay gateway involves three ESP32-CAM devices, each dedicated to one of the wireless communication protocols, utilizing BLE, ZigBee, and Wi-Fi transmitters respectively. This version of ESP32 modules integrates a camera module, which we utilized for scanning QR codes used in the distribution of authentication keys. The ESP32-CAM, as documented in reference (Esp32-s2 series datasheet, 2024), features a 32-bit dual-core microprocessor operating at a clock speed of 160 MHz. It is compatible with camera modules such as the OV2640, which we utilized, as well as the OV7670. The ESP32-CAM utilizes a microSD card for storing captured images. Additionally, it can be connected to a computer via an FTDI programmer for code loading purposes. In our secure multi-protocol design, it is exclusively used for verifying IoT devices through QR-based authentication methods.

The relay gateway lacks human intervention input/output functionalities. Therefore, employing passkeys for device authentication is not feasible. To fulfill this

requirement, we suggest implementing an application-layer security mechanism relying on key sharing and the utilization of a 6-digit HOTP value obtained by a camera from QR codes displayed on the screens of end devices. This presents a feasible alternative to conventional BLE security functionalities, given that our ESP32 devices are equipped with cameras. Additionally, it proves beneficial for ZigBee and Wi-Fi mesh protocols. This approach enables HOTP-based authentication without necessitating human interaction on the device. This authentication process guarantees that only trusted devices can connect to the network and interact with other nodes. However, the authentication process for both the node and the relay gateway node occurs on the main gateway side, thereby enhancing the network's security with an additional layer of verification. It should be noted that although each utilizes a distinct wireless technology for communicating with the end nodes, all three ESP32-CAM devices integrated into the relay gateway utilize Wi-Fi to establish a connection with the main gateway, as MQTT operates over TCP/IP.

To use AES encryption with MBED TLS in BLE, ZigBee, and Wi-Fi communication, both the relay gateway and end node devices, as well as the main gateway, must support AES encryption and have the same encryption key. Within the scope of our study, the exchange of encryption keys is securely facilitated using the QR code method with the utilization of ESP32-CAM modules. Alternatively, it can be accomplished through a secure key exchange protocol such as Diffie-Hellman or RSA, or by employing a pre-shared key securely stored on both devices. After securely exchanging the encryption key, the data is encrypted using AES-128 in cipher block chaining (CBC) mode. Subsequently, the encrypted data is transmitted via the relevant wireless communication technology (i.e., Wi-Fi, BLE and ZigBee). At the recipient end (main gateway), the encrypted data is decrypted using the identical encryption key through AES decryption in CBC mode. In summary, a viable approach to security and reliability within the IoT system has been proposed, which is essential for safeguarding the security and confidentiality of user data.

3.2.4 IoT Edge Devices

End devices is a broad term encompassing any sensor or actuator device wirelessly connected to the respective IoT network. In our configuration, it is presumed that they

have the capability to facilitate communication via one of the BLE, ZigBee, or Wi-Fi mesh protocols. However, they are not required to support TCP/IP communication or be connected to the Internet.

The ESP32, a series of development boards created and manufactured by Espressif Systems, stands as a prevalent choice for end devices in IoT applications. Its popularity stems from its affordability, low power consumption, and efficient communication capabilities using both Wi-Fi and BLE technologies. We have opted for the ESP32-S2 to establish the mesh network of end devices, primarily due to its ability to support infrastructure station, SoftAP, and promiscuous modes simultaneously. ESP32-S2 features a reduced duty cycle, awakening intermittently solely upon message arrival to minimize power usage. Hence, we opted for BLE and ZigBee communication. Moreover, ESP32-S2 accelerates cryptographic functions including AES, SHA-256, RSA, random number generator (RNG), HMAC, and digital signature through hardware support (Esp32-s2 series datasheet, 2024). All end devices in our configuration are equipped with a 3.5-inch thin film transistor (TFT) liquid crystal display (LCD) to display unique QR codes for authentication. This TFT LCD features a 16-bit interface and a 16-bit color display with a resolution of 480x320 pixels. It can be directly connected to an ESP32 IoT device.

The XBee PRO S2C, a data transmission module utilizing ZigBee, operates on the IEEE 802.15.4 network protocol. It is equipped with its own microcontroller. Hence, it includes pins dedicated to functions such as analog inputs, pulse-width modulation (PWM) outputs, and voltage reference (VREF). It facilitates either point-to-point or point-to-multi-point communication. Operating at an optimal frequency of 2.4 GHz, it offers an indoor coverage range of 90 meters and an outdoor coverage range of 3200 meters, with a RF data rate of 250 kbps (Pardede, Hutajulu, & Sirait, 2020). Two modules are set up using the XBee configuration and testing utility (XCTU) (Adi et al., 2021) software to communicate via the ZigBee protocol, and are paired with our two ESP32 devices, one serving as the transmitter and the other as the receiver within our relay gateway. Upon receiving data, the receiver serves as an MQTT client, publishing the data from the transmitter to the main gateway.

3.2.5 Server and Cloud Database

To set up a multi-protocol, multi-gateway network with improved security requires servers that store and verify configuration parameters, as well as store data for analysis. In our configuration, we utilize the OAuth 2.0 authorization protocol (Hardt, 2012) through the WSO2 identity server (WSO2IS) (Da Silva Torres, 2020). As a result, end devices such as ESP32 mesh nodes, ESP32 XBee receivers and transmitters, and ESP32 BLE slaves and masters are registered on WSO2IS to facilitate authentication and authorization processes by the main gateway. In our implementation, WSO2IS is responsible for interfacing with the Mosquitto broker on the gateway and the three ESP32 devices comprising the relay gateway. Relay gateways acquire access tokens from WSO2IS before publishing messages to Mosquitto on the gateway. Mosquitto validates the access token by invoking the introspection API, which initiates an HTTPS connection request to WSO2IS. Upon successful validation, WSO2IS confirms the token's validity by returning the device's ID and scope information. Moreover, a MongoDB instance is linked with Mosquitto on the main gateway to authenticate devices using HOTP as a passphrase in their connection messages. How this is facilitated in this context is clearly explained in chapter 4. This configuration guarantees that the main gateway verifies the authenticity of the end device. Furthermore, MongoDB stores incoming messages along with details of the communication technologies used, allowing for performance analysis related to payload growth and the effectiveness of the security measures implemented. The system hosting WSO2IS version 5.11.0 and MongoDB Atlas version 5.0.7 is a 64-bit Windows 10 personal computer equipped with 16 GB of RAM and an Intel i7 processor.

3.3 Configuration of Wireless Communication Technologies

Setting up a Wi-Fi mesh network involves configuring the ESP32 devices to communicate seamlessly, ensuring robust messaging and implementing security measures to protect the network. For BLE communication, the ESP32 devices need to be configured to establish reliable Bluetooth connections, enabling efficient data exchange. The configuration of ZigBee communication requires the use of XBee modules, which are set up using the XCTU program to ensure proper network

functionality. This includes setting up the necessary parameters for ZigBee communication and establishing the network for reliable data transfer among the devices.

3.3.1 To set up ESP32 Wi-Fi PainlessMesh Network

Utilizing the PainlessMesh library within the Arduino Studio IDE, data collected from four ESP32 devices, all configured with the same mesh SSID, mesh password, and mesh port, is exchanged over a shared wireless local area network. The Espressif mesh network structure, which utilizes the simultaneous AP-STA feature in Espressif SoCs, provides wireless communication when WiFi is not available (ESP32-S2 ESP-IDF). It allows up to 1000 connected devices. When the number of devices was increased to 4, 5, and 8 and tested (Şeker et al., 2022), there was a decrease in packet transmission rate due to which our mesh structure in our multi-gateway multi-protocol design was limited to 4 devices. These nodes operate without a TCP/IP connection. To enable the functionality provided by this library, dependencies such as ArduinoJson (by bblanchon), TaskScheduler, and AsyncTCP (ESP32) are required.

One of the devices is defined as the root node (NodeWRG), and every 5 seconds, a scan is conducted to check if there are any devices willing to join the network within the coverage area. In case of a potential disruption in the network, if the root node drops from the network, data transmission from the mesh network to the main gateway is halted, and the developer redefines the root node. Due to the addition of a security mechanism, where the roles of the root (NodeWRG) and childrens (called as NodeW1, NodeW2 and NodeW3) differ, options such as allowing the mesh structure to reorganize itself based on RSSI value for root selection have been disabled.

PainlessMesh is a library compatible with ESP8266 or ESP32 boards that provides a real ad-hoc network. Unlike traditional Wi-Fi networks, nodes can communicate directly with neighboring nodes without the need for a central access point. In our extended PainlessMesh setup, nodes are responsible for relaying each other's transmissions, eliminating the requirement to connect to a central node. This prevents the limitation of the maximum number of nodes by a central node's capacity. Our

design consists of two layers, with each authenticated device placed in the second layer. The network structure is illustrated in Figure 3.2.

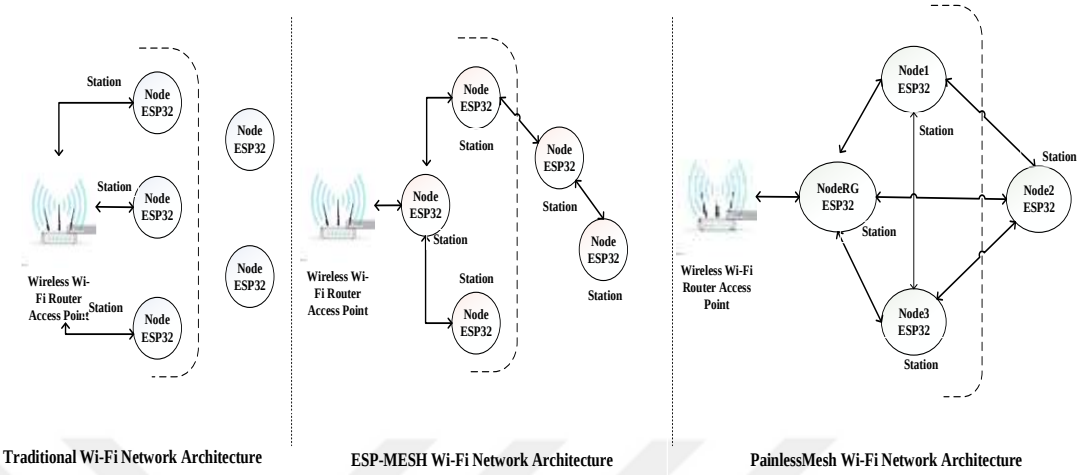


Figure 3.2 Compare PainlessMesh Wi-Fi network structure with traditional and ESP-Mesh network architectures

The setup, message transmission, and security of the mesh structure are detailed in the diagram as shown in Figure 3.3. And also, it is explained on the following pages.

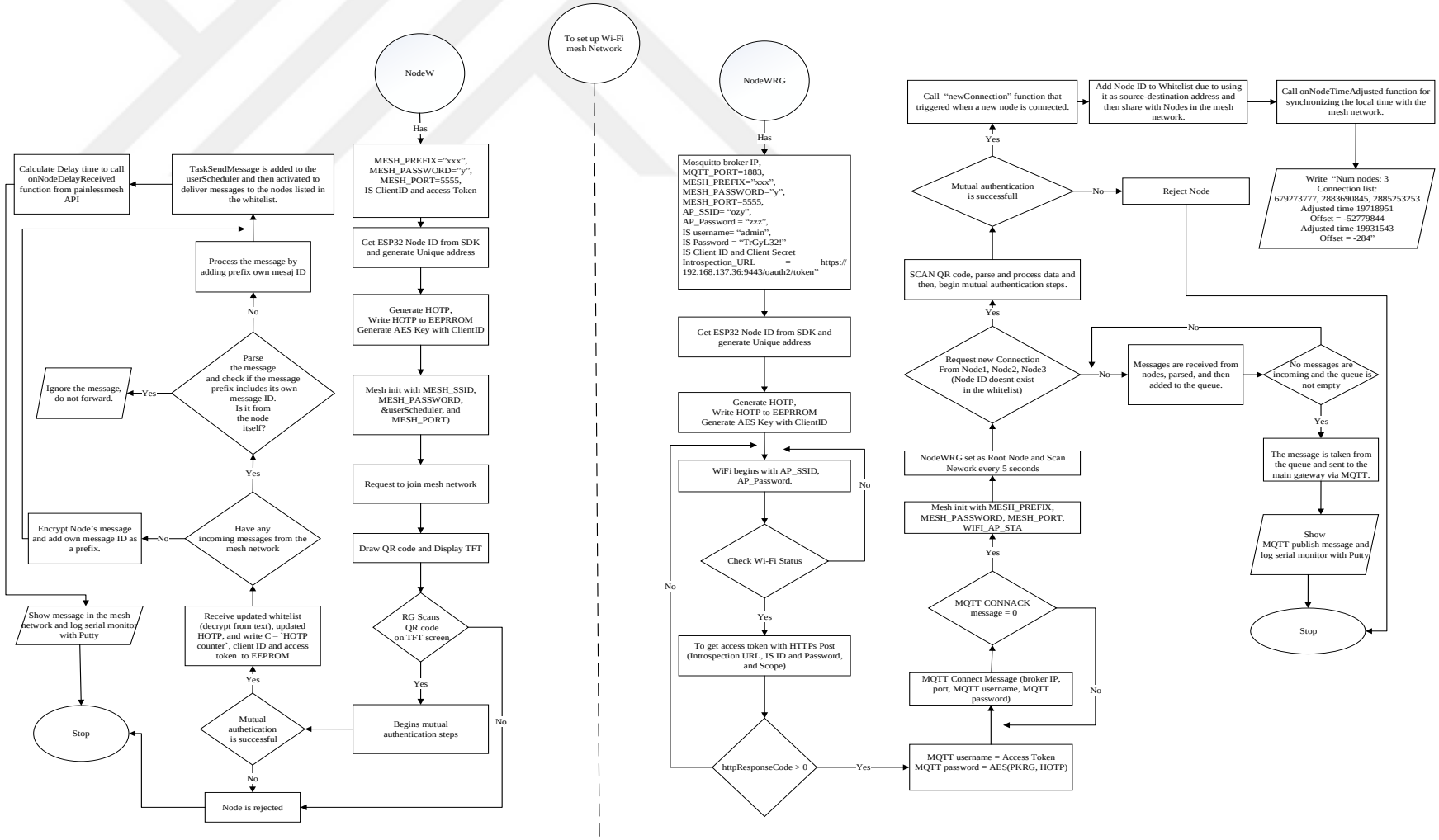


Figure 3.3 The setup, message transmission, and security of the Wi-Fi mesh structure

Below is an explanation of how to initiate wireless communication between ESP32 devices using the PainlessMesh network.

NodeWRG acts as the root device, while the others are configured as subNodes (Node1, Node2, Node3). Only NodeWRG establishes the MQTT connection. NodeWRG is responsible for forwarding messages from other Nodes connected to the mesh network to the main gateway. SubNodes, on the other hand, are responsible for forwarding their own messages and messages from other nodes to the RG.

In case of a potential disruption in the network, if the root node drops from the network, data transmission from the mesh network to the main gateway is halted, and the developer redefines the root node. Due to the addition of a security mechanism, where the roles of the root (NodeWRG) and subNodes (called as NodeW1, NodeW2 and NodeW3) differ, options such as allowing the mesh structure to reorganize itself based on RSSI value for root selection have been disabled.

Firstly, NodeWRG connects to wireless Wi-Fi router (modem) with access point SSID and password and then, NodeWRG establishes an HTTPS connection to the authorization server and obtains an access token. Then, it establishes an MQTT connection to main gateway. The mesh network is not initiated before the MQTT connection is established.

In our multi-protocol multi gateway design, a mesh network set up among four ESP32 devices. They need to have the same "mesh SSID," "mesh password," and "mesh port" parameters to connect to the same network. The 32-bit ESP32 chip ID is used to assign a unique NodeID to each device. The ESP32 chip ID is utilized for the destination and source addresses of the Nodes, and it is called from the Espressif SDK.

When a new Node connects, the taskSendMessage is disabled, and the onNewConnection function is triggered on all devices. Then, to ensure synchronization between the Nodes in mesh network, the onNodeTimeAdjusted function is called, and the local time and mesh network time are adjusted with offset parameter, which represents the calculated adjustment difference applied to the local clock. The mesh network checks every 3 seconds whether the Node is currently connected to the network or if there is any potential issue, such as internet

disconnection. If there is a change in the number of nodes in the network, the onChangedConnections function is triggered. This function is also called when a node is added to the network. When an old connection is lost, it is reconnected to the network upon reestablishment. However, a new connection undergoes an authentication steps. An example log for connection establishment and time adjustment is provided below in Figure 3.4.

```
Connecting to WiFi..
Connected to the WiFi network
Get Token Values:
D959dc49-d4c3-3711-b2c8-481cf121f44d
My IP is 192.168.1.37
Attempting MQTT connection…
CONNECTION: eventSTAGotIPHandler: ARDUINO_EVENT_WIFI_STA_GOT_IP
MQTT Connected Time = 1332 us
setLogLevel: ERROR|START UP| CONNECTION |
STARTUP: init (): 0
STARTUP: init (): AP tcp server established on port 5555
After mqtt connection: 216664
CONNECTION: stationScan(): "ozy"
CONNECTION: eventScanDoneHandler: ARDUINO_EVENT_WIFI_SCAN_DONE
CONNECTION: scanComplete(): Scan finished
CONNECTION: scanComplete(): - -> Cleared old Aps.
CONNECTION: scanComplete(): num=9
CONNECTION:    found: "ozy", -41 dBm
CONNECTION:    found 1 nodes
CONNECTION: connectToAP(): Best AP is 3802943388
CONNECTION: connectToAP(): Trying to connect, scan rate set to 4*normal
CONNECTION: New AP connection incoming
CONNECTION: Painlessmesh::Connection: New connection established.
CONNECTION: newConnectionTask();
CONNECTION: newConnectionTask(): adding 679273777 now= 72443578
- ->Start: New Connection, nodeId= 679273777
- ->Start: New Connection, {
     "nodeId": 3175689321,
     "root": true,
     "subs": [
     {
     "nodeId": 679273777,
     "subs":[
     {
     "nodeId": 2883690845
     },
     {
     "nodeId": 2885253253
     }
    ]
   }
  }
}
Changed connections
Num nodes:3
Connection list: 679273777 2883690845 2885253253
Adjusted time 19718951 Offset = -52779844
Adjusted time 19931543 Offset= -284
```

Figure 3.4 Log files of setting up and new connection of PainlessMesh network

In the continuation of the page, the message transmission process in the PainlessMesh network is addressed.

A limitation of the library is that while nodes in the mesh network can receive messages, each device only broadcasts its own data and does not forward data from other nodes to the NodeWRG. To address this issue, Node type information has been added as a prefix to messages. Additionally, a Node ID whitelist of authenticated devices verified by NodeWRG is included, and the whitelist is encrypted and transmitted to authenticated Nodes in the mesh network. During message transmission, the onReceive, sendSingle, or sendBroadcast functions are triggered. When a message is received, the parameters include the sender's message and sender Node ID. When a message is sent, it contains the Node ID of the authenticated node found in the whitelist, along with either the node's own message or messages from other nodes in the mesh network.

For each node, the prefix consists of 5 bytes: 1 byte for the prefix information, 1 byte for the node type (indicating whether it's Node1, Node2, or Node3), and the remaining 3 bytes for the Message ID information. When a message is received, its prefix is checked. If the node type matches its own, the message is ignored, preventing it from being sent back to the mesh network and causing an infinite loop. In the case of belonging to a different node, the message is forwarded to the authenticated nodes found in the whitelist by triggering the SentSingle function, with its own prefix information added.

Calling the sendSingle function ensures the addition of the TaskSendMessage task to the userScheduler and then enables this task. Consequently, the task of sending messages is scheduled and executed at a random time between 1 second and 5 seconds.

The process of sending a 16-byte message from Node1 to Node2, Node3, and NodeWRG in the mesh network is detailed in Figure 3.5.

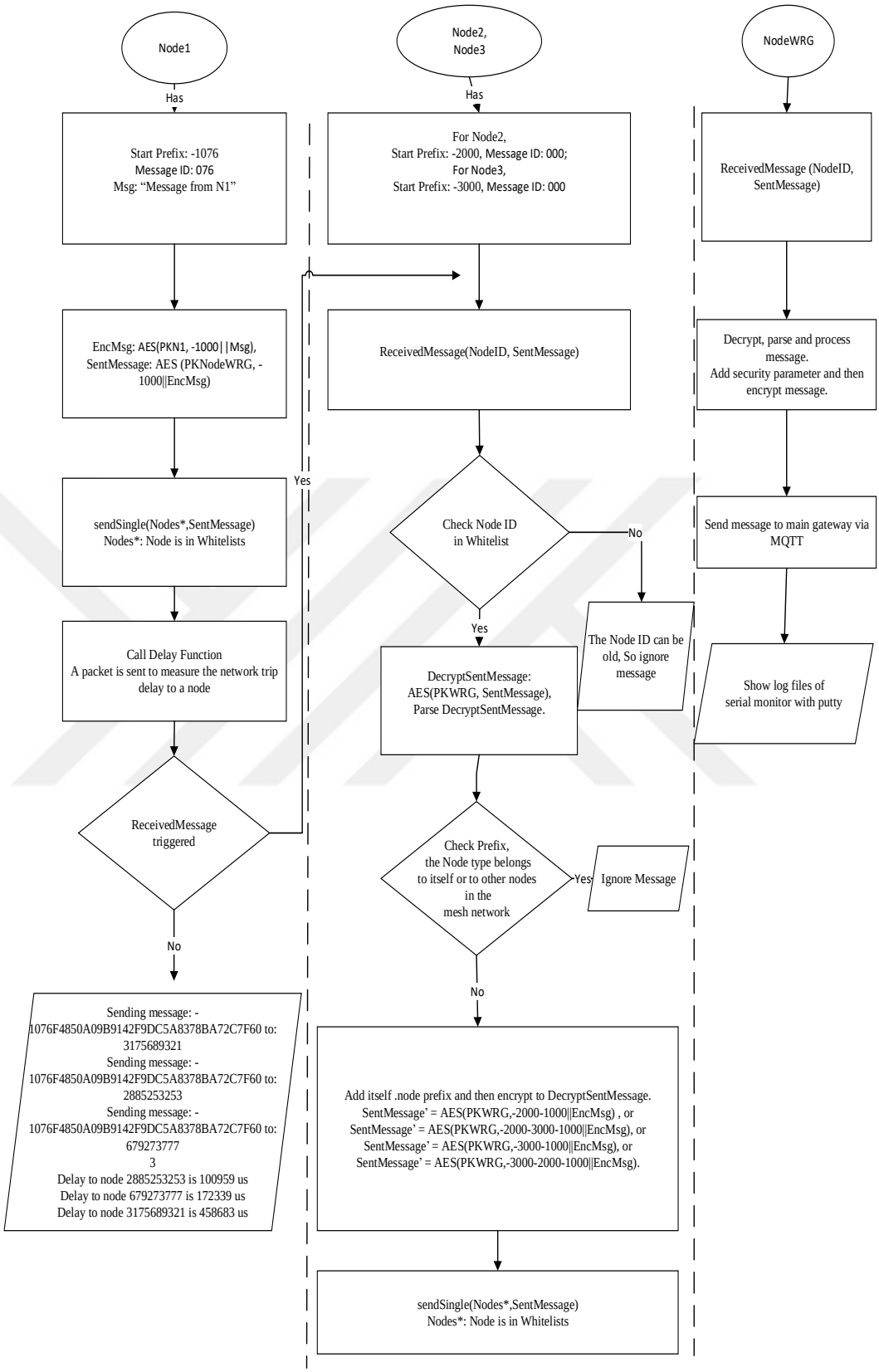


Figure 3.5 The process of sending a 16-byte message from Node1 to Node2, Node3, and NodeWRG in the mesh network

The message from Node1 reaches Node2 through Node1 and Node3, and vice versa. Upon receiving the messages, they are forwarded to other nodes in the mesh network with their own prefixes added. Similarly, Node3 receives the message from Node1 through Node1 and Node2, and it is then forwarded with its own prefixes added. Node1, however, ignores the message with its own prefix and does not redirect it back to the mesh network. As a result, NodeWRG receives the message from Node1 five times in the mesh network. A message ID is added to each message sent. Section 5 reports on how message reliability is affected by message sending frequency, transmission intervals, and payload growth. If we were to provide an example, the process of forwarding the message "*1005E0C74D8F4EFAB84B9C2B5A8BC9E33147*" sent by Node1, with the Node destination addresses being "*2885253253*", "*679273777*", and "*2883690845*", which are the Node IDs whitelisted, to the RG via the mesh network is as follows.

"Received from 2885253253 enc msg=-1005E0C74D8F4EFAB84B9C2B5A8BC9E33147 PubencTime=4028 us Free heap: 209644 bytes.

Received from 679273777 enc msg=-2007-1005E0C74D8F4EFAB84B9C2B5A8BC9E33147 PubencTime=4187 us Free heap: 209720 bytes.

Received from 2883690845 enc msg=-3006-2007-1005E0C74D8F4EFAB84B9C2B5A8BC9E33147 PubencTime=5016 us Free heap: 209704 bytes.

Received from 2883690845 enc msg=-3006-1005E0C74D8F4EFAB84B9C2B5A8BC9E33147 PubencTime=4191 us Free heap: 209540 bytes.

Received from 679273777 enc msg=-2007-3006-1005E0C74D8F4EFAB84B9C2B5A8BC9E33147 PubencTime=5044 us Free heap: 209492 bytes"

Lastly, the security of the PainlessMesh network is discussed. The PainlessMesh structure does not include any customized or universal security mechanism. PainlessMesh structure does not include any customized or universal security mechanism. In our scope of work, the authentication of Nodes joining the mesh network is done based on QR codes and HOTP. Further details will be provided in later stages with full elaboration. Node IDs authenticated by NodeRG are added to a whitelist, ensuring that message transfer within the network is only allowed between Nodes contained in the whitelist.

Each message contains a unique message ID, helping to prevent message duplication and ensuring message reliability. Message transmission between nodes in the mesh network occurs through AES encryption. Sending plain text is avoided.

This setup ensures efficient message transmission within the mesh network while maintaining security and preventing issues such as message duplication and infinite message loops.

3.3.2 To set up BLE Communication

Our multi-protocol design covers communication technologies such as BLE, with two types of ESP32 devices: NodeB (slave) and NodeBRG (master). Generally, NodeB shares the data it possesses (such as sensor data), while NodeBRG is the device that wishes to receive this data and transmit it to the main gateway. Therefore, to share its data, NodeB can "advertise" its presence to NodeBRG. NodeBRG can scan the area and attempt to establish a connection with NodeB to receive data. However, before establishing a connection, NodeBRG must initiate an MQTT connection with the main gateway. The main gateway authenticates NodeBRG using OAuth 2.0 token-based authentication. Consequently, NodeBRG sends an HTTPS post to request an access token from the IS and then forwards the MQTT CONNECT message to the main gateway using this access token. If the MQTT CONNACK message returns successfully, a BLE connection is established with NodeB. Once connected, the authentication phase begins, and NodeB and NodeBRG authenticate each other using QR code-based and HOTP methods. Once the authenticated connection is established, NodeB starts updating NodeBRG immediately via notifications when sending data. Notifications from NodeB indicate that data can be continuously transmitted to NodeBRG without expecting a response or acknowledgment.

In the overview of BLE that we introduced in Chapter 2, we discussed the GATT hierarchy, which defines a layered structure enabling devices to discover each other, perform read-write operations, and share data. The profile found in the top layer encompasses a collection of services previously compiled by the Bluetooth SIG. Services, which can contain various characteristics, are identified uniquely using a 128-bit UUID, facilitating the logical segmentation of data. At the lowest level of the

GATT structure, the characteristic encompasses the data array of NodeB in the "value" field. characteristics identify themselves using either 16-bit or 128-bit UUIDs. Within the scope of the thesis, the characteristic value is the primary area where NodeB and NodeBRG interact; it is writable and readable. The value, within the thesis scope, is a string text data type, and while the GATT structure has a maximum transmission unit (MTU) defined payload of 23 bytes, it has been expanded to 517 bytes by NodeBRG which is observed in the Figure 3.6.

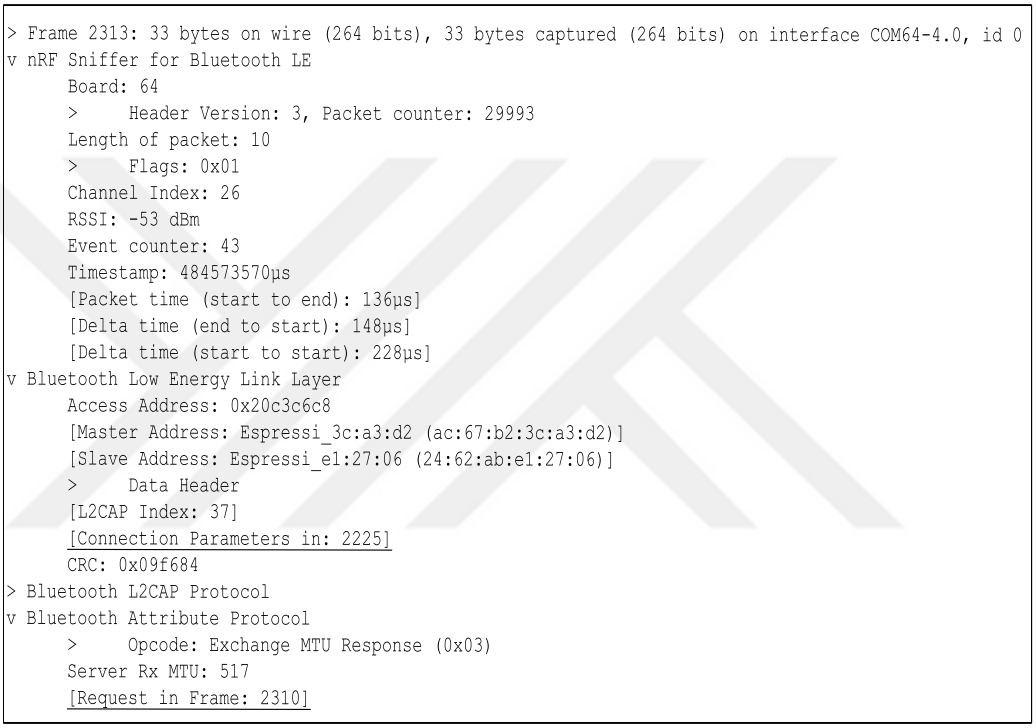


Figure 3.6 Set maximum transmission unit as 517 bytes

In the Figure 3.7, the NodeB software flow diagram covers the transmission of a 32-byte BLE data packet, starting from the boot-up process. If data is collected from the sensor, it is stored in a buffer. The process begins by defining custom Service and Characteristic UUID values, registering with the IS by specifying the Client ID, and generating HOTP. Then, NodeB is initialized as a BLE slave, Service and Characteristic properties are created, and it starts advertising its presence. It continues advertising until a connection is established with NodeBRG. Upon successful connection establishment, mutual authentication begins. NodeB generates a QR Code and displays it on the TFT screen to verify its presence. Once mutual authentication is

successful, it checks if there is data to be transmitted in the buffer. If data exists, it is encrypted, then the Characteristic value is set, and a BLE notification is sent. The process ends with handling file system and BLE hardware changes. In case of a possible network issue causing disconnection, when reconnected, it rejoins the network as it was before.

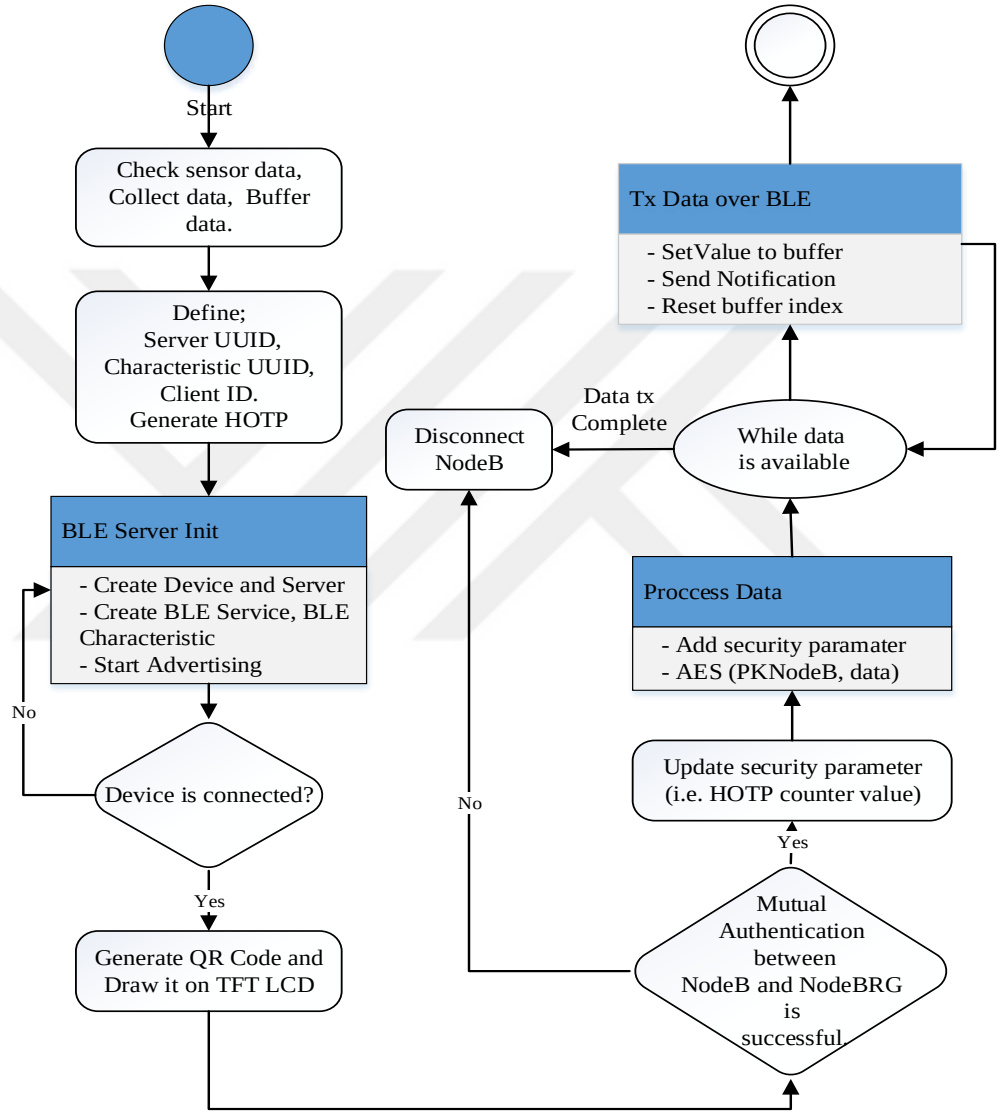
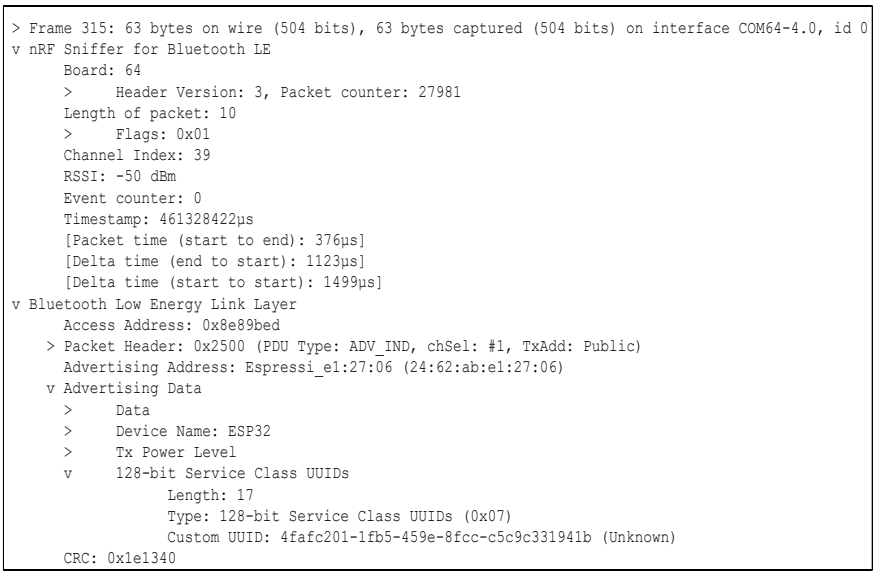


Figure 3.7 Data transmission from NodeB over BLE

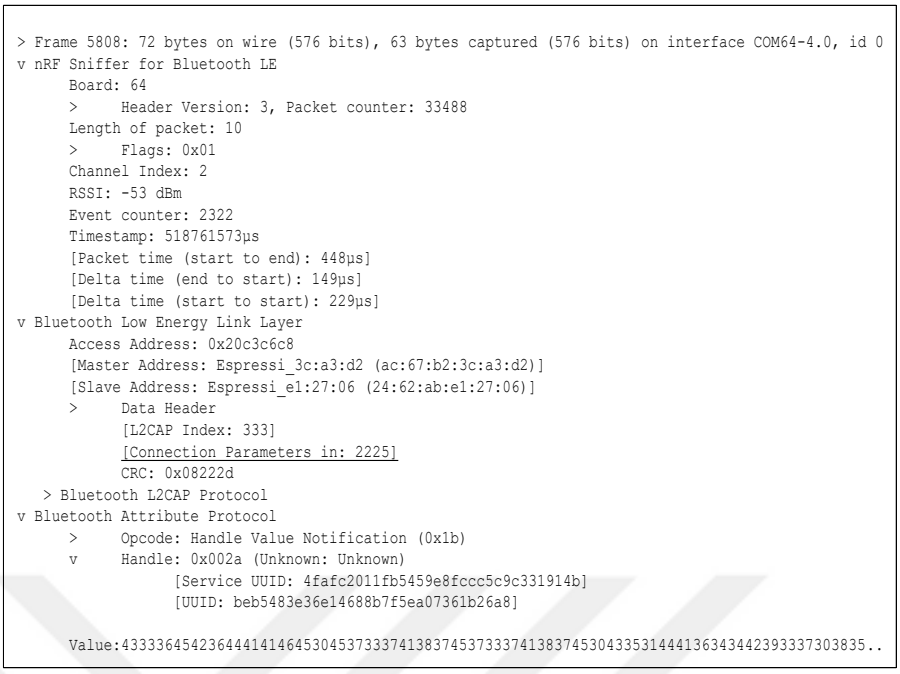
In our scope of thesis, we utilized a custom UUID generator. We used the UUIDs *"4fafc201-1fb5-459e-8fcc-c5c9c331914b, beb5483e-36e1-4688-b7f5-ea07361b26a8,* and *1c95d5e3-d8f7-413a-bf3d-7a2e5d7be87e"* as the service, characteristic1, and

characteristic2, respectively. Subsequently, the BLE device instance is created with the name "ESP32", which will be visible to other NodeBRGs during scanning.

Afterwards, the BLE server, Service, and Characteristic enabled with READ, WRITE, and NOTIFY properties are created. By employing the “getAdvertising” function (which is included in this source code), we append the service UUID to the advertising object, facilitating recognition of the Node B by the NodeBRG. For advertising, only channels 37, 38, and 39, i.e., 3 channels out of 0-39, are utilized, while data transmission uses the remaining channels. Subsequently, we initiate wireless advertising to enable its connection. The connection establishment process with the NodeBRG and data reception were monitored using Wireshark integrated with Nordic nRF. As seen in the Figure 3.8 (a), when NodeB advertises, with the Nordic nRF 52840 Wireshark integration, the name of the instantaneously listened NodeB is ESP32, it uses channel 39, and the Service UUID is *"4fafc201-1fb5-459e-8fcc-c5c9c331914b"*. The Figure 3.8 (b) displays real-time Service and Characteristic UUID values along with the current Characteristic value using nRF. After the connection is established, the mutual authentication between NodeB and NodeBRG is detailed in the following pages. The data is processed and transmitted over BLE after being encrypted with AES. This process continues until the queue is empty. Chapter 5 covers the increase in the “setValue” and “send Notification” process time with different data size.



(a) NodeB advertises



(b) Received handle value notification

Figure 3.8 Packet captured by Wireshark integration with Nordic nRF 52840 when NodeB advertises (a) and received handle value notification (b)

First, before establishing BLE communication, the NodeBRG attempts to establish an MQTT connection with the main gateway. For this purpose, it connects to the IS and obtains an access token via HTTPS. Since the usage of parameters generated during the authentication phase (such as AES key, HOTP, etc.) and authentication steps are specified in the following pages, this section and Figure 3.9 focuses only on establishing the connection and receiving messages.

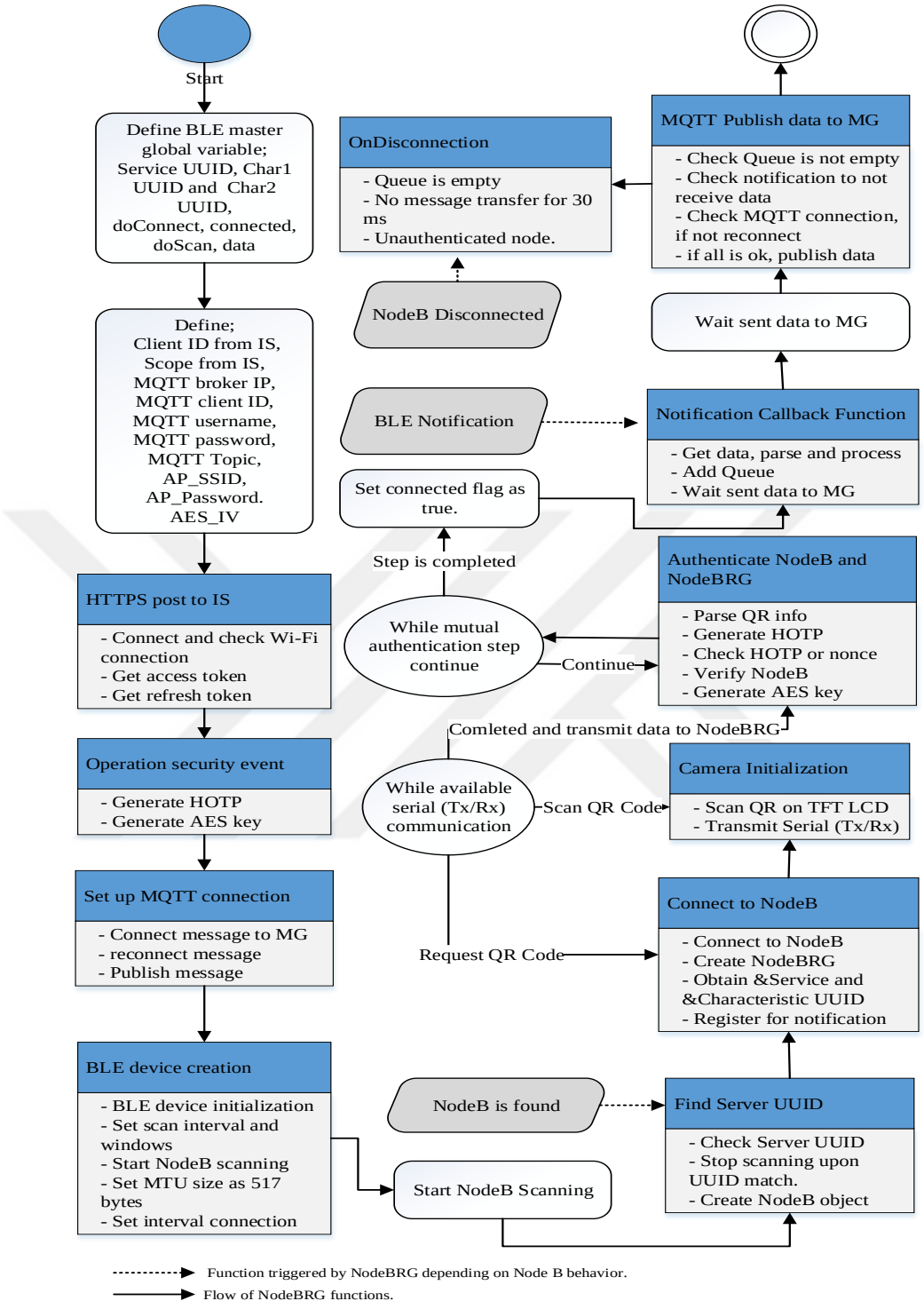


Figure 3.9 Creation of BLE master (NodeBRG), initiation of communication with NodeB, and transfer of NodeB data to the MG

After successful connection to the main gateway via HTTPS and MQTT, the NodeBRG initiates BLE communication between NodeBRG and NodeB. For this,

initially, the NodeBRG attempts to connect to the Service UUID, Characteristic 1 UUID, and Characteristic 2 UUID belonging to NodeB. “doConnect”, “connected”, and “doScan” flags respectively determine whether to connect to the NodeB device, whether NodeB and NodeBRG are connected to each other, and whether NodeBRG will perform device scanning. Data will hold the Characteristic value as a string type. It is the value to be transmitted to the MG via MQTT after necessary parsing, preprocessing, and encryption with AES. As shown in Figure 3.9, to initiate BLE communication, the device is initialized similarly to NodeB (BLE slave). Unlike NodeB, there is no need to specify the device name. When the device is detected, we expect to be notified by the callback function as followed in the source code found in the GitHub repository (ref). NodeBRG, every 5 seconds, performs a scanning process over channels 37, 38, and 39. When it finds each BLE device, the “MyAdvertisedDeviceCallbacks” function is triggered, and then it creates an object for the found device, checking if it matches with the “Service UUID”. NodeBRG then connects to Node B and obtains references to the Service, Characteristic1 and Characteristic2 UUIDs. It defines a “notifyCallback” function to be notified when there is new data in NodeB's characteristic. In the absence of enhanced security, the connection flag is updated to true at this stage. However, in our thesis scope, this step involves mutual authentication, and NodeBRG initializes the camera module to scan the QR Code on NodeB's TFT LCD screen. The camera module transmits the serially read code to NodeBRG. This process is carried out by checking for data every 3 ms. After a successful scan of the QR Code, mutual authentication is expected to occur following parsing, computation-checking, processing, and AES encryption. Upon successful completion, the connection flag is now set to true. Each incoming message is parsed, processed, and enqueued, and then, when no messages are incoming, messages are dequeued and checked for MQTT connections to publish to the MG. If there is incoming data, data acquisition occurs first to avoid data loss and delay. If the connection to NodeB is lost, and at the same time, the Queue is empty and no new data is triggered for 30 ms, if no response is received within 30 ms while the authentication steps are taking place, the “ondisconnection” function is triggered.

3.3.3 ZigBee Configuration with XCTU

Our multi-protocol multi-gateway design supports peer-to-peer ZigBee wireless communication, facilitating the remote monitoring and control of data collected through ZigBee communication by devices that do not have ZigBee support, via the main gateway. The ZigBee communication protocol consists of three components: coordinator, router, and end device. In our multi-protocol multi-gateway design, NodeZRG acts as the coordinator. It is responsible for initiating the network, assigning a network identifier (PAN ID) to the end device NodeZ, and forwarding the data received from the end device to the main gateway. Since communication is only established between two devices, there is no router responsible for routing between nodes. NodeZ (end device) is responsible for transmitting its data to the coordinator. To enable ZigBee communication, our Xbee modules have been configured as coordinator and end device using the XCTU program, as depicted in Figure 3.10.

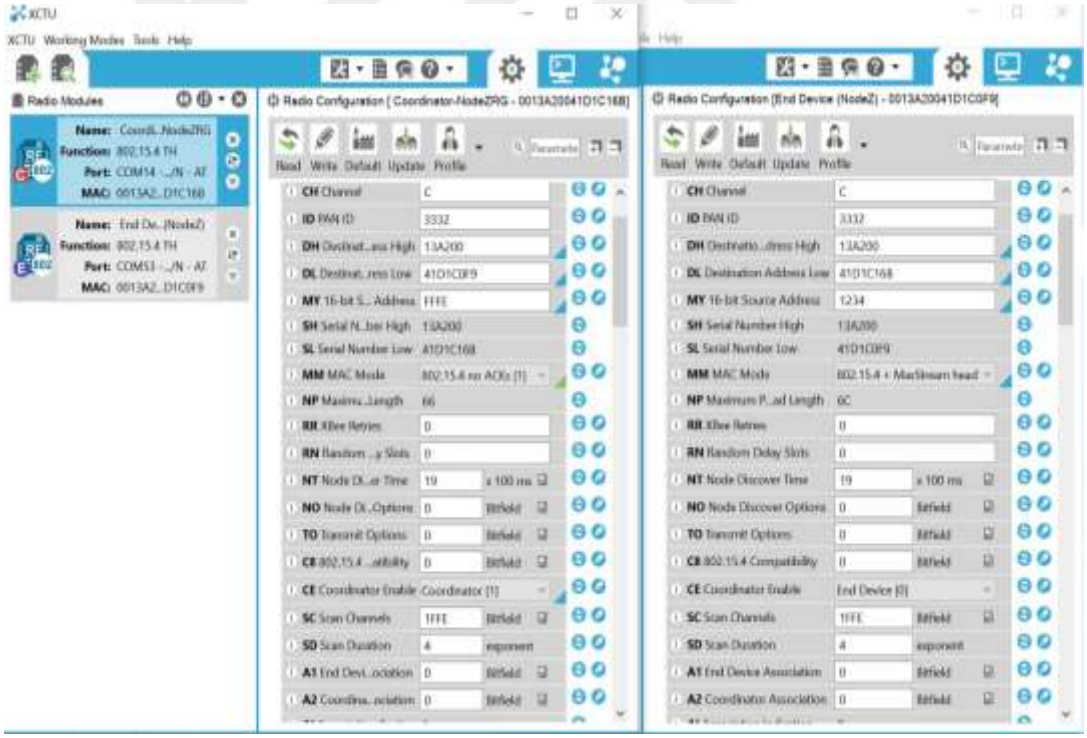


Figure 3.10 Configuration of XBEE S2C pro modules with XCTU program

- The XBee S2C modules used in the SIMMS design were configured using the XBee Explorer USB adapter produced by Sparkfun. The XBee USB

adapter board serves as a USB to Serial TTL converter. It acts as an intermediate connection device between the XBee modules and the computer, enabling programming of the XBee modules. The board features pins where XBee modules can be attached. This means that the TX, RX, +5V, and GND (ground) pins on the module are connected to the TX-IN, RX-OUT, VCC, and GND pins, respectively, on the XBee USB adapter board.

- Once the XBee Explorer USB board is connected using a mini USB cable, it will create a virtual COM port. Programming of the XBee modules is possible through this COM port using the X-CTU program.

- The Coordinator (NodeZRG) determines the communication channel (as "C" in the configuration file in Figure 3.10) and the PAN ID value (which is 3332). PAN ID defines the network and allows devices to join the network when they have the same PAN ID. Once NodeZ joins the network with a matching PAN ID to NodeZRG, mutual authentication steps begin. Upon successful authentication, devices between NodeZRG and NodeZ can exchange data.

- NodeZRG has been configured to communicate with its target device (NodeZ) by entering the address "13A200" into the "Destination Address High" field and "41D1C0F9" into the "Low" field. Additionally, the optional node name "Coordinator-NodeZRG" has been entered into the "Node Identifier" field. Since NodeZ only communicates with the NodeZRG device, the serial number of the coordinator module has been entered into the "Destination Address High" field as "13A200," corresponding to the "SH Serial High" part of the coordinator module's serial number. Similarly, the "Destination Address Low" field has been populated with the "41D1C168" address, corresponding to the "SH Serial Low" part of the coordinator module's serial number. Optionally, the node name "End device (NodeZ)" has been entered into the "Node Identifier" field. Devices can join the network by scanning and detecting each other at regular intervals. The "Node Discover Time" setting has been configured to

2500 ms for the scanning and detection process between NodeZRG and NodeZ.

- For ESP32 to communicate with the XBee module, the XBee module has been configured to communicate over the ESP32's universal asynchronous receiver-transmitter (UART) protocol at a baud rate of 9600, with no parity, and 1 stop bit. By connecting the appropriate GPIO pin (16 and 17 for Rx/Tx) of the ESP32 to the UART pins and utilizing the Arduino IDE development tool, communication between ESP32 and the XBee module has been facilitated.

QR code and HOTP-based authentication phase has been added to the application layer, similar to BLE communication, and implemented with the initiation of the protocol. The BLE event flow has been modularly designed to adapt to ZigBee communication as well.

3.4 Secure Tiers

The suggested system architecture integrates modularly designed security features, considering the resource limitations of devices and the precise security demands of the IoT application. The security measures are organized into tiers to accommodate the diverse protection requirements.

Tier 1: Physical Tracking. Basic monitoring of the device's physical location without involving device authentication. Data is transmitted in plain text.

Tier 2: Enhanced Authentication and Access Control. Authentication is strengthened by transmitting HOTP through displaying (by end nodes) and scanning (by relay gateway) predefined QR codes. Access control is enforced using the OAuth 2.0 authorization protocol and WSO2IS.

Tier 3: Data Security with AES Encryption. Beyond the functionalities provided in Tier 2, Tier 3 introduces persistent data encryption through AES.

Tier 4: Digital Signing for Enhanced Data Integrity. At the highest tier, diversification includes digital signatures in addition to AES encryption. This

guarantees not only data confidentiality but also confirms the data's origin, thereby preventing repudiation.

Within the scope of this study, security implementations have been extensively applied in detail, following a gradual approach tailored to the evolving requirements and constraints of IoT devices, encompassing tiers 1, 2, 3, and 4. Embedding secret (e.g., encryption) keys into hardware could enhance security; however, it is crucial to consider the risks associated with such interventions and potential physical damage or failures. Utilizing alternative channels like QR codes to transmit necessary parameters serves as a viable means to mitigate these risks.

CHAPTER FOUR
PROTOTYPING AND VALIDATION OF MULTI-PROTOCOL GATEWAY IN THE SECURE TIERS

This section comprehensively addresses the detailing of security levels and the design and implementation of a multi-protocol gateway.

Security tier 1 discusses the application, test environment, and performance results for Wi-Fi and BLE wireless communication, as the thesis work was conducted in a closed environment. For ZigBee communication technology, RSSI value will indicate tampering with the device when it is taken outside the closed environment due to significant signal loss.

Security tiers 2 and 3 have been implemented together, and steps for authentication, access permission, and confidentiality have been developed for the gateway that supports Wi-Fi mesh, BLE, and ZigBee wireless communications simultaneously. The test environment and performance results for security tiers 2 and 3 are detailed in Chapter 5 due to their extensive scope.

Finally, security tier 4, which includes message integrity and offers more comprehensive security, presents the application environment of the ECDSA algorithm, the processing times of the required cryptographic methods with BLE communication. Security tier 4 is modularly designed, allowing it to be directly integrated into Wi-Fi and ZigBee communication. It has not been repeated as it is implemented with ESP32 devices having the same features.

4.1 An Overview of Secure Tier 1: Physical Tracking

The increase in the number of devices with Internet access and the diversity of areas they are used in have made ensuring the security of transmitted data essential. The security requirements of each IoT application may vary. While some applications may contain sensitive personal or commercial information, in others, only the presence of the device may be important. Unique values embedded in hardware can be used as keys or secret values within cryptographic algorithms to ensure the confidentiality and

integrity of data. In this case, ensuring the security of the IoT devices used emerges as an equally important factor as the confidentiality of the data.

Received signal strength indicator (RSSI) is a metric that measures the received signal strength of a wireless communication connection. The RSSI value is typically expressed in dBm (decibel milliwatts) and indicates the signal strength from one device to another. However, the RSSI value does not directly represent a distance because the received signal strength can be influenced by environmental conditions, obstacles, and other factors.

There are different methods for converting RSSI to distance, and these methods generally vary depending on the specific wireless communication technology and equipment used. However, a commonly used method is based on a logarithmic approach and relies on the path loss in free space.

According to this approach, there is not a linear relationship between RSSI and distance, but an approximate estimation can be made. Field tests or calibration may be required for this estimation. The relationship between RSSI values and distance is usually determined by experimentation in a specific application, and this relationship is typically not expressed with a single standard formula.

Nevertheless, there are several mathematical models and algorithms commonly used to convert RSSI values to distance. One of these is the Friis Free Space Path Loss (FSPL) model, which is a logarithmic model that is based on the signal attenuation in free space and allows for an estimation to be made between RSSI value and distance (Carpi et al., 2023).

In this section, distance information was calculated based on the RSSI data obtained from an ESP32 IoT device located indoors using the FSPL model. Distance estimation based on RSSI information was used to observe the error rate in determining the current distance between devices via Wi-Fi communication. The legitimacy of the data-transmitting device was tested by employing indoor positioning through distance calculation using RSSI.

When considering the physical tracking of devices, determining the distance between devices or the distance to the wireless access point provides valuable location information. In the scope of thesis and benchmark tests, the location of ESP32 IoT devices, is determined using indoor positioning system (IPS). If the gateway possesses the location information of all devices in the IoT application, it can distinguish between legitimate devices and external threats. The ESP32 supports both Wi-Fi and BLE, RSSI a suitable indicator of signal strength. Signal strength is typically expressed in decibel milliwatts (dBm), with a decrease in signal strength observed as distance increases (Akleylek, Kılıç, Söylemez, Aruk, & Çavuş, 2020). While dBm has a universally recognized value, RSSI values are determined by individual companies and are expressed as percentages. Signal strength below -90 dBm indicates a significant drop in signal quality, leading to loss of control. Signal loss over short distances is attributed to logarithmic variation, and the ideal signal strength is typically found between -30 and -80 dBm.

Within the scope of this thesis, no calculation based on RSSI for ZigBee wireless communication has been conducted.ZigBee is suitable for short-range communication, and assuming our devices are positioned in an indoor environment, moving the device may lead to disconnection. Furthermore, there are no advanced devices capable of replicating our devices with ZigBee wireless communication capability (such as computers, tablets, and mobile devices, etc.).

4.1.1 Implementation and Test Configuration for Wi-Fi

Secure tier 1 aims to physically track resource-constrained devices that cannot run compute-intensive cryptographic algorithms. This is due to the fact that cryptographic algorithms ensuring data confidentiality and device verification involve computations requiring processing power. For this purpose, our example case involves four ESP32 IoT devices placed at fixed points indoors, communicating with each other, and transmitting data to the gateway (Şeker, Şahin, Akdoğan, & Dalkılıç, 2024). The goal is to ensure that our devices remain in their positions without human intervention so that the data flow from these devices is not interrupted by physical intervention. ESP32 devices function as both stations and access points. Therefore, RSSI signals were collected and sent to the gateway using the MQTT protocol for distance calculation.

The fixed positions and distance information of our devices are shown in Figure 4.1. At the same time, the actual distance information of the IoT devices is recorded on server (MG) with the device's MAC information to compare with the calculated RSSI to distance. Distance from RSSI information (Misal, Prajwal, Niveditha, Vinayaka, & Veena, 2020) was calculated with the following formula 4.1. Indicated n in the formula, used as 4 in our calculation, is an external factor varying between 2 and 4.

$$Distance = 10^{Measured\ Power - RSSI)/(10\ x\ n)} \quad (4.1)$$

As seen from Figure 4.1, four ESP32 devices are positioned to form a rectangular area and act as access points. An ESP32 device operating in STA mode is located in this area and scans Wi-Fi periodically according to the four devices at the reference points, collecting RSSI data. The collected RSSI values related to each device are sent to the Raspberry Pi (main gateway) using the MQTT protocol. RSSI values are also stored on the server. Additionally, the server maintains the actual distance values of the ESP32 devices according to the reference points. Mosquitto includes a Python script plugin to calculate the distance based on the RSSI-to-distance formula. The most frequently occurring value among the collected 10 RSSI values is taken as an input for the formula. The error rate is determined by comparing the calculated distance value with the actual distance value.

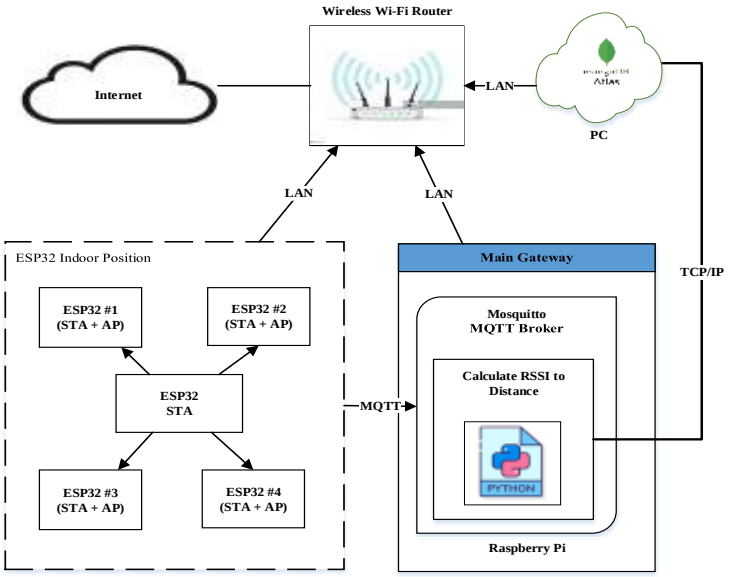


Figure 4.1 The component, relation and interaction of indoor position scheme

When setting up the test environment, four ESP32 devices were used as AP and strategically placed as reference points in an indoor environment. The ESP32 in station mode measured RSSI values against these reference points, which were observed to be in the range of -61dBm to -68 dBm. The device in Station mode was tested in various scenarios both indoors and outdoors, with respect to being positioned near either 4 or 2 reference points. RSSI values were measured accordingly. Using these RSSI values, the distance was calculated using the RSSI to Distance formula. Four different scenarios, dependent on the number of reference points and the different positions of the ESP32 in station mode, are illustrated in Figure 4.2.

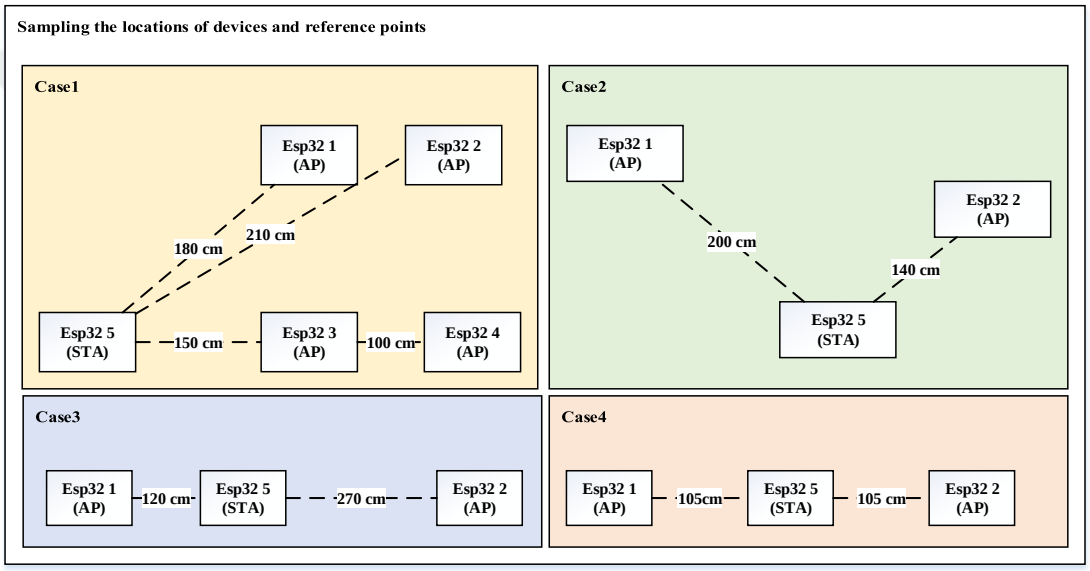


Figure 4.2 Sampling the location of ESP32 in station mode, and reference points

In Case 1, the suggested sample places the ESP32 device in station mode outside the defined rectangular area, with distances to the 4 reference points being 180 cm, 210 cm, 150 cm, and 250 cm respectively. The actual distance between the device's current location and the reference points is intentionally set to exceed 1 meter. This deliberate configuration aims to facilitate the detection of unauthorized devices that do not belong to the designated application. Consequently, data transmitted by devices whose locations are not authenticated will be excluded from the network. As stated in Table 4.1, the RSSI measurement is repeated 10 times, and the most frequently occurring value (that is -74 dBm, -85 dBm, -77 dBm and -90 dBm) is selected for the RSSI-to-Distance calculation.

Table 4.1 Case1: RSSI values (dBm) for ESP32 device positioned outside the designated area, with fixed four reference points

ESP32 (AP1)	ESP32 (AP2)	ESP32 (AP3)	ESP32 (AP4)
-74	-85	-75	-90
-75	-88	-77	-90
-68	-81	-71	-92
-75	-88	-77	-92
-70	-85	-78	-88
-72	-86	-79	-90
-70	-90	-77	-88
-74	-85	-77	-90
-74	-81	-76	-93
-74	-73	-77	-80

In Case 2 and Case 3, the number of reference nodes has been reduced to 2, and the device is positioned between these reference points. The distance based on RSSI values is calculated by ensuring that there is more than 1 meter of distance between the reference points and the ESP32 device. In the final case, the RSSI value was measured at a distance of approximately 1 meter between the 2 reference points. The RSSI values corresponding to case 2, 3, and 4 are presented in Table 4.2.

Table 4.2 Case2, 3, and 4: RSSI values (dBm) for ESP32 device positioned within the designated area, with fixed two reference points

Case2 (dBm)		Case3 (dBm)		Case4 (dBm)	
ESP32#1	ESP32#2	ESP32#1	ESP32#2	ESP32#1	ESP32#2
-31	-40	-31	-46	-68	-71
-31	-40	-31	-45	-71	-70
-31	-39	-33	-46	-71	-71
-33	-40	-36	-44	-71	-68
-33	-40	-28	-46	-70	-70
-31	-40	-31	-49	-69	-69
-33	-40	-32	-44	-74	-69
-32	-40	-31	-46	-71	-67
-33	-38	-29	-53	-71	-71

The most frequently occurring value of the RSSI measurement, repeated 10 times, and the results of the RSSI-to-distance formula calculation for the 4 cases are presented in Table 4.3.The measured power value in the formula represents the RSSI value measured at 1m. When considering the actual distance and the calculated distance, the error rate is indicated as a percentage.

Table 4.3 Actual distance compared with RSSI-to-Distance

	References Nodes	RSSI	Measured Power	N	Distance	Calculation	Average Error (%)
Case1	ESP32 1	-74	-68	4	180	141	27.34
	ESP32 2	-85	-68	4	210	266	
	Esp32 3	-77	-68	4	150	167	
	Esp32 4	-90	-68	4	250	354	
Case2	ESP32 1	-40	-28	4	200	199	2.35
	ESP32 2	-33	-28	4	140	133	
Case3	ESP32 1	-31	-27	4	120	125	8.46
	ESP32 2	-46	-27	4	270	298	
Case4	ESP32 1	-71	-69	4	105	112	6.67
	ESP32 2	-71	-69	4	105	112	

When we examine the error rates based on the distance calculated with Table 4.3, we observe that with 4 reference points, the error margin ranges from 0.39 to 1.04 meters. However, this margin significantly decreases to between 0.01 and 0.5 meters when there are only 2 reference points. This suggests that as the number of reference points decreases, the accuracy of the distance calculation improves. It is possible to observe that as the distance between the device and the reference points increases depending on the number and location of reference points, the RSSI values deviate from the generally accepted range of -30 dBm to -80 dBm, indicating a decrease in signal strength. Using more reference points generally allows for more accurate and reliable distance calculations. Therefore, observing a larger margin of error when 4 reference points are used may indicate that these points are more effective in making a better distance estimation. Calculations with 4 reference points show a wider range of error margins. This may suggest that measurements taken under different conditions exhibit more variability, and the results can vary over a wider range. In the first case, since 4 devices are used as access points, the coverage area is much larger, resulting

in higher error rates. Another significant reason for the errors is electromagnetic interference, as the experiments were not conducted in a laboratory environment. Additionally, the ESP32 device is not very stable in wireless communication. However, having a lower range of error when using 2 reference points could indicate that these measurements are more consistent and reliable. In the remaining cases, the error rate is much lower. Considering all the cases, error rates are acceptable to detect whether the device is in the coverage area or not.

4.1.2 Selecting the Experimental Environment for BLE and Testing

Considering the smooth floor and its size, room 129 on the 1st floor of the Department of Computer Engineering at Dokuz Eylül University has been selected as the application environment. Since the room floor is tiled with 40x40 cm tiles and to observe the movements of our ESP32 station mode device, our working environment has been designed as depicted in Figure 4.3, and the coordinates and movements of the device have been adjusted accordingly. Access points are placed at the corners of a 6x6 square plane, and within a 5x5 area, an ESP32 Node is moved to each point, sending data every 10 seconds. A total of 100 RSSI measurements are taken from each point to the access points. Starting from the (0,0) point, movement occurs in the X direction by 40 cm while keeping Y constant until reaching point (5,5). All nodes are visited. A total of 14400 RSSI (dBm) data points are logged into a file, with 400 measurements at each point.

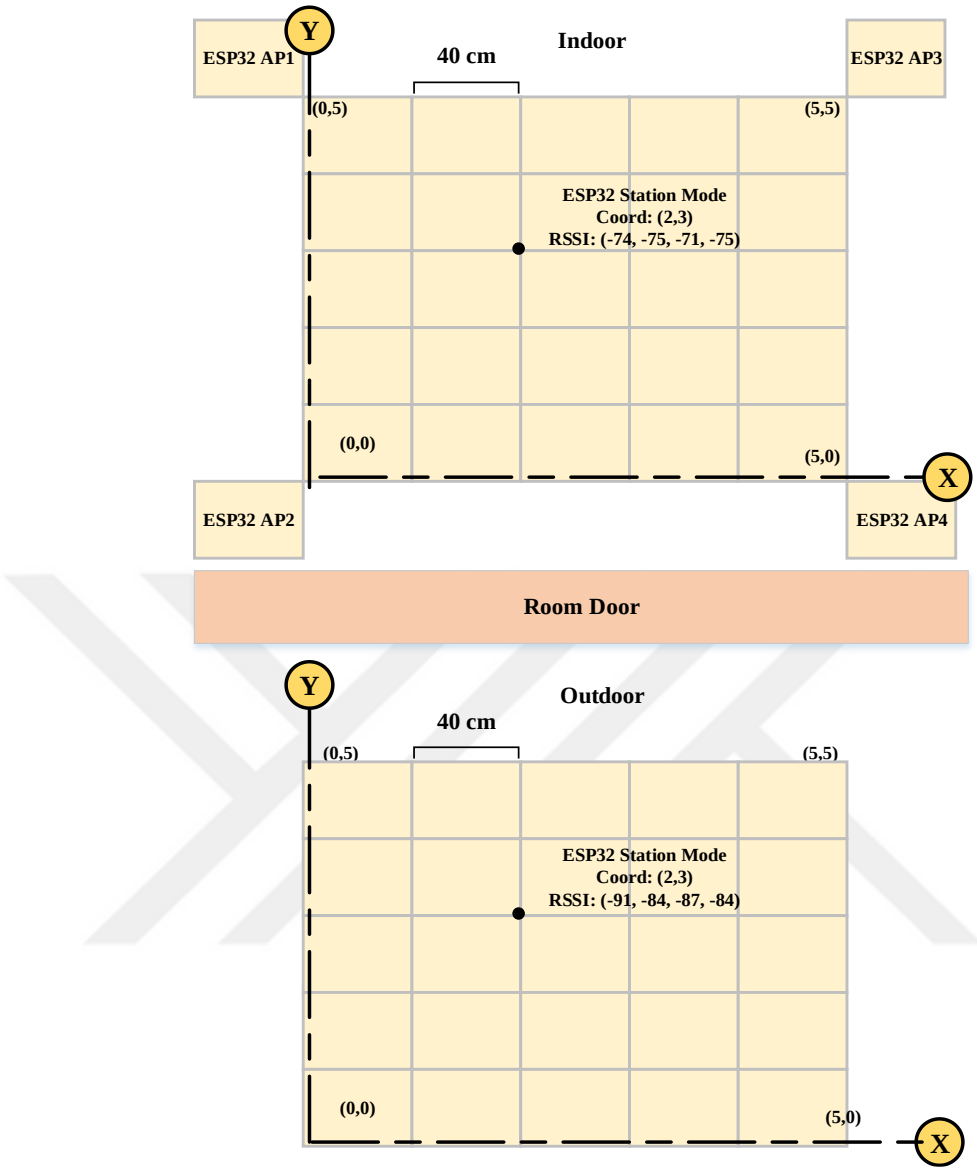


Figure 4.3 Test environment, positioning of ESP32 access points, and RSSI measurement of ESP32 station mode

The same process is carried out by moving outside the room to cover the same 5x5 area. As indicated in Figure 4.3, in the indoor environment, the most frequently occurring RSSI values obtained from points AP1, AP2, AP3, and AP4 are respectively -74 dBm, -74 dBm, -71 dBm, and -75 dBm. However, in the outdoor scenario, these values were measured as -91 dBm, -84 dBm, -87 dBm, and -84 dBm, indicating signal loss.

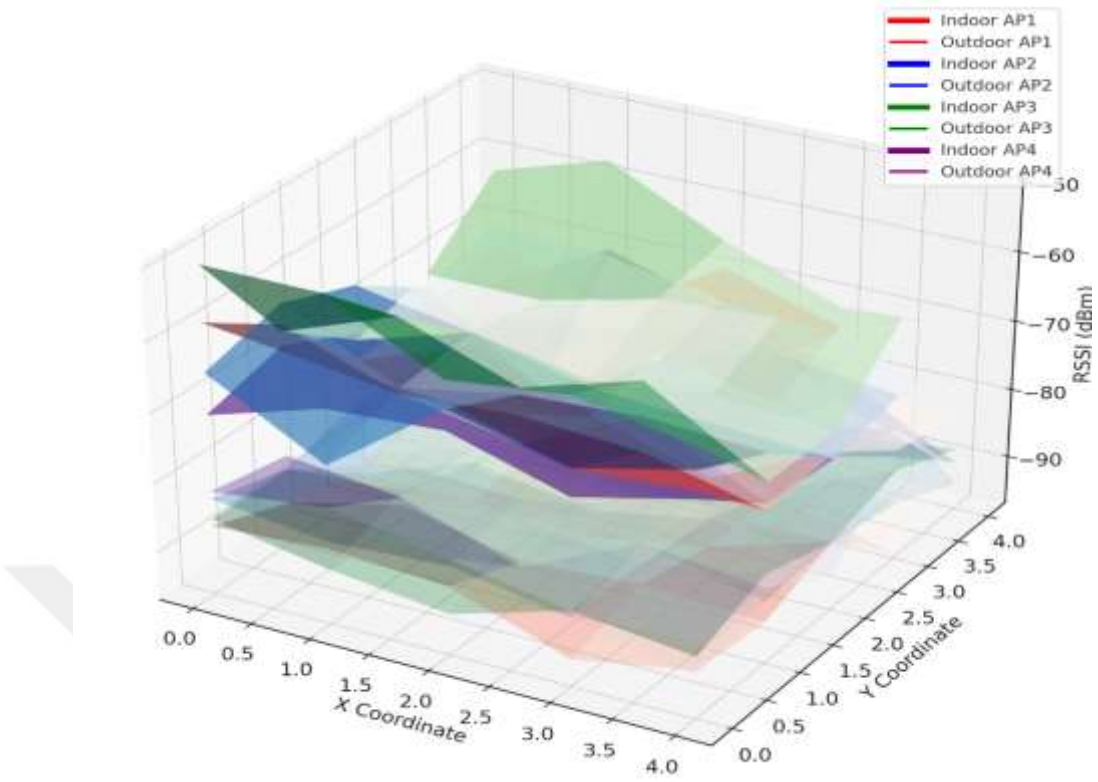


Figure 4.4 Indoor and outdoor RSSI distribution for all access points (ESP32).

In Figure 4.4, RSSI values are generally higher indoors compared to outdoors and are closer to 0 dBm. This indicates stronger signal strength in the indoor environment. Additionally, there is more variability in indoor RSSI values, with certain access points (APs) showing stronger signals at specific coordinates. For example, high RSSI values are observed in areas close to AP1, and this signal strength decreases as the distance from AP1 increases. The same applies to AP2, AP3, and AP4; each exhibits strong signals in areas near their locations, and the signal strength diminishes as the distance increases. This consistent pattern demonstrates a predictable signal attenuation in the indoor environment.

In contrast, outdoor RSSI values are generally lower and more negative, indicating weaker signal strength. The distribution in the outdoor environment is more uniform and shows less variability compared to the indoor environment. All outdoor APs exhibit much lower RSSI values compared to indoor APs. The relatively weak signal strength suggests significant attenuation due to obstacles and environmental factors.

Based on this data, a threshold value can be used to determine whether a measurement is taken indoors or outdoors based on RSSI values. Indoor RSSI values typically range from -50 dBm to -70 dBm, while outdoor values range from approximately -80 dBm to -90 dBm.

Using four access points fixed in both indoor and outdoor locations, the most frequently occurring value (mod operation) is extracted from the RSSI values. These values are then used with the RSSI-to-Distance formula in equation (4.1) to calculate the distance to each point. The RSSI value measured with BLE communication technology at 1 meter is -69 dBm, and an n coefficient of 4 is taken for environmental factors-induced errors. Additionally, the distance from each point in the known 6x6 area to the ESP32 nodes is calculated using the Euclidean distance formula and stored in MongoDB. Table 4.4 presents the actual distance, measured distance in the indoor environment, and measured distance in the outdoor environment. Based on this table, we can establish a threshold value within the indoor area, and any values exceeding this threshold may indicate external interference. Appendix 1 provides detailed RSSI measurements for indoor and outdoor locations in Room 129, 1st Floor, DEU. Complementing Table 4.4, the table includes RSSI values from four different access points (APs) and the corresponding calculated and actual distances. The data is organized to reflect the signal strength and distance relationships in both settings, offering insights into signal propagation characteristics.

Table 4.4 The actual distance, measured distance in the indoor environment, and measured distance in the outdoor environment

Coord	Actual Distance				Calculated Distance (Indoor)				Calculated Distance (Outdoor)			
	AP1	AP2	AP3	AP4	AP1	AP2	AP3	AP4	AP1	AP2	AP3	AP4
(0,5)	40	200	243.31	312.41	53.09	79.43	33.50	112.20	281.84	223.87	266.07	211.34
(1,5)	56.57	203.96	282.84	203.96	59.57	133.35	33.50	84.14	266.07	251.19	316.23	188.36
(2,5)	89.42	120	164.92	256.12	63.10	66.83	53.09	79.43	281.84	237.14	354.81	223.87
(3,5)	126.49	233.34	126.49	233.24	84.14	84.14	47.32	105.93	398.11	281.84	266.07	266.07
(4,5)	164.92	256.12	89.44	215.41	79.43	66.83	56.23	84.14	334.97	354.81	298.54	281.83
(5,5)	203.96	282.84	56.57	203.96	79.43	66.83	84.14	94.41	354.81	266.07	266.07	237.13
(0,4)	80	160	252.98	288.44	112.20	105.93	59.57	112.20	281.84	199.53	251.19	266.07
(1,4)	89.44	164.92	215.41	256.12	105.93	89.13	84.14	133.35	354.81	211.35	251.19	375.83

Table 4.4 Continues

Coord	Actual Distance				Calculated Distance (Indoor)				Calculated Distance (Outdoor)			
	AP1	AP2	AP3	AP4	AP1	AP2	AP3	AP4	AP1	AP2	AP3	AP4
(2,4)	113.14	178.89	226.27	178.89	112.20	84.14	63.10	84.14	334.97	237.14	281.84	237.13
(3,4)	144.20	200	144.20	200	141.25	89.13	112.20	133.35	354.81	298.54	237.14	266.07
(4,4)	178.89	226.27	113.14	178.89	158.49	84.14	133.35	125.89	421.70	223.87	421.70	354.81
(5,4)	215.41	256.12	89.44	164.92	112.20	100.00	100.00	141.25	421.70	316.23	298.54	266.07
(0,3)	120	120	268.33	268.33	158.49	125.89	118.85	133.35	446.68	334.97	354.81	281.83
(1,3)	126.49	126.49	233.24	233.24	211.35	133.35	105.93	149.62	375.84	211.35	298.54	266.07
(2,3)	144.22	144.22	200	200	133.35	141.25	112.20	141.25	281.84	281.84	281.84	375.83
(3,3)	169.71	169.71	169.71	169.71	149.62	125.89	112.20	211.35	354.81	237.14	281.84	237.13
(4,3)	200	200	144.22	144.22	141.25	237.14	112.20	158.49	354.81	251.19	316.23	298.53
(5,3)	233.24	233.24	126.49	126.49	141.25	149.62	89.13	141.25	251.19	316.23	266.07	251.18
(0,2)	160	80	288.44	252.98	167.88	112.20	84.14	199.53	375.84	223.87	354.81	266.07
(1,2)	164.92	89.44	256.12	215.41	199.53	125.89	66.83	167.88	473.15	316.23	237.14	211.34
(2,2)	178.89	113.14	226.27	178.89	125.89	105.93	66.83	177.83	334.97	266.07	473.15	223.87
(3,2)	200	144.22	144.22	200	177.83	112.20	50.12	105.93	354.81	211.35	421.70	223.87
(4,2)	226.27	178.89	178.89	113.14	100.00	133.35	79.43	141.25	266.07	177.83	237.14	211.34
(5,2)	256.12	215.49	164.92	89.44	133.35	141.25	112.20	251.19	375.84	177.83	334.97	199.52
(0,1)	200	40	312.41	243.31	223.87	223.87	105.93	354.81	354.81	398.11	298.54	334.96
(1,1)	203.96	56.57	282.84	203.96	211.35	177.83	118.85	149.62	446.68	316.23	354.81	251.18
(2,1)	215.41	89.44	256.12	164.92	211.35	100.00	118.85	118.85	421.70	398.11	334.97	398.10
(3,1)	233.24	126.49	233.24	126.49	177.83	158.49	100.00	167.88	473.15	223.87	334.97	251.18
(4,1)	256.12	164.92	215.41	89.44	211.35	112.20	112.20	199.53	398.11	237.14	530.88	237.13
(5,1)	282.84	203.96	203.96	56.57	105.93	112.20	125.89	149.62	298.54	298.54	281.84	158.48
(0,0)	240	0	339.41	240	177.83	105.93	94.41	112.20	354.81	281.84	375.84	223.87
(1,0)	243.31	40	312.41	200	94.41	118.85	79.43	133.35	281.84	237.14	298.54	199.52
(2,0)	252.98	80	288.44	160	94.41	211.35	63.10	158.49	334.97	237.14	334.97	223.87
(3,0)	268.33	120	268.33	120	199.53	237.14	237.14	211.35	354.81	266.07	316.23	188.36
(4,0)	288.44	160	252.98	80	141.25	112.20	112.20	223.87	334.97	211.35	298.54	266.07
(5,0)	312.41	200	243.31	40	133.35	100.00	167.88	112.20	354.81	316.23	281.84	281.83

Based on the results in the Table 4.4, the error margin between the calculated distance and the actual distance in indoor and outdoor environments has been determined. According to this error margin, data points with an error margin above the average in the indoor environment are classified as outdoor environment data and presented in Figure 4.5.

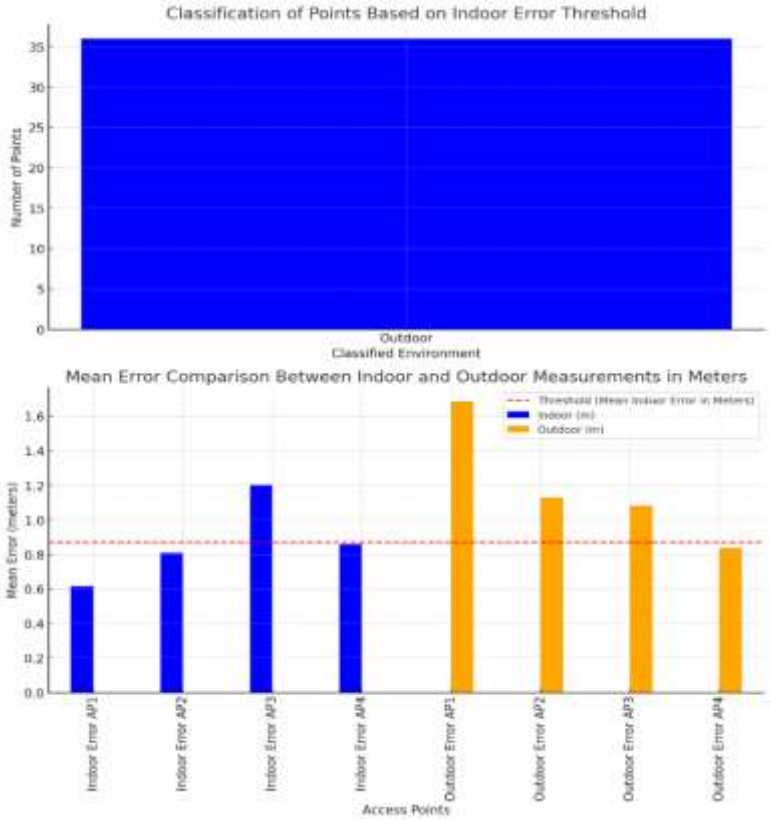


Figure 4.5 Mean error comparison between indoor and outdoor measurements in meters

In the Figure 4.5, we show the error margins in meters. The top graph displays the number of data points classified as "Indoor" and "Outdoor," and it can be seen that all data points are classified as "Outdoor." The bottom graph shows the average error margins in meters for each access point (AP1, AP2, AP3, AP4) in both indoor and outdoor environments. The red dashed line represents the average threshold value of the indoor error margin in meters.

While the average error margin obtained in the indoor environment is 0.872 meters, it is observed that these values are much higher in the outdoor environment. The average error margins for each Access Point are calculated as 0.617 meters for AP1, 0.809 meters for AP2, 1.201 meters for AP3, and 0.860 meters for AP4 in the indoor environment. In the outdoor environment, the error margins are found to be 1.684 meters for AP1, 1.130 meters for AP2, 1.081 meters for AP3, and 0.836 meters for AP4. These results indicate that the error margins of the outdoor data are above the threshold value of the indoor environment, and therefore are classified as "Outdoor.

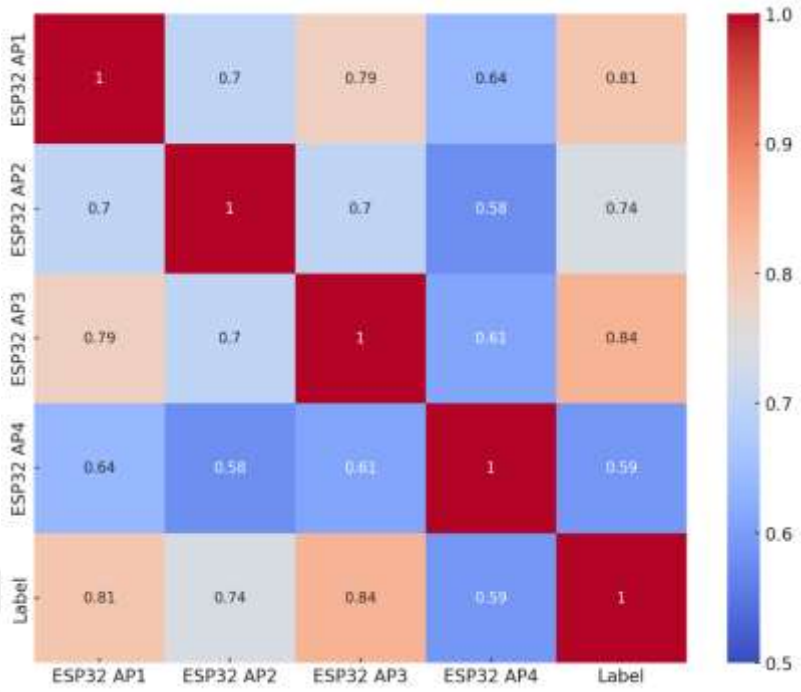


Figure 4.6 Correlation matrix of features (including label).

In Figure 4.6, the correlation matrix reveals significant insights into the relationships between RSSI values from different access points (APs) and the classification label. Notably, ESP32 AP1 shows strong positive correlations with ESP32 AP2 (0.70) and ESP32 AP3 (0.79), and a moderate correlation with ESP32 AP4 (0.64). Importantly, the correlation between ESP32 AP1 and the label is 0.81, indicating that AP1's RSSI values significantly influence the label, which likely distinguishes between indoor and outdoor environments.

Similarly, ESP32 AP2 exhibits strong correlations with ESP32 AP3 (0.70) and a moderate correlation with ESP32 AP4 (0.58). The correlation between AP2 and the label is 0.74, suggesting that AP2's RSSI values also have a substantial impact on the label. ESP32 AP3 demonstrates moderate correlation with ESP32 AP4 (0.61) and a very strong correlation with the label (0.84), highlighting AP3's critical role in influencing the classification.

ESP32 AP4, while showing moderate correlations with the other APs, has a lower correlation with the label (0.59) compared to the other APs, indicating that its influence on classification is less significant. Overall, this analysis identifies ESP32 AP1, AP2,

and AP3 as key determinants in distinguishing indoor from outdoor environments, with AP4 playing a relatively minor role. This understanding is crucial for refining the classification model by prioritizing the more influential APs.

In addition to computing the distances, RSSI values were used to predict whether the device is indoors or outdoors using default versions of basic learners. Two files containing indoor and outdoor RSSI data were provided as input, where 90% of the data was used for training and 10% for testing. Three basic classifiers, namely Support Vector Machines (SVM), k-Nearest Neighbors (KNN), and Random Forest (RF), were assessed and compared. Performance metrics comprised accuracy, precision, recall and F1-score. All models underwent 10-fold cross-validation. The parameters of the base classifiers were chosen as default. For creating the KNN model, the number of neighbors was set to 5, and the distance metric used was Euclidean. The "auto" method, which determines the most suitable algorithm based on the values provided to the fit method, was chosen for computing the nearest neighbors.

The total number of samples in the indoor and outdoor data files was 7200. Due to the 10-fold cross-validation, in each iteration, the training and test sets consist of 6480 and 720 samples, respectively. Table 4.5 compares the accuracy results, weighted precision, recall, and F-score values of the applied methods. The high accuracy indicates the effectiveness of BLE technology in short ranges, and the decrease in signal outside closed areas demonstrates the consistency of real-time collected data. It has been observed that external interference from a device can be detected in our application located in a closed area. Additionally, if the device is moved from its physical location, there will still be signal loss in RSSI values, enabling the detection of an attack scenario.

Table 4.5 The comparisons of the applied techniques in terms of accuracy, precision, recall and fscore

Base Classifier	Acurracy	Precision	Recall	FScore
SVM	99.31	99.32	99.31	99.31
KNN	99.59	99.60	99.59	99.59
RF	99.60	99.61	99.61	99.61

4.1.3 Performance Discussion of Secure Tier 1

In a closed environment with security level 1, attempts were made to observe whether there is a relationship between the weakening of the RSSI signal and the distance as the device is moved from its location, as stated above in the test environments. The Friis model, which takes into account environmental factors related to distance and calculates approximate distance with logarithmic changes, serves as a basis in Wi-Fi communication. This model is used to predict communication quality by considering how signal strength varies with distance and environmental conditions. Particularly in indoor or complex environments, walls, furniture, and other obstacles can affect signal strength. By considering such interactions, the Friis model is preferred for calculating communication distance and power.

In measurements conducted by excluding the device from the designated reference points to achieve greater coverage area with four reference points, the RSSI value significantly demonstrated signal attenuation with distance. With an error rate of 27.34% between the actual and calculated distances and a maximum error rate of 1.04 meters, it is possible to say that, considering the tests were not conducted in a magnetic field, i.e., laboratory environment, the results are acceptable. In measurements conducted within distances of up to 1 meter with fewer reference points, it was observed that the margin of error was very minimal, approximately 0.07 meters.

Using reference points placed at the corners of a 6x6 grid, approximate distance calculations were performed with the Friss model based on 100 RSSI data points collected from each point within a 5x5 inner area on the BLE side. Similarly, RSSI data was collected from outside the room with the same grid size for comparison. Threshold values can be discussed to provide information on whether the device is inside or outside. Considering this, machine learning methods were employed to predict whether the device is indoors or outdoors based on RSSI information. The accuracy of predicting whether the device is indoors or outdoors was found to be 99.31% for SVM, 99.59% for KNN, and 99.60% for RF. These results demonstrate the consistent performance of RSSI signals in BLE communication with ESP32 devices.

Within the scope of this thesis, the relationship between RSSI and distance has been considered. In future studies, location prediction instead of distance will be tested using machine learning and deep learning methods.

4.2 Outlook of Secure Tiers 2 and 3

Under this heading and its subheadings, the steps for applying lightweight authentication and authorization mechanisms to IoT devices, developed through the hybrid use of existing cryptographic methods, are detailed, covering security tiers 2 and 3.

4.2.1 Procedures of Security Mechanism

The ultimate outcome of our thesis study is to present a prototype of an IoT multi-protocol, multi-gateway system based on comprehensive encryption levels that align with security requirements. A prototype as shown in Figure 3.2 has been designed and implemented to support various security tiers, enabling users to select the appropriate level of security based on their specific requirements. The prototype implementation entailed the utilization of a minimum of ten hardware modules, comprising ESP32s, ESP32-CAMs, an ESP32 with XBee-pro S2C, a Raspberry Pi 4, and a PC. The preceding section introduced the hardware selection details. While this section expands on the functionality and features, the subsequent section presents the performance tests.

The developed prototype of the IoT multi-protocol multi-gateway allowed us to verify the security features we assert to offer, such as a secure internet connection, encrypted communication between the gateway and connected devices, implementation of access control, and monitoring of the activity of connected devices. Moreover, the prototype provides mutual authentication and authorization for devices and entities within the network. This guarantees that only authorized and trusted devices are permitted to communicate with the gateway, thereby reducing the potential for unauthorized access or malicious attacks.

Communication between edge devices and the main gateway is facilitated via the MQTT protocol. However, in the current state, only a token and a username-password

pair are utilized to authenticate IoT devices over the MQTT protocol. There is no standardized or universally accepted security mechanism within the MQTT protocol to ensure data integrity, confidentiality, and availability. Two primary reasons can be discussed for the necessity of a new authentication (and authorization) mechanism for the MQTT protocol.

Without an authentication mechanism in place, any malicious or compromised device can act as an attacker by sending fake messages or gaining access to sensitive data. One of the main challenges in developing an appropriate authentication mechanism is ensuring "uniqueness" for each device in a huge network. When examining the literature, it is evident that authentication is predominantly carried out using token-based solutions. However, a weakness arises from the vulnerability of fixed tokens, making identity information susceptible to interception. To address this vulnerability, the possibility of connecting devices to an authentication server via a secure channel like HTTPS was explored. However, due to the constraints of IoT devices and reported security vulnerabilities, this approach was found to be impractical. Consequently, within the scope of this thesis, a significant contribution was made by implementing a second authentication factor to mitigate the risk of a compromised fixed token. A lightweight one-time password scheme based on HMAC was employed to update passwords during each connection.

Finally, regulating device access rights is contingent upon their topic-related interests, ensuring that messages remain inaccessible to all devices through suitable access restrictions. Employing the security mechanism established with OAuth 2.0 for implementing access control on Mosquitto prevents devices from engaging in malicious behavior.

The proposed security procedure comprises three sub-steps: registration, authentication, and authorization, as detailed below.

4.2.2 Registration Stage

The initialization of the system requires manual intervention from a developer or administrator, who is responsible for registering the end devices intended for network usage onto the identity server (IS), also known as the authorization server (AS). In IoT

applications, managing unique credentials for each device becomes increasingly challenging with the growing number of devices, particularly in the absence of user interfaces and input-output units. The management of tasks such as setting, modifying, and revoking access permissions for these devices can pose challenges. To address these complexities, system developers or administrators need to follow a structured process that ensures secure device registration, token acquisition, and periodic renewal to maintain secure communication within the IoT application. Figure 4.7 provides a concise overview of the device registration process. Further elaboration on this procedure is presented below:

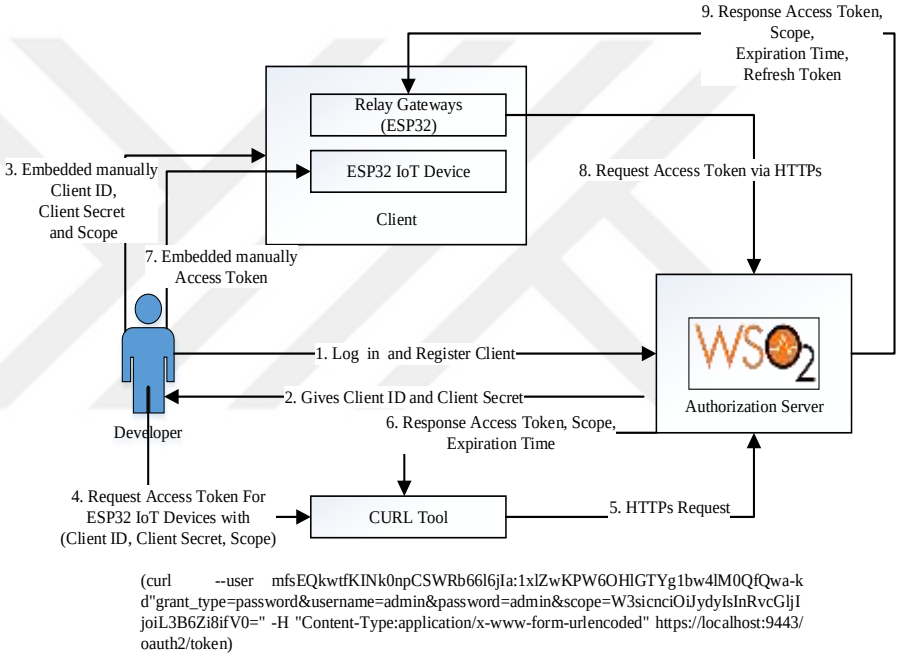


Figure 4.7 Procedure for end device registration and token acquisition

1. **Identity Server Login and Device Registration**. The system administrator or developer accesses the IS console using predefined credentials, such as a username ("admin") and password ("TrGyL32!"), and proceeds to register IoT end devices with the IS. During registration, scopes are defined, encompassing the topics of interest and access permissions for the devices.
2. **Client ID and Client Secret Generation**. The IS identifies each edge devices such as (ESP32 mesh node, ESP32 Zigbee transmitter and receiver and ESP32

BLE slave and master) and allocates a unique string value known as the client ID, along with a secret value termed as the client secret, for every device.

3. **Manual Embedding of Credentials.** The developer manually inserts the client ID, client secret, and scope information into each IoT edge device.
4. **Access Token Claim for End Device.** Developers acquire access tokens for end devices (like ESP32 Node) utilizing the Curl tool.
5. **Connection of Curl to IS.** Establishing a connection between Curl and the IS involves executing Curl commands with appropriate parameters (such as client ID and client secret) to communicate with the IS and retrieve access tokens for end devices.
6. **Access Token Issuance by IS**. IS issues access tokens to IoT edge devices after successfully authenticating the developer and obtaining authorization.
7. **Manual Embedding of Access Token.** The developer manually integrated the access token into the edge devices.
8. **Access Token Claim for Relay Gateways.** Wi-Fi mesh, BLE, and ZigBee client modules in the relay gateway request an access token by sending an HTTPS request.
9. **Refresh Token Acquisition.** Upon receiving the access token from AS, the relay gateway obtains a refresh token via HTTPS to prevent potential security attacks. After each successful connection between the relay gateway and the main gateway, the relay gateway requests a refresh token from the IS via an HTTPS connection.

The access token, represented in string format, provides authorization to the client. These tokens contain scopes associated with the device's topic and the duration of access granted by the developer, a regulation enforced by both the Mosquitto broker and the identity server.

4.2.3 Authentication Stage

The registration procedure for edge devices was carried out according to the guidelines outlined in previous subsection and depicted in the Figure 4.7. This subsection presents specific implementation considerations and details in addition to the theoretical procedure. Henceforth, for improved readability, we will refer to the

end devices based on their communication technology. As illustrated in the Figure 3.1, Nodes 1, 2, and 3 will be denoted as NodeW (W for Wi-Fi) without any numerical designation, as they were identical. Node 6 will be referred to as NodeB (B for BLE), and Node 8 will be designated as NodeZ (Z for ZigBee). Additionally, the three ESP32-CAM modules integrated into the relay gateway (abbreviated as RG) will be named NodeWRG, NodeBRG, and NodeZRG, following the same logic.

During the authentication procedure, the IS distributes unique identifiers (IDs) for each NodeW, NodeB, and NodeZ, along with their respective access tokens. These ID and access token combinations are manually integrated into the ESP32 end devices. The default length for an ID and an access token on the IS is 28 and 32 characters, respectively. A 32-character access token represents 4 bits per character, resulting in a total of 128 bits, which is equivalent to 16 bytes.

The size of both the refresh token and access token remain uniform. Upon reception of the ID and access token, each ESP32 end node (NodeW, NodeB, and NodeZ) autonomously formulates and transmits authentication parameters for three distinct communication technologies. The comprehensive process for device registration and initialization is delineated in Figure 4.8 as Algorithm 1, complemented by further clarifying remarks below:

- Every node produces a *HOTP* utilizing its individual *ID* as the secret along with a *count* parameter (Alg. 1 line 4).
- NodeW acquires a distinctive *NodeID* (Chip ID) from the ESP32 SDK, setting it apart from other nodes (Alg. 1 line 6).
- NodeW calculates the message digest of the unique identity of ESP32, denoted as H(*NodeID*), and subsequently incorporates it into the parameters of the QR codes (Alg. 1 line 7).
- Following this, the end nodes (NodeW, NodeB, and NodeZ) produce QR codes that display their *HOTP*, *access token*, *ID*, and *NodeID* parameters. Wi-Fi nodes generate the QR code by combining the 17 bytes ESP32 chip ID hash value (20 bytes) with the 6-digit HOTP and the 16 bytes access token. Conversely, BLE and ZigBee nodes formulate the QR code to include a 6-digit HOTP and a 28 bytes ID (Alg. 1 lines 8-9).

- The end nodes present their QR code on their TFT LCD screen. (Algorithm 1, line 10).

Algorithm 1 Device Registration Procedure

1: IS → Generates IDs and keys: ID_{NodeW} , ID_{NodeB}, ID_{NodeZ} , $ID_{NodeW\ RG}$, $ID_{NodeBRG}$, $ID_{NodeZRG}$, $SK_{NodeW\ RG}$, $SK_{NodeBRG}$, $SK_{NodeZRG}$
2: Developer → Embeds into corresponding nodes: ID_{NodeW} , ID_{NodeB}, ID_{NodeZ} , IDN_{odeWRG}, $ID_{NodeBRG}$, $ID_{NodeZRG}$, $SK_{NodeWRG}$, $SK_{NodeBRG}$, $SK_{NodeZRG}$
3: IS → Generates access tokens: AT_{NodeW} , $AT_{NodeW\ RG}$, $AT_{NodeBRG}$, $AT_{NodeZRG}$
4: Developer → Embeds into NodeW: AT_{NodeW}
5: NodeWRG, NodeBRG, NodeZRG → Request from IS via HTTPS: $AT_{NodeWRG}$, $AT_{NodeBRG}$, $AT_{NodeZRG}$
6: NodeW, NodeB, NodeZ → Calculate HOTP: *HOTPN(ID, N) = Truncate(HMAC-SHA-1(ID, C))*
7: NodeW → Calculates Hash: *Hash1 = H(ID)*
8: NodeW → Creates QR code: *QRCodeW = QRCode(HOTP ||*AT_{NodeW} *||Hash1)*
9: NodeB, NodeZ → Create QR code: *QRCode = QRCode(HOTP ||ID)*
10: NodeW, NodeB, NodeZ → Display on LCD: QRCode

Figure 4.8 Algorithm 1: Device Registration Procedure

4.2.3.1 End Nodes Authenticated by Relay Gateways

The authentication process of end nodes connected to the relay gateway's modules is carried out in four distinct phases:

1. The relay gateway initiates the discovery and verification of end nodes' IDs.
2. End nodes then proceed to verify the IDs of relay gateway modules.
3. Subsequently, relay gateway modules authenticate the end nodes.
4. Finally, an additional whitelisting scheme is implemented between the relay gateway and Wi-Fi end nodes.

In order to mitigate the risk of eavesdropping, QR codes are employed to convey security parameters, such as the symmetric key, via a secure channel. Therefore, to commence the authentication procedure, NodeW, NodeB, and NodeZ generate a QR code and transmit the message 'Scan QR' to the relay gateway to establish a connection. The ESP32-CAMs equipped with the OV2640 camera module within the relay gateway scan the QR code displayed on the LCD screens. In contrast to Wi-Fi mesh, BLE and ZigBee do not utilize a mesh network feature, which simplifies the authentication process for them. However, this leads to minor variations in specific steps, while the main flow remains largely similar. To mitigate this issue, we primarily

consolidate BLE and ZigBee actions whenever feasible and present Wi-Fi actions separately.

In the Wi-Fi PainlessMesh network, NodeWRG establishes a connection to Wi-Fi by utilizing the SSID and functions as an access point. However, it establishes an MQTT connection during the second phase of authentication and configures the mesh network. When a new node seeks to join the mesh network, a mutual authentication process commences with the first phase. The node connection request is declined if there is no QR information in the initial step. Our design presupposes that the devices are situated in a physically secure environment, with the aim of validating the devices seeking to join the network by other devices within the radio range.

For BLE communications, NodeBRG, referred to as the master or client (not the slave), employs both Wi-Fi and BLE communication, facilitated by the ESP32 device. NodeBRG establishes an MQTT connection to the main gateway via Wi-Fi while simultaneously seeking a slave node (server) to initiate the BLE connection. The slave node periodically broadcasts its presence and awaits connections. When it receives a connection request from NodeBRG, it showcases a QR code on the LCD, initiating the authentication process. Each time a connection request is made, the relay gateway demands QR code data. Devices without QR code data are barred from the system. The procedures for authenticating the Node to the relay gateway using QR code and HMAC-based OTP are outlined in Figure 4.9 and Figure 4.10 as Algorithm 2.

In the realm of ZigBee communications, comparable protocols are employed as with BLE, although with nuanced differences due to the absence of inherent ZigBee support in ESP32S. On the contrary, communication initiates through serial communication using an XBee module to establish ZigBee communication. The initial phase encompasses the the relay gateway (referred to as NodeZRG equipped with the XBee receiver), which initiates an MQTT connection with the main gateway (Raspberry Pi). Subsequently, it commences receiving ZigBee data and triggers the authentication process. The subsequent authentication steps mirror those employed for BLE.

Step 1 of Figure 4.9 involves the generation of QR code components to facilitate the inclusion of end devices into the network, with subsequent display on the TFT LCD. Next, the relay gateway initiates a "SCAN QR" request.

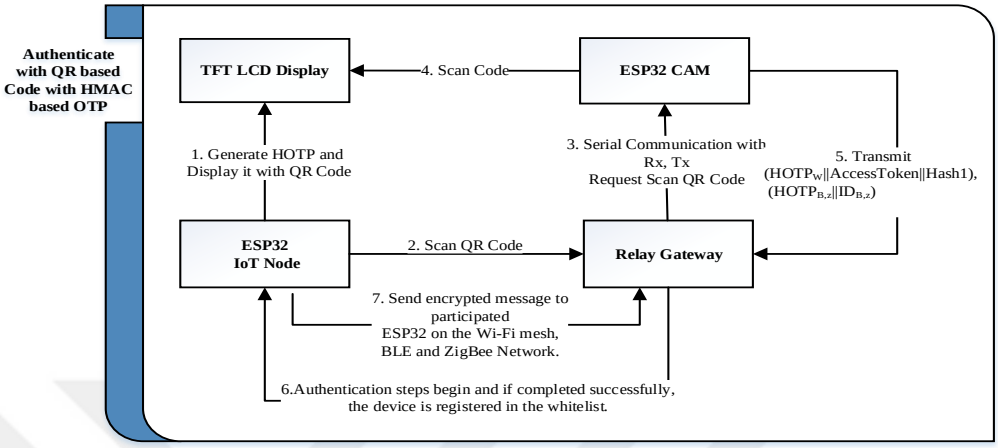


Figure 4.9 Authentication of end node by relay gateway with QR code and HOTP

In step 3, the relay gateway communicates serially with the ESPCAM to prompt QR code scanning. Step 4 involves the ESP32-CAM scanning the data, followed by step 5 where the relay gateway parses the scanned data, conducts ID verification, and generates a symmetric encryption key using this information. In step 6, the relay gateway encrypts the symmetric encryption key with the PK_{Node} by incorporating the $HOTP_{Node}$ value and transmits it to the Node. Finally, Algorithm 2 delineates the identity verification steps, enacted upon successful mutual authentication between the Node and RG, thus enabling data transmission with AES encryption to ensure data confidentiality.

The comprehensive procedure for authenticating end nodes by the relay gateway (involving all four phases) is precisely detailed in Figure 4.10 (Algorithm 2), accompanied by noteworthy remarks presented below:

- The ESP32-CAM module integrated into the relay gateway scans the QR codes displayed on the LCD screens of the end nodes (Alg. 2, line 1).
- Within the relay gateway, the ESP32-CAM module transmits the credentials (HOTP || AccessToken || Hash1) to the corresponding module through wired serial communication (Alg. 2, lines 2-3).

- After receiving Hash1, NodeW is authenticated by comparing the computed Hash1 with the received Hash1' (Alg. 2, lines 4-5).
- Upon receiving the IDs for NodeB and NodeZ, the node undergoes verification by comparing the computed HOTP with the received HOTP' (Alg. 2, lines 8-9).
- After extracting the access token from the QR code payload of a node, it is utilized to generate the symmetric key for AES encryption (Alg. 2, lines 6, 11).
- Upon reception of the message, the node decrypts the ciphertext using SKNodeW, an access token of up to 128 bits (Alg. 2, line 13).
- Upon reception of the message, the node decrypts the ciphertext using SKNode, a client ID of up to 128 bits (Alg. 2, line 14).
- Upon receiving the ciphertext, NodeBRG and NodeZRG are authenticated by comparing the received HOTP' to the calculated HOTP (Alg. 2, lines 11-20).
- Following decryption, the node verifies NodeW by comparing the calculated Hash2 with the received Hash2' (Alg. 2, lines 26-27).
- Upon successful verification, the validated node is appended to the whitelist containing authenticated nodes. Subsequently, the updated whitelist is disseminated to all authenticated nodes along with the relay gateway's nonce (Alg. 2, lines 28-30).
- Upon decryption and parsing of ciphertext2, NodeB and NodeZ undergo authentication by comparing the calculated HOTP to the received Nonce (Alg. 2, lines 32).
- Upon successful authentication of NodeB and NodeZ, the message is permitted for transmission (Alg. 2, line 35).
- In case the relay gateway fails to receive the nonce message within the stipulated timeframe, it retries sending the whitelist message thrice. If the issue persists, the node is eliminated from both the network and the whitelist (Alg. 2, line 37).
- NodeW confirms the reception of the updated whitelist by comparing the calculated Hash3 with the received Hash3' (Alg. 2, lines 38-39).
- Following these phases, all communication between the node and the relay gateway is conducted using AES encryption, ensuring secure data transmission.

Algorithm 2 Authentication of End Nodes by Relay Gateway and Data Transfer.

PHASE 1: RELAY GATEWAY DISCOVERS AND VERIFIES END NODES' IDS.

1: NodeWRG, NodeBRG, NodeZRG → *read corresp. QR code from NodeW, NodeB, NodeZ.*
2: NodeWRG → parse QR code of NodeW: *HOTP , AT, Hash1.*
3: NodeBRG, NodeZRG → parse QR code of NodeB, NodeZ: *HOTP , ID.*
4: NodeWRG → calculates Hash1 ′ : $Hash1' = H(ID_{NodeW})$
5: NodeWRG → verifies NodeW via Hash1 ′ : *if (Hash1 ′ == Hash1) then verified*
6: NodeWRG → constructs CipherText1: $CipherText1 = AES\text{-}CBC(PK_{NodeW}, HOTP_{NodeW}||PK_{NodeWRG})$
7: NodeWRG → sends CipherText1 to NodeW.
8: NodeBRG, NodeZRG → calculate HOT P ′ : *HOTP ′ = HOTP(ID, C) ▷ ID is IDNodeB or IDNodeZ* **.**
9: NodeBRG, NodeZRG → verify NodeB, NodeZ via HOTP ′ : *if (HOT P ′ == HOTP) then verified.*
10: NodeBRG, NodeZRG → calculate Nonce: $Nonce = HOTP(ID_{RG}, C) \triangleright ID_{RG}$ *is* $ID_{NodeBRG}$ *or* $ID_{NodeZRG}$.
11: NodeBRG, NodeZRG → construct CipherText1: *CipherText = AES-CBC(SK, HOTP ||Nonce).*
12: NodeBRG, NodeZRG → send CipherText1 to NodeB, NodeZ.

PHASE 2: END NODES VERIFY RELAY GATEWAY MODULES' IDS.

13: NodeW → decrypts received CipherText1 ′ : $DecryptText1 = AES\text{-}CBC(PK_{NodeW}, CipherText1')$
14: NodeB, NodeZ → decrypt received CipherText1 ′ : $DecryptText1 = AES\text{-}CBC(PK, CipherText1')$
15: NodeW → verifies NodeWRG via HOTP ′ : $if\ (HOTP_{NodeW} == HOTP'_{NodeW})$ *then verified*
16: NodeW → calculates Hash2: $Hash2 = H(PK_{RG})$
17: NodeW → calculates Nonce': $Nonce = HOTP^2 = HOTP(HOTP_{NodeW}, C')$
18: NodeW → constructs CipherText2: $CipherText2 = AES\text{-}CBC(PK_{RG}, Hash2||Nonce)$
19: NodeW → sends CipherText2 to NodeWRG.
20: NodeB, NodeZ → verify NodeBRG, NodeZRG via HOTP : *if (HOTP == HOTP ′) then verified*
21: NodeB, NodeZ → calculate Nonce′ : $Nonce' = HOTP_{RG}$
22: NodeB, NodeZ → constructs CipherText2: $CipherText2 = AES\text{-}CBC(SK, HOTP||Nonce')$
23: NodeB, NodeZ → send CipherText2 to NodeBRG, NodeZRG.

PHASE 3: RELAY GATEWAY MODULES VERIFY END NODES' AUTHENTICITY.

24: NodeWRG → decrypts received CipherText2 ′ : $DecryptText2 = AES\text{-}CBC(PK_{RG}, CipherText2')$
25: NodeBRG, NodeZRG → decrypt received CipherText2′ : $DecryptText2 = AES\text{-}CBC(PK_{RG}, CipherText2')$
26: NodeWRG → calculates Hash2: $Hash2 = H(PK_{RG})$
27: NodeWRG → verifies NodeW via Hash2 ′ : *if (Hash2 ′ == Hash2) then verified*
28: NodeWRG → adds NodeW to whitelist of authentic nodes: $Whitelist.append(ID_{NodeW})$
29: NodeWRG → constructs CipherText3 and CipherText3∗: $CipherText3 = AES\text{-}CBC(PK_{RG}, Whitelist||Nonce_{RG}||Nonce')$, $CipherText3* = AES\text{-}CBC(PK_{RG}, Whitelist||Nonce_{RG})$
30: NodeWRG → sends CipherText3 to NodeW and CipherText3∗ to all other nodes in its Whitelist.
31: NodeWRG → finalize authentication.
32: NodeBRG, NodeZRG → verify NodeB, NodeZ via HOTP ′ : $if\ (HOTP_{RG} == HOTP'_{RG})$ *then verified.*
33: NodeBRG, NodeZRG → finalize authentication.
34: NodeB, NodeZ → constructs CipherText3: $CipherText3 = AES\text{-}CBC(PKRG, message)$
35: NodeB, NodeZ → sends CipherText3 to NodeBRG, NodeZRG.

PHASE 4: RG AND WI-FI END NODES RUN ADDITIONAL WHITELISTING SCHEME.

36: NodeW → decrypts received CipherText3 ′ : $DecryptText3 = AES\text{-}CBC(PK_{RG}, CipherText3')$
37: NodeW → verifies NodeWRG via Nonce: *if (Nonce == Nonce′) then verified*
38: NodeW → calculates Hash3: $Hash3 = H(Nonce_{RG})$
39: NodeW → sends Hash3 to NodeWRG.
40: NodeW → constructs CipherText4: $CipherText4 = AES\text{-}CBC(PK_{RG}, message)$
41: NodeW → sends CipherText4 to NodeWRG.
42: NodeW → adds NodeWRG to whitelist of authentic nodes: $Whitelist.append(ID_{NodeWRG})$
43: NodeWRG → ensures NodeW has whitelist: if Hash3 ′ == H(NonceRG) then ensured

Figure 4.10 Algorithm 2: Authentication of end nodes by relay gateway and data transfer

4.2.3.2 Relay Gateways Authenticated by Main Gateway

Upon completion of the authentication process between the nodes and the relay gateway, the subsequent phase entails authentication and access authorization between

the relay gateway and the main gateway. This communication between the entities is enabled via the MQTT protocol, which has been integrated with a robust security mechanism to guarantee secure communication.

The relay gateway strengthens security by encrypting and sending messages from verified ESP32 end nodes to the main gateway using its modules. It also employs multifactor authentication between the relay gateway and the main gateway, using MQTT user-password authentication and OAuth 2.0 access tokens. For ensuring secure communication, an HTTPS connection is initiated using a base64 encoding scheme, which includes the client ID, client secret, and scope information. Subsequently, an access token is acquired from the IS. The acquired access token acts as the username in the MQTT connect message. To bolster security, the access token undergoes periodic refreshment, mitigating the risk of eavesdropping during each MQTT connection and thwarting potential misuse by a third party persistently attempting to access the gateway using the access token.

Upon receiving a connection request from the relay gateway, the Mosquitto broker operating on the main gateway initiates an HTTPS connection between the integrated authentication plugin and the introspection API on the IS. The introspection API validates the legitimacy of the access token employed as the username. Upon confirmation of its validity, the API provides relevant information encompassing the client ID, scope, and expiration period embedded within the token. In the outlined method, the HOTP is encrypted utilizing AES, with the client ID of the relay gateway serving as the encryption key. This encrypted HOTP functions as the password for authentication purposes. Once authenticated, the ESP32 client ID serves as the MQTT client ID, enabling MongoDB to log every incoming message linked with the ESP32 and its corresponding communication technology.

In MongoDB, the system checks the HOTP value associated with the ESP32 chip ID. If it's not present, the system generates and stores it. After successfully verifying the access token, the authentication server supplies information about the client ID, scope, and expiration time of the access token for the relay gateway. The password is decrypted using the client ID. The decrypted HOTP within the password is compared with the stored HOTP in the database. If they match, the ESP32 relay gateway is

authenticated through the encrypted transmission of credentials from the RG to the MG. If both the access token and password are valid, the gateway issues an MQTT connack message with a zero code, signifying successful authentication of the relay gateway and establishment of the connection. The general procedure is illustrated in Figure 4.11 and detailed in Figure 4.12 as Algorithm 3.

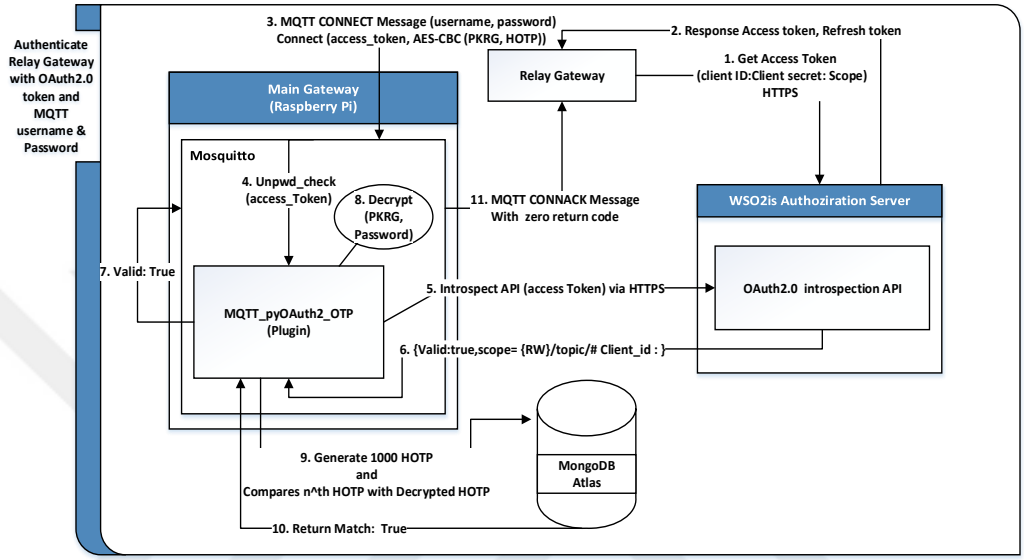


Figure 4.11 Authenticate ESP32 relay gateway by gateway on Raspberry Pi

The MQTT connect message comprises crucial details, including the username, password, Broker IP, port, and client ID. If a client ID is not explicitly specified, the protocol generates one randomly (1 up to 23 bytes). The username is denoted by the $AccessToken_{RG}$, while the password field contains encrypted $HOTP_{RG}$ information. To enhance security, the access token is granted a brief validity period, and the refresh token is utilized for new connections. Moreover, the HOTP value changes with each connection, adopting a dynamic approach to prevent static credentials and mitigate the risks of eavesdropping and replay attacks.

The comprehensive process for authenticating the relay gateway by the main gateway through the identity server is meticulously outlined in Figure 4.12 as an Algorithm 3, accompanied by explanatory notes provided below:

In the Mosquitto Gateway (MG), a Python plugin is integrated to facilitate the execution of the lightweight security mechanism. This plugin orchestrates the

invocation of functions responsible for managing security aspects within Mosquitto. Specifically, it interacts with the introspection API hosted on the IS to validate access tokens and ascertain their legitimacy, as outlined in Algorithm 3, lines 2-12.

Upon successful validation of the access token originating from the RG on the gateway's side, sensitive information such as the secret and key are retrieved through a secure channel, namely HTTPS, wherein all communication is encrypted (Algorithm 3, lines 7-14).

It's noteworthy that the key for RG (PK_{RG}) is derived from the ClientID and comprises up to 16 characters, ensuring a unique and secure identifier (Algorithm 3, line 12).

Following the validation process, if the Relay Gateway is verified, Mosquitto Gateway proceeds to send a MQTT Connack message with the code '0', indicating a successful connection establishment. Conversely, if authorization fails, the code will be '5', indicating that the client is not permitted to connect (Algorithm 3, lines 14-15).

Throughout this process, it's emphasized that the key and secret are never exchanged or transmitted in plaintext, thereby maintaining the confidentiality and integrity of the system's security measures.

Algorithm 3 Authentication of Relay Gateway by Main Gateway.

PHASE 1: RG INITIATES CONNECTION TO MG AND IS

1:RG → constructs MQTT Connect Message: *MQTT Connect = {ClientID = rand string(≤ 23 chars)||Username||Password = AES-CBC(SK_{RG}, $HOTP_{RG}$)||BrokerIP = 193.140.150.251, Port = 1883}*
2:RG → requests from IS via HTTPS: *RT_{RG}*
3:IS → sends RG via HTTPS: *RT_{RG}*
4:RG → updates Username: ***Username = RT_{RG} ▷ Received RefreshToken will replace AccessToken in future***
5:RG → sends MG: *MQTT Connect ▷ MG runs Mosquitto broker*

PHASE 2: MG INITIATES AUTHENTICATION OF RG

6:MG → receives and parses MQTT Connect: *Username'*
7:MG → calculates credentials: *introspectURL = "https : //localhost : 9443/oauth2/introspect", introspectUsername = "admin", introspectP assword = "admin", Username = Username' ▷ Username' parsed from received MQTT Connect*
8:MG → calls introspectAPI from IS via HTTPS: *introspectAPI.call(Username' , introspectURL)*

PHASE 3: IS AUTHENTICATES RG

9:IS → verifies RG with introspectAPI via RT_{RG}: *if (RT_{RG} ∈ RT database) then verified*
10:IS → looks up ScopeRG from local DB: *Scope = Base64decode(CodedScope)*
11:IS → returns introspectAPI to MG via HTTPS: *introspectAPI.return(True, $ClientID_{RG}$, $Scope_{RG}$)*

PHASE 4: MG AUTHENTICATES RG

12:MG → receives and parses introspectAPI response: *$HOTP_{RG}$ = DecryptText5 = AES-CBC(SK_{RG}, Password)*
13:MG → calculates $HOTP_{RG}$: *$HOTP_{RG}$ = HOTP($ClientID_{RG}$, C)*
14:MG → verifies RG with $HOTP_{RG}$: *if ($HOTP_{RG}$ == HOTP ' $_{RG}$) then verified*
15:MG → sends to RG: *MQTT Connack(0) ▷ Code '0' indicates successful connection*

PHASE 5: RG PUBLISHES DATA SOURCED FROM END NODES

16:NodeWRG, NodeBRG, NodeZRG → collect data from end nodes
17:NodeWRG → constructs data message: *CipherText6 = $Payload_{WRG}$ = AES-CBC($SK_{NodeWRG}$,AT_{NodeW})||$HOTP_{NodeW}$||data,$Topic_{NodeW}$="PainlessMesh/to/gateway"*
18:NodeBRG, NodeZRG → constructs data message: *CipherText5 = $Payload_{BRG,ZRG}$ = AES-CBC($SK_{NodeBRG,NodeZRG}$, $ID_{NodeB,NodeZ}$)||$HOTP_{NodeB,NodeZ}$ ||data, $Topic_{NodeB,NodeZ}$ = "BLE,ZigBee/to/gateway"*
19:NodeWRG, NodeBRG, NodeZRG → send MQTT Publish Message: MQTT Publish = {Topic||Payload}

Figure 4.12 Algorithm 3: Authentication of relay gateway by main gateway

4.2.4 Authorization Stage

A client can transmit messages associated with its specified topic(s) to the Mosquitto broker, thereby identifying itself as a publisher (of the specified topic). Furthermore, the client can solicit incoming messages from the broker pertaining to its topic(s) of interest, signifying its role as a subscriber (of the specified topic). Consequently, selective data forwarding plays a crucial role in MQTT. Access control

management is enforced to restrict the reception of data to only the relevant clients. Clients are granted access solely to data corresponding to the topics they have expressed interest in.

The authorization procedures are consistent across all three protocols. When an MQTT client (such as RG) sends a PUBLISH message, it includes both a topic and a payload. The topic functions as a string identifier representing the message subject, whereas the payload carries the actual data being transmitted. While the payload can comprise any data type, UTF-8 strings are frequently utilized for textual data. Please note that the upper limit for an MQTT message size stands at 268435455 bytes, roughly equivalent to 256 MB. This restriction, as delineated in the MQTT standard, encompasses both the message payload and any accompanying headers or metadata.

Upon receipt of an MQTT publish message, the MQTT broker (specifically Mosquitto, operating on MG) parses the message header to extract information regarding the topic and payload. A custom plugin has been developed to intercept messages and execute additional tasks, such as payload encryption or decryption if required. At this phase, the plugin decrypts the payload utilizing SK_{RG} and then parses it to retrieve SK_N, $HOTP_N$, and the message content. Following verification of $HOTP_N$ with MongoDB, access control procedures are initiated. The plugin utilizes the introspection API function for access control, leveraging the scope to ascertain which clients are authorized to publish messages on specific topics, in accordance with the OAuth 2.0 protocol.

Mosquitto utilizes MongoDB for message storage purposes. Upon receiving a message, Mosquitto logs it into the database, capturing metadata such as arrival time and relevant status information. Additionally, the system monitors message loss rates to assess the overall health of the system. The message is accepted if the node's topic name and scope information are correctly aligned, and the node possesses writing permission. Subsequently, the message is recorded in MongoDB, and the message's arrival time and loss status from the node to the multi-protocol gateway are analyzed. Furthermore, the plugin verifies access control for an MQTT publish message and records the message in MongoDB. This process involves extracting the topic name

and payload information from the MQTT Publish packet and using them to determine access permission and topic information.

The comprehensive process for authenticating the relay gateway by the main gateway through the identity server is meticulously outlined in Figure 4.13 as an Algorithm 4, accompanied by detailed explanatory notes provided below:

Algorithm 4 Authorization and Access Control for End Nodes by Relay Gateway and Main Gateway

PHASE 1: MG COLLECTS DATA MESSAGES FROM RG

1:MG → decrypts received CipherText6 ′ : *DecryptT ext6 = AES($SK_{NodeWRG}$, CipherText6 ′) ▷ CipherT ext6 ′ comes from NodeWRG (see Algorithm 3-17)*
2:MG → decrypts received CipherText5 ′ : *DecryptText5 = AES(SK_{NodeBR}G,NodeZRG, CipherText5 ′) ▷ CipherText5 ′ comes from NodeBRG, NodeZRG (see Algorithm 3-18)*
3:MG → parses DecryptText6: *DecryptPayload = AES(SKNodeWRG, Payload), HOTPN′ = DecryptPayload[6](6bytes), UsernameN′ = AccessT okenN′ = DecryptPayload[7..39](32bytes), CipherT ext4′ =Message′=DecryptPayload[39..len(DecryptPayload)],Topic′$_{NodeW}$ = {“Topic”: “PainlessMesh/to/gateway”}*
4:MG → parses DecryptText5: *DecryptPayload = AES($SK_{NodeBRG,NodeZRG}$, Payload), HOTPN′ = DecryptPayload[6](6bytes), IDN′ = DecryptPayload[7..34](28bytes), CipherText3′ = Message′ = DecryptPayload[34..len(DecryptPayload)], Topic′ $_{NodeB,NodeZ}$ = {“Topic”: “BLE,ZigBee/to/gateway”}*

PHASE 2: TOPIC-BASED AUTHORIZATION OF END NODES

5: MG → calls introspectAPI from IS via HTTPS: *introspectAPI.call(Username′ , introspectURL)*
6:IS → returns introspectAPI to MG via HTTPS: *introspectAPI.return(True, $ClientID_{NodeW,NodeB,NodeZ}$, $Scope_{NodeW,NodeB,NodeZ}$)*
7:MG → calculates $HOTP_N$: *$HOTP_N(ID_N, C)$*
8:MG → verifies NodeW via HOTP′: *if ($HOTP_N$ == $HOTP_N$′) then verified*
9:MG → parses $Scope_{NodeW,NodeB,NodeZ}$: *$Topic_{NodeW,NodeB,NodeZ}$ = Base64encode(“rw”:“rw”,“Topic”:“PainlessMeshBLE,ZigBee/to/gateway”)*
10:MG→ verifies Topic′ for authorization: *if Topic′ NodeW,NodeB,NodeZ == TopicNodeW,NodeB,NodeZ then verified ▷ End nodes authorized for that topic*
11:MG → decrypts CipherText3 ′ : *DecryptText5∗ = AES($SKNode_{BRG,NodeZRG}$, CipherText3′)*
12:MG → decrypts CipherText4 ′ : *DecryptText6∗ = AES($SK_{NodeWRG}$, CipherText4′)*
13:MongoDB → records: *Message ▷ End of all procedures*

Figure 4.13 Algorithm 4: Authentication of relay gateway by main gateway

- The payload undergoes decryption using AES with the key derived from ID_{RG}. Following decryption, the $HOTP_N$ value, along with the access token and message, is extracted from the payload. The scope of the access token, obtained via base64 decoding, encompasses access permissions and topic information.
- MG confirms the validity of the access token sent by NodeW in the IS (Alg. 4, lines 5-8).

- MG compares the received topic to the current topic record and verifies if the node has been granted publishing privileges for that topic; if both conditions are met, the node is authorized (Algorithm 4, line 10).

4.3 Outlook of Secure Tier 4: Digital Signing for Enhanced Data Integrity

The scope of the thesis has also taken into account the integrity of the data with the highest security level: digital signature. Elliptic Curve Digital Signature Algorithm (ECDSA), one of the asymmetric encryption methods, is a cryptographic algorithm used for generating digital signatures. It is responsible for generating and verifying signatures using private and public keys. The mathematical properties of points on elliptic curves form the basis of cryptographic algorithms like ECDSA. These properties, such as point addition, inversion, and scalar multiplication, enable ECDSA to efficiently and securely generate digital signatures. The security of ECDSA relies on the "elliptic curve discrete logarithm problem," which is mathematically complex. This problem involves the difficulty of finding the position of a point on an elliptic curve, and currently, the most effective solutions are generally very expensive in terms of computational power. Therefore, ECDSA is a secure and widely accepted digital signature algorithm. Consequently, we chose the ECDSA algorithm for security tier 4, especially for our ESP32 devices with limited resources and applications requiring high security.

4.3.1 Implementation of ECDSA on ESP32

In the implementation phase, the fundamental steps of key generation, signing a message, and verifying the signature using the ECDSA algorithm have been addressed. Additionally, within the scope of the multi-protocol multi-gateway design discussed in Chapter 3, the ECDSA algorithm has been designed for communication between nodes and the Relay gateway. Authentication and authorization between the Relay gateway and the main gateway are based on token-based OAuth 2.0 protocol.

Our ESP32 Arduino code involves using the mbedtls library to generate a key pair with ECDSA, creating random numbers using entropy and CTR_DRBG (Counter Mode Deterministic Random Bit Generator) (Alg. 5 line 1-2), computing the SHA-

256 digest of a message, generating a digital signature with the private key, and verifying the authenticity of this signature with the public key. The comprehensive process of the ECDSA algorithm is delineated in Algorithm 5 specified in Figure 4.14.

Algorithm 5 Key generating, signing and verifying

1: Nodes → Init entropy: *Randomness is necessary for the process of key generation.*
2: Nodes → Init CTR-DRBG: **Counter Mode Deterministic Random Byte Generator is a type of random number generator used to produce random numbers*
3: Nodes → Use SECP256R1: *Elliptic Curve Cryptography specifies the curve*
4: Nodes → Generate private key: d_{NodeW1}, d_{NodeW2}, d_{NodeW3}, d_{NodeB}, d_{NodeZ}
5: Nodes → Generate public key: $Q_{NodeW1} = d_{NodeW1} X G$ *(G is a starting point on the curve)*
6: Nodes → Calculate hash value of the message: $h = H(m)$
7: Nodes → Select random number: k
8: Nodes → Calculate point: $(x1, y1) = k X G$ *(G is a point on the curve and is a general parameter)*
9: Nodes → Calculate r: $r = x1 \bmod n$ *(n is the number of points on the curve)*
10: Nodes → Calculate s: $s = k^{-1} X (h + r X d_{NodeW1}) \bmod n$ *(d is the private key, and k^{-1} is the modular inverse of k with respect to h, r, d, and n)*
11: Nodes sends the signature (r, s), message (m) and public key (Q)
12: NodeRG → Calculates hash value of the message: $h = H(m)$
13: NodeRG→ Checks for the given signature (r, s): *verify that s is non-zero and less than n, and validate that the values of r and s are valid.*
14: NodeRG→ Calculates w: $w = s^{-1} \bmod n$
15: NodeRG→ Calculates u1 and u2: $u1 = h X w \bmod n$, $u2 = r X w \bmod n$
16: NodeRG→ Calculates (x1, y1) = $(x1, y1) = u1 X G + u2 X Q_{NodeW1}$
17: NodeRG→ Verifies signature = $r = x1 \bmod n$ *(If this condition holds, the signature is verified).*
** For the SECP256R1 curve, the n and G values are specified in a standard established by the National Institute of Standards and Technology (NIST), known as FIPS PUB 186-4.*

Figure 4.14 Algorithm 5: Key generation, signing and verification

The function mbedtls_ecdsa_genkey is called to generate a private key pair. This function creates a private key pair using the ECDSA algorithm on a specified curve. In our application, the SECP256R1 curve is used (Alg. 5 line 3).

Mathematically, the private key pair is expressed as (d, Q) (Alg. 5 line 4-5), where:

- d is the private key. It is a random number and is typically selected within a certain range (e.g., [1, n-1]), where n is a large prime number for a specific curve.
- Q is the public key and is a point on the curve. Q = d X G, where G is a starting point on the curve (usually referred to as the "generator point"). This point is calculated using the group structure defined by a specific curve.

The signing process of ECDSA begins by taking a hash of the message to be signed (Alg. 5 line 6). Then, a random number is generated using this hashed value and the private key. This random number represents a point on the curve, and its x-coordinate is used as the r value in the signature (Alg. 5 line 7-8-9). Next, during the signing process, the product of the r value calculated and the private key is added to the hash of the message, and then multiplied by another random number. The resulting value forms the s value (Alg. 5 line 10). The combination of r and s values obtained through these steps represents the signature of the message.

The verification process involves taking the hash of the signed message and using the r and s values from the signature for the verification process (Alg. 5 line 13). A point calculated using the r value is combined with another point derived from the public key for a specific mathematical operation (Alg. 5 line 14-15-16). The resulting r value is used to verify the validity of the signature. If the calculated r value matches the r value in the signature, the signature is deemed valid, and the message's integrity is confirmed (Alg. 5 line 17).

4.3.2 Evaluating the performance of the ECDSA

The thesis ensures message integrity and security with the ECDSA algorithm, augmented to provide security tiers 2 and 3. In addition to encryption with AES, the message is further enhanced by appending signature information before transmission. Performance evaluation was conducted to measure the durations of key pair generation, message signing, and verification processes of the ECDSA algorithm. These durations were separately measured for different message sizes (16, 32, and 64 bytes). The measurements were taken as averages for each operation. With these measurements, it is possible to observe how much delay the messages introduce to the transmission time.

In Table 4.5, t_E represents the time taken for symmetric encryption (specifically AES). T_h indicates the duration required for the SHA256 algorithm used in the ECDSA algorithm to detect whether a message has been modified by a third party. Additionally, t_{keyGen} specifies the time needed for a Node to generate the public-private key pair using entropy, a random number, and the SECP256R1 curve. $t_{signing}$ refers to

the time required to sign the message using the public key. t_{BLE} represents the one-way transmission time in BLE communication, while tprocess encompasses the time taken for a Node to encrypt and then sign the message before sending it via BLE communication. This duration will vary for Wi-Fi and ZigBee communication. $t_{verifying}$ denotes the time taken to verify the signature of the received message by NodeRG. In the Node side, using the SECP256R1 curve with the ECDSA algorithm, key generation and message sending, followed by verification on the NodeRG side, the total elapsed time is indicated as the total span time. The values are averaged over 20 repeated runs and are presented in milliseconds (ms). Appendix 3 contains information on key generation time, encryption time (in microseconds), message lengths, encrypted (AES) messages, hashing time, signing time, message (HEX), public key X (HEX), public key Y (HEX), signature (HEX), BLE RTT (in microseconds), process time (in milliseconds), received message, signature validity, and verification time (in milliseconds) for various payloads. This appendix provides detailed insights into the performance of cryptographic operations under different payload conditions.

Table 4.5 Approximate time estimates in ms for various cryptographic operations on ECDSA

Operation	**Symbol**	**16 bytes**	**32 bytes**	**64 bytes**	**128 bytes**
AES	t_E	0.045	0.052	0.054	0.063
SHA256	t_h	0.089	0.094	0.105	0.117
Generate key	t_{keyGen}	219	219	219	219
Sign	$t_{signing}$	100	102	107	114
Process	$t_{process}$	147	151	179	257
$BLE_{Write+ACK}$	t_{BLE}	1.848	2.251	2.479	4.043
Verification	$t_{verifying}$	512	588	590	591
Total span time (Node)	$(t_E + t_h + t_{keyGen} + t_{signing})$	319	321	326	333
Total span time (NodeRG)	$t_{verifying}$	512	588	590	591

Using BLE, operations such as encryption, digital signing, and public key sharing are performed. To calculate the communication cost, the size of the transmitted data needs to be determined. This data consists of the message, public key, and signature. In the test environment, messages of 16, 32, 64, and 128 bytes are encrypted using

AES 128-CBC before signing, resulting in message sizes of 16, 32, 64, and 128 bytes, respectively.

On the SECP256R1 curve, a public key is an elliptic curve point, usually represented in compressed or uncompressed format. In our application, the public key is represented by two coordinates (x, y), each 256 bits (32 bytes) long. Therefore, the total size of an uncompressed public key is 64 bytes (32 bytes x 2) plus 1 byte for the format indicator at the beginning, resulting in a total of 65 bytes.

A private key is used to authenticate the identity of a person or entity and to sign digital transactions. It is usually randomly selected and typically is a 256-bit long number. Each private key corresponds to a point on the SECP256R1 curve.

A signature is used to authenticate the message and ensure its integrity. On the SECP256R1 curve, signatures are 72 bytes long. This signature is created by an algorithm using a given private key and is used to verify the authenticity and integrity of the message, ensuring it was created by the sender and has not been altered.

CHAPTER FIVE
PERFORMANCE EVALUATIONS

In this section, we conduct a comparative analysis of the performance implications of our proposed security scheme across different communication technologies. This analysis utilizes a uniform set of resource-constrained devices, specifically ESP32 and Raspberry Pi. We draw references from our previous study, MARAS (Şeker et al., 2023) to emphasize any disparities. The assessment concentrates on the runtime performance of the proposed security scheme, SIMMS. We assess encryption overhead, computational burden, and communication efficiency on these devices, providing valuable insights into its scalability and efficacy.

5.1 Test Configuration for Secure Tiers 2 and 3

Utilizing SIMMS within the introduced network setup, we measured the transmission times of messages with different data sizes by periodically transmitting them at random intervals of 9600 baud rate for BLE with 5 different connection interval, for ZigBee with 3 different connection interval, and for Wi-Fi with random intervals ranging from 1 to 5 and 3 to 7 seconds. This process was repeated five times, and the results were averaged. We examined how the frequency of message transmission (i.e., sensor data) and the growth payloads introduced by encryption affected the message delay in the network. Below are the main configurations applied to the testbed, which also describe the test scenario:

- Prior to accepting any node connection requests, an MQTT connection is established between the relay gateway (RG) and the main gateway (MG) via Wi-Fi.
- In RG, enable NodeWRG's Wi-Fi mesh connection and commence advertisement scans between NodeBRG and NodeB. These scans are conducted at intervals of every 5 seconds until a device is discovered. This process is repeated in the event of a disconnection of an existing BLE device. If there is no activity within 30 milliseconds during the authentication steps, the devices are rebooted, and the authentication steps are repeated. The ZigBee coordinator within RG (NodeZRG) generates a unique personal area network (PAN) ID,

which will be utilized by other devices within the ZigBee network for communication. This PAN ID must be distinct from those used by all other devices within the network. NodeZ, which is an end device, must be configured with the same PAN ID to join the network successfully. Additionally, NodeZ should include the destination address information of the coordinator (NodeZRG) to ensure proper connection establishment. During the network joining process, NodeZ scans for the PAN ID and searches for the appropriate coordinator. XBee modules scan for each other at specified intervals (e.g., every 2500 ms) to complete the network joining process. Throughout this process, devices conduct scans to discover and establish communication with the coordinator.

- In Painless Mesh, all nodes utilize a shared SSID, which is distinguished by their BSSID. This shared SSID is accompanied by a network password, and the mesh server operates on a default TCP port of 5555.
- In the mesh architecture, when a device seeks to join the network, the reception of data from all nodes is halted until the identity of the node is authenticated, as outlined in Algorithm 1-4. Upon successful authentication, synchronization is achieved among all devices listed on the whitelist.
- Every 3 seconds, NodeWRG examines the network, eliminates any disconnected nodes from the whitelist, and proceeds to re-synchronize the system.
- BLE operates with varying connection intervals, remaining in sleep mode when not actively transmitting data to reduce power consumption. These connection intervals are configured as multiples of 1.25 ms. In our experiments, we utilized intervals of 7.5, 40, 80, 100, and 150 ms. Initially, messages were transmitted 100 times at these intervals, and then increased to 500 times to evaluate data reliability under both encrypted and non-encrypted conditions.
- Determine the throughput for both secure and non-secure scenarios considering one-way transmission, which encompasses data transmission and notification times (NodeBRG signaling new data). While ideally throughput should be independent of the connection interval, it is factored in due to varying message counts across different intervals. Incoming messages are stored in the queue within individual RG modules and are processed from the queue whenever

possible to prevent delays. If there are no incoming messages, data is fetched from the queue and sent to the main gateway via MQTT.

- ZigBee message transmission intervals are configured at 80, 100, and 150 ms to facilitate comparison with BLE. However, in an encrypted scenario, frequent data transmission every 40 ms led to notable message loss.

The NRF52840, a compact USB dongle, played a crucial role in our experiments. This cost-effective network device facilitates data transmission and monitoring across various wireless protocols including Bluetooth 5.4, Bluetooth mesh, Thread, ZigBee, 802.15.4, ANT, and other proprietary protocols operating at 2.4 GHz. We selected this device for our study because of its affordability, extensive support for all short-range wireless standards employed in nRF devices, user-friendly interface with PCs through nRF Connect, and compatibility with Wireshark integration.

5.2 Complexity Discussion for Secure Tiers 2 and 3

The time complexity of the process for registering Nodes and RG to WSO2IS to acquire access tokens, depicted in Algorithm 1, is determined to be O(N), where N represents the number of clients in the network. This is because the registration process needs to be iterated for each ESP32 client device. The complexity of establishing a connection between a single Mosquitto broker and WSO2IS via HTTPS to validate an access token, as depicted in Algorithm 3, is O (1); however, this process can be executed concurrently if there are multiple brokers. In the authentication process depicted in Figure 4.12, the second step, where a broker authenticates a RG using MongoDB records, necessitates N calculations for HOTP if the client is not registered in MongoDB. This operation must be repeated for each of the N clients, resulting in a complexity of $O(N^2)$. As introduced in Figure 4.10 Algorithm 2: Authentication of end nodes by relay gateway and data transfer the client's process of verifying the corresponding broker (for mutual authentication) necessitates N calculations of HOTP. Although this step is performed for each RG, the execution is parallel on each client device, resulting in an O(N) complexity. Therefore, the time complexity for a PUBLISH message (in Algorithm 4) remains at $O(N^2)$ as it will be the dominant term in the sum of all partial complexity classes mentioned above. The time complexity of encryption and decryption operations with AES on both clients and brokers is O(1),

and thus can be neglected. However, the message complexity for both types of messages turns out to be O(M), where M represents the message to be sent.

5.3 Comparative Analysis for Secure Tiers 2 and 3

Continuing with the runtime performance analysis of SIMMS obtained in our tests, we provide a brief comparative overview referring to similar schemes from the literature. The information presented in this section is not a comprehensive analysis due to the non-equivalence of test environments, yet it offers insights into how SIMMS compares to alternatives in terms of computation and communication overhead. Additionally, it highlights the increased cost on resource-constrained devices compared to our study MARAS (Şeker et al., 2023) applied on advanced devices. Our focus is limited to the mutual authentication phase of SIMMS and MARAS since the registration phase is manually performed once by a developer. For this purpose, the computation and communication costs of both SIMMS and MARAS (Şeker et al., 2023) are compared to those of (Patel & Doshi, 2023) and (Saqib, Jasra, & Moon, 2022).

The primary objective behind the design of MARAS (Şeker et al., 2023) is to reduce the processing burden on publisher and subscriber clients with limited resources, while simultaneously executing computationally demanding tasks on the broker(s). The computational expense in MARAS focuses on the quantity of cryptographic operations involved during mutual authentication and the duration needed for these operations on both client and broker ends.

In our MARAS study, on our test PC with an Intel i7 3630QM processor running Ubuntu OS, the time for an Eclipse Paho subscriber/publisher client and a broker to complete the given operations is as follows: T_E, the time for symmetric encryption (i.e., AES in MARAS), is 7ms; T_D, symmetric decryption equivalent to encryption; T_H, time for a one-way hash function, is 0.35 ms; T_{HOTP}, time for calculating HOTP(s), is 0.42 ms; and T_{AS}, time for token-based authentication steps performed with introspection API over HTTPS, is 10.4 ms. In the notation used for Table 5.2, T_{ECA} represents ECC point addition, T_{ECM} represents ECC point multiplication, and so on.

The runtime performance of SIMMS, which we examined within the scope of the thesis, is detailed in Table 5.1 and Table 5.2.

5.3.1 Computational Costs

Table 5.1 outlines the duration required by a node (i.e., end nodes, RG modules, and MG) to execute various common operations essential for the proposed scheme on our testbed. The testbed employed ESP32s for end nodes and RG modules, while the MG utilized a Raspberry Pi 4 running Raspbian OS, featuring an ARM Cortex-A72 quad-core processor.

Table 5.1 Approximate time estimates in ms for various cryptographic operations on SIMMS

Operation	Symbol	Time to Complete (ms)						
		NodeB	**NodeZ**	**NodeW**	**NodeBRG**	**NodeZRG**	**NodeWRG**	**MG**
AES	(t_E | t_D)	0.027	0.024	0.044	0.054	0.036	0.092	0.621_B 0.492_Z 0.724_W
HOTP	(t_{HOTP})	0.214	0.214	0.214	0.230	0.230	0.230	0.461
Hash	(t_H)	0.351	0.351	0.351	0.351	0.351	0.351	0.610
Access Control	(t_{AC})	N/A	N/A	N/A	N/A	N/A	N/A	136.43
Scan QR code	(t_{CAM})	N/A	N/A	N/A	0.082	0.082	0.082	N/A
Authentication	(t_{Auth})	10	84	801	107	166	1010	0.113
MQTT Publish	(t_{PubM})	N/A	N/A	N/A	3	2	3.454	N/A
IntrospectionAPI	(t_{AS})	N/A	N/A	N/A	942	942	942	71.862

**** NA: Does not include the operation.***

In both Table 5.1 and Table 5.2, t_E represents the time taken for symmetric encryption (specifically AES), while t_D stands for the time needed for symmetric decryption, which is equivalent to encryption. t_H indicates the duration required for the one-way hash function, and t_{HOTP} denotes the time taken for calculating HOTP(s). Additionally, t_{AC} signifies the time spent on authorization on Mosquitto (MG), and tAS reflects the time required for token-based authentication steps performed with the introspection API over HTTPS. Following the token-based verification between RG and MG, the duration to publish a message is denoted as t_{PubM}, while the time required to manage the access permission of the published message is represented by t_{AC}. Moreover, t_{CAM} stands for the duration taken for ESP32 devices to scan and read a QR

code. T represents the time in milliseconds for QR code scanning and mutual authentication steps at the relay gateway. The values were averaged after 10 runs.

Table 5.2 Computation costs comparison in ms

References	End Nodes	Relay Gateway	Main Gateway	Time (ms)
Şeker et. al, 2023 (MARAS)	$t_E + t_{HOTP} + t_H$	N/A	$2t_D+2t_{HOTP}+202t_H+2t_{AS}$	121.8
Patel et al., 2023	$2 \times (6T_H + 2T_E)$	N/A	$13T_H + 6T_E$	792
Saqib et al., 2022	$2T_{ECM} +4T_H + 3T_{ECM} + 1T_H + 2T_{ECA}$	N/A	$5T_{ECM} + 2T_H + 2T_{ECA}$	190
Wi-Fi (SIMMS)	$2t_{HOTP}+3t_H + 2t_D+2t_E$	$t_{AS}+t_{HOTP}+2t_H+4_{tE}+t_D+t_{CAM}+2t_{PubM}$	$2t_{AS}+ 101t_{HOTP} + 3t_D$	1146
BLE and ZigBee (SIMMS)	$2t_{HOTP} +t_D + 2t_E$	$t_{AS}+2t_{HOTP}+3t_E +t_D +t_{CAM}+2t_{P\ ubM}$	$t_{AS} + 101t_{HOTP} + 3t_D$	1069 and 1066
** NA: Does not include multi-protocol gateway - (relay gateway)*				

For an end node to conduct data transmission within the network, it needs to undergo authentication by the relay gateway. The authentication steps are detailed in Section 4. Table 5.2 provides a comparative view of the number and types of operations required for different communication technologies and the time needed to complete the entire set of introduced authentication procedures. Additionally, the table offers a performance comparison with a previous scheme, MARAS (Şeker, Dalkılıç, & Çabuk, 2023), which we implemented on advanced devices. The notable increase in operations is due to the additional authentication steps involved.

5.3.2 Communication Costs

Table 5.3 presents a comparison of communication costs in terms of transmitted bits in a scenario comprising a slave node (each of NodeW, NodeB, NodeZ), a relay gateway (acting as a publisher), and a broker (main gateway). Consequently, a slave node joins the network with authentication, while the relay gateway seeks authorization to transmit the slave's data to the server through the broker.

Table 5.3 Communication costs comparison in bits

Protocol	Node-to-RG		RG-to-MG		MG-to-IS	Total
	Msg. Ct.	**Cost (b)**	**Msg. Ct.**	**Cost (b)**	**Cost (b)**	**Cost (b)**
Wi-Fi	5	1488	3	1184	2672	5344
BLE, ZigBee	4	704	3	1224	2672	4600
MARAS	N/A	N/A	3	2432	2672	5104
**** NA: Does not include communication between components.***						

In Wi-Fi communication, mutual authentication entails a 42-byte QR code (comprising 6 bytes for HOTP, 16 bytes for the access token, and 20 bytes for the hash(ID)), along with a total of 144 bytes of AES encryption output across 5 stages. Conversely, in both BLE and ZigBee communication, mutual authentication involves a 34-byte QR code data (including 6 bytes for HOTP and 28 bytes for ID), as well as a total of 38 bytes of AES encryption output. At the relay gateway and main gateway (i.e., broker), a MQTT CONNECT message encompasses 73 bytes (equivalent to 584 bits). This includes 1 byte for header flags, 1 byte for message length, 2 bytes for protocol name length, 4 bytes for protocol name, 1 byte for protocol version, 1 byte for connect flag, 2 bytes for keep alive, 2 bytes for client ID length, up to 23 bytes for client ID value, 2 bytes for username length, 16 bytes for username value (i.e., token), 2 bytes for password length, and 16 bytes for password value. The CONNACK message serves as a response from the main gateway (MG) to the relay gateway (RG) within the MQTT protocol, following a connection request. This message signifies whether the connection request has been successfully accepted or not. The CONNACK message consists of a 2-byte response code. Furthermore, the PUBLISH message comprises 73 bytes, including 2 bytes for the fixed header, 23 bytes for the topic, and 48 bytes for the payload. Communication via ZigBee and BLE, as detailed in Table IV, fluctuates based on the length of the topic and payload. Both the main gateway (MG) and the relay gateway (RG) transmit a combined total of 334 bytes (equivalent to 2672 bits) of authentication server data over TLS. Subsequently, they dispatch a confirmation to the client totaling 2 bytes (16 bits).

5.4 Throughput, Message Reliability and Delay Time for Secure Tiers 2 and 3

While we've detailed the separate costs for computation, data, and communication, looking at the achieved throughput in practice could give us better insight into overall performance. In the BLE, ZigBee, and Wi-Fi mesh network setup, messages from the end nodes are meant for the relay gateway. Once received there, they're sent to the main gateway via MQTT. Each incoming message gets encrypted before publication and then forwarded to the main gateway, which uses the Mosquitto broker. Since BLE and ZigBee work similarly in our setup, we combined their results, while the Wi-Fi mesh results were shown separately.

The tests mentioned in this section involved several components: the BLE end node (referred to as NodeB), the BLE module of the relay gateway (referred to as NodeBRG), the ZigBee end node (referred to as NodeZ), the ZigBee module of the relay gateway (referred to as NodeZRG), and the main gateway (referred to as MG). While MG is not directly involved in BLE or ZigBee communication lines, it connects to the NodeBRG and NodeZRG modules of the relay gateway via Wi-Fi (as shown in Figure 4.13). These tests aimed to assess packet delivery performance under varying conditions, including different message sizes, time intervals, and payload sizes. For this, Appendix 2 contains information on message reliability, throughput, and transmission time for BLE and ZigBee, based on different connection ranges, various payloads, and different message sending frequencies. This appendix provides detailed insights into the performance characteristics of BLE and ZigBee under varying conditions.

Within the network, BLE communication primarily involves 2 ESP-32 nodes, with initiation from the relay gateway (NodeBRG). The end node (NodeB) begins advertising across the three available BLE channels. NodeB continuously broadcasts advertisements, and upon receiving the advertising PDU, NodeBRG notifies by transmitting the CONNECT IND PDU. The slave (NodeB) hosts the generic attribute (GATT) server, while the master (NodeBRG) acts as the GATT client. The master is configured to discover and connect to a single slave, adjusting the characteristics of the slave using the notification attribute protocol.

Figure 5.1 depicts the structure of the CONNECT IND message sent by the NodeBRG and the process of establishing a connection with the node. Upon sending the CONNECT IND message, the relay gateway initiates a transmission window of WindowSize. WindowDelay remains constant, while the CONNECT IND message contains details about WindowOffset and WindowSize. The initial packet transmitted within this transmission window serves as the reference point for the first connection event. The node remains in listening mode until it receives the first connected packet from the center. Subsequently, both devices periodically initiate connection events at specific intervals and take turns transmitting data packets within these connection events.

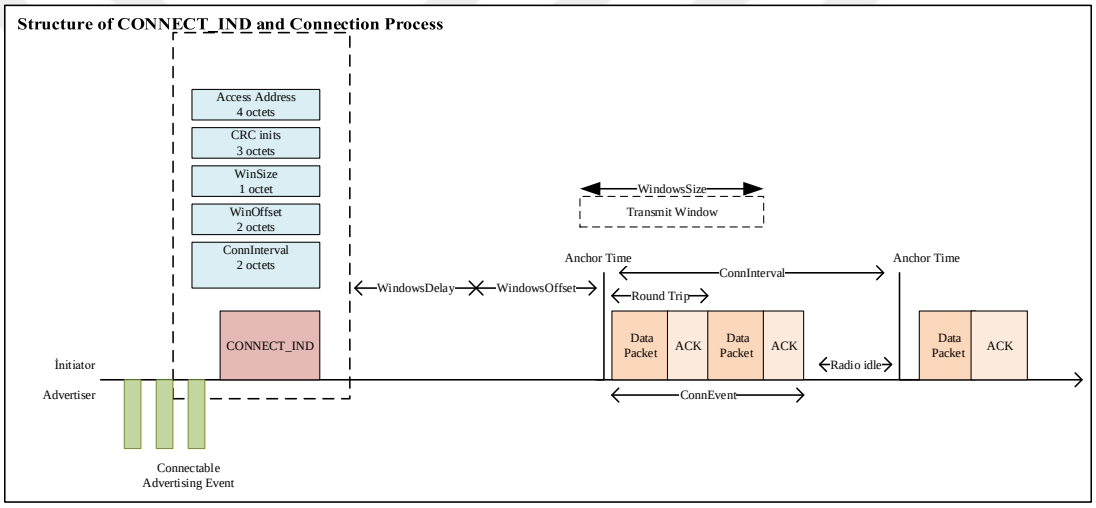


Figure 5.1 BLE packet structure

The process of data transfer from the NodeBRG to the NodeB is analyzed using notifications. This allows data to be sent from the node to the relay gateway without the need for the relay gateway to verify the receipt of the data. The term ConnEvent refers to the time during which data packets are exchanged between the node and relay gateway, encompassing the round-trip times of data packets and their acknowledgments (ACKs). The remaining time, when communication is inactive, is referred to as Radio Idle. The anchor point marks the beginning of a ConnEvent, where the relay gateway initiates the transmission of packets to the node. The wait time between packets is expressed as time inter-frame space (TIFS). The Connection interval (ConnInterval) represents the time between the beginnings of two consecutive

ConnEvents. This time is a multiple of 1.25 ms and falls within the range of 7.5 to 4000 ms (Bluetooth SIG, 2019). The BLE packet consists of a header, an access address, protocol data unit (PDU), and cyclic redundancy check (CRC), which are 1, 4, (2~257), and 3 bytes respectively. The transmission times for the master (T_M) and slave (T_S) packets are as follows:

$$T_s = T_{pr} + T_{aa} + T_{llh} + T_{CRC} + T_{PDU} + T_{op} + T_{ATTh} + T_{L2CAPh} \quad (5.1)$$

$$T_M = T_{pr} + T_{aa} + T_{llh} + T_{CRC} \quad (5.2)$$

TPr, Taa, Tllh, TCRC, Top, TATTh, TL2CAPh, and TPDU represent the transmission durations of the following packet components: Preamble, Access Address, Link Layer Header, CRC, ATT Opcode, ATT Handle, L2CAP Header, and PDU, respectively.

$$T_{SM} = T_S + T_{IF} + T_M + T_{IF} \quad (5.3)$$

Once pairing is established, the end node sends data messages of 16, 32, and 64 bytes to its counterpart within the relay gateway, both in plain text and encrypted forms (using 128-bit AES). For testing, we consider five different transmission intervals: 7.5 ms, 40 ms, 80 ms, 100 ms, and 150 ms, permitting only one write data operation from the slave side. The procedure is replicated for larger payload sizes, both in AES-encrypted and unencrypted formats.

Throughput is quantified as the volume of data messages (such as sensor data) transmitted from an end node to the relay gateway within one second. In the case of BLE, this refers to the transmission from NodeB to NodeBRG, while in ZigBee, it pertains to the transmission from NodeZ to NodeZRG. According to the setup, NodeBRG does not transmit any data to NodeB. However, as per the BLE protocol, the master must regularly send packets to initiate communication or enable the slave to transmit more data. Consequently, all packets transmitted from NodeBRG to NodeB are considered as overhead. Given that each packet carries varying amounts of application data, the throughput (T) can be mathematically expressed as follows:

$$T(bps) = N_{round} \times Payload(B) \times 8/ConnInterval \quad (5.4)$$

where the maximum theoretical number of round-trips (N*round*) per connInterval can be calculated as the floor of the ratio of connInterval and troundtrip:

$$N_{round} = (connInterval/t_{roundtrip}) \quad (5.5)$$

Equations 5.4 and 5.5 allow us to compute the packet count for five distinct connInterval values: 7.5 ms, 40 ms, 80 ms, 100 ms, and 150 ms. The throughput, in this context, is defined as the number of bits per connInterval, divided by the connInterval itself. Figure 5.2 illustrates the throughput performance of both BLE and ZigBee nodes across various connection intervals (80 ms, 100 ms, and 150 ms). Notably, ZigBee encountered prohibitively high packet loss rates at 7.5 ms and 40 ms, rendering those intervals impractical, and thus they were omitted from the ZigBee data plotted on the chart. Across different experimental runs, data payload sizes of 16, 32, and 64 bytes were utilized. Additionally, the breakdown includes results obtained with and without the implementation of the proposed security mechanism, offering insights into its impact on performance.

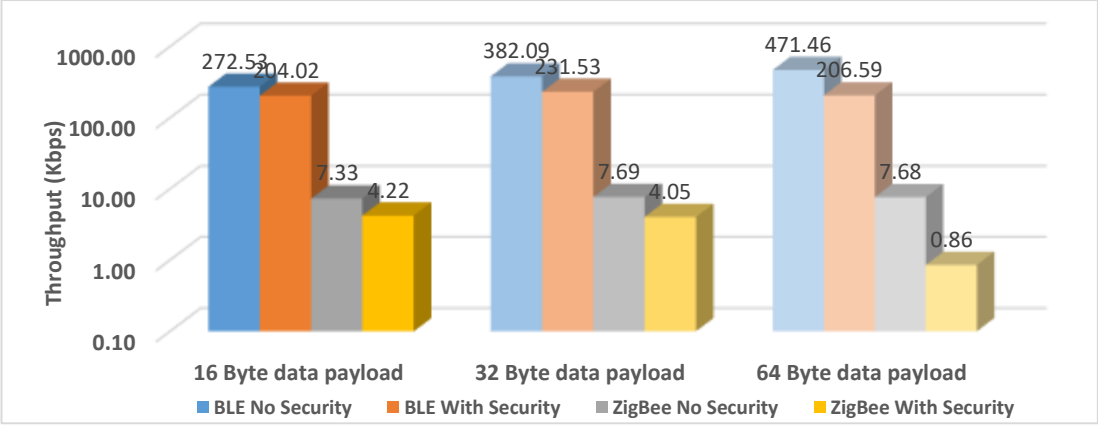


Figure 5.2 Average throughput for 80, 100, and 150 ms connection intervals

ESP32 BLE data sheet stated that the one-way data transmission speed between two ESP32 devices can reach up to 1900 kbps (approximately 235 KB/s). In this paper (Dian, Yousefi, & Lim, 2018), which facilitates the communication of sensor data between two devices using BLE, the maximum efficiency was observed by considering different BLE connection intervals, BLE packet size, and ACK. The maximum throughput was observed to be 221.7 kbps with a connection interval of 70 ms for 20-byte data. In another study (Farej & Saeed, 2020), using both single-slave

and multiple-slave configurations with a data payload of 60 bytes and a connection interval of 7.5 ms, enabling Data Length Extension (DLE) in OMNET ++ simulator resulted in observed maximum data transmission rates of 755.8 and 689.6, respectively. In our study, enabling DLE increased the maximum transmitted data size from 20 bytes to 517 bytes. When 100 and 500 data were sent over 5 different connection intervals, data losses were observed at the 7.5 ms interval for both 100 and 500 messages, with an increased loss rate for 500 messages. The parameters added for encryption increased the data size, negatively impacting the throughput. In other connection intervals, as shown in the Figure 5.2, the average throughput value remains the same in both encrypted and unencrypted states within the same interval. However, due to the decrease in the number of transmitted messages, this value has decreased at the 7.5 ms interval. Thus, at the 7.5 ms connection interval, the throughput values decreased from 325.42, 221.68, and 170.66 kbps for 64, 32, and 16 bytes of unencrypted data, respectively, to 88.27, 90.32, and 66.01 kbps for encrypted data of the same sizes.

Figure 5.3 contains the data transmission times from NodeB and NodeZ to the RG (precisely to NodeBRG and NodeZRG), as well as the acknowledgment time indicating the arrival of new data by the RG in ms for BLE. Additionally, an increase in these times is observed when security parameters are added.

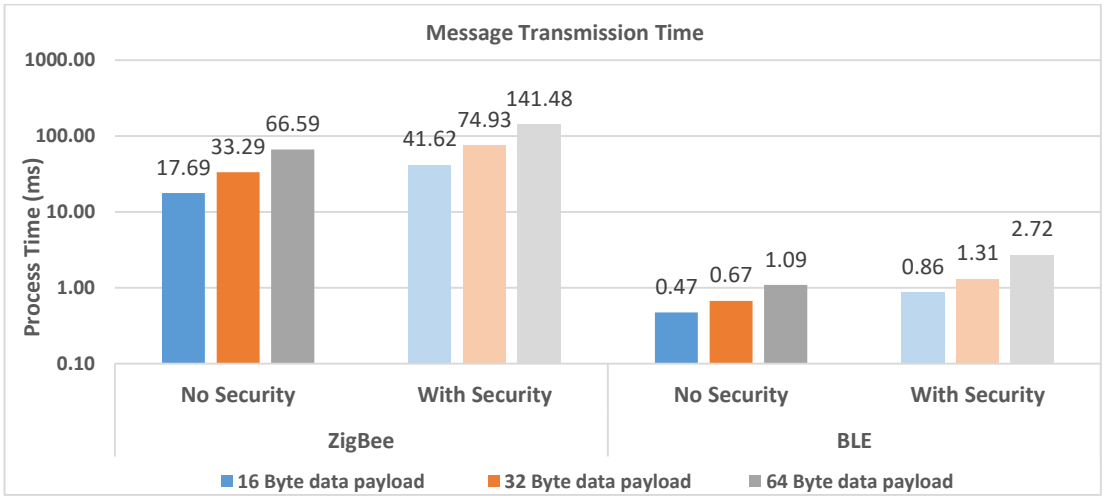


Figure 5.3 Comparison of the one-way message transmission times per data size for BLE and ZigBee

For BLE and ZigBee, we define a communication reliability ratio (ρ), which can be calculated as follows:

$$p = \frac{SuccessfullyForwardedMsgs}{SentMsgs} \times 100 \quad (5.6)$$

Where *SentMsgs* is the number of end node messages received by the relay gateway, *SuccessfullyForwardedMsgs* is the number of end node messages received and successfully forwarded by RG. This ratio indicates the proportion of messages that were successfully forwarded compared to the total number of messages. A higher ratio suggests reliable transmission with BLE and ZigBee, while a lower one indicates unreliable communication due to lost messages.

Figure 5.4 and Figure 5.5 assess the reliable message ratio between the end node and the relay gateway, between the relay gateway and the main gateway, and the end-to-end transmission between the end node and the main gateway, for BLE and ZigBee, respectively. Upon increasing the message transmission frequency from 100/s to 500/s, a decrease in message reliability from 100 was observed at connection intervals of 7.5 ms and 40 ms.

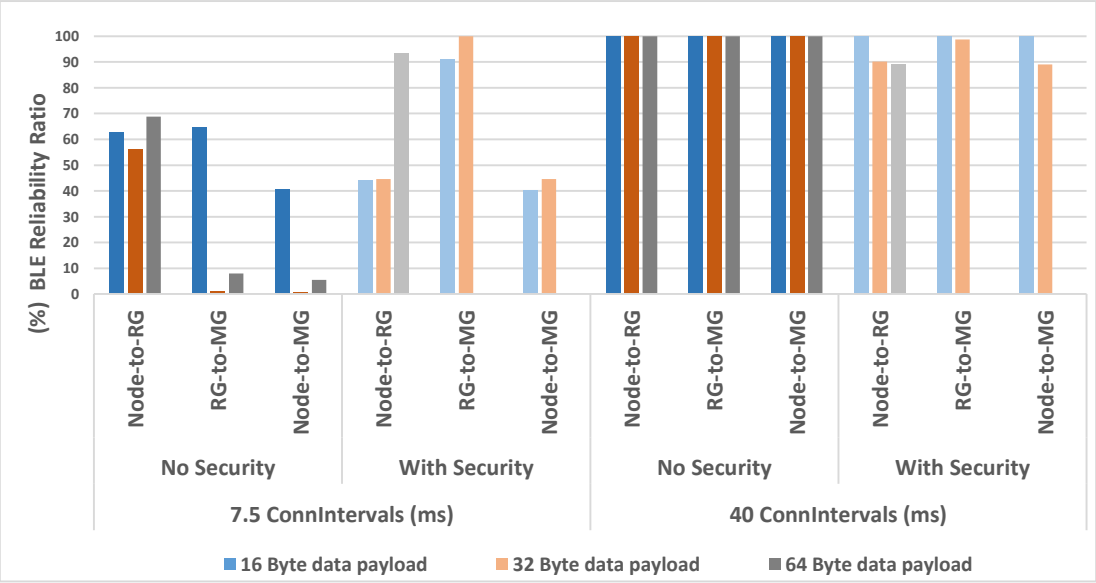


Figure 5.4 BLE message realibility

In this study (Pang, Claeys, Jonck, Hallez, & Boydens, 2022), the relationship between reliability, defined as the ratio of received messages to transmitted messages,

and throughput in BLE communication was investigated. 7.5 ms connection interval and payload sizes of 100 bytes, 150 bytes, and 200 bytes were utilized. As reliability decreased from 90% to 80% with increasing payload, the measured throughput values were determined to be 120,96 Kbps, 166,81 Kbps, and 224,43 Kbps, respectively. In our study, transmissions were conducted with AES encryption using 16, 32, and 64-byte data payloads at a 7.5 ms connection interval, repeated 2500 times. According to the obtained data, the reliability of 16-byte encrypted data transmissions from Node to RG and from RG to MG was calculated to be 44.04% and 40%, respectively, as shown in Figure 5.4. Transmission of 64-byte payloads to MG failed due to message size overrun in MQTT with increasing payload. Real-world testing revealed that the 7.5 ms connection interval is not preferable for both encrypted and unencrypted transmissions. For 500 transmissions, the reliability was 100%, and even with an increase in the connection interval to 40 ms for 2500 transmissions, a 100% reliability was observed. In this context, our study concludes that both data transmission frequency and connection interval have an impact on data reliability.

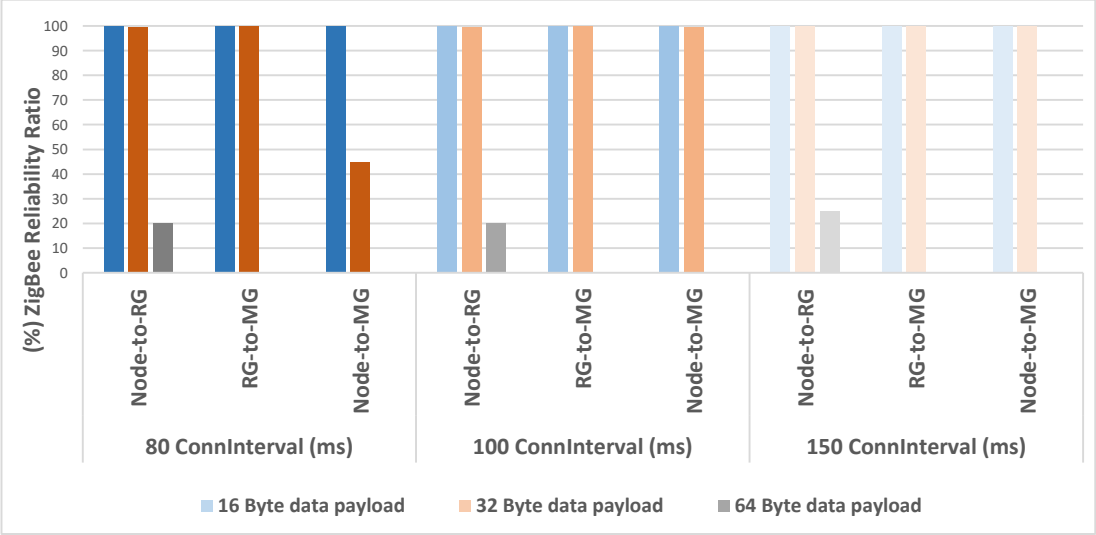


Figure 5.5 ZigBee message realibility

In the ZigBee context, it was observed that unencrypted transmissions experienced no message losses, indicating a high level of reliability under such circumstances. However, the introduction of encryption resulted in some losses, which are solely illustrated in the Figure 5.5 for encrypted scenarios. These observations have been quantitatively detailed in a table stored on the drive for potential reference to numerical

data during discussions. The analysis encompasses the count of messages sent from end nodes to RG, those forwarded to MG, and the end-to-end transmission status. The use of Wi-Fi between RG and MG generally leads to higher reliability in transmissions within this segment. Nevertheless, the reliability notably decreases when transmitting 64-byte data packets, impacting ZigBee more severely. This degradation in reliability became evident with both BLE and ZigBee when the test progressed from 100 messages/s to 500 messages/s.

End nodes, NodeW, NodeB, and NodeZ encrypt the payload using AES encryption and append HOTP information before sending it to the RG. The RG processes the data by adding additional security mechanism information to the payload after performing HOTP parsing and comparison. After encrypting the data with AES, it sends the data to the MG with an MQTT publish message. The MQTT PUBLISH message includes the topic, payload, and header information. Figure 5.6 illustrates the increase in the payload size in end-to-end transmission due to added security mechanisms when the size of the original data sent by end nodes was set to 16, 32, and 64 bytes.

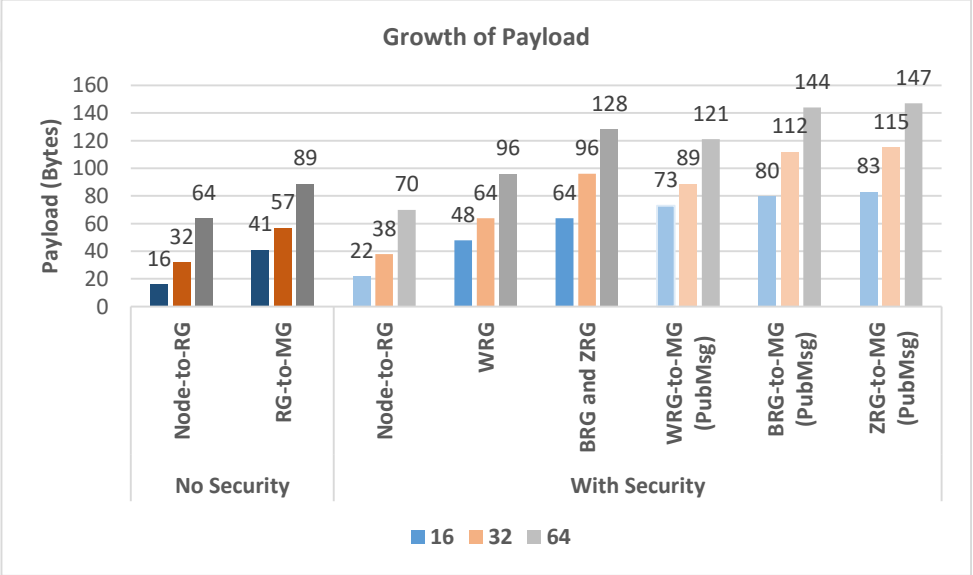


Figure 5.6 The growth of node payload during the authentication process (Bytes)

The experiments described in this section involve a Wi-Fi mesh network comprising three end devices, designated as NodeW1, NodeW2, and NodeW3, along with the Wi-Fi module of the relay gateway, referred to as NodeWRG. Additionally, it includes the main gateway, denoted as MG, which is not part of the mesh network

but is connected via Wi-Fi. The experiments aim to assess packet delivery performance across various message sizes, time intervals, and payload sizes. In particular, the investigation focuses on payload sizes of 16 and 32 characters. Moreover, the tables provide detailed information regarding message ID and node type, which are further elaborated upon in Section 4. Throughout the experiments, the quantity of messages dispatched from a single node fluctuated between 100 and 500, while the time interval was modulated within the scope of 1 to 5 seconds or 3 to 7 seconds. Message transmissions were executed across three devices interconnected within the mesh network to ensure the validity of the test results.

Table 5.4 presents the results of the unique message transmission rate. The initial column, denoted as 'P', indicates the payload size, which incorporates the message ID. The column 'M' denotes the observed message count during the testing, while 'T' signifies the time range in seconds. Columns 2, 3, and 4 depict the number of messages transmitted from NodeW1, NodeW2, and NodeW3, respectively, to the relay gateway. It is expected that these numbers closely correspond to the message size. The NodeWRG column displays the total count of unique messages transmitted by clients to NodeWRG. Similarly, the MG column indicates the anticipated number of messages to be dispatched by NodeRG and received by the MG. Lastly, the "Loss" column illustrates the proportion of lost packets observed at the destination (MG).

Table 5.4 Unique message transmission rate

P, M, T	NodeW1	NodeW2	NodeW3	RG	MG	Loss (%)
(15, 100, 1-5)	100	100	100	300	300	0
(15, 100, 3-7)	88	100	100	288	288	4
(15, 500, 1-5)	449	445	446	1340	1340	10.67
(15, 500, 3-7)	492	494	500	1486	1486	0.93
(31, 100, 1-5)	100	100	100	300	300	0
(31, 100, 3-7)	100	95	96	291	291	3
(31, 500, 1-5)	450	431	432	1313	1313	12.47
(31, 500, 3-7)	469	491	500	1460	1460	2.67

In Table 5.5, when there are either 100 or 500 messages, it is expected that NodeW1, NodeW2, and NodeW3 will each transmit 500 or 2500 messages to the relay gateway, respectively. This expectation arises from the mesh structure, where each node adds its own message ID before forwarding it. For example, NodeW1 includes its message ID and forwards the message to NodeW2, NodeW3, and NodeW3. Upon receiving the message, NodeW2 and NodeW3 add their respective message IDs. Once NodeW2 adds its message ID, it forwards the message to NodeW3 and NodeWRG. After NodeW3 adds its message ID, it sends the message to the relay gateway. NodeW3, after receiving the message from NodeW1, adds its own message ID and forwards it to NodeW2 and NodeWRG. NodeW2 then adds its message ID and sends it to NodeWRG. This flooding process effectively reduces message losses, resulting in the relay gateway's NodeWRG receiving the generated message a total of 5 times.

Table 5.5 The transmission rates of all messages sent in the mesh network

P, M, T	NodeW1	NodeW2	NodeW3	NodeRG	MG	Loss(%)
(15, 100, 1-5)	497	500	500	1497	1497	0.20
(15, 100, 3-7)	440	446	441	1327	1327	11.53
(15, 500, 1-5)	2243	2222	2223	6688	6688	10.83
(15, 500, 3-7)	2429	2441	2448	7318	7318	2.43
(31, 100, 1-5)	500	500	500	1500	1500	0
(31, 100, 3-7)	478	473	477	1428	1428	4.80
(31, 500, 1-5)	2144	2092	2110	6346	6346	15.39
(31, 500, 3-7)	2315	2351	2371	7037	7037	6.17

After examining Table 5.4 and Table 5.5, a notable finding emerges concerning message transmission within the designed IoT environment. When the quantity of messages transmitted is modest, maintaining a transmission interval spanning from 1 to 5 seconds ensures the successful conveyance of all data within a shorter duration and with minimal losses. Conversely, when there is a considerably larger volume of messages to be dispatched, the findings from the tests imply that augmenting the message transmission interval proves to be more effective in attaining successful message delivery. Moreover, both tables underscore that there are no instances of message loss during MQTT publishing.

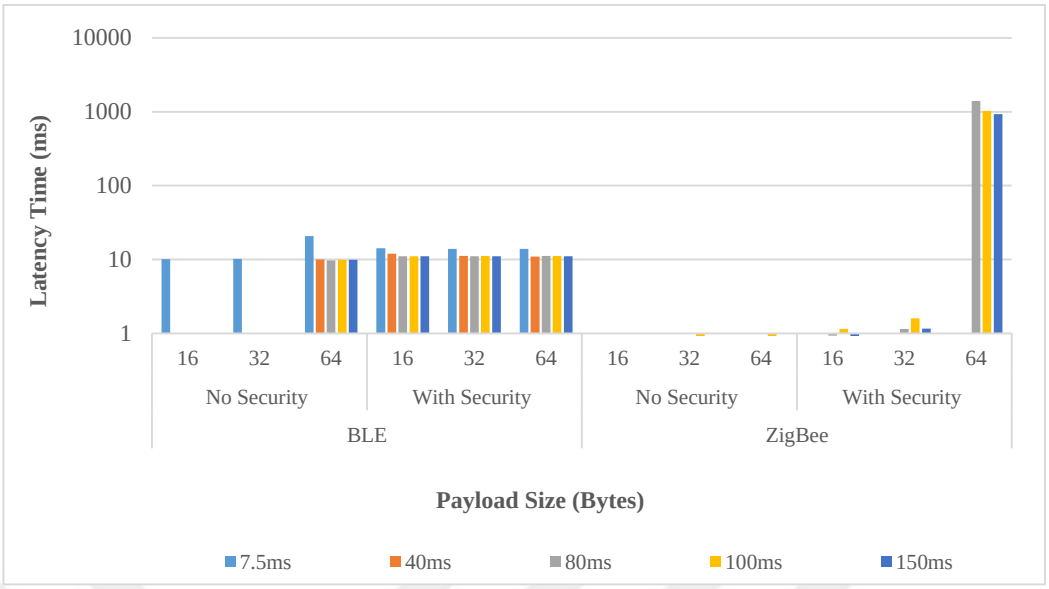


Figure 5.7 The Delay time (ms) in ZigBee and BLE communication

Figure 5.7 depicts the latency of single-hop Zigbee communication with variations in 9600 baud rates and packet sizes for both AES encrypted and unencrypted communication at connection intervals of 80 ms, 100 ms, and 150 ms. Additionally, it includes delays for BLE communication at connection intervals of 7.5 ms and 40 ms. The mean of 2500 readings is considered for all data points in the plots.

For AES encrypted and unencrypted data in Zigbee communication, the latency significantly increases with a packet size of 64 bytes. Due to the lower transmission speed of Zigbee communication, the delays in BLE communication are comparatively more acceptable. However, it is possible to mention that Zigbee communication is more stable for small data transmissions. In the case of BLE communication, the delay starts at a connection interval of 7.5 ms and with a data transmission of 64 bytes in the unencrypted scenario. In the range of 40 ms to 150 ms connection intervals, the delay time averages 10 ms across 4 different connection scenarios, while this rate doubles to an average of 20 ms at a 7.5 ms connection interval. In the encrypted scenario, with 4 different connection intervals and data transmissions of 16 and 32 bytes, the average delay is 11.13 ms, but it shows a linear increase at a 7.5 ms connection interval across all message sizes, averaging 14 ms.

For Wi-Fi mesh network, time synchronization action repeats periodically in every 10 minutes by taking into the account time sync conflicts by using delay ±35% for node clocks. Time synchronization initiates when each node joins the mesh network. The new node sends a time request (T0), to calculate the delay. Responding node replies with T1 and T2 timestamps that respectively indicate when the request is received and response is sent. The requesting node calculates the timestamp as T3 when it takes a response time (Arjadi, Setyaningsih, Wibowo, & Sudrajat, 2019). The calculation of offset and round trip time according to T0, T1, T2, and T3 are given in (5.7) and (5.8).

$$Offset = \frac{T1-T0}{2} + \frac{T2-T3}{2} \tag{5.7}$$

$$Trip\ delay = (T3 - T0) - (T2 - T1) \tag{5.8}$$

While the transmission time between two data with NodeW and NodeWRG is about 2.90-2.98 ms, this time also increases to 3.47 ms with the increasing number of devices. When Security tiers 2 and 3 are applied, the average delay time due to the synchronization of the system when a new device is added to the network is 161.07 ms.

5.5 Security Analysis of Multi-Protocol Multi Gateway

As a conceptual application, SIMMS has been demonstrated to provide an easy solution for securing bidirectional MQTT communication among numerous sensor/actuator devices (i.e., clients), relay gateways, and servers. The flawless operation of authentication (for clients and relay gateways) and bidirectional identity verification (for clients and relay gateways) has been shown, enabling all devices within a SIMMS-enabled MQTT network environment (i.e., clients, relay gateways, main gateways, and servers) to ensure the authenticity of each connected device. SIMMS employs dynamic encryption using HOTP and tokens for authentication, effectively neutralizing replay attacks and man-in-the-middle attacks by invalidating password reuse. Additionally, permission-based authorization (for data sending and receiving) is implemented on sensor/actuator devices via an identity server through brokers. Using OAuth 2.0, the IoT application can effectively manage access control and authorization management for an increasing number of devices. Each device is

individually authenticated and authorized, utilizing its unique identity to enforce access restrictions. This granular level of authorization enhances security and control within the IoT ecosystem, ensuring that devices can only access messages and resources for which they have explicit permission.

Proposal 1: Security level 1 provides physical protection in closed environments.

Proof: With security level 1, distance information is obtained through the RSSI data collected by access points installed at fixed points. This information allows us to detect when the device is moved away from its location, triggering alarms or alerts in case the boundaries of physically protected areas are breached.

Proposition 2: The proposed mechanism ensures that Nodes have unique identities.

Proof: In the proposed mechanism, the unique identity of Nodes relies on a random number, known as the client ID, which is registered for each device in the authorization server. The client ID information is manually added to the devices. Therefore, assuming the physical security of the devices and that they are not shared in any insecure environment, an adversary cannot guess the random number.

Proposition 3: The proposed mechanism enables mutual authentication between Node and NodeRG.

Proof: During the authentication phase, the HOTP, seed information for generating AES keys, and the hashed Node device ID obtained from the QR code are transmitted serially to NodeRG. In Algorithm 2, steps 5 and 9, RG verifies the Node using the HOTP information, while in Algorithm 2, steps 15 and 20, Node follows the same process to authenticate the NodeRG. If an attacker aims to create a valid Node/NodeRG replica, they must generate valid messages. However, since the attacker does not have information about random numbers such as the client ID and access token, they cannot generate valid HOTP and AES keys. After transmitting the security parameters securely via QR code, mutual communication occurs encrypted with AES. After each connection, the HOTP information becomes invalid.

Proposition 4: The proposed mechanism ensures the authentication of NodeRG by the main gateway.

Proof: During the authentication phase, NodeRG establishes an HTTPS connection with the IS to obtain an access token and refresh token, valid for 3 seconds. When a connection is established between RG and the main gateway, the access token becomes invalid. If an attacker eavesdrops on the access token and tries to reuse it, the token will be invalid. Without the client ID and client secret information, it is not possible for the attacker to establish an HTTPS connection with the IS.

Proposition 5: The proposed mechanism prevents replay attacks.

Proof: As described in Algorithm 2, Phase 1, in Chapter 4, the legitimate Node device ID hash is verified by NodeRG using a QR code. We assume that the legitimate Node sends the message (i.e., HOTP, access token, and hash(NodeID)) to the NodeRG via QR code. If an attacker attempts to impersonate the legitimate Node by replaying this message, they cannot transmit it because they cannot read the QR code. In subsequent mutual communications, each message contains a nonce and HOTP information. NodeRG rejects the authentication request from the attacker because the HOTP secret information is calculated only by the legitimate Node using a random number and counter information. Additionally, authentication is performed only by verifying the device ID hash known only to the Node.

Proposition 6: The proposed mechanism demonstrates resilience against man-in-the-middle attacks.

Proof: An attacker, by obtaining the Node identity (i.e., ID) or NodeRG (i.e., client ID), cannot initiate a man-in-the-middle attack and compute a HOTP and an access token because they cannot obtain the client secret, which is known only to the IS. The client secret is securely stored in the IS and is not transmitted to any other entity.

Proposition 7: The proposed mechanism withstands impersonation attacks.

Proof: Let's consider Node1 intending to impersonate another Node, Node2. However, Node1 cannot obtain the access token and HOTP generated by Node2 and NodeRG because it does not have information about Node2's random number (i.e., client ID).

Proposition 8: The proposed mechanism encompasses access permission and authorization.

Proof: Each Node is registered with the IS specifying the scope (i.e., subject matter and read-write permission) during registration. Upon successful connection establishment with the main gateway, the scope information is queried via an HTTPS connection between the IS and the MQTT broker using the access token. Authorization is granted if the received response matches the topic information and write permission for the MQTT publication. The mechanism prevents each device from accessing every message and publishing to every topic on the network.

Proposition 9: The proposed mechanism does not require clock synchronization.

Proof: There is no need to synchronize the clocks of the nodes (e.g., Node, NodeRG, main gateway, and AS) because the exchanged messages between devices are based on HMAC-based OTP and nonce-based authentication mechanisms, which do not rely on timestamps.

Proposition 10: The proposed security tier 4 guarantees that the message has not been altered.

Proof: By using the ECDSA algorithm with a security level of 4, it is ensured that the message has not been altered. The public key is sent to NodeRG via a QR code. This approach is an effective method for ensuring the integrity and authenticity of the transmitted data.

5.6 Performance Discussion of Results

This section provides a summary and discussion of the functionality validation outlined and the performance evaluation presented in subheading of Section 4 and 5.

The implementation of the SIMMS security scheme in environments utilizing BLE, ZigBee, and Wi-Fi technologies, particularly on devices such as ESP32 and Raspberry Pi, highlights certain trade-offs between security enhancements and network performance. The computational burden imposed by encryption is demonstrated by the differing AES encryption durations, which range from 0.024 ms for ZigBee on

ESP32 to 0.724 ms for Wi-Fi on the Raspberry Pi. The computation of HOTP also represents another time-intensive operation. The duration required for reading QR codes remains consistent at 0.082 ms across all scenarios. The typical duration for publishing an MQTT message averages around 3 ms for encrypted messages and 1-2 ms for non-encrypted messages. Nonetheless, the influence on communication expenses and payload dimensions remains tolerable for numerous applications.

The analysis demonstrates that the balance between connection intervals, message frequency, and payload size significantly affects network throughput and reliability. For instance, when the message frequency rises from 100/s to 500/s, both BLE and ZigBee networks experience a noticeable decrease in message reliability, particularly with shorter connection intervals. This decline is particularly pronounced within ZigBee networks at 7.5 ms intervals, highlighting the increased vulnerability of high-frequency transmissions to the negative impacts of encryption. Throughput also fluctuates with alterations in payload size and connection intervals, demonstrating decreases in secured transmissions across different payload sizes (16, 32, 64 bytes) and intervals (80, 100, 150 ms).

Despite the added complexity brought by SIMMS, the overall system resilience largely remains intact. This is evidenced by minimal (for Wi-Fi) to acceptable (for BLE and Zigbee) packet losses, and minimal (for Wi-Fi and BLE) to acceptable (for ZigBee) delays even under high message frequencies and larger payloads. Some limitations were observed for BLE and ZigBee, yet the feasibility of effectively securing IoT communications using SIMMS is demonstrated.

Our examinations also brought to light the consequences and costs associated with integrating the suggested security mechanism into an MQTT setup. The results clearly show a trade-off between the additional communication load and the security enhancement achieved. The specific expenses of implementing SIMMS (in terms of data overhead) can be identified by referring to Table 5.6, where the three largest message types are compared. It's worth noting that other message types, being both substantially smaller and minimally impacted by SIMMS, can be safely disregarded.

Table 5.6 Extra network traffic caused by SIMMS (in terms of message size)

Message Type	N*	Naïve MQTT	SIMMS	Cost increase (%)
CONNECT	N=1	37 B	73 B	97.29
	N=1000	37 B	73 B	97.29
PUBLISH (16bytes payload data)	N=1	$41B_{Wi\text{-}Fi,BLE,ZigBee}$	$73B_{Wi\text{-}Fi,}$ $80B_{BLE,}$ $83B_{ZigBee}$	78.04 $_{Wi\text{-}Fi,}$ 95.12 $_{BLE,}$ 102.43 $_{ZigBee}$
	N=1000	41 kB	$73kB_{Wi\text{-}Fi,}$ $80kB_{BLE,}$ $83kB_{ZigBee}$	78.04 $_{Wi\text{-}Fi,}$ 95.12 $_{BLE,}$ 102.43 $_{ZigBee}$
PUBLISH (32bytes payload data)	N=1	$57B_{WiFi,BLE,ZigBee,}$	$89B_{Wi\text{-}Fi,}$ $112B_{BLE,}$ $115B_{ZigBee}$	56.14 $_{Wi\text{-}Fi,}$ 96.49 $_{BLE,}$ 101.75$_{ZigBee}$
	N=1000	57kB	$89kB_{Wi\text{-}Fi,}$ $112kB_{BLE,}$ $115kB_{ZigBee}$	56.14 $_{Wi\text{-}Fi,}$ 96.49$_{BLE,}$ 101.75$_{ZigBee}$
PUBLISH (64bytes payload data)	N=1	89B $_{Wi\text{-}Fi,BLE,ZigBee}$	$121B_{Wi\text{-}Fi,}$ $144B_{BLE,}$ $147B_{ZigBee}$	35.95 $_{Wi\text{-}Fi,}$ 61.79 $_{BLE,}$ 65.16$_{ZigBee}$
	N=1000	89kB	$121kB_{Wi\text{-}Fi,}$ $144kB_{BLE,}$ $147kB_{ZigBee}$	35.95$_{Wi\text{-}Fi,}$ 61.79 $_{BLE,}$ 65.16$_{ZigBee}$
*** N solely indicates nr. of data packages to be published in entire network lifetime.**				

In scenarios where a sensor network operates over an extended period and needs to transmit more than 1000 data packages, it is evident the discussion of results from (Şeker et al., 2023) and Table 5.6 that PUBLISH messages are likely to dominate network traffic. This dominance stems from the fact that other message types are typically sent only once or infrequently. However, in many scenarios, 1000 data packages are relatively modest, considering that a sensor network may generate millions of messages throughout its lifetime. Consequently, while the cost increase is more pronounced in CONNECT messages, the primary expense arises from securing published data. As a result, SIMMS increases the total traffic overhead by approximately 1.9 times compared to insecure conditions. Nevertheless, assuming there is no multimedia content, message sizes remain small, making this overhead generally tolerable, especially when network security is paramount.

We present a comparative analysis in Table 5.7, contrasting the security features of our mutual authentication and role-based authorization scheme (MARAS), developed for IoT applications and implemented on advanced devices, with existing schemes. In this table, "Authorization" refers to simple access control methods (e.g., OAuth 1.0, access control lists, tree hierarchy), while "Role-based access control" denotes advanced node/client authorization via assigned roles (e.g., using OAuth 2.0). "Client authentication" indicates brokers authenticating clients (e.g., using tokens, usernames, and passwords), "HOTP" signifies the use of HMAC-based one-time passwords for authentication, "Mutual authentication" refers to clients and brokers mutually

authenticating each other (e.g., via HOTP, SSL, nonces), and "Confidentiality" represents payload encryption (e.g., AES). As depicted in Table 5.7, no prior endeavor collectively offers all the security features provided by MARAS.

Table 5.7 Comparative analysis of existing works and MARAS (Şeker, Dalkılıç, & Çabuk, 2023)

Literature & MARAS	Authorization	Role-based Access Control	Client Authentication	HOTP	Mutual Authentication	Confidentiality (Payload Encryption)
Fremantle et al.,2014	✓	✓	✓	×	×	×
Singh et al., 2015	✓	×	✓	×	×	✓
Upadhyay et al., 2016	✓	×	✓	×	×	×
Niruntaskrat et al.,2016	✓	×	✓	×	×	×
Zamfir et al., 2016	×	×	✓	×	✓	✓
Nagarajan et al., 2021	×	×	✓	×	✓	✓
Alshammari, 2023	✓	×	✓	×	✓	✓
Fathy et al., 2023	×	×	×	×	×	✓
Shilpa et al., 2022	✓	×	✓	×	✓	✓
Winarno et al., 2022	×	×	×	×	×	✓
Ramyasri et al., 2023	×	×	✓	×	✓	✓
Patel et al., 2023	×	×	✓	×	✓	✓
Saqib et al., 2022	×	×	✓	×	✓	×
Katsikeas et al., 2017	×	×	×	×	×	✓
Bhawiyuga et al., 2017	×	×	✓	×	×	×
Bashir et al., 2017	×	×	×	×	×	✓
Wardana et al., 2018	✓	×	✓	×	×	✓
Calabretta et al., 2018	✓	×	✓	×	×	×
Bali et al., 2019	×	×	✓	×	✓	✓
Sundarrajan et al., 2019	×	×	✓	×	×	✓
Aknin et al., 2023	×	×	✓	✓	✓	×
Sanjuan et al., 2020	×	×	✓	×	✓	✓
Amoretti et al., 2020	✓	✓	✓	×	×	×
Blazy et al., 2021	✓	✓	✓	×	×	✓
Azzedin et al., 2023	✓	✓	×	×	×	×
MARAS	✓	✓	✓	✓	✓	✓

Additionally, we present a comparative analysis of SIMMS' multi-protocol design and the security features we developed, compared to existing schemes, in Table 5.8. The term "Gateway" indicates the presence of multiple gateways involved in interaction. (For example, our design includes both relay gateway and main gateway, considering multi-gateway communication and requirements.) "Security features" represent whether authentication, authorization, and data privacy are included. "Methods" cover the cryptographic techniques and methods used when the security mechanism is present. "Hardware/Software" addresses the IoT devices and software environments used in the study's test environment, if applicable. As depicted in Table 5.8, no prior endeavor has collectively provided all the security features of both multi-gateway secure communication and multi-protocol gateway handling simultaneously, as SIMMS does.

Table 5.8 Comparative analysis of existing works and SIMMS

Study	**Gateway**	**Protocol Support**	**Security Features**	**Methods**	**Hardware/Software**
Jabbar et al., 2019	N/A	Wi-Fi	RFID auth.	N/A	NodeMCU, Adafruit IO, smartphone, Google Home
Chaudhary et al., 2021	Single	Wi-Fi, BT, ZigBee	N/A	N/A	Wearables, Xbee HUB, smartphone, Raspberry Pi 3
Hajian et al., 2022	Single	N/A	3FA w/ ID, passw., biometrics	XOR, hash	No testbed, theoretical
AlQahtani et al., 2022	Single	Wi-Fi	Proximity-based auth.	RSSI	Raspberry Pi
Saqib et al., 2022	Single	Wi-Fi	3FA w/ ECC, hash chains, dig. signat.	Hash, ECC	Formal verif. via Scyther
Marktscheffel et al., 2016	N/A	Wi-Fi	QR-based mutual auth.	Hash	N/A
Pearson et al., 2019	Single	Wi-Fi	ECDSA signature	AES, HMAC, ECC	CC3220SF, ESP8266, ESP32
Saputra et al., 2019	Single	Wi-Fi	QR-based authentication	PBKDF2	ILI9255, NodeMCU, ESP32, Android smartphone
Naveen et al., 2021	N/A	Wi-Fi	QR-based authorization	N/A	ESP32 CAM, L293D, CP2102
Khanchuea et al., 2019	Multi	Wi-Fi, ZigBee, Eth.	N/A	N/A	CubieTruck SBC, ESP8266 ESP WROOM32, XBee 2
Hanggoro et al., 2019	Single	Wi-Fi, BLE, ZigBee	N/A	N/A	Raspberry Pi 3B, XBee Znet 2.5,
Amiruddin et al., 2018	Single	Wi-Fi, BLE, ZigBee, Eth.	Data security	AND, XOR, circ. shift	Raspberry Pi, ESP32, XBee, HM10 BLE, Arduino

Table 5.8 Continues

Study	Gateway	Protocol Support	Security Features	Methods	Hardware/Software
Zhang et al., 2019	N/A	BLE	Auth. and availability	TOTP, AES, ECDHA	nRF52832, mobile device
Gupta et al., 2023	Single	BLE	Lightweight dig. cert.-based auth.	AES-CMAC, ECDH, ECDSA	Arduino Nano 33, ESP-32, Android smartphone
SIMMS	Multi	Wi-Fi, BLE, ZigBee	3FA w/ QR, passw., token auth. And author.	AES-CBC, HOTP, hash, OAuth 2.0, ECDSA	ESP-32, Raspberry Pi, Xbee pro, WSO2IS, MongoDB

Considering the resource constraints of our ESP32 devices, it can be said that the ECDSA algorithm is a better choice compared to other public key algorithms like RSA and DSA, especially when security tier 4 is added in addition to security tiers 2 and 3. When comparing the performance of ECDSA, RSA, and DSA algorithms, several factors come into play, including computational efficiency, key size, and security levels (Badawy, 2023). ECDSA is known for its high efficiency and smaller key sizes, providing equivalent security with significantly less computational power and bandwidth compared to RSA and DSA. For instance, a 256-bit key in ECDSA offers comparable security to a 3072-bit key in RSA, resulting in faster computations and reduced storage and transmission requirements. RSA, on the other hand, is widely recognized for its robustness but requires larger key sizes to achieve the same security levels, leading to slower performance and higher computational costs, particularly in environments with limited resources. DSA falls somewhere in between, offering good performance and security but generally requiring larger key sizes than ECDSA. Additionally, DSA's performance can be less consistent due to its reliance on random number generation during the signing process. Overall, ECDSA is preferred for applications demanding high efficiency and lower resource consumption, while RSA is often chosen for its widespread adoption and proven security, and DSA is utilized where a balance between performance and security is acceptable.

The implementation of cryptographic methods (such as AES encryption, hash operations, and ECDSA signatures) for each security tier leads to a significant increase in payload size. This increase is illustrated in Figure 5.8, which shows the data sizes of 16, 32, 64, and 128 bytes for different security levels. Specifically, as the security level escalates from secure tier 1 to tiers 2 and 3, and further to tier 4, the added cryptographic operations significantly expand the payload size. For example, in secure

tier 1, the payload overhead is minimal due to the lack of cryptographic operations, relying instead on physical tracking of the device. However, transitioning to tiers 2 and 3, where AES encryption is introduced, the payload size moderately increases due to the more complex operations involved. In tier 4, the integration of hash functions and ECDSA results in a substantial increase in payload size. While tier 4 maximizes security, it also leads to the highest payload size. This detailed analysis underscores the balance between enhanced security and increased payload size, highlighting its critical importance for optimizing system performance and security.

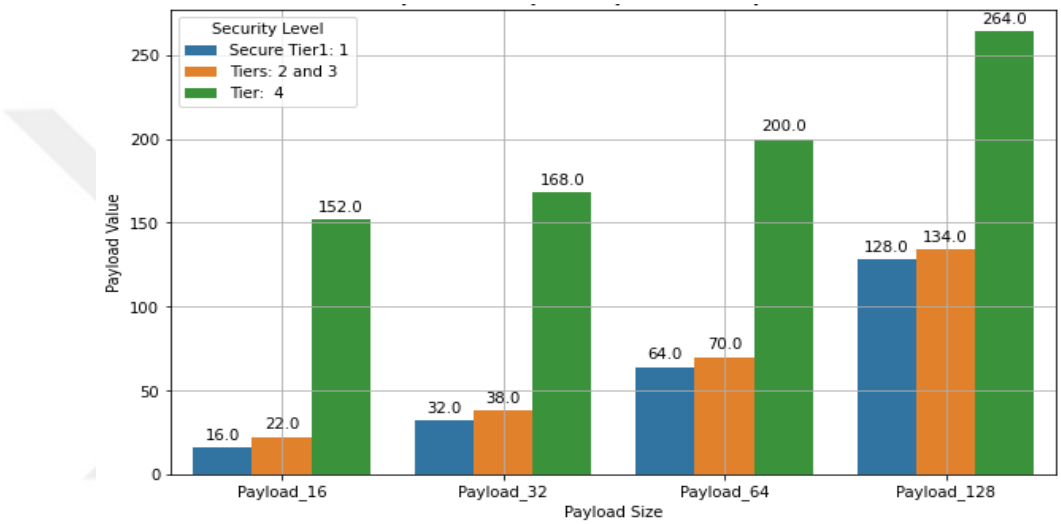


Figure 5.8 Payload increase based on security tiers

The round trip time (RTT) durations (ms) in BLE communication are compared and presented in relation to the increasing payload sizes for these security levels. As shown in Figure 5.9, changes in RTT values are observed across different security tiers with data sizes of 16, 32, 64, and 128 bytes. As the security tier escalates from secure tier 1 to tiers 2 and 3, and further to tier 4, the RTT durations increase significantly due to the added cryptographic operations.

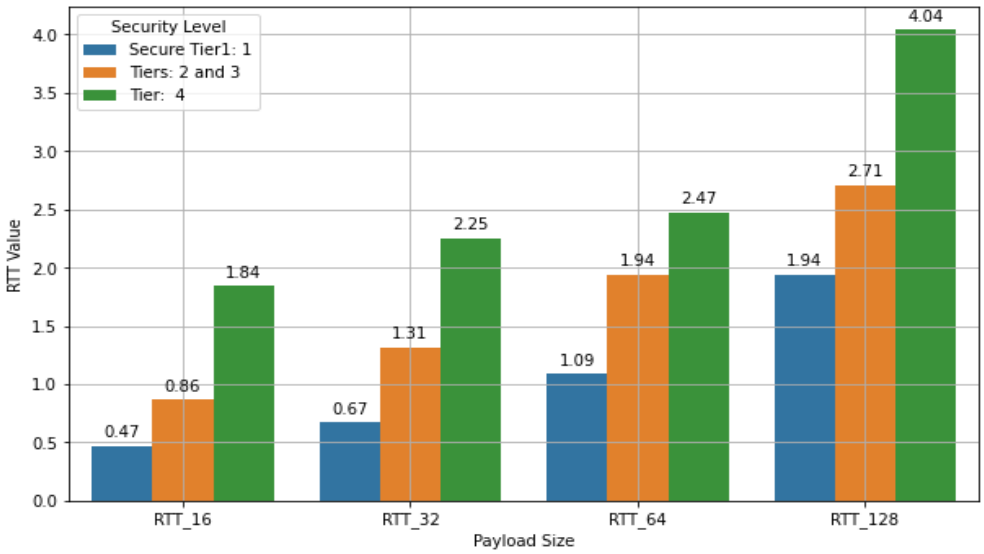


Figure 5.9 Increase in BLE RTT based on security tiers, shown in milliseconds (ms)

In conclusion, as the security tiers increases, there are significant increases in both payload and BLE RTT values, as indicated in Figure 5.8 and Figure 5.9, respectively. When transitioning from secure tier 1 to tiers 2 and 3, the payload values increase by an average of 4.69% to 37.5%, and the RTT values increase by 39.69% to 95.52%. When moving from secure tier 1 to tier 4, the payload values rise by 106.25% to 850%, and the RTT values increase by 108.25% to 291.49%. These percentage changes highlight that higher security levels require significantly higher data and transmission times. In this context, it is important to emphasize the selection of an appropriate security tier, considering the device constraints and application requirements in IoT applications.

CHAPTER SIX
CONCLUSION AND FUTURE WORK

6.1 Conclusion

The MQTT protocol provides a fast and reliable communication scheme for IoT deployments composed of lightweight sensors and actuator devices. However, this protocol does not inherently implement any advanced security mechanisms to adapt to the stringent resource constraints of such devices. Nevertheless, securing sensitive data in critical applications is of utmost importance. The use of SSL/TLS and HTTPS can provide significant security, but this comes with high energy consumption, low performance, intensive computation, and inflated bandwidth costs. Therefore, there is a significant need for a lightweight security protocol for resource-constrained IoT devices. The thesis proposes an extension to the MQTT protocol to ensure the authorization of requesting nodes, mutual authentication of communicating parties, confidentiality of related topics, and encryption of sensitive data.

With mutual authentication and role based scheme (MARAS), a Mosquitto broker can authorize client nodes (and control their rights to publish or subscribe to certain topics) through an authentication server (running WSO2IS with OAuth 2.0). Similarly, nodes and brokers can authenticate each other by comparing HMAC-based one-time passwords stored in their local databases (e.g., MongoDB). Therefore, such trusted communications can be encrypted with AES to ensure confidentiality. This also inherently provides integrity protection. Thus, comprehensive security can be brought to lightweight IoT devices using MQTT. Previous studies have not achieved the level of security provided by MARAS without using SSL/TLS and HTTPS; these technologies are heavy and costly.

In practice, the cost of applying MARAS to a sufficiently long-lived sensor network is doubling the existing data traffic in the network. This is mainly due to the 90.74% increase in the size of PUBLISH messages (as noted by Şeker, Dalkiliç, & Çabuk, 2023). When the number of PUBLISH messages sent within the network exceeds 1000, they start to account for over 99% of the cost increase caused by MARAS. Nonetheless, this extra load is still negligible for many wireless sensor network (WSN) or IoT scenarios, and the delays are much smaller compared to those caused by

SSL/TLS or HTTPS. Additionally, the solution is not dependent on the technologies specified in this thesis; it can be implemented using alternatives (e.g., RabbitMQ instead of Mosquitto).

MARAS has been tested on advanced devices and offers a comprehensive framework to secure IoT through a multi-protocol multi-gateway system called SIMMS. SIMMS aims to evaluate the applicability of the MARAS approach on a security-enhanced multi-protocol gateway that includes BLE, Wi-Fi, and ZigBee communication protocols with various security levels. Supported on hardware platforms such as ESP32 and Raspberry Pi, SIMMS leverages multi-factor authentication, role-based access authorization, and cryptographically secure communication. SIMMS includes QR code-based secret transmission and HMAC-based one-time passwords to enhance data privacy and prevent unauthorized access, setting a new standard in IoT network security. The framework proposes the cooperation of multiple gateways to connect end nodes via Wi-Fi, BLE, and ZigBee through one or more relay gateways and includes a main gateway that provides internet connectivity and access control features by connecting the relay gateway(s). Unlike MARAS, SIMMS can dynamically change tokens to provide better security and support seamless authentication with three protocols.

Our prototyping efforts played a significant role in validating SIMMS. By integrating ESP32 modules (for end nodes and relay gateway), Raspberry Pi (for the main gateway), and a Windows PC (for the identity server), we created a prototype to simulate IoT environments and demonstrated the comprehensive security mechanisms of SIMMS in action. This practical approach validated the feasibility of the proposed security architecture. During this process, we effectively demonstrated SIMMS' scalability and interoperability with various protocols.

With SIMMS, as security levels increase from tier 1 to tier 4, there are significant rises in both payload and BLE Round Trip Time (RTT) values. Specifically, transitioning from secure tier 1 to tiers 2 and 3 results in payload increases ranging from 4.69% to 37.5%, while RTT values rise by 39.69% to 95.52%. Moving from secure tier 1 to tier 4 causes payloads to increase by 106.25% to 850% and RTT values

to increase by 108.25% to 291.49%. These percentage changes highlight that higher security levels demand substantially more data and longer transmission times.

Additionally, as security levels increase, there is a notable reduction in throughput values for both BLE and Zigbee communications, with the extent of the decrease varying significantly depending on the data size. As security levels rise, throughput decreases approximately 1.3 times for BLE and 4.6 times for Zigbee. This underscores the importance of carefully managing the balance between security and communication efficiency.

Performance evaluations conducted over BLE, ZigBee, and Wi-Fi revealed minimal impact on message reliability and acceptable latency despite the added complexity of integrated security measures. In summary, SIMMS presents a robust and flexible approach to securing IoT networks by enhancing existing systems with modular authentication for multiple communication technologies and dynamic token changes. This research marks significant advancements in the security and energy efficiency of IoT-based smart systems, offering scalable and adaptable solutions for secure IoT deployments.

Furthermore, the MARAS approach cannot prevent physical attacks. For this, SIMMS has achieved an additional security level (i.e., security level 1) by checking devices' RSSI information and indoor location and detecting non-application devices. Additionally, with security level 4, it established another security level for message integrity and non-repudiation.

6.2 Future Work

Within this thesis, the relationship between RSSI and distance has been explored. Future studies will focus on predicting locations using machine learning and deep learning methods instead of relying solely on distance. Additional future work may involve ensuring compatibility with decentralized ledger systems, supporting multi-broker networks, and implementing further security measures such as availability and non-repudiation. Research will also explore integrating additional communication technologies like LoRa and adopting advanced cryptographic algorithms to address diverse security needs.

REFERENCES

Adi, P. D. P., Sihombing, V., Siregar, V. M. M., Yanris, G. J., Sianturi, F. A., Purba, W., & Prasetya, D. A. (2021, April). A performance evaluation of ZigBee mesh communication on the Internet of Things (IoT). In *3rd East Indonesia Conference on Computer and Information Technology (EIConCIT)*, 7-13.

Akleylek, S., Kılıç, E., Söylemez, B., Aruk, T. E., & Çavuş, A. (2020). Kapali mekan konumlandirma üzerine bir çalişma. *Mühendislik Bilimleri ve Tasarım Dergisi*, *8*(5), 90-105.

Aknin, R., & Bentaleb, Y. (2023). Enhanced MQTT architecture for smart supply chain. *International Journal of Advanced Computer Science and Applications*, *14*(4).

Alassaf, N., Gutub, A., Parah, S. A., & Al Ghamdi, M. (2019). Enhancing speed of SIMON: A light-weight-cryptographic algorithm for IoT applications. *Multimedia Tools and Applications*, *78*, 32633-32657.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorial*s, *17*(4), 2347-2376.

AlQahtani, A. A. S., Alamleh, H., & Al Smadi, B. (2022, June). Iot devices proximity authentication in ad hoc network environment. In *2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 1-5.

Alshammari, H. H. (2023). The internet of things healthcare monitoring system based on MQTT protocol. *Alexandria Engineering Journal*, *69*, 275-287.

Amiruddin, A., Ratna, A. A. P., Harwahyu, R., & Sari, R. F. (2018, April). Secure multi-protocol gateway for Internet of Things. In *Wireless Telecommunications Symposium (WTS)*, 1-8.

Amoretti, M., Pecori, R., Protskaya, Y., Veltri, L., & Zanichelli, F. (2020). A scalable and secure publish/subscribe-based framework for industrial IoT. *IEEE Transactions on Industrial Informatics*, *17*(6), 3815-3825.

Arjadi, R. H., Setyaningsih, E., Wibowo, P., & Sudrajat, M. I. (2019, July). Performance evaluation of ESP8266 mesh networks. In *journal of physics: Conference series*, *1230*(1), 012023.

Ashton, K. (2009). That 'internet of things' thing. *RFID journal*, *22*(7), 97-114.

Azzedin, F., & Alhazmi, T. (2023). Secure data distribution architecture in IoT using MQTT. *Applied Sciences*, *13*(4), 2515.

Badawy, M. (2023). Security evaluation of different hashing functions with RSA for digital signature. *IJCI. International Journal of Computers and Information*, *10*(2), 99-116.

Bali, R. S., Jaafar, F., & Zavarasky, P. (2019, January). Lightweight authentication for MQTT to improve the security of IoT communication. In *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*, 6-12.

Bashir, A., & Mir, A. H. (2017). Securing communication in MQTT enabled Internet of things with lightweight security protocol. *EAI Endorsed Transactions on internet of things*, *3*(12), e1-e1.

Beniwal, G., & Singhrova, A. (2022). A systematic literature review on IoT gateways. *Journal of King Saud University-Computer and Information Sciences*, *34*(10), 9541-9563.

Bhardwaj, I., Kumar, A., & Bansal, M. (2017, September). A review on lightweight cryptography algorithms for data security and authentication in IoTs. In *2017 4th International Conference on Signal Processing, Computing and Control (ISPCC)*, 504-509.

Bhawiyuga, A., Data, M., & Warda, A. (2017, October). Architectural design of token based authentication of MQTT protocol in constrained IoT device. In *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, 1-4.

Blazy, O., Conchon, E., Klingler, M., & Sauveron, D. (2021). An iot attribute-based security framework for topic-based publish/subscribe systems. *IEEE Access*, *9*, 19066-19077.

Bluetooth SIG, (2014). *Specification of the Bluetooth System, Covered Core Package*; Version 4.2; Bluetooth Special Interest Group: Kirkland, WA, USA.

Boland, H., & Mousavi, H. (2004, May). Security issues of the IEEE 802.11 b wireless LAN. In *Canadian Conference on Electrical and Computer Engineering*, *1*, 333-336.

Calabretta, M., Pecori, R., & Veltri, L. (2018, September). A token-based protocol for securing MQTT communications. In *2018 26th International conference on software, telecommunications and computer networks (SoftCOM)*, 1-6.

Carpi, F., Martalò, M., Davoli, L., Cilfone, A., Yu, Y., Wang, Y., & Ferrari, G. (2023). Experimental analysis of RSSI-based localization algorithms with NLOS pre-mitigation for IoT applications. *Computer Networks*, *225*, 109663.

Cäsar, M., Pawelke, T., Steffan, J., & Terhorst, G. (2022). A survey on Bluetooth Low Energy security and privacy. *Computer Networks*, *205*, 108712.

Chatterjee, R., & Chakraborty, R. (2020, March). A Modified Lightweight PRESENT Cipher for IoT Security. In *International Conference on Computer Science, Engineering and Applications (ICCSEA)*, 1- 6.

Chaudhary, S. K., Yousuff, S., Meghana, N. P., Ashwin, T. S., & Guddeti, R. M. R. (2021). A multi-protocol home automation system using smart gateway. *Wireless Personal Communications*, *116*(3), 2367-2390.

Chen, J. C., Jiang, M. C., & Liu, Y. W. (2005). Wireless LAN security and IEEE 802.11 i. *IEEE Wireless Communications*, *12*(1), 27-36.

Da Silva Torres, R. J. (2020). *Identity management: analysis of secure authentication propositions* [Master's Thesis]. Universidade Do Porto, Porto, Portugal.

Deniz, E., & Samet, R. (2019). Nesnelerin internetinde ZigBee 3.0 ağlarına güvenli katılım için yeni bir model önerisi. *Uluslararası Bilgi Güvenliği Mühendisliği Dergisi*, *5*(1), 35-45.

Dian, F. J., Yousefi, A., & Lim, S. (2018, November). A practical study on Bluetooth Low Energy (BLE) throughput. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 768-771.

Dolev, D., & Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on information theory*, *29*(2), 198-208.

Ebleme, M. A., Bayilmis, C., & Cavusoglu, U. (2018, September). Examination and performance evaluation of MQTT. In *3rd International Conference on Computer Science and Engineering*, 246-250.

Ertaul, L., & Rajegowda, S. K. (2017). Performance analysis of CLEFIA, PICCOLO, TWINE Lightweight block ciphers in IoT environment. In *Proceedings of the International Conference on Security and Management (SAM)*, 25-31.

Esfahani, A., Mantas, G., Matischek, R., Saghezchi, F. B., Rodriguez, J., Bicaku, A., & Bastos, J. (2017). A lightweight authentication mechanism for M2M communications in industrial IoT environment. *IEEE Internet of Things Journal*, *6*(1), 288-296.

Esp32-s2 series datasheet. (n.d). Retrieved March 19, 2024, from https://www.espressif.com/sites/default/files/ documentation/esp32-s2 datasheet en.pdf.

Farooq, M. S., Abdullah, M., Riaz, S., Alvi, A., Rustam, F., Flores, M. A. L., & Ashraf, I. (2023). A survey on the role of industrial iot in manufacturing for implementation of smart industry. *Sensors*, *23*(21), 8958.

Fathy, C., & Ali, H. M. (2023). A secure IoT-based irrigation system for precision agriculture using the expeditious cipher. *Sensors*, *23*(4), 2091.

Farej, Z. K., & Saeed, A. M. (2020). Analysis and performance evaluation of bluetooth low energy piconet network. *Open Access Library Journal*, *7*(10), 1-11.

Fremantle, P., Aziz, B., Kopecký, J., & Scott, P. (2014, September). Federated identity and access management for the internet of things. In *International Workshop on Secure Internet of Things*, 10-17.

Ge, J., Wang, A., Zhu, L., Liu, X., Shang, N., & Zhang, G. (2019, October). Power Analysis and Protection on SPECK and Its Application in IoT. In *International Conference on Security and Privacy in Communication Systems*, 350-362).

Ghael, H. D., Solanki, L., & Sahu, G. (2020). A review paper on raspberry pi and its applications. *International Journal of Advances in Engineering and Management (IJAEM)*, *2*(12), 4.

Gupta, C., & Varshney, G. (2023). An improved authentication scheme for BLE devices with no I/O capabilities. *Computer Communications*, *200*, 42-53.

Hajian, R., Erfani, S. H., & Kumari, S. (2022). A lightweight authentication and key agreement protocol for heterogeneous IoT with special attention to sensing devices and gateway. *The Journal of Supercomputing*, *78*(15), 16678-16720.

Hanggoro, D., Rosyidi, L., & Sari, R. F. (2019, July). Design and implementation of multi-protocol gateway for Internet of Things. In *2019 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, 64-69.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework (RFC6749). *Internet Engineering Task Force (IETF)*, 1-76.

Hung, S. H., Yao, C. Y., Fang, Y. J., Tan, P., Lee, R. R., Sheffer, A., & Chu, H. K. (2019). Micrography QR codes. *IEEE transactions on visualization and computer graphics*, *26*(9), 2834-2847.

Jabbar, W. A., Kian, T. K., Ramli, R. M., Zubir, S. N., Zamrizaman, N. S., Balfaqih, M., & Alharbi, S. (2019). Design and fabrication of smart home with internet of things enabled automation system. *IEEE access*, *7*, 144059-144074.

Katsikeas, S., Fysarakis, K., Miaoudakis, A., Van Bemten, A., Askoxylakis, I., Papaefstathiou, I., & Plemenos, A. (2017, July). Lightweight & secure industrial

IoT communications via the MQ telemetry transport protocol. In *IEEE Symposium on Computers and Communications (ISCC)*, 1193-1200.

Khanchuea, K., & Siripokarpirom, R. (2019, March). A multi-protocol IoT gateway and WiFi/BLE sensor nodes for smart home and building automation: Design and implementation. In *2019 10th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES)* (pp. 1-6). IEEE.

Kim, H. J., & Kim, H. S. (2011, June). AUTHHOTP-HOTP based authentication scheme over home network environment. In *International Conference on Computational Science and Its Applications*, 622-637.

Lee, J. H., & Lim, D. G. (2014). Parallel architecture for high-speed block cipher, HIGHT. *International Journal of Security and Its Applications*, *8*(2), 59-66.

Lee, S. W., & Sim, K. B. (2021). Design and hardware implementation of a simplified DAG-based blockchain and new AES-CBC algorithm for IoT security. *Electronics*, *10*(9), 1127.

Li, R., Zhang, W., Wu, L., Tang, Y., & Xie, X. (2023). ZPA: a smart home privacy analysis system based on ZigBee encrypted traffic. *Wireless Communications and Mobile Computing*, *2023*(1), 6731783.

Madakam, S., Lake, V., Lake, V., & Lake, V. (2015). Internet of Things (IoT): A literature review. *Journal of Computer and Communications*, *3*(05), 164.

Mahmood, K., Chaudhry, S. A., Naqvi, H., Shon, T., & Ahmad, H. F. (2016). A lightweight message authentication scheme for smart grid communications in power sector. *Computers & Electrical Engineering*, *52*, 114-124.

Mahto, D., Khan, D. A., & Yadav, D. K. (2016, June). Security analysis of elliptic curve cryptography and RSA. In *Proceedings of the world congress on engineering*, *1*, 419-422.

Marktscheffel, T., Gottschlich, W., Popp, W., Werli, P., Fink, S. D., Bilzhause, A., & de Meer, H. (2016, June). QR code based mutual authentication protocol for

Internet of Things. In *IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 1-6.

Misal, S. R., Prajwal, S. R., Niveditha, H. M., Vinayaka, H. M., & Veena, S. (2020, March). Indoor positioning system (IPS) Using ESP32, MQTT and bluetooth. In *2020 fourth international conference on computing methodologies and communication (ICCMC)*, 79-82.

M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., & Ranen, O. (2005). Hotp: An hmac-based one-time password algorithm. *In No. rfc4226.*

Munshi, A. (2022). Improved MQTT secure transmission flags in smart homes. *Sensors*, *22*(6), 2174.

Nagarajan, S. M., Deverajan, G. G., Chatterjee, P., Alnumay, W., & Ghosh, U. (2021). Effective task scheduling algorithm with deep learning for Internet of Health Things (IoHT) in sustainable smart cities. *Sustainable Cities and Society*, *71*, 102945.

Naveen, P., Teja, M. S., Kalyan, K. P., & Basha, S. M. (2021). Bank locker security system using QR code. *Annals of the Romanian Society for Cell Biology*, 4218-4227.

Niruntasukrat, A., Issariyapat, C., Pongpaibool, P., Meesublak, K., Aiumsupucgul, P., & Panya, A. (2016, May). Authorization mechanism for MQTT-based Internet of Things. In *2016 IEEE international conference on communications workshops (ICC)*, 290-295.

OASIS MQTT Version 3.1.1 Plus Errata 01. (n.d). Retrieved May 19, 2024, from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf.

Pang, B., Claeys, T., T'Jonck, K., Hallez, H., & Boydens, J. (2022, September). Bluetooth low energy reliability and throughput under Wi-Fi interference. In *XXXI International Scientific Conference Electronics (ET)*, 1-4).

Pardede, M., Hutajulu, E., & Sirait, R. (2020). Laboratory Room Control Access and Monitoring System Using Fingerprint and XBee Pro S2C. *Jurnal Mantik*, *4*(3), 1921-1928.

Park, C. S. (2018). One-time password based on hash chain without shared secret and re-registration. *Computers & Security*, *75*, 138-146.

Patel, C., & Doshi, N. (2023). LDA-2IoT: A level dependent authentication using two factor for IoT paradigm. *Computer Networks*, *223*, 109580.

Pearson, B., Luo, L., Zhang, Y., Dey, R., Ling, Z., Bassiouni, M., & Fu, X. (2019, May). On misconception of hardware and cost in IoT security and privacy. In *ICC IEEE International Conference on Communications (ICC)*, 1-7.

Qiu, T., Chi, J., Zhou, X., Ning, Z., Atiquzzaman, M., & Wu, D. O. (2020). Edge computing in industrial internet of things: Architecture, advances and challenges. *IEEE Communications Surveys & Tutorials*, *22*(4), 2462-2488.

Qureshi, I. A., & Asghar, S. (2023). A systematic review of the IEEE-802.11 standard's enhancements and limitations. *Wireless Personal Communications*, *131*(4), 2539-2572.

Ragothaman, K., Wang, Y., Rimal, B., & Lawrence, M. (2023). Access control for IoT: A survey of existing research, dynamic policies and future directions. *Sensors*, *23*(4), 1805.

Rajan, M. A., Varghese, A., Narendra, N., Singh, M., Shivraj, V. L., Chandra, G., & Balamuralidhar, P. (2016, March). Security and privacy for real time video streaming using hierarchical inner product encryption based publish-subscribe architecture. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 373-380.

Ramyasri, G., Murthy, G. R., Itapu, S., & Krishna, S. M. (2023). Data transmission using secure hybrid techniques for smart energy metering devices. *e-Prime-Advances in Electrical Engineering, Electronics and Energy*, *4*, 100134.

Rizzardi, A., Sicari, S., & Coen-Porisini, A. (2022). Analysis on functionalities and security features of Internet of Things related protocols. *Wireless Networks*, *28*(7), 2857-2887.

Sanjuan, E. B., Cardiel, I. A., Cerrada, J. A., & Cerrada, C. (2020). Message queuing telemetry transport (MQTT) security: A cryptographic smart card approach. *IEEE Access*, *8*, 115051-115062.

Saputra, L. K. P., & Raharjo, W. S. (2019, October). Implementation of password-based key derivation function for authentication scheme in patrolling system. In *5th International Conference on New Media Studies (CONMEDIA)*, 31-35.

Saqib, M., Jasra, B., & Moon, A. H. (2022). A lightweight three factor authentication framework for IoT based critical applications. *Journal of King Saud University-Computer and Information Sciences*, *34*(9), 6925-6937.

Saxena, A. (2008, January). Dynamic authentication: Need than a choice. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*, 214-218.

Shilpa, V., Vidya, A., & Pattar, S. (2022). MQTT based secure transport layer communication for mutual authentication in IoT network. *Global Transitions Proceedings*, *3*(1), 60-66.

Shin, S., Kobara, K., Chuang, C. C., & Huang, W. (2016, October). A security framework for MQTT. In *IEEE Conference on Communications and Network Security (CNS)*, 432-436.

Sinha, R., Srivastava, H. K., & Gupta, S. (2013). Performance based comparison study of RSA and elliptic curve cryptography. *International Journal of Scientific & Engineering Research*, *4*(5), 720-725.

Singh, M., Rajan, M. A., Shivraj, V. L., & Balamuralidhar, P. (2015, April). Secure mqtt for internet of things (iot). In *fifth international conference on communication systems and network technologies*, 746-751.

Singh, S., Sharma, P. K., Moon, S. Y., & Park, J. H. (2017). Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, 1-18.

Stallings, W. (2006). Cryptography and network security principles and practice (6th ed.). India: Pearson Education.

Sundarrajan, M., & Narayanan, A. E. A. (2019). An authentication scheme for an IoT environment using advanced multiple encryption system. *Int. J. Innov. Tech. Expl. Eng*, *9*, 868-874.

Şeker, Ö., & Dalkiliç, G. (2022, September). Implementation and performance analysis of a multi-protocol gateway. In *Innovations in Intelligent Systems and Applications Conference (ASYU)*, 1-6.

Şeker, Ö., Dalkılıç, G., & Çabuk, U. C. (2023). MARAS: Mutual authentication and role-based authorization scheme for lightweight Internet of Things applications. *Sensors*, *23*(12), 5674.

Şeker, Ö., Şahin, B., Akdoğan, T., & Dalkılıç, G. (2024). A physical tracking of ESP32 IoT devices with RSSI based indoor position calculation. *Journal of Millimeterwave Communication, Optimization and Modelling*, *4*(1), 13-16.

Tosi, J., Taffoni, F., Santacatterina, M., Sannino, R., & Formica, D. (2017). Performance evaluation of bluetooth low energy: A systematic review. *Sensors*, *17*(12), 2898.

Upadhyay, Y., Borole, A., & Dileepan, D. (2016, March). MQTT based secured home automation system. In *Symposium on Colossal Data Analysis and Networking (CDAN)*, 1-4.

Wardana, A. A., & Perdana, R. S. (2018, July). Access control on internet of things based on publish/subscribe using authentication server and secure protocol. In *10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 118-123.

Winarno, A., & Sari, R. F. (2022). A novel secure end-to-end IoT communication scheme using lightweight cryptography based on block cipher. *Applied Sciences*, *12*(17), 8817.

Woolley, M. (2019, January). Bluetooth core specification v5. 1. In *Bluetooth*.

Valea, E., Da Silva, M., Flottes, M. L., Di Natale, G., Dupuis, S., & Rouzeyre, B. (2019, April). Providing confidentiality and integrity in ultra low power iot devices. In *14th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 1-6.

Valchanov, H., Edikyan, J., & Aleksieva, V. (2019, October). A study of Wi-Fi security in city environment. In *IOP Conference Series: Materials Science and Engineering*, *618*(1), 012031.

Yerlikaya, Ö., & Dalkılıç, G. (2017, May). Security of message queue telemetry transport protocol. In *25th Signal Processing and Communications Applications Conference (SIU)*, 1-4.

Zamfir, S., Balan, T., Iliescu, I., & Sandu, F. (2016, October). A security analysis on standard IoT protocols. In *international conference on applied and theoretical electricity (ICATE)*, 1-6.

Zhang, Q., Liang, Z., & Cai, Z. (2019). Developing a new security framework for Bluetooth Low Energy devices. *Computers, Materials & Continua*, *59*(2).

Zohourian, A., Dadkhah, S., Neto, E. C. P., Mahdikhani, H., Danso, P. K., Molyneaux, H., & Ghorbani, A. A. (2023). IoT Zigbee device security: A comprehensive review. *Internet of Things*, 100791.

APPENDICES

Appendix 1: RSSI Measurements and Distances for Indoor and Outdoor Environments

	Indoor								Outdoor								Indoor			
	(RSSI dBm)				Distance (cm)				RSSI (dBm)				Distance (cm)				Actual Distance (cm)			
Coord	AP1 (0,0)	AP2 (6,0)	AP3 (0,6)	AP4 (6,0)	AP1 (0,0)	AP2 (6,0)	AP3 (0,6)	AP4 (6,0)	AP1 (0,0)	AP2 (6,0)	AP3 (0,6)	AP4 (6,0)	AP1 (0,0)	AP2 (6,0)	AP3 (0,6)	AP4 (6,0)	AP1 (0,0)	AP2 (6,0)	AP3 (0,6)	AP4 (6,0)
(0,5)	-58	-65	-50	-71	53.09	79.43	33.50	112.20	-87	-83	-86	-82	281.84	223.87	266.07	211.35	40.00	200.00	243.31	312.41
(1,5)	-60	-74	-50	-66	59.57	133.35	33.50	84.14	-86	-85	-89	-80	266.07	251.19	316.23	188.36	56.57	203.96	282.84	203.96
(2,5)	-61	-62	-58	-65	63.10	66.83	53.09	79.43	-87	-84	-91	-83	281.84	237.14	354.81	223.87	89.42	120.00	164.92	256.12
(3,5)	-66	-66	-56	-70	84.14	84.14	47.32	105.93	-93	-87	-86	-86	398.11	281.84	266.07	266.07	126.49	233.34	126.49	233.24
(4,5)	-65	-62	-59	-66	79.43	66.83	56.23	84.14	-90	-91	-88	-87	334.97	354.81	298.54	281.84	164.92	256.12	89.44	215.41
(5,5)	-65	-62	-66	-68	79.43	66.83	84.14	94.41	-91	-86	-86	-84	354.81	266.07	266.07	237.14	203.96	282.84	56.57	203.96
(0,4)	-71	-70	-60	-71	112.20	105.93	59.57	112.20	-87	-81	-85	-86	281.84	199.53	251.19	266.07	80.00	160.00	252.98	288.44
(1,4)	-70	-67	-66	-74	105.93	89.13	84.14	133.35	-91	-82	-85	-92	354.81	211.35	251.19	375.84	89.44	164.92	215.41	256.12
(2,4)	-71	-66	-61	-66	112.20	84.14	63.10	84.14	-90	-84	-87	-84	334.97	237.14	281.84	237.14	113.14	178.89	226.27	178.89
(3,4)	-75	-67	-71	-74	141.25	89.13	112.20	133.35	-91	-88	-84	-86	354.81	298.54	237.14	266.07	144.20	200.00	144.20	200.00
(4,4)	-77	-66	-74	-73	158.49	84.14	133.35	125.89	-94	-83	-94	-91	421.70	223.87	421.70	354.81	178.89	226.27	113.14	178.89
(5,4)	-71	-69	-69	-75	112.20	100.00	100.00	141.25	-94	-89	-88	-86	421.70	316.23	298.54	266.07	215.41	256.12	89.44	164.92
(0,3)	-77	-73	-72	-74	158.49	125.89	118.85	133.35	-95	-90	-91	-87	446.68	334.97	354.81	281.84	120.00	120.00	268.33	268.33
(1,3)	-82	-74	-70	-76	211.35	133.35	105.93	149.62	-92	-82	-88	-86	375.84	211.35	298.54	266.07	126.49	126.49	233.24	233.24
(2,3)	-74	-75	-71	-75	133.35	141.25	112.20	141.25	-87	-87	-87	-92	281.84	281.84	281.84	375.84	144.22	144.22	200.00	200.00
(3,3)	-76	-73	-71	-82	149.62	125.89	112.20	211.35	-91	-84	-87	-84	354.81	237.14	281.84	237.14	169.71	169.71	169.71	169.71
(4,3)	-75	-84	-71	-77	141.25	237.14	112.20	158.49	-91	-85	-89	-88	354.81	251.19	316.23	298.54	200.00	200.00	144.22	144.22
(5,3)	-75	-76	-67	-75	141.25	149.62	89.13	141.25	-85	-89	-86	-85	251.19	316.23	266.07	251.19	233.24	233.24	126.49	126.49
(0,2)	-78	-71	-66	-81	167.88	112.20	84.14	199.53	-92	-83	-91	-86	375.84	223.87	354.81	266.07	160.00	80.00	288.44	252.98
(1,2)	-81	-73	-62	-78	199.53	125.89	66.83	167.88	-96	-89	-84	-82	473.15	316.23	237.14	211.35	164.92	89.44	256.12	215.41
(2,2)	-73	-70	-62	-79	125.89	105.93	66.83	177.83	-90	-86	-96	-83	334.97	266.07	473.15	223.87	178.89	113.14	226.27	178.89
(3,2)	-79	-71	-57	-70	177.83	112.20	50.12	105.93	-91	-82	-94	-83	354.81	211.35	421.70	223.87	200.00	144.22	144.22	200.00
(4,2)	-69	-74	-65	-75	100.00	133.35	79.43	141.25	-86	-79	-84	-82	266.07	177.83	237.14	211.35	226.27	178.89	178.89	113.14
(5,2)	-74	-75	-71	-85	133.35	141.25	112.20	251.19	-92	-79	-90	-81	375.84	177.83	334.97	199.53	256.12	215.49	164.92	89.44
(0,1)	-83	-83	-70	-91	223.87	223.87	105.93	354.81	-91	-93	-88	-90	354.81	398.11	298.54	334.97	200.00	40.00	312.41	243.31
(1,1)	-82	-79	-72	-76	211.35	177.83	118.85	149.62	-95	-89	-91	-85	446.68	316.23	354.81	251.19	203.96	56.57	282.84	203.96
(2,1)	-82	-69	-72	-72	211.35	100.00	118.85	118.85	-94	-93	-90	-93	421.70	398.11	334.97	398.11	215.41	89.44	256.12	164.92
(3,1)	-79	-77	-69	-78	177.83	158.49	100.00	167.88	-96	-83	-90	-85	473.15	223.87	334.97	251.19	233.24	126.49	233.24	126.49
(4,1)	-82	-71	-71	-81	211.35	112.20	112.20	199.53	-93	-84	-98	-84	398.11	237.14	530.88	237.14	256.12	164.92	215.41	89.44
(5,1)	-70	-71	-73	-76	105.93	112.20	125.89	149.62	-88	-88	-87	-77	298.54	298.54	281.84	158.49	282.84	203.96	203.96	56.57
(0,0)	-79	-70	-68	-71	177.83	105.93	94.41	112.20	-91	-87	-92	-83	354.81	281.84	375.84	223.87	240.00	0.00	339.41	240.00
(1,0)	-68	-72	-65	-74	94.41	118.85	79.43	133.35	-87	-84	-88	-81	281.84	237.14	298.54	199.53	243.31	40.00	312.41	200.00
(2,0)	-68	-82	-61	-77	94.41	211.35	63.10	158.49	-90	-84	-90	-83	334.97	237.14	334.97	223.87	252.98	80.00	288.44	160.00
(3,0)	-81	-84	-84	-82	199.53	237.14	237.14	211.35	-91	-86	-89	-80	354.81	266.07	316.23	188.36	268.33	120.00	268.33	120.00
(4,0)	-75	-71	-71	-83	141.25	112.20	112.20	223.87	-90	-82	-88	-86	334.97	211.35	298.54	266.07	288.44	160.00	252.98	80.00
(5,0)	-74	-69	-78	-71	133.35	100.00	167.88	112.20	-91	-89	-87	-87	354.81	316.23	281.84	281.84	312.41	200.00	243.31	40.00

Appendix 2: Message Reliability, Throughput, and Transmission Time for BLE and ZigBee

BLE	Connection Interval	Send Number of Messages	Receive Number of Messages	Publish Number of MQTT message	BLE_RTT(us)	Running Code Process (us)	MQTT Publish Messages(us)	Delta (ms)	Througput (Kbps)
No Security 2500 messages									
16 Bytes	**7.5**	2500	1570	1014	471	609	2345	14.10	170.67
	40	2500	2500	2500	470	606	2618	0.01	272.34
	80	2500	2500	2500	469	605	2726	0.02	272.92
	100	2500	2500	2500	470	605	2699	0.02	272.34
	150	2500	2500	2500	470	605	2787	0.01	272.34
32 Bytes	**7.5**	2500	1405	15	648	806	10939	34.58	222.02
	40	2500	2500	2500	670	832	2758	0.43	382.09
	80	2500	2500	2500	669	839	2755	0.30	382.66
	100	2500	2500	2500	671	831	2792	0.42	381.52
	150	2500	2500	2500	668	838	2818	0.22	383.23
64 Bytes	**7.5**	2500	1720	137	1081	1253	5568	22.02	325.86
	40	2500	2500	2500	1084	1256	2999	11.31	472.32
	80	2500	2500	2500	1086	1254	2804	10.99	471.45
	100	2500	2500	2500	1085	1253	2873	10.97	471.89
	150	2500	2500	2500	1085	1261	2872	10.98	471.89
No Security 500 messages									
16 Bytes	**7.5**	500	500	500	470	608	2309	13.78	272.34
	40	500	500	500	467	604	3011	0.13	274.09
	80	500	500	500	459	593	2699	0.16	278.87
	100	500	500	500	469	604	2605	0.13	272.92
	150	500	500	500	468	603	2415	0.12	273.50
32 Bytes	**7.5**	500	500	500	674	837	2123	17.56	379.82
	40	500	500	500	671	833	2677	0.59	381.52
	80	500	500	500	671	831	2725	0.45	381.52
	100	500	500	500	670	830	2732	0.41	382.09
	150	500	500	500	668	828	2792	0.39	383.23
64 Bytes	**7.5**	500	500	500	1085	1274	2001	20.63	471.89
	40	500	500	500	1085	1275	2502	11.78	471.89
	80	500	500	500	1087	1255	2768	10.92	471.02
	100	500	500	500	1085	1273	2724	10.96	471.89
	150	500	500	500	1085	1253	2875	10.96	471.89

Appendix 2: Continues

BLE	Connection Interval	Send Number of Messages	Receive Number of Messages	Publish Number of MQTT message	BLE_RTT(us)	Running Code Process (us)	MQTT Publish Messages(us)	Delta (ms)	Througput (Kbps)
With Security 2500 messages									
16 Bytes	**7.5**	2500	1101	1000	854	1674	3196	13.28	66.01
	40	2500	2500	2500	864	1674	5766	11.98	148.15
	80	2500	2500	2500	863	1672	4639	11.98	148.32
	100	2500	2500	2500	862	1670	3454	11.98	148.49
	150	2500	2500	2500	863	1671	4029	12.00	148.32
32 Bytes	**7.5**	2334	1115	1115	1264	8328	3196	15.26	90.33
	40	2500	2255	2226	1309	8328	5766	12.51	176.40
	80	2500	2500	2500	1314	8299	4639	12.45	194.82
	100	2500	2500	2500	1310	8288	3454	12.47	195.42
	150	2500	2500	2500	1315	8248	4029	12.46	194.68
64 Bytes	**7.5**	2335	1116	-	2717	29152	3196	15.22	84.12
	40	2500	2227	-	2717	14739	5766	12.16	167.87
	80	2500	2500	2500	2713	29087	4639	10.99	188.72
	100	2500	2500	2500	2710	29045	3454	10.97	188.93
	150	2500	2500	2500	2709	29043	4029	12.34	189.00
With Security 500 messages									
16 Bytes	**7.5**	500	500	500	855	1737	3196	13.39	149.71
	40	500	500	500	856	1739	5766	11.98	149.53
	80	500	500	500	856	1.738	4639	11.94	149.53
	100	500	500	500	855	1737	3454	11.85	149.71
	150	500	500	500	855	1736	4029	11.86	149.71
32Bytes	**7.5**	500	500	500	1264	12801	3196	15.63	202.53
	40	500	500	500	1263	12841	5766	12.12	202.69
	80	500	500	500	1262	12860	4639	12.09	202.85
	100	500	500	500	1264	12801	3454	11.95	202.53
	150	500	500	500	1263	12799	4029	11.87	202.69

Appendix 2: Continues

BLE	Connection Interval	Send Number of Messages	Receive Number of Messages	Publish Number of MQTT message	BLE_RTT(us)	Running Code Process (us)	MQTT Publish Messages(us)	Delta (ms)	Througput (Kbps)
With Security 500 messages									
64Bytes	**7.5**	500	500	500	2717	29152	3196	40.22	188.44
	40	500	500	500	1937	14739	5766	24.76	264.33
	80	500	500	500	2713	29087	4639	39.05	188.72
	100	500	500	500	2710	29045	3454	38.80	188.93
	150	500	500	500	2709	29043	4029	38.72	189.00

ZigBee	Connection Interval (ms)	Send Number of Messages	Receive Number of Messages	Publish Number of MQTT message	ZigBee_RTT(ms)	Running Code Process (ms)	MQTT Publish Messages(ms)	Delta (ms)	Througput (Kbps)
No Security - 2500 messages									
16 Bytes	**80**	2500	2500	2500	17.69	17.80	1.88	17.16	7.24
	100	2500	2500	2500	17.69	17.80	1.92	17.00	7.24
	150	2500	2500	2500	17.00	17.80	1.94	17.00	7.53
32 Bytes	**80**	2500	2500	2500	33.29	33.41	1.90	32.93	7.69
	100	2500	2500	2500	33.29	33.41	1.92	33.00	7.69
	150	2500	2500	2500	33.29	33.41	1.95	33.00	7.69
64 Bytes	**80**	2500	2500	2500	66.59	66.71	2.01	65.86	7.69
	100	2500	2500	2500	66.58	66.70	1.86	66.01	7.69
	150	2500	2494	2494	66.58	66.70	1.96	66.01	7.67
No Security - 500 messages									
16 Bytes	**80**	500	500	500	17.69	17.80	1.88	17.16	7.24
	100	500	500	500	17.69	17.80	1.92	17.00	7.24
	150	500	500	500	17.69	17.80	1.94	17.00	7.24
32 Bytes	**80**	500	500	500	33.29	33.41	1.90	32.93	7.69
	100	500	500	500	33.29	33.41	1.92	33.00	7.69
	150	500	500	500	33.29	33.41	1.95	33.00	7.69
64 Bytes	**80**	500	500	500	66.59	66.71	2.01	65.86	7.69
	100	500	500	500	66.58	66.70	1.86	66.01	7.69
	150	500	500	500	66.58	66.70	1.96	66.01	7.69

Appendix 2: Continues

ZigBee	Connection Interval (ms)	Send Number of Messages	Receive Number of Messages	Publish Number of MQTT message	ZigBee_RTT(ms)	Running Code Process (ms)	MQTT Publish Messages(ms)	Delta (ms)	Througput (Kbps)
With Security - 500 messages									
16 Bytes	**80**	500	500	500	41.61	42.29	1.92	42.00	3.08
	100	500	500	500	41.61	42.29	1.97	42.01	3.08
	150	500	500	500	41.61	42.29	2.05	42.81	3.08
32 Bytes	**80**	500	500	500	74.90	76.14	1.99	76.12	3.42
	100	500	500	500	74.90	76.13	2.05	75.32	3.42
	150	500	500	500	74.90	76.14	2.06	72.92	3.42
64 Bytes	**80**	500	200	0	141.48	154.20	0.00	1502.00	1.45
	100	500	100	0	141.46	154.20	0.00	1256.00	0.72
	150	500	125	0	141.47	154.19	0.00	1070.00	0.90
With Security - 2500 messages									
16 Bytes	**80**	2500	2494	2494	41.62	42.29	1.92	47.54	3.07
	100	2500	2500	2499	41.61	42.29	1.97	43.20	3.08
	150	2500	2500	2500	41.62	42.29	2.05	42.00	3.08
32 Bytes	**80**	2500	2492	2492	74.90	76.14	1.99	76.06	3.41
	100	2500	2500	2490	74.90	76.13	2.05	76.50	3.42
	150	2500	2500	2500	74.90	76.14	2.06	76.09	3.42
64 Bytes	**80**	2500	100	0	141.48	154.20	0.00	1541.00	0.14
	100	2500	100	0	141.46	154.20	0.00	1170.00	0.14
	150	2500	100	0	141.47	154.19	0.00	154.14	0.14

Appendix 3: Performance Metrics for Key Generation, Encryption, and Signing with Various Payloads Using ECDSA Algorithm on ESP32

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
16 bytes	219														
		42	32	04058DCE4A2A3FB8A929A053B93F0ECE	153	101	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1817	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	516
		50	32	04058DCE4A2A3FB8A929A053B93F0ECE	80	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1777	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	516
		43	32	04058DCE4A2A3FB8A929A053B93F0ECE	87	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1782	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	515
		44	32	04058DCE4A2A3FB8A929A053B93F0ECE	80	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1777	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	516

Appendix 3: Continues

Payload	Key generation	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
16 Bytes		42	32	04058DCE4A2A3FB8A929A053B93F0ECE	95	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1793	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	516
		44	32	04058DCE4A2A3FB8A929A053B93F0ECE	80	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1778	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	515
		42	32	04058DCE4A2A3FB8A929A053B93F0ECE	87	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1783	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	516
		44	32	04058DCE4A2A3FB8A929A053B93F0ECE	80	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1777	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	515

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
16 Bytes		42	32	04058DCE4A2A3FB8A929A053B93F0ECE	87	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1782	146	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	517
		45	32	04058DCE4A2A3FB8A929A053B93F0ECE	80	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1778	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	507
		42	32	04058DCE4A2A3FB8A929A053B93F0ECE	86	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1781	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	510
		44	32	04058DCE4A2A3FB8A929A053B93F0ECE	80	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1812	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	507

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
16 Bytes		42	32	04058DCE4A2A3FB8A929A053B93F0ECE	87	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1782	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	509
	218	44	32	04058DCE4A2A3FB8A929A053B93F0ECE	80	100	3034303538444345344132413346423841393239413035334239334630454345	21D2FDD98CDB4B3E1C3226A34FCABB3024628AAE885C29EAE4FCC8F3E61	4C40407CB01FD28F7E7BB1793DB99F1EDFA594D38D5C409C5D404B87B12BB3	30452206C5039A0E9F4D8E2F731BCD95F14E74F5FF782AA1B19BDF6FD61C6B904FD2722210BB74286A66C3EA9DD8CDF93FB666C224C061B2E4C5F1F5C1D69986198A7C49D4	1777	146	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	510
		42	32	04058DCE4A2A3FB8A929A053B93F0ECE	139	100	3034303538444345344132413346423841393239413035334239334630454345	C7CB293865D4D980819C51E8389F4B622D6F911325E4A11B73FC924A6864D170	E28F80787AACA62D104E25137D3BC46BC9A8D30C2FBBF223508CCCAD061BA	30462210F087732FF2C87B0C8AB65361672D7353D8F465356A2F1F3FB1CCC65CC35FF22108E1BE78EBBF5BD55BCAB581FD41611168F519137201A6D3F4F981D4850FA70	2059	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	510
		58	32	04058DCE4A2A3FB8A929A053B93F0ECE	81	100	3034303538444345344132413346423841393239413035334239334630454345	C7CB293865D4D980819C51E8389F4B622D6F911325E4A11B73FC924A6864D170	E28F80787AACA62D104E25137D3BC46BC9A8D30C2FBBF223508CCCAD061BA	30462210F087732FF2C87B0C8AB65361672D7353D8F465356A2F1F3FB1CCC65CC35FF22108E1BE78EBBF5BD55BCAB581FD41611168F519137201A6D3F4F981D4850FA70	1980	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	510
		52	32	04058DCE4A2A3FB8A929A053B93F0ECE	85	100	3034303538444345344132413346423841393239413035334239334630454345	C7CB293865D4D980819C51E8389F4B622D6F911325E4A11B73FC924A6864D170	E28F80787AACA62D104E25137D3BC46BC9A8D30C2FBBF223508CCCAD061BA	30462210F087732FF2C87B0C8AB65361672D7353D8F465356A2F1F3FB1CCC65CC35FF22108E1BE78EBBF5BD55BCAB581FD41611168F519137201A6D3F4F981D4850FA70	1991	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	509

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
16 Bytes		53	32	04058DCE4A2A3FB8A929A053B93F0ECE	81	100	3034303538444345344132413346423841393239413035334239334630454345	C7CB293865D4D980819C51E8389F4B622D6F911325E4A11B73FC924A6864D170	E28F80787AACA62D104E25137D3BC46BC9A8D30C2FBBF223508CCCAD061BA	30462210F087732FF2C87B0C8AB65361672D7353D8F465356A2F1F3FB1CCC65CC35FF22108E1BE78EBBF5BD55BCAB581FD41611168F519137201A6D3F4F981D4850FA70	1981	147	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	507
		51	32	04058DCE4A2A3FB8A929A053B93F0ECE	86	101	3034303538444345344132413346423841393239413035334239334630454345	C7CB293865D4D980819C51E8389F4B622D6F911325E4A11B73FC924A6864D170	E28F80787AACA62D104E25137D3BC46BC9A8D30C2FBBF223508CCCAD061BA	30462210F087732FF2C87B0C8AB65361672D7353D8F465356A2F1F3FB1CCC65CC35FF22108E1BE78EBBF5BD55BCAB581FD41611168F519137201A6D3F4F981D4850FA70	1987	148	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	510
		53	32	04058DCE4A2A3FB8A929A053B93F0ECE	81	101	3034303538444345344132413346423841393239413035334239334630454345	C7CB293865D4D980819C51E8389F4B622D6F911325E4A11B73FC924A6864D170	E28F80787AACA62D104E25137D3BC46BC9A8D30C2FBBF223508CCCAD061BA	30462210F087732FF2C87B0C8AB65361672D7353D8F465356A2F1F3FB1CCC65CC35FF22108E1BE78EBBF5BD55BCAB581FD41611168F519137201A6D3F4F981D4850FA70	1984	148	04058DCE4A2A3FB8A929A053B93F0ECE	Valid	510
	Average:	**45.95**			**89.75**	**100.15**					**1848.9**	**147**			**512.05**
32 Bytes	219														
		41	64	81EF992CB35A9AE114C4E79BE81ACE85F8805F9BD9118A307CF073C9F2D3D058	148	102	3831454639393243423335413941453131344334453739424538314143453835463838303546394244393131384133303743463037334339463244334430 3538	1DDCFC9862DC67985A2BDCB6255DA3E82CF88CC9CAF0ABF9305223F0F5B1C0	201C8597CA1BBB63C1385BC5DF651DA6A1A19479371EC3F6133D72E353B6B	304622108B69DBD07B4042DA20EE204EE5AD7E7B8F725ECA7955C921F4E434C5A2652210D76222F88C1D1E8A5D5AB662F4A9AFFDEDDF366BC19ADF01B77AFB7E4A12	2095	160	81EF992CB35A9AE114C4E79BE81ACE85F8805F9BD9118A307CF073C9F2D3D058	Valid	588

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
32 Bytes		50	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	102	383145463939324342333541394145313134433344537394245383141434538354638383035463944243931381383303074346303733433946324433443035538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2286	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	589
		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	94	101	383145463939324342333541394145313134433344537394245383141434538354638383035463944243931381383303074346303733433946324433443035538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2272	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	101	383145463939324342333541394145313134433344537394245383141434538354638383035463944243931381383303074346303733433946324433443035538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2246	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	93	101	383145463939324342333541394145313134433344537394245383141434538354638383035463944243931381383303074346303733433946324433443035538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2269	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	589
		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	101	383145463939324342333541394145313134433344537394245383141434538354638383035463944243931381383303074346303733433946324433443035538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2273	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588

Appendix 3: Continues

Payload	Payload	Key generation time (ns)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity
32 Bytes		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	94	101	3831454639393243423335413941453 1313443344537394245383141434538 3546383830354639424439313138413 3303743463037334339463244334430 3538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2254	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	90	101	3831454639393243423335413941453 1313443344537394245383141434538 3546383830354639424439313138413 3303743463037334339463244334430 3538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2274	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	589
		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	94	101	3831454639393243423335413941453 1313443344537394245383141434538 3546383830354639424439313138413 3303743463037334339463244334430 3538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2248	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		55	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	102	3831454639393243423335413941453 1313443344537394245383141434538 3546383830354639424439313138413 3303743463037334339463244334430 3538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2253	150	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	94	100	3831454639393243423335413941453 1313443344537394245383141434538 3546383830354639424439313138413 3303743463037334339463244334430 3538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2249	148	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	590

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
32 Bytes		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	101	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931381383303074346303733433946324433443038538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2277	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	94	101	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931381383303074346303733433946324433443038538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2253	149	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	90	103	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931381383303074346303733433946324433443038538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2249	151	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	589
		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	94	104	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931381383303074346303733433946324433443038538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2273	151	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	103	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931381383303074346303733433946324433443038538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2256	151	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
32 Bytes		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	94	104	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931331384133037434630373343394632443344303538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2253	152	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	590
		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	106	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931331384133037434630373343394632443344303538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2247	154	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	589
		52	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	96	106	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931331384133037434630373343394632443344303538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2255	154	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
		53	64	81EF992CB35 A9AE114C4E7 9BE81ACE85F 8805F9BD9118 A307CF073C9 F2D3D058	89	108	383145463939324342333541394145313134433344537394245383141434538354638383035463942443931331384133037434630373343394632443344303538	1DDCFC9862 DC67985A2B DCB6255DA3 E82CF88CC9C AF0ABF93052 23F0F5B1C0	201C8597CA1BBB63 C1385BC5DF651DA6 A1A19479371EC3F61 33D72E353B6B	304622108B69DBD07 B4042DA20EE204EE 5AD7E7B8F725ECA7 955C921F4E434C5A2 652210D76222F88C1 D1E8A5D5AB662F4A 9AFFDEDDF366BC1 9ADF01B77AFB7E4A 12	2254	156	81EF992CB3 5A9AE114C4 E79BE81ACE 85F8805F9B D9118A307C F073C9F2D3 D058	Valid	588
	Average:	**51.9**	**64**		**94.35**	**102.45**					**2251.8**	**150.85**			**588.45**
9	218														

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
64 Bytes		44	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	157	100	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445354634544544374445344434131364339413533353935384542384442324637414338313131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2296	183	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590
		53	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	102	101	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445354634544544374445344434131364339413533353935384542384442324637414338313131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2529	172	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	107	100	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445354634544544374445344434131364339413533353935384542384442324637414338313131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2475	171	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
64 Bytes		55	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	103	101	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445354363531324239423345374613635341394334383336354338373945346354133453635313242394233453746413636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741 43383131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2453	172	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	108	101	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445354363531324239423345374613635341394334383336354338373945346354133453635313242394233453746413636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741 43383131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2466	172	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		55	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	96	101	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445354363531324239423345374613635341394334383336354338373945346354133453635313242394233453746413636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741 43383131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2460	172	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
64 Bytes		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	102	102	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445363531324239423345374613635341373945383531383134374238454534454637444534434431364339413533353935384542384442324637414338313 1	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2448	173	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590
		55	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	96	102	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445363531324239423345374613635341373945383531383134374238454534454637444534434431364339413533353935384542384442324637414338313 1	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2462	174	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	107	103	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445363531324239423345374613635341373945383531383134374238454534454637444534434431364339413533353935384542384442324637414338313 1	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2485	175	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	591

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
64 Bytes		56	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	97	104	383145463939324342333541394145313134433445373942453831414345385333333039393641383834374641344635413943343833336354338373945344233453635313242394233453746413636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741433831313	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2456	176	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		64	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	101	108	383145463939324342333541394145313134433445373942453831414345385333333039393641383834374641344635413943343833336354338373945344233453635313242394233453746413636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741433831313	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2478	179	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		55	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	103	107	383145463939324342333541394145313134433445373942453831414345385333333039393641383834374641344635413943343833336354338373945344233453635313242394233453746413636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741433831313	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2474	178	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
64 Bytes		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	107	109	383145463939324342333541394145313134433445373942453831414345385333330393936413838343746413445354139433438333635433837394534423345363531324239423345374641363633413739453835313831334742384545344546374445344344313643394135333539353845423844423246374143383131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2506	180	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590
		55	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	103	110	383145463939324342333541394145313134433445373942453831414345385333330393936413838343746413445354139433438333635433837394534423345363531324239423345374641363633413739453835313831334742384545344546374445344344313643394135333539353845423844423246374143383131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2528	181	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	591
		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	108	110	383145463939324342333541394145313134433445373942453831414345385333330393936413838343746413445354139433438333635433837394534423345363531324239423345374641363633413739453835313831334742384545344546374445344344313643394135333539353845423844423246374143383131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2483	182	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
64 Bytes		55	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	97	121	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445363531324239423345374613636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741433831	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2473	192	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	591
		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	102	121	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445363531324239423345374613636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741433831	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2540	192	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		57	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	97	122	383145463939324342333541394145313134433445373942453831414345385330393936413838343746413445363531324239423345374613636334137394538353138313437423845453445463744453443443136433941353335393538454238444232463741433831	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2494	193	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590

Appendix 3: Continues

Payload	Key generation time (ms)	Encryption time (us)	msg_len (HEX)	Encrypted (AES) messages	Hashing time (ms)	Signing time (ms)	Message (HEX)	Public key X (HEX)	Public key Y (HEX)	Signature (HEX)	BLE RTT (us)	Process time (ms)	Received message	Signature validity	Verification time (ms)
64 Bytes		53	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	102	101	383145463939324342333541394145313134433445373942453831414345385333330393936413838343746413446354139433438333635433837394534423345363531324239423345374641363633413739453835313831334743746454534454637444534434431364339413533353935384542384442324637414338313131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2529	172	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	589
		54	128	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	115	120	383145463939324342333541394145313134433445373942453831414345385333330393936413838343746413446354139433438333635433837394534423345363531324239423345374641363633413739453835313831334743746454534454637444534434431364339413533353935384542384442324637414338313131	618D24C4E1542EAC1CDCD8BABCFA3088C2E627FE3880EE50BC7F3D83CB078	C342E660E1F75B39C763619A99211D68ADC5247EE0FCA7ABAFFC0D6FC3AD6	3044220702990B5904FE4C1C3B4B23B20B2A61863B30D94BCBF6247523597A95187522206F1195EA86E13281C18FD687E68E3A5E4C3DC5EC71587F07653CA39C7344468	2536	191	81EF992CB35A9AE114C4E79BE81ACE85330996A8847FA4F5A9C48365C879E4B3E6512B9B3E7FA663A79E8518147B8EE4EF7DE4CD16C9A535958EB8DB2F7AC811	Valid	590
	Average:	**54.45**			**105.5**	**107.2**					**2478.55**	**179**			**589.75**