Tampere University

Nusrath Jahan Mouri

IOT PROTOCOLS AND SECURITY

Faculty of Computing and Electrical Engineering
Master of Science Thesis
July 2019

ABSTRACT

Nusrath Jahan Mouri: IoT Protocols and Security

Master of Science Thesis

Tampere University

Information Technology

July 2019

During the past years, there has been an exponential growth of internet connected devices all over the world. In future the growth of these devices is expected to grow at the higher rate. There are some studies estimating that Internet of Things (IoT) will be able to connects 500 billion devices by 2030. IoT smart devices are remotely accessible and are possible to control using existing network infrastructure.

At present, the usage of Internet of Things has increased rapidly. IoT is a dynamic global network between smart objects or things connected over the internet. IoT wireless network can connect anyone with anything at any place. With the rapid growth of IoT, security threats and vulnerabilities of the linked objects are also increasing continuously. Now, IoT security has become the most paramount technological research work over the world. The main objective of all IoT applications is to maintaining privacy and secure data transmission between devices. Due to the heterogeneous characteristics and constrained devices it is challenging to deploy security mechanisms in IoT compare to traditional network.

In this thesis, we highlight the importance of security in the IoT sector by studying a wide range of IoT security issues. Furthermore, we described several challenges derived from the existing IoT protocols and the security features of IoT protocols are also explained. In addition, implementation of UDP communication protocol and MQTT protocol using Contiki OS and Zolertia RE-Mote devices are added to the work. Cryptographic methods AES [1] and ECC [2] are described in the thesis and the implementation of AES-128 to secure device communication and ECC key generation process are also added to the thesis work.

Keywords: Internet of Things, IoT, IPv6, MQTT, CoAP, encryption, secure communication.

Preface

This M.Sc. thesis work was completed at the Department of Pervasive Computing, Tampere University and was supported by the Internet of Things research team, working for a project under the supervision of Antonis Michalas.

First of all, I would like to start by thanking my supervisors, Antonis Michalas for introducing me to the topic and for guiding me throughout the complete process and Elena-Simona Lohan for providing the guidance to complete the thesis work.

I would like to thank my husband Syed Golam Mahmud, without his belief and support it was not possible to complete the thesis work.

Finally, I would like to thank my family, especially my father who have always stood by me and his belief in me has been the source of motivation for me in my entire life.

Tampere, 2 July 2019

Nusrath Jahan Mouri

Contents

List of Figures

List of Tables

LIST OF SYMBOLS AND ABBREVIATIONS

IoT	Internet of Things
BLE	Bluetooth Low Energy
WSN	Wireless Sensor Network
IPv6	Internet Protocol version 6
IETF	Internet Engineering Task Force
6LoWPAN	IPv6 over Low power Wireless Personal Area Network
MQTT	MQ Telemetry Transport
CoAP	Constrained Application Protocol
REST	Representational State Transfer
DTLS	Datagram Transport Layer Security
HTTP	Hypertext Transfer Protocol
AES	Advanced Encryption Standard
ECC	Elliptic-Curve Cryptography
ESP	Encapsulating Security Payload
IPSec	Internet Protocol Security
ICMPv6	Internet Control Message Protocol for IPv6
6LBR	6LoWPAN Border Router

1. INTRODUCTION

1.1 Motivation

At present, communication technology is no longer limited to person-to-person communication but can also connect people with a wide range of devices all over the world. In near future, it is expected that everything and everyone in our society will be connected. Internet of Things (IoT) was introduced to fulfill this present technology demands. The term IoT was invented by Kevin Ashton in 1999 [3]. Even though the concept exists for almost 20 years there is still no unique definition about the field. Internet of Things allows people to connect with devices, sensors, services using a wired or wireless network from anywhere to achieve a specific goal. 'Thing' is used as a physical or virtual object that can communicate with other objects or users in IoT. Different organizations and group of researchers elucidated IoT definition in their own way. Stephan Haller et al. [4] defined IoT as "A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these 'smart objects' over the Internet, query their state and any information associated with them, considering security and privacy issues". According to Gartner [5], "The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense [6] or interact with their internal states or the external environment". IoT smart devices are remotely accessible and are possible to control using existing network infrastructure. To manage the efficient and secure connectivity between devices and objects over the internet, Internet Engineering Task Force (IETF) [7] designed a lightweight IPv6 network routing protocol for IoT [8]. According to Cisco [5], IoT will be able to connect 500 billion devices to the internet by 2030.

Along with the exponential increase in the number of IoT usage, security threats and effects on privacy has grown severely in communication. Secure data transmission and maintaining privacy and trust are the primary motive for all IoT applications. Due to the heterogeneous characteristics of the Internet of Things, privacy protection is a great challenge for any IoT applications [3]. IoT is mostly using constrained devices with low processing power, limited memory, battery supported compared to traditional internet. Therefore, it is difficult to extend security functionalities on these light IoT devices.

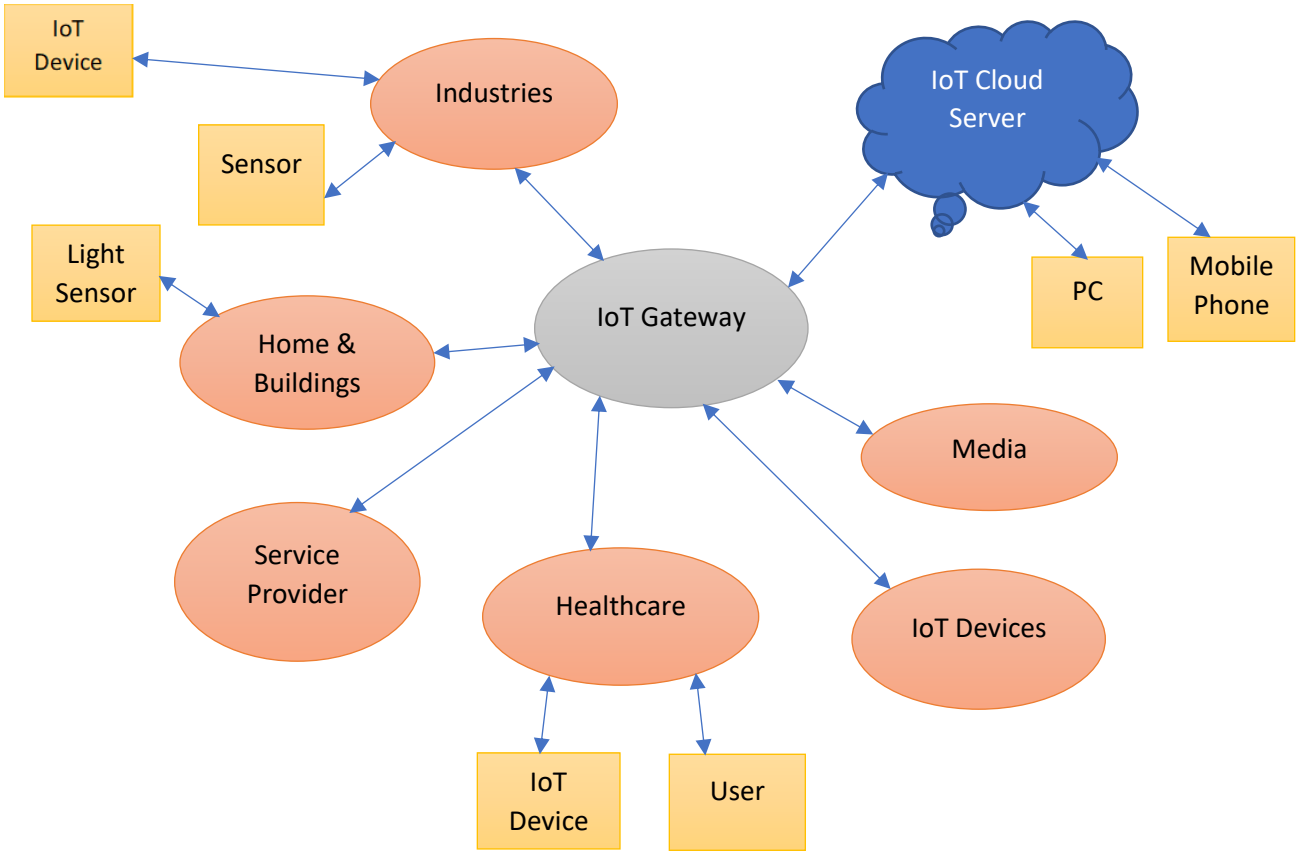

Figure 1: IoT Architecture [9]

1.2 Thesis Objectives

The purpose of this thesis is to describe the importance of IoT security, security protocols that are introduced for implementing in different IoT layers and the security challenges in IoT. Chapter 3 will cover all these mentioned contexts. In case of transferring sensitive data over the Internet using IoT, security and privacy are important topics to focus on by the organizations or businesses before implementing IoT network [10]. Since all IoT devices will be connected over the Internet, it is important to have security schemes that will protect the devices from malicious attack and to keep transmitted data immune. Applying security could be the biggest challenge for compatible speed and performance. Nowadays, enormous research work is going on to evolve challenges associated with the security and privacy issues in IoT. Finally, the implementation of some example programs in Zolertia Re-Mote hardware devices will be added to this thesis work to evaluate the performance of Zolertia IoT constrained devices at Chapter 4.

1.3 Author's Contribution

In this thesis, the Author's contribution are as follows:

- Presenting the importance of IoT security and the challenges which make it difficult to implement the security scheme in the IoT devices as well as in the IoT networks.
- Describing the IoT protocols and the security features of these protocols that are now popular to the developers.
- Implementing an experimental study based on Zolertia RE-Mote devices (see Chapter 4).
- Implementation of UDP (User Datagram Protocol) and MQTT (Message Queuing Telemetry Transport) protocols and secure communication between devices using AES-128 encryption method.
- Drawing conclusions on the theoretical and experimental studies and discussing the possible future directions in the addressed research field.

1.4 Limitations

This thesis is focused on the theoretical description of IoT protocols and the security features of all those protocols that have been proposed as mean to protect IoT from several security and privacy threats. In addition, some security protocols and cryptographic algorithms are described in Chapter 3. Apart from the theoretical work, we also conducted some experiments that help us to test the applicability of popular cryptographic functions on IoT devices with constraint resources. Our testbed conformed from a number of interconnected Re-Mote Zolertia devices running on Contiki OS [14] - an operating system that has been built to fit squarely the specific needs of IoT. Implementation of MQTT communication protocol will be tested using Contiki OS, but other protocols like COAP, DTLS, and other operating systems will not be covered. Zolertia Re-Mote devices are used as hardware platforms for the testing and implementation part in Chapter 4, because of the availability and for some attractive features like the communication range between 100 meters and 20 km, ARM Cortex-M3 32 MHz clock speed. Contiki OS supports the software that is developed for Zolertia hardware. A digital Grove light sensor is used to obtain real data for the testing purpose. A brief description of Zolertia hardware and the utilized sensors will be presented in Chapter 4.

1.5 Organization

Chapter 2 presents the background knowledge about the technologies and protocols that are being used for both hardware and software implementation in this thesis writing.

Chapter 3 describes a list of security issues and challenges for IoT. Furthermore, some security protocols that are designed for different IoT layers are described. More precisely, a brief description of MQTT [11], DTLS [12], COAP [13] protocols are provided to analyze security issues for constrained Node Networks.

Chapter 4 illustrates the concepts of hardware devices and tools that are used for the experiments to get the desired results. All the test procedures are included in this chapter.

Finally, in **Chapter 5** we conclude this study by highlighting the main points as well as some possible future directions.

2. IoT OVERVIEW

In this chapter, we present the main technologies that will be used in this thesis work. A variety of protocols and communication technologies are used for the IoT. A brief description of IoT technologies, protocols and operating system are presented below for making it easier to understand the use of each technology and protocols. Further details will be added to other chapters based on the thesis work.

2.1 IEEE 802.15.4

IEEE 802.15.4 [14] is a broadly used mesh network communication standard, the first version was released in 2003. This standard is intended to use in low-data-rate and low-cost communications between inexpensive devices to achieve the lowest power consumption. At present, various network layer protocols such as 6LoWPAN, ZigBee and IEEE 802.15.5 run over IEEE 802.15.4 based networks.

This standard provides physical layer, medium access control sublayer (MAC) of the OSI model specifications for low-data-rate wireless connectivity with portable or fixed devices with limited battery consumption requirements. As a MAC mechanism this standard uses a CSMA / CA protocol. These characteristics make IEEE 802.15.4 contrasted from Wi-Fi technology, which requires more power and offers large bandwidth. It describes the smaller range of wireless communication compared to Wi-Fi. It supports 10-meter communication range with a transfer rate of 250 Kbit/s.

IEEE 802.15.4 standard supports network topologies such as star and peer-to-peer and two types of network nodes: Full Function Device (FFD) and Reduced Function Device (RFD). FFD can operate all operations and can communicate to all other devices for sending and receiving data in the network. RFD is simpler and can only communicate with FFD nodes in the network. It uses 64-bit or short 16-bit IEEE addressing modes for each node. Devices that support IEEE 802.15.4 can use one of 868/915/2450 MHz frequency bands for operations.

2.2 Bluetooth Low Energy

Bluetooth Low Energy (BLE) [15] is a wireless communication technology designed for short-range communication, developed by the Bluetooth Special Interest Group (SIG). BLE succeed in the market as Bluetooth 4.0 by 2011. It is quite different than classic Bluetooth. The primary difference is in power consumption. BLE is used only for the applications that are applicable to transferring the short amount of data and therefore it can run over a small battery for several years. But it is not ideal for phone calls [15].

The key features of BLE are low cost, low power consumption, enhanced range, multi-vendor interoperability. Classic Bluetooth and BLE both operate in the same spectrum range 2.4 GHz ISM band, but they use different channels. Bluetooth Low Energy has the data transfer rate up to 1 Mbit/s and the possible range is over 100 m. It supports full AES-128 encryption to provide encryption and authentication of data packets. BLE technology is now being used by numerous numbers of applications like security, low power sensor devices, entertainment, healthcare [16] [17] [18] [19], automotive industries to provide a wireless connection between manufacturers and consumers [20].

2.3 WSN

Wireless Sensor Network (WSN) [21] can be generally defined as a network that may control and sense surrounded environment as well as establishes device-gateway communication where devices are acting as network nodes and border routers are performing as gateway. In WSN, data is forwarded to multiple nodes and data forwarded to other networks using a gateway. Wireless Sensor Network nodes act as sensor nodes, actuator nodes, clients and gateways. Sensor nodes can sense some data like temperature, pressure, humidity, power of light from the environment and forward it to the sink node that can process data to produce some control commands. The Border router is used between a WSN mesh network and the Internet. Nowadays WSN is mostly used for IoT applications and systems for its low power needs.

The wireless sensor network nodes are organized according to the topologies illustrated in Figure 2. Sensor nodes publish data to other nodes and receive status from other nodes to detect each other. Data forwarding process between nodes differs for different topologies. WSN devices

are battery dependent and spending low energy. Due to these energy-saving criteria, there is the possibility of security threats [22].

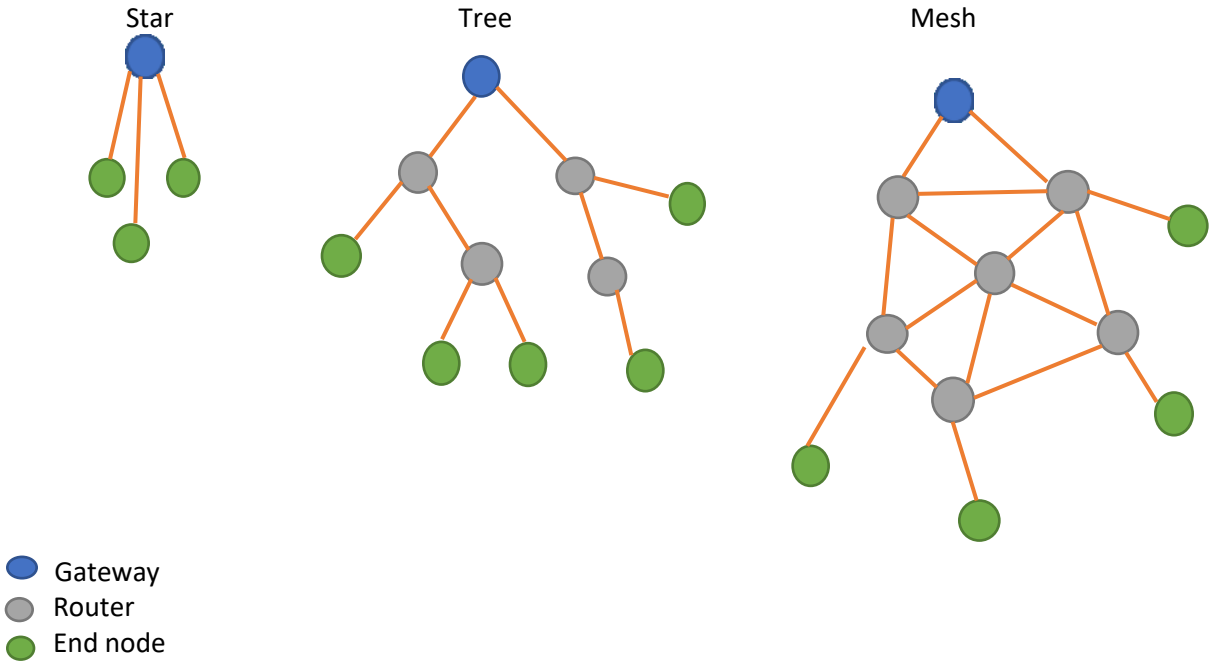

Figure 2: Wireless Sensor Network Topologies [23]

Operating systems that are used for WSN are: Tiny OS, Contiki, MANTIS, BTunt [22]. Mostly common communication standards for WSN are 6LoWPAN, ZigBee and Bluetooth [22]. Some applications for wireless sensor network are the security system, healthcare, military applications, industrial applications, environmental monitoring, and smart buildings [23].

2.4 Internet Protocol version 6 (IPv6)

Internet Protocol version 6 (IPv6) [24] is the newest version of Internet Protocol standardized by the Internet Engineering Task Force (IETF) to conquer old version IPv4 to get over new technical challenges. IPv6 uses 128 bits for each IPv6 address which means it has 2^{128} options for the addresses while IPv4 uses 32 bits only for each address. This extension of IP address length brings some benefits like the immense number of addresses, easier address management, end-to-end IPsec possibility, and address auto-configuration methods. IPv6 is not only differing on a large number of addresses than IPv4 but also offers divers improvements over IPv4 [25]:

- Simplifies efficient routing process
- Allocation of the hierarchical address

- Mandatory IPsec network layer security
- Assure Quality of Service (QoS) in terms of performance
- IPv6 provides Stateless Address Autoconfiguration (SLAAC)
- Extension of the header without including any additional fields.

2.5 6LoWPAN

6LoWPAN [24] is the acronym for IPv6 over Low power Wireless Personal Area Networks has been developed by IETF. It is a protocol used to transmit IP packets in networks based on IEEE 802.15.4 standard. This 6LoWPAN concept comes with the idea of using low power devices with limited processing (CPU, Memory) capabilities as the network nodes for the Internet of Things (IoT). 6LoWPAN layers are presented in Figure 3.

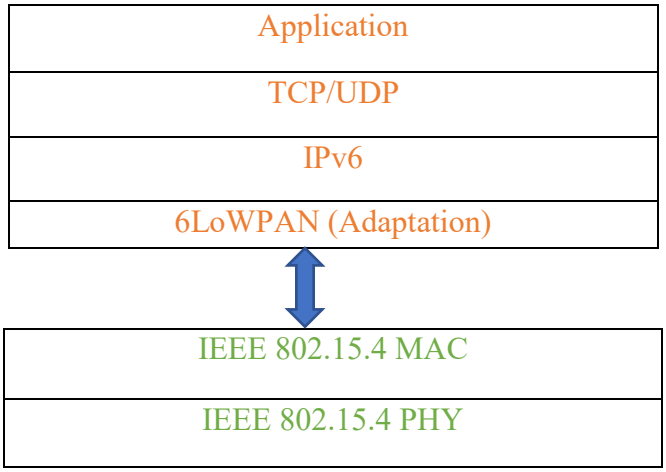

Figure 3: 6LoWPAN Layers

The main goal of 6LoWPAN is to act as an adaptation layer between the standard IEEE 802.15.4 and IPv6 [24]. It provides the encapsulation and header compression methods to solve the different packet size problems between IEEE 802.15.4 and IPv6 layers [24]. Stateless header compression is used to reduce the large IPv6 and UDP header to some bytes [24]. Communication range for 6LowPAN is 45 to 90 meters [22]. Some functions of the 6LoWPAN protocol are device and service discovery, security, packet size adaptation, address resolution and optimization [26]. Star and Mesh topologies are included in 6LoWPAN protocol [24]. Due to uncertain radio connectivity, battery drain, and physical difficulty devices supported by 6LoWPAN tends to be unreliable [24].

2.6 Contiki

Contiki is an open source operating system for the Internet of Things (IoT) and Wireless Sensor Networks (WSN) [24]. It connects small microcontrollers with low cost and low power properties to the Internet [24]. It was developed by a group of developers from the Swedish Institute of Computer Science [27]. Contiki Applications are written in C programming language [27]. Contiki supports both IPv4 and IPv6 along with 6LoWPAN, RPL and COAP low-power wireless standards [24].

Instant Contiki consists of a single file download for users that have all tools and compilers included to ease software development for Contiki [27]. Some key features of Contiki OS are [27]:

- Multi-tasking Operating System
- Power and Memory efficient
- Highly portable Operating System
- Possible to load and unload application programs in run time using event-driven kernel

2.7 TCP

Transmission Control Protocol (TCP) is designed as a network communication protocol to transmit packet data between nodes over the Internet [28]. TCP is a transport layer protocol in the Open Systems Interconnection (OSI) model. It was mainly designed for the wired network [29]. It is a connection-oriented protocol. Before data transmission between the source node and destination nodes, TCP establishes a connection between nodes. This connection establishment is renowned as handshaking, which needs three packets transmission before sending any user data packets.

TCP divides a large amount of data packet to smaller packets for transmission. Header size for TCP is 20 bytes, included in each packet. It sends all data packets sequentially. It provides acknowledgment (ACKs) for packets received by the destination node. If the source node does not receive the correct ACK from the receiver node then It resends the data packet [30]. Thus, TCP ensures data reliability and integrity. These make TCP communication process a bit slower. This protocol is responsible for error checking and error recovery. HTTP, HTTPs, FTP, SMTP protocols could be used as TCP traffic source [31]. Applications that need more reliability and

needs to receive data in a sequence like email, web browsing, files downloading uses TCP protocol.

2.8 UDP

User Datagram Protocol (UDP) is similarly a communication protocol that is used to transmit data between nodes. It is a connectionless protocol. UDP does not need to establish any connection between sender and receiver before user data transmission. When an application using UDP protocol, it just sends all data packets to the destination directly. If the receiver does not receive data accurately or some part of the data is missing, UDP does not resending the data that is not reached correctly [30]. Thus, UDP is faster than TCP protocol but not ensuring data reliability and integrity.

UDP is responsible for error checking but does not attempt to recover error, just discard the packet. It is not receiving any acknowledgment from the destination node. UDP header size is 8 bytes [14]. Data packets are not sent in any sequence. Each packet sent independently, and integrity is checked in the destination if the packets arrive [31]. DNS, DHCP, NFS, RIP protocols could be used as UDP traffic source [31]. Applications that do not need any error correction and needs faster data transmission like the online game, VoIP, video or audio streaming are suitable for UDP protocol.

2.9 MQTT

MQ Telemetry Transport (MQTT) is a messaging protocol designed in 1999 and works on top of TCP/IP protocol [24]. Two types of messages “Publish and Subscribe” are used on an MQTT connection [32]. It is designed to minimize network bandwidth and to work with low resources consumption [24]. An MQTT broker is used to publish data on a specific topic and for receiving data on a topic.

MQTT Architecture is designed in Figure 4. A client node is publishing data with a topic name to the broker and the clients also can subscribe data from the broker with a specific topic name. The broker is ready to reply for data sending to the clients based on their requirements. MQTT

provides three different levels of Quality of Service (QoS) that defines how the messages are delivered between nodes [24]:

- **QoS 0:** broker or client delivers the message once without any confirmation.
- **QoS 1:** broker or client delivers the message with confirmation at least once.
- **QoS 2:** broker or client delivers the message with four-step handshaking exactly once.

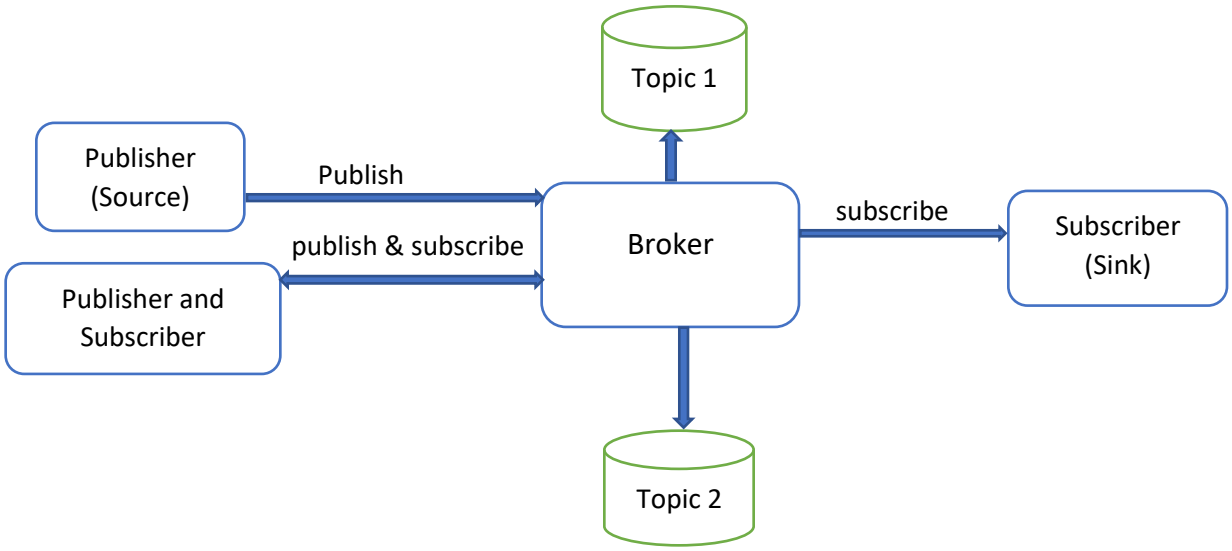

Figure 4: MQTT Architecture

MQTT minimizes protocol overheads and provides high-efficiency communication for IoT [32]. Furthermore, it reduces bandwidth consumption while at the same time increases scalability. It has keep-alive message function, if any clients disconnected broker can detect that node [24]. Broker keeps published data of a topic, therefore, if any new subscriber connected with the same topic can subscribe all data based on this topic from MQTT broker [24]. MQTT is mostly suitable for IoT because of remote sensing and controlling capability. It is possible to provide security by adding a username and password to the data packet in the protocol.

3. IoT SECURITY AND CHALLENGES

3.1 Security Issues in IoT

IoT can be defined as the internet of objects. IoT objects interact with the other components like mobile, data collectors, laptops, smart devices for data sharing, collecting and management in the context of service provided. With the rapid growth of IoT, IoT devices are responsible for transmitting and collecting a large number of data between devices. Due to the vast number of connected devices, there is a high risk of data stealing, device, and network manipulation. With the increasing number of IoT devices, security concerns are becoming the major issues all over the world. A research study on Internet of Things [33] by Hewlett-Packard Development Company, 2014 found that 80% of devices failed to require strong passwords, 60% did not use encryption during downloading software updates, 70% failed to encrypt data transmission via local network and the Internet and 80% of devices collecting personal information which could be harmful if it is transmitted by any devices without encryption.

IoT is the biggest revolution in the IT industry [34]. It is easier to conduct attacks on IoT devices. Some dynamic threats and challenges for IoT devices are listed below [34, 35, 36]:

- Due to the lack of security mechanisms and impotent encryption process transmitted data can be attacked or misused by the unauthorized interference during the data transmission across the network. Maintaining data integrity and confidentiality are must be needed during data transmission.
- Personal information or messages from the sender to the receiver can be affected by the attacker and wrong information can be sent [37].
- IoT devices are resource constrained means they have limited computational and storage capabilities. It is not possible for the developers to adding required features to defend against the security threats.
- Life becomes easier with the use of IoT that provides several facilities to everyday lives. But it influences the users' privacy [38] [39] [40]. If the IoT device compromised the collected data, then it will be difficult for the user to keep trust in IoT.
- Some security vulnerabilities in network services like Exploitable UDP Services, DoS via Network Device Fuzzing, Buffer Overflow, Denial-of-Service [41], [42], [43]might allow an attacker to acquire unauthorized access to the IoT device.

- Insecure cloud interface [44], [45], [46], [47], [48] and insecure mobile interface due to the poor authentication and unencrypted data transmission might allow unauthorized access to the device and the collected data.
- If the web interface that is used to the user and device interaction is not secure, then it is easier for the attackers to gain access to the IoT device.
- In some IoT applications, device life is shorter, and some applications have longer device life. Security techniques should be designed based on device lifetime and application. If the device life is longer than it should have the option to update security features to defend from new attacks and security threats.
- Open source software is useful for quick development in IoT systems. To protect from malicious attacks, developers need to be careful to choose the software and for open source software validity must have a framework. Most of the IoT edge devices use same open source software which may have the same vulnerabilities. It increases the possibility of full system collapse if an attacker uses the same vulnerabilities to all edge devices and attacks all network devices.

To protect devices from the above-mentioned security threats it is important not only to provide reliable but also realistic (i.e. efficient) solutions. As the IoT devices are resource constrained, it is not possible to directly employ the traditional security mechanisms like the other Internet devices. There are some limitations to apply traditional security schemes directly in the IoT devices based on hardware, software, and network which are discussed below [49]:

- Most of the IoT devices are battery dependent. Thus, using low power CPU that has the lowest computation power is considered as of paramount importance. However, using such a processor makes it difficult to execute computationally heavy cryptographic algorithms.
- IoT devices have limited memory than other traditional devices and use a lightweight version of Operating System. Therefore, traditional security algorithms cannot be used in these devices with small memory as the process might not get enough space after booting up the operating system and some software. However, Traditional security algorithms are not designed with memory efficiency consideration.
- Sometimes IoT devices are needed to place in some remote area and not possible to monitoring continuously. If a single IoT device from a network is attacked by an unauthorized person, an attacker might change the programs of the device or may replace the device with another malicious node.

- Installing a dynamic security scheme in IoT device is not possible as the operating system and protocol stack might not be able to receive and integrate new code and library.
- IoT devices might use IP networking to communicate with IoT service provider and proprietary network protocol for network communication simultaneously. For these characteristics of IoT devices, traditional security methods become unsuitable for the devices.
- An IoT device can join or leave the network at any time and this adding or removing device characteristics make a network topology dynamic. Security models that exist for the digital systems do not work duly with these topological changes.
- IoT devices might be connected via wired or wireless link to the local and global network. It is difficult to find any security protocol that can work for both wired and wireless connectivity.

3.2 Protocols for IoT Layers and Security

A Protocol is the set of rules that are used by the end node of a network connection to communicate with the other end nodes of the same network or the different network. In this subsection 3.2, IoT layers are mentioned in Figure 5 and the most common security protocols listed in Table 1 that are used for Machine-to-Machine (M2M) communication in IoT different layers will be described briefly.

Perception Layer is responsible for data collection. It is the lowest layer of the IoT. This layer uses sensors and actuators to perform different measurements like temperature, humidity, and pressure. IoT nodes perform the required function for data collection and RFID, ZigBee, some other sensor nodes are involved in this layer [50]. It is important to secure this layer to protect the data that could be damaged by a malicious attack. Some possible security threats are DoS attack, routing attack and hardware damage. Attackers can destroy a sensor node by replacing the node or by electronically interrogating the nodes to get the access and to change the important information. Physical security like abnormal sensor reading detection and cryptographic elements implementation has to be provided at the perception layer [50].

Network Layer in IoT is the same as the TCP/IP network layer. Data transmission is the main task of this layer. Similar to the traditional TCP/IP network layer this layer has several security

issues as well. Some of those common security threats are unauthorized access to the networks, lack of confidentiality and data integrity, DoS attack, storage attack, eavesdropping information, gateway attack and so on [50]. To protect the network layer from such attack key management, precise authentication, unauthorized access detection, and negotiation should be implemented.

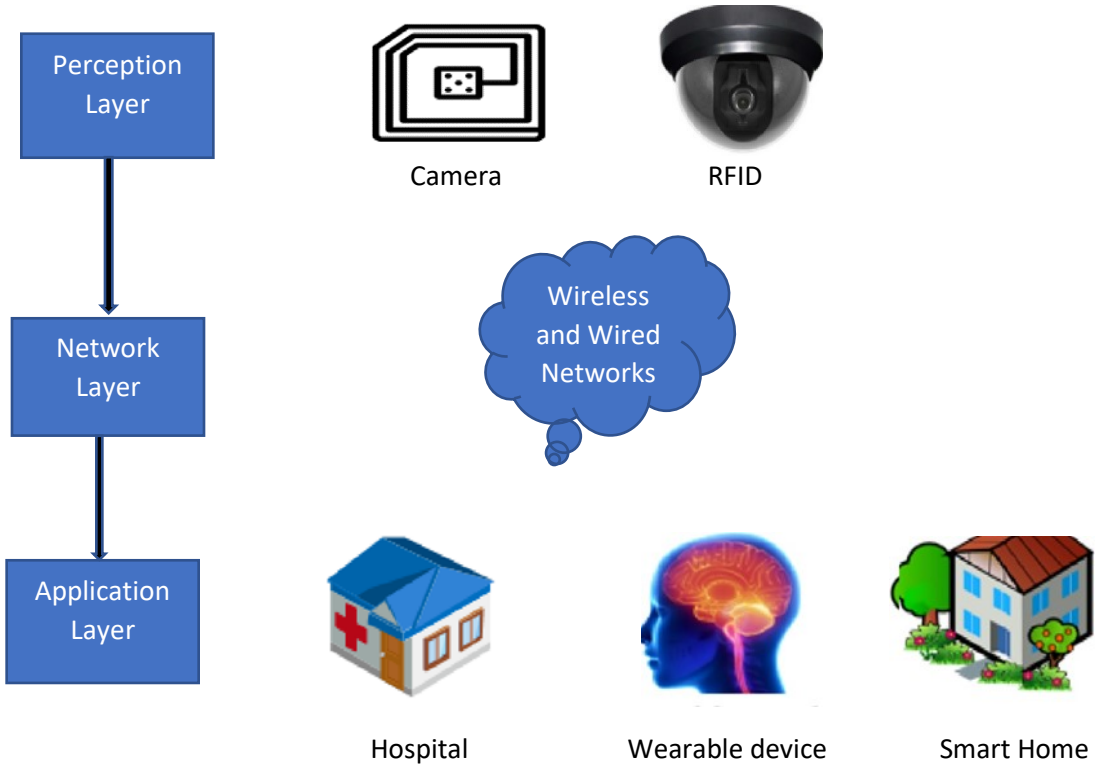

Figure 5: IoT Layered Architecture [51]

Application Layer provides the services that are requested by the user. There is no universal standard for the IoT application layer construction as because of the availability of many different devices and manufacturers [50]. This layer can provide high-quality services to meet user demands. A large number of devices and nodes makes it challenging for the layer to secure data efficiently because of the massive amount of data and information transformation. Availability of so many different applications and the user makes it difficult to manage authentication and authorization. There are many security threats to the application layer. Attackers can steal user data by known application vulnerabilities. If the coders are not more conscious about their codes in the application, attackers can upload malicious codes which can lead to the software attack. Attackers can destroy the complete application or service.

IoT Layer	IoT Protocol	Security Protocol
Application	MQTT, COAP	User defined
Transport	UDP	DTLS
Network	IPv6, RPL	IPsec, RPL security
Adoptation	6LoWPAN	---
Perception	IEEE 802.15.4	802.15.4 security

Table 1: IoT layers and protocols [52]

3.2.1 IEEE 802.15.4 Protocol and Security

IoT PHY and MAC both layers are supported by the IEEE 802.15.4 protocol [10] which was established in 2003. It was designed as a principle for a protocol stack towards short range and energy efficient communication. The frequency band 2.42 GHz is mostly used for this protocol. It supports short-range communication at 250 Kbit/s. IEEE 802.15.4 security is implemented in the MAC layer, not designed for the PHY layer. IEEE 802.15.4 PHY layer is responsible to manage signal, energy and channel selection [53]. IEEE 802.15.4 MAC layer provides other services like physical channel access, frames validity, time slots, and security and node organizations [53]. Carrier Sense Multiple Access (CSMA/CA) method is used to avoid collisions during data transmissions. MAC layer provides multiple security levels.

IEEE 802.15.4 protocol has a stable mechanism and it provides various security services [10]. The security depends on symmetric cryptography and Advanced Encryption Standard (AES). This provides data authentication, data encryption, and confidentiality. Access control is another security service that can activate the nodes to find more security information for processing the security of the message [10]. Security-enabled flag inside the control field of the frame used to secure MAC frames during the communication [54]. Auxiliary Security Header is responsible to have information about the frame security and security enabled control flag is responsible for the signals of the presence of this header. A new feature known as *Time Synchronized Channel Hopping* was introduced in 2008 which provides services to protect from replay attacks. It requires

acknowledgment-based or frame-based synchronization between end devices. Receiver node calculates the difference between the estimated arrival time and the exact arrival time of a frame and then send it to the sender with the acknowledgment to synchronize the time clock according to receiver's clock [53]. The usage of Absolute Slot Number (ASN) that stores the total number of timeslots as a global frame counter value can be used to protect from the replay attack and semantic attack [53]. But the IEEE 802.15.4 specification does not define how to do key management. Reem et al [10] state that, "the IEEE 802.15.4 protocol security has a limitation that it is not using key management model for the cases in which the end nodes might reuse the nonce value". The AES-128 block cipher uses 128-bit symmetric keys, but the upper layers are responsible to do the key generation, distribution, and replacement [54].

3.2.2 6LoWPAN Protocol and Security

The IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) is a network protocol standardized by IETF to enable direct connection to the Internet for constrained resource devices [54]. The widely supported standard IEEE 802.15.4 for low-power wireless embedded communications released in 2003 was the reason for 6LoWPAN standardization. The first 6LoWPAN specification was introduced in 2007. 6LoWPAN defines a set of rules that are generally used to integrate all sensor nodes into IPv6 networks. It supports star and mesh topologies and the combination of both star and mesh topology. As it supports low-power, low cost, battery supplied, IP-driven devices and large mesh network, it is becoming a suitable option for the Internet of Things (IoT) applications. It defines the frame format and header compression mechanisms for IPv6 packets during the transmission over IEEE 802.15.4 networks [54]. Different specifications have been developed to support IPv6 over Bluetooth Low Energy (BLE) and some other networks.

This 6LoWPAN protocol has the freedom of physical layer and frequency band [10]. It is possible for the constrained devices to establish network connectivity with any other IPv6 enabled device using the 6LoWPAN protocol. The adaptation layer can be classified in various RFC (Request for Comments) documents. RFC 4919 is responsible to discuss the goals and assumptions used to perform the works by the group of IETF 6LoWPAN [53]. RFC 6282 presents how the UDP header compression could be done following the context of the 6LoWPAN adaptation layer [53]. RFC 4944 defines the mechanisms that are used to transmit IPv6 packets over IEEE 802.15.4 networks with header compressions [53]. RFC 6606 represents the

6LoWPAN routing requirements. 6LoWPAN adaptation layer does not contain any special security mechanisms. It relies on the security mechanisms that are available for the lower layer (Section 3.2.1-IEEE 802.15.4 security) or upper layer (Section 3.2.4-COAP) for securing communications [41]. But the adaptation layer RFC documents include some of the security vulnerabilities and requirements to consider the importance of network-layer security [40]. Identification was one of the problems that represent RFC 4944 is involved with the possibility of extracting EUI-64 interface that can compromise the 6LoWPAN unique interface identifier [40]. RFC 4944 document is also involved with the discussion of mesh routing mechanisms and neighbor discovery on IEEE 802.15.4 which might be a threat to security. MAC layer AES security may provide some fundamentals for developing the mechanisms to protect from such threats [40]. The usage of compression UDP was another major issue which reduces an UDP port numbers (16) to 4 bits [7]. Hence, it creates the possibility of receiving the wrong message or incorrect payload by an application [7]. Difference between 6LoWPAN and IP protocol stack is presented in Figure 6.

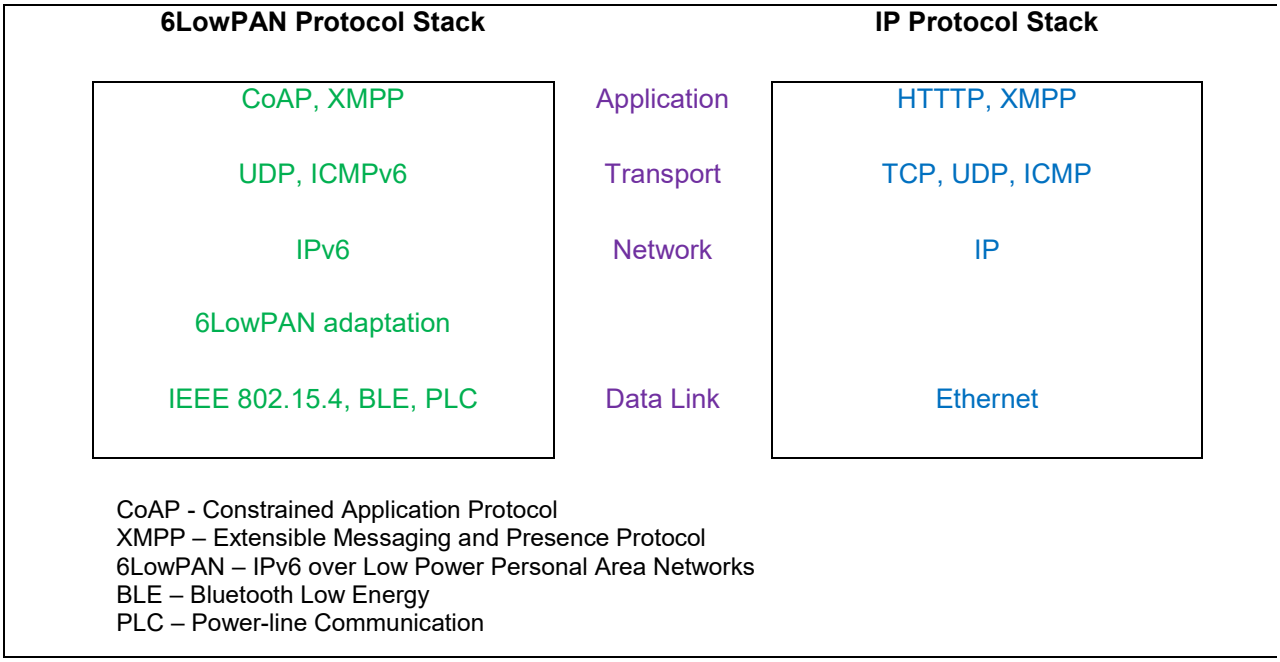

Figure 6: 6LowPAN vs. Traditional IP Protocol Stack [55]

Jorge Granjal et. al [53] has mentioned some possible solutions for the protection from the security threats. The use of Internet Protocol Security (IPsec) for end-to-end IoT provides authentication and encryption at the network layer. Design of compressed security header for the

Authentication Header (AH) and Encapsulating Security Payload (ESP) for 6LoWPAN in transport and tunnel modes was another proposed solution [10]. Security implementation against packet fragmentation attacks was one more approach. The fragmentation attacks may cause the buffer overflow or the misuse of available computational capability. The proposed solution is to add a new field namely timestamp for the protection against unidirectional replays and a nonce for the protection against bidirectional replays to the fragmentation header of 6LoWPAN to solve these issues [53]. Key Management is one of the important security functionalities to ensure long-term and effective security for applications. One possible solution for key management is to reduce the payload information and IKE headers using the 6LoWPAN protocol [10].

3.2.3 RPL Protocol and Security

RPL is a lightweight IPv6 distance vector routing protocol for Low power and Lossy Networks (LLNs) was standardized in 2011 by the IETF group with a goal of providing routing solutions for IoT. It supports Point-to-point, Multipoint-to-point, and Point-to-multipoint traffic patterns [24]. Different functions can be used based on scenarios for the path selection in the RPL routing protocol to make the network more efficient and for flexible structure [56]. RPL builds a Destination Oriented Directed Acyclic Graph (DODAG) for each root device which is identified by a DODAGID [53]. RPL can response on a sudden topology change of the network and is able to provide optimal path selection based on functions [56]. The RPL routing protocol supports several types of control messages and all these messages are encapsulated in ICMPv6 packets [53]. In table 2 functions of these control messages are listed.

RPL Control Messages	Functions
DODAG Information Object (DIO)	Carries information to join a DODAG which exists and select list of parents.
DODAG Information Solicitation (DIS)	Request for a DIO message from a RPL node.
Destination Advertisement Object (DAO)	Used to propagate destination and routing information from leaf nodes to the root.

DAO Acknowledgment (DAO-ACK)	Sent by the parent as an acknowledgment on the reception of DAO.
Consistency Check (CC)	Used to synchronize counter values among communicating nodes.

Table 2: RPL control messages with functions

The current RPL specifications include SHA-256 to use digital signatures and AES/CCM including128-bit keys for MAC to support authenticity and integrity [53]. RPL message is entirely maintaining RPL security. In an RPL ICMPv6 message, IPv6 header and the ICMPv6 header is not encrypted but this fields required for the correct packet decryption. A Consistency Check RPL control message can help a sensor device to establish a challenge response with a motive to authenticating the counter value of another node [10]. For example, if any receiver is maintaining incoming counter value for message originator while a message is received with the initialized counter value, then the receiver can start resynchronization with the sender by sending a consistency check message to the source [53]. This counter fields technique provides security against replay attacks.

Self-configuration	The path of networks are dynamically discovered using IPv6 neighbor discovery mechanisms.
Routing Type	Source routing and Distance-vector routing.
Target Networks	6LowPAN network, Low Power and Lossy networks and other IPv6 networks.
Identifiers	RPL instance ID, DODAG ID, DODAG version and Rank.
Traffic Flows	P2P, MP2P and P2MP.
Data Transmission	Unicast and Multicast.

Table 3: RPL Features [55]

Table 3 represents some key features of RPL protocol. RPL also has three different security modes which can be applied to routing control messages [53]:

- Unsecured: RPL uses this mode as default. It does not apply any additional security to the routing control messages.
- Preinstalled: The security is applied by a node to join the RPL instance as a router or host device using a preconfigured symmetric key. This symmetric key is used to provide data authentication, confidentiality, and data integrity for RPL control messages.
- Authenticated: RPL applies this security mode only to the routers. When a device joining the network with a preconfigured symmetric key means using preinstalled mode since it can obtain a cryptographic key from the authority to act as a router. The authentication and authorization of the devices are assured by the key authority.

The RPL specifications recently define that it is not mandatory to use symmetric cryptography to support the authenticated mode, but how the asymmetric cryptography can be used to authenticity and to retrieve the key by the node which is ready to act as the router is not specified. The future RPL versions may clarify this issue. Lack of security mechanisms and internal attacks are the limitations of RPL security. Adding version number, authentication based on signatures and one-way hash chain can be used to protect against the internal attack [10].

3.2.4 CoAP Protocol and Security

Constrained Application Protocol (CoAP) [57] is a web transfer protocol, developed for constrained networks and constrained nodes in IoT [58]. IETF designed CoAP using a subset of the HTTP methods to target constrained devices [59]. It uses REST (Representational State Transfer) architecture where resources are identified by Universal Resource Identifier (URI) [60]. REST is commonly used with HTTP but runs over UDP for the transactions [10]. The main goal for developing CoAP protocol is to match the special requirements of IoT especially machine-to-machine (M2M) applications and considering energy [57]. Some key features of CoAP protocol are [57] [58]:

- Constrained web protocol supports M2M requirements.
- Asynchronous message exchanges.
- UDP protocol binding to avoid TCP handshakes

- Supports content-type and Universal Resource Identifier.
- UDP supports uni-cast and multi-cast requests.
- Security binding to Datagram Transport Layer Security (DTLS).
- Simple proxy and caching capabilities.
- Parsing complexity and Low header overhead.
- HTTP mapping for integration with existing networks and Web technologies.
- Confirmable and Acknowledgement messages can be used to provide reliable communications.

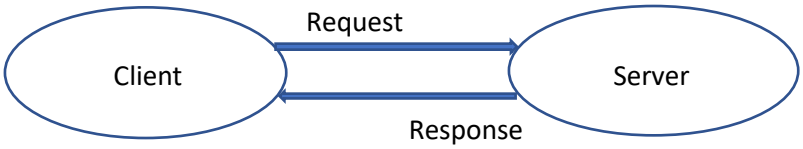

Figure 7: Client-server model in CoAP

CoAP architecture can be divided into two different layers: response/request layer and message layer [10]. The message layer is responsible for exchanging messages between end nodes over UDP while the second layer is used to response on transferred messages to avoid arrival delay, packet lost. CoAP is a reliable protocol with some valuable features like multicast requests, retransmissions and duplicate detection [10]. The communication between CoAP server and client is peer-to-peer [57]. CoAP uses binary format for message encoding and a fixed-length binary header. Each message contains message ID for reliability and to detect duplicate of the message [57]. To provide reliability sender sends message as Confirmable (CON) with message ID and recipient sends back Acknowledgement (ACK) with the same message ID [57]. If the recipient fails to process a Confirmable message, it sends back Reset (RST) message instead of ACK [57]. Four different methods are defined in CoAP based on RESTful architecture [60] [10]:

- **GET**: is used for retrieving resource representation identified by URI request.
- **POST**: is used for transferring information representation.
- **PUT**: is used for updating resources on server.
- **DELETE**: is used for deleting identified resources by the URI request.

CoAP protocol requires additional protocol to provide security as like the traditional protocols [50]. Datagram Transport Layer Security (DTLS) and IPsec is generally used for the message encryption. DTLS protocol is applicable to the transport layer. The basic encryption algorithms

AES/CCM provides authentication, integrity and confidentiality. DTLS adds 13 bytes/ datagrams overhead after the initial handshake process completed [51]. 6LowPAN header employs10 bytes of IPv6 header while 4 bytes are required by the CoAP header therefore, the DTLS compression is needed [50]. DTLS also can provide security for replay attacks using a nonce value to each CoAP packet and each nonce value should be different from another [40]. Figure 8 represents the CoAP Architecture for constrained node networks.

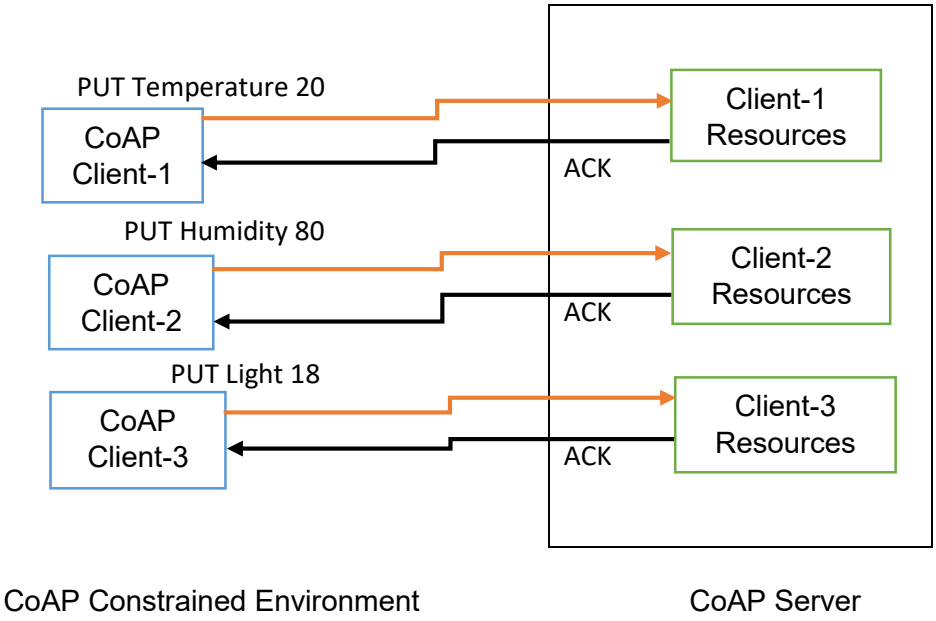

Figure 8: CoAP architecture [61]

In addition to DTLS security protocol, CoAP defines four different security modes that can be used in the applications are [49] [7]:

- **NoSec:** This mode does not provide any security and CoAP messages are not including any security elements before transmission.
- **PreSharedKey:** This mode is used by the devices that are pre-cofigured with symmetric cryptographic keys to confirm secure communication. Applications can use one key for group of devices or one key for one device. This is suitable for the devices that does not supports public key cryptography.

- **RawPublicKey:** It requires public key for authentication. Devices using this method are programmed with the list of asymmetric keys so the devices does not need any certificate to start a DTLS session.
- **Certificates:** This mode also supports authentication with public key and the applications that are participating to certificate chain for checking the validity of certificates. Devices that has asymmetric key pair and an X.509 certificate that can be validated using certificates mode and root keys.

ECC (Ecliptic Curve Cryptography) is public key cryptography which supports both RawPublicKey and Certificates modes [51]. ECC using the Elliptic Curve Digital Signature Algorithm (ECDSA) to support device authentication and for key agreement using the Elliptic Curve Diffie-Hellman Algorithm with Ephemeral keys (ECDHE) [40]. The devices using NoSec security mode sending packets without any security elements, using “coap” method in URI addresses [51]. In addition, different modes need to support mandatory-to-implement cipher suite for each mode based on AES/CCM and ECC. Applications with PreSharedKey mode supports TLS_PSK_WITH_AES_128_CCM_8 suite for the authentication using pre shared key and 8-byte nonce value [53]. The RawPublicKey security mode required TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 suite using ECDSA-capable public keys [51]. This mode uses SHA-256 to compute hash function. Applications using Certificates mode also required TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 security suite and supports public key transport in X.509 certificate [40].

CoAP security is still under scrutiny because of the controversy and challenges to implement security elements. Maintaining the high performance as well as managing security and protection together is the biggest challenge. To provide end-to-end security in CoAP protocol DTLS can be implemented as the official application layer security protocol but DTLS also has some limitations [10]. Since DTLS was not designed for constrained devices [62], it is heavyweight protocol with long headers. The long header of DTLS is compressed by using 6LowPAN as well as the large message and handshake compression is also needed [62]. DTLS does not support the CoAP proxy modes [10]. In addition, the author [63] indicates that in some end-to-end communication process a HTTP client needs the CoAP server in the backend to access resources. A mapping is required between CoAP and HTTP in the application layer and 6LowPAN Border Router (6lbr) cab be used as proxy [63]. To ensure if there is a malicious code the proxy must translate the packet without scanning [10]. Here the challenge is multi-cast messages in group communications are not supported by DTLS. Hence it is possible to translate the proxy from HTTP to CoAP and

has to decide the message as uni-cast or multi-cast [10]. As like all other protocols robust key management could be the solution for CoAP [10]. Key Management can help to ensure security standards that are applied and to solve the multi-cast message issue in DTLS [10].

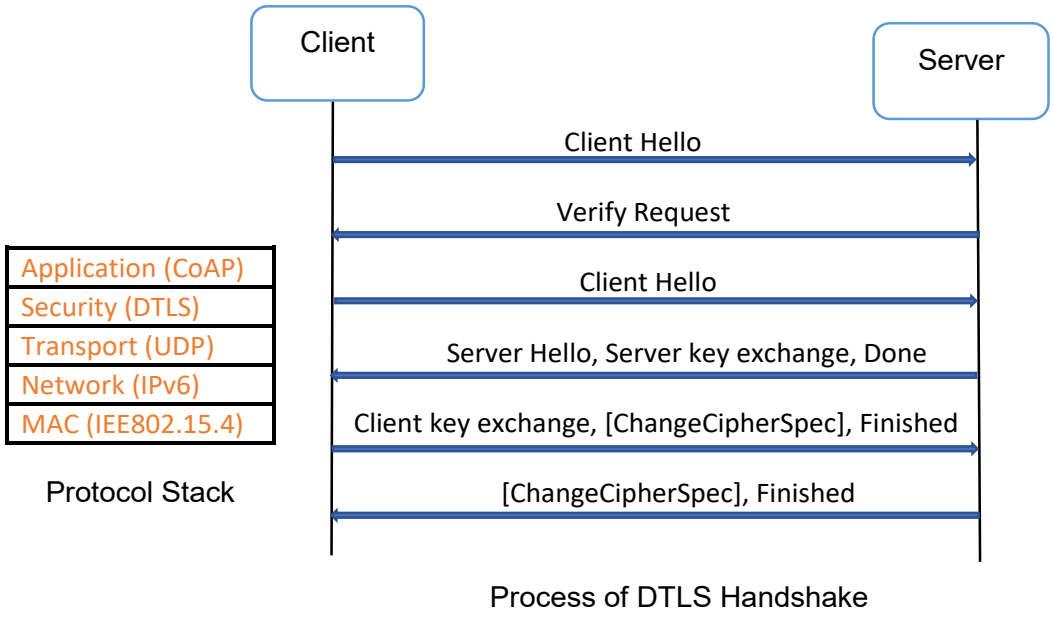

Figure 9: DTLS Handshake Process [61]

3.2.5 MQTT Protocol

MQTT [11] is an application layer protocol as like CoAP which is designed by Andy Stanford-Clark and Arlen Nipper [62]. MQTT protocol standard v3.1.1 is widely used for IoT system. A client device using MQTT always establishes connection to the server. A client publishes application message to the server/broker that is interesting topic to other clients in the network. All published data to the server has a specific topic name. Client can subscribe to request application message that it is interested to receive and can unsubscribe to remove a request [63]. Application message is the data carried by MQTT protocol across the network. Server acts as an intermediary between clients which are publishing and subscribing application messages on same topic. Some sessions

between clients and the server lasts as long as the network connection and others can be connected consecutively [63].

MQTT protocol do not have overall security mechanism. It does not include encryption capabilities but only has authentication mechanism [62]. A MQTT Mosquitto broker can authenticate the client connection using three different ways: client id, username and passwords, client certificates [64].

- Client ID: All clients must have a client ID. When a client wants to subscribe a topic, client id links the topic to the client and to the TCP connection. MQTT broker remembers client id and subscribed topic. To provide client security Mosquitto broker allows to impose client id prefix restrictions on the client name [64].
- Username and passwords: To establish a connection with client, an MQTT broker requires a valid username and password from the client. The username and password is transmitted using a clear text which is not secure without transport encryption form like SSL [64].
- Client certificates (X.509): A digital certificate provided by the recognized authority to each client which is used for Transport Layer Security (TLS). It is the most secure authentication method but difficult to implement on large number of clients [64].

3.3 CoAP and MQTT

Both the IoT application layer protocol CoAP and MQTT are described in the previous sections. Different IoT applications are commonly using these two protocols based on the protocol features. Some of the key differences between CoAP and MQTT are discussed in Table 4.

Features	CoAP	MQTT
Communication Model	Publish-Subscribe or Request-Response	Publish-Subscribe
Messaging	Synchronous and Asynchronous	Asynchronous
Header size	4 bytes	2 bytes
RESTful	Yes	No

Application reliability	Confirmable and Non-Confirmable messages, Acknowledgements and Retransmissions	3 levels Quality of Service
Transport Layer	UDP-based but TCP can be used	TCP-based but UDP can be used
Application Layer	Single Layer with two sub-layer Messages Layer and Request-Response Layer	Single Layer
Security	DTLS or IPSec	Not defined
Performance	Lower overhead and packet loss	Lower delays

Table 4: Differences between CoAP and MQTT [65]

3.4 DTLS Security Protocol

Datagram Transport Layer Security is employed as the security protocol to provide a secure channel between constrained nodes to transfer secure application messages [12]. Constrained-Node Networks (CNN) should concentrate on the device lightweight-ness before the security implementation. Traditional security protocols are not suitable for constrained nodes since the resources are limited [12]. The DICE (DTLS In Constrained Environments) group is delivering absolute efforts to support the use of DTLS for IoT environments [66]. Now, some of the hardware designer of constrained nodes started to integrate additional cryptographic modules as well as AES. However, still now there are no specific cryptographic algorithms that is designed specifically for CNN [12].

CNN relies on the UDP protocol for data transmission due to the simple connection-less architecture of this protocol. But operating connection-less protocol is more complex than the connection-oriented protocol while for flow control it does not maintain any sequence number. Employing security functionalities on UDP protocol is also complex. DTLS is one of the protocols to secure UDP for CNN. Many works have been done by the IETF standardization group DICE to fit large and heavy DTLS into CN as it was not designed for CNN [12]. DTLS is a heavy protocol for constrained IoT devices in terms of time duration and energy consumption. It consists of three sub-protocols which are handshake protocol, alert protocol and change cipher spec protocol [67].

Figure 9 represents the DTLS handshake process. To establish a secure end-to-end connection DTLS handshake process uses three rounds of message communication. To prevent DOS attack for multiple 'ClientHello' message transmission, cookie mechanism is added to the first hello transmission. Server and client negotiate the cipher suite to use in their sessions after all mutual hello message transmission done. At last after the reception of 'Finished' message both devices start to exchange encrypted messages which is not possible to decrypt by the third party [12]. This handshake process requires various message exchanges between client and server which is a heavy process for constrained nodes. In the networks with low transmission quality like CNN, retransmission of handshake messages occurs as regular practice. While the handshake process should be completed to start the secure session, it is important to analyze the network condition earlier.

Session duration is important to constrained nodes because the setup of repeated session requires the additional handshake. Session restarting process is specified in the standard to avoid the complete handshake every time. However, excessive session preservation could be the reason for poor performance of servers, specifically if servers are constrained devices. Hence, administrator should examine the network flow first and then they can start configuring the session duration and session resumption. Longer session duration increases the possibility of being affected by the attackers [12]. To process DTLS in constrained nodes required memory buffer is needed. In the DTLS standard 2^{14} bytes are fixed as maximum plaintext fragment length. For this, in 'ClientHello' message an extension of Maximum Fragment Length Negotiation can be added, which allows the client to inform the server how much memory buffers (maximum size) it can use [12]. Among all application layer protocols for CNN, CoAP and LwM2M has defined the usage of DTLS protocol to establish a secure communication.

DTLS uses three different authentication modes to setup the secure communication are [66] [12]:

- **Pre-shared Key (PSK) based Authentication:** To use this mode for authentication, the secret key should be known to the server and client previously. This secret key needs to be made of various random combinations to keep it safe from adversary attack. Each PSK pair forms with a client ID and the corresponding secret key. Hence, server should know the correspond PSK pair for the client that the client will use to connect to it beforehand. This Pre-shared key mode is the most popular authentication mode that consumes less bandwidth and limited resources [12].

- **Raw Public Key (RPK) based Authentication:** RPK is lightweight in comparison to traditional PKI and does not require any additional structure to convey the public key. It stores the information of public key into the 'SubjectPublicKeyInfo' field of the existing certificate message structure. To exchange the public keys, additional messages and 'server_certificate_type' and 'client_certificate_type' are used in the handshake process [12].
- **Certificate based Authentication:** This Certificate based Authentication mode provides the highest security level. It provides mutual authentication. Since it generates a large communication overhead, it requires the use of cached info extension to minimize the overhead. For this extension, there is no need to send the entire certificate chain by the server in every DTLS handshake [12].

Cipher-suites that are used for these three different authentication modes in CoAP protocol are already mentioned in section 3.2.4. These cipher-suites for different credential types need AES for their confidentiality. To provide AES functionality embedded hardware module or software library can be used. Constrained nodes are encouraged to use embedded hardware module to achieve the performance efficiencies for example low network latency and lowest power consumption [12]. If a device chip does not support hardware module, then have to use the software AES in it. However, software AES is not efficient for high speed required applications or real time applications and it also requires higher memory than the hardware AES. Therefore, now manufacturers started to load AES in hardware as well as other modules like SHA, CCM and even Public Key HW Accelerator driver for ECC calculation [12].

3.5 The Advanced Encryption Standard (AES)

Advanced Encryption Standard [1] is a symmetric key algorithm. Symmetric key algorithms are used for cryptography in which only one key is used to encrypt plaintext and to decrypt the ciphertext [68]. It uses fixed block size. AES contains three block ciphers AES-128, AES-192 and AES-256 [1]. Each of these ciphers using a block length of 128, 192 and 256 bits respectively. Both the sender and receiver must have to know the same secret key. AES-128 uses 10 rounds for 128-bit keys, 192-bit uses 12 rounds and 14 rounds for 256-bit keys [1]. A round is used to process several steps including substitution, transposition and finally to transform the plaintext to the ciphertext. AES key features and advantages are presented in Table 5.

Features	Operations	Advantages
AES gives full specification and details of design.	It performs computations on bytes.	AES provides high/fast security.
It uses larger key sizes.	In the final round, it performs XOR operation to get the ciphertext.	It uses low power consumption.
It is faster and stronger than DES.	The number of rounds in AES is variable and depends on the key length.	AES is fast equally in hardware and software implementation.
AES design is based on substitution-permutation network.	AES uses 128-bits block as 16 bytes which is arranged in 4x4 matrix.	For 128 bits, almost 2^{128} attempts are needed to break. So, it is difficult to hack it.

Table 5: AES Features and Advantages [69]

AES has two main functions: key schedule and round transformation. Key schedule is a process that derives the round keys from the cipher key. AES algorithm defines number of round transformations to be performed on the data stored in an array. The first step is to put the data into an array. Then the cipher transformations are repeated over several encryption rounds. The first encryption round is the substitution of data using a fixed substitution table. The second round shifts the data rows where each row is shifted to the left and then the mixing of columns using special mathematical function [65]. In the next transformation a simple XOR operation performed between the 4x4 matrix and the encryption key [63].

3.6 Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography [70] is an asymmetric key algorithm that is using public key encryption technique to create efficient cryptographic keys. Asymmetric key algorithm uses pair of keys: public key and private key to complete the message encryption and decryption process. If the message is encrypted using a public key from the pair, then it is only possible to decrypt using the other private key of the same pair. ECC algorithm was first proposed for cryptography in 1985 by Neal Koblitz (University of Washington), and Victor S. Miller (IBM) [71] [2]. It can be used in conjunction with some asymmetric encryption methods such as Diffie-Hellman and RSA. Instead of using some traditional methods ECC uses the properties of the elliptic curve equation

to generate keys. According to researchers, this method can accomplish the same level of security with 164-bit key while other system requires 1024-bit key [72]. Elliptic curve supports several cryptographic schemes such as:

- **Key agreement scheme (ECDH, ECDHE):** Elliptic Curve Diffie-Hellman (ECDH) is a method where clients can agree for an elliptic curve public/private key pair over an insecure channel which is normally used to derive another secret key. This derived key can be used for symmetric key cipher. It is also possible to share the key directly but due to the Diffie-Hellman exchange it contains weak bits. Hash of this shared key can be used as solution to remove the weak bits [67].
- **Encryption scheme (ECIES, PSEC):** Elliptic Curve Integrated Encryption Scheme (ECIES) is a public key mechanism that provides encryption, key exchange capabilities and digital signature. It is responsible to organize both message authentication scheme and symmetric key encryption [67].
- **Digital signature scheme (ECDSA):** Elliptic Curve Digital Signature Algorithm is based on the elliptic curve cryptography and it has important differences in comparison to the Digital Signature Algorithm (DSA). It uses ECC private key to sign the data and for the signature verification uses ECC public key. For the process repetition the signature of the data changes because of the involvement of a random value in the ECDSA algorithm. This random value is automatically created by the signature creation process and during signature verification it is recalculated [68]. Some key features and advantages of ECC are presented in Table 6.

Features	Advantages
ECC uses a shorter encryption key.	The shorter ECC key is faster.
It provides same level of security strength like RSA but uses much shorter key length than RSA.	Due to the use of a key with a shorter length, the generated ciphertexts and signatures are also smaller.
It uses public key and private key for each node.	It requires less computational power in comparison to other asymmetric encryption methods such as RSA.

Most suitable for wireless devices with limited memory and computing power.	Signatures can be computed in two stages, allowing latency much lower than inverse throughput.

Table 6: ECC Features and Advantages

3.7 Security Threats in IoT Application Layer

Security is the most important topic for IoT. The main goal of IoT security is to provide reliable connection, confidentiality and providing proper authentication mechanisms about the transmitted data for each node connected to the network. Many security principles should be enabled to each IoT layer for efficient use of the IoT applications. Security threats to any part of the Data integrity, confidentiality and data availability could be the serious reason behind the damage to the system. There are many security threats to IoT application layer, and it is also challenging to overcome these security problems. Some of the possible application layer security problems are:

- DOS attack where the attacker behaves like an authorized user, login to the system and interrupt normal activities of the network [3].
- Phishing attack in which hacker succeed to achieve the credential access of an authenticated user and damage data using email of the network authority [3].
- Malicious code injection is the technique where attacker injects some malicious code into the system to manipulate user data or to steal data [3].
- Sniffing attack in which hacker commence a sniffer application into the system to force an attack on it to reach the network information [3].

To solve the above mentioned IoT application layer security threats it is important to employ some possible steps. However sometimes for some cases it becomes difficult to apply proper solutions to solve the security problems. Sowmya et al. [3] presented the difficulties related to security to overcome the security threats and some possible security measures to succeed the problems related to security which are discussed below and in table 7:

- Authentication of User ID: At the same time different users' requests for different applications. Each application has large number of users. Hence, to prevent the network from unauthorized access proper authentication mechanism should be deployed [3].

- Data Handling: A huge number of network user involves with the large amount of data processing. During the communication process it makes the possibility of data loss which can affect network efficiency [3].
- Data storage: The data storage comprises data transmission using different channels to several locations which includes data integrity and user privacy. There is a high risk of attack during data transmission, so it is important to organize appropriate data storage and recovery at every steps of data transmission [3].
- Software vulnerabilities: Due to the non-standard codes in software, vulnerabilities of software may occur. Such software bugs might be exploited by an adversary and allowed him to launch powerful attacks [3].

Risk assessment	This process can identify the threats in the network. It can be used to analyze situation and to check the level of risk acceptance.
Intrusion Detection	This method can produce alarm on any ambiguous activity in the network and provides solution for threats by cloud computing, uninterrupted monitoring and virtualization.
Data Security	To preserve data from unauthorized user encryption methodologies can be includes. In addition, updated malwares and up-to-date anti-dos-firewalls can be included to achieve data security.

Table 7: Security measures [3]

3.8 IoT Security Challenges

IoT opens great opportunities for devices, software and applications to share information and communicate over the internet. This shared data may contain many private information of users. Hence, it is important to protect this information from any possible adversary that might try to get unauthorized access. Due to the heterogeneity and large scale of objects in IoT new security issues emerges compared to traditional security aspects. These IoT security problems are more complex than the traditional problems. Some possible IoT attack surfaces are presented in table 7. To provide the proper solution for security problems IoT must encounter numerous challenges [73] that are presented below:

- **Authentication and Authorization:** Public-key cryptosystem can provide the authentication process or authorization but without global root certificate authority it is difficult to design the complete authentication scheme for IoT. In addition, it may not be feasible to issue certificate for large number of objects as IoT often designed with a huge number of objects [73].
- **Privacy:** The IoT challenges to privacy preserving can be divided into data anonymization and data collection process. Data collection process describes type of data, amount of information and the access control of a device to the data. If the collected data and private information storage is restricted than ensuring the privacy is possible. Cryptographic protection and secrecy of data relations are required to ensure data anonymity. For IoT resource constrained devices lightweight cryptographic methods are suitable. Data encryption can be used to maintain the secrecy of data relations. Hence, the encrypted data computation for data anonymization is a challenge for IoT constrained devices [73].
- **Object Identification:** For object identification main challenge is to assure the integrity of records in the naming architecture. Although DNS system provides name translation services, but it is not a secure naming system. Domain Name Service Security Extension (DNSSEC, IETF) is extended as the DNS security extension which ensures the authenticity and integrity of resource records. DNSSEC requires high communication and computation overhead hence for IoT devices this naming scheme might not be suitable. Therefore, it is still challenging for IoT to overcome this problem.
- **Malware:** As because the IoT devices has limited resources so the threat specifically targeted to IoT malware is significant. Traditional security mechanism may not be feasible against malware. Therefore, antivirus could be the effective solution to detect known malware. But the antivirus scanning functionality may results the overhead to IoT devices.
- **Lightweight Cryptosystem:** Asymmetric key cryptosystem can provide some more security features compared to private-key cryptosystem, but the problem is computational overhead. In the case of authenticity and data integrity public-key cryptography are mostly desirable. Therefore, maintaining complex security protocols and reducing computational overhead for the asymmetric key cryptosystem has become a great challenge for IoT [73].

Global Network	IoT Service ⟷ Service
	Controller ⟷ IoT Service Provider
Local Network	IoT Device ⟷ IoT Device
	Device ⟷ Coordinator
	Coordinator ⟷ Gateway

Table 8: IoT Attack Surfaces [74]

4. EXPERIMENTAL RESULTS WITH ZOLERTIA

4.1 Zolertia IoT Devices

Zolertia *[75]* is a company located in Barcelona, Spain that provides hardware solutions for Internet of Things Applications. Zolertia company produces their own hardwires and firmware but also design the hardwires with specific solutions based on their client demands. It allows the integration of long-range communication technology between IoT constrained devices *[76]*. Their projects claim to be appreciated for academic and organizational purpose as well as for the developers.

Three Main Zolertia hardware platforms are:

Z1 Mote: Zolertia's first commercially available platform was Z1 Mote, developed for researchers and enthusiasts. It is a general-purpose card and has an ultra-low power MCU with 16 MHz clock speed, 8 Kb RAM and 92 Kb Flash. It includes 2.4 GHz Transceiver with data rate of 250 Kbps and two on-board digital sensors [77].

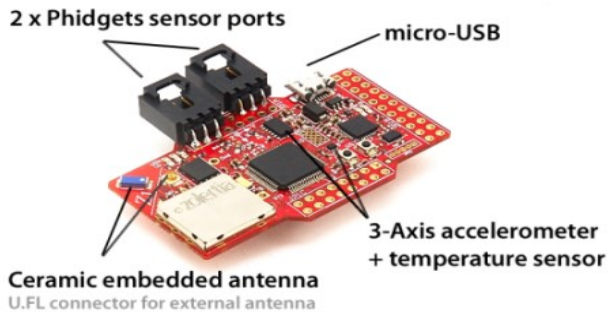

Figure 10: Zolertia Z1 Mote [77]

Figure 11: Zolertia Firefly [77]

Firefly: Firefly is a small card with some essential features. It reveals the basic Zolertia Zoul features. It includes powerful CC2538 core by Texas Instruments, an 32-bit ARM Cortex-M3 with 512KB flash, 32KB RAM and 32MHz clock speed [77].

RE-Mote: RE-Mote is the latest hardware development platform designed jointly with the industrial partners and universities. It includes all Zoul features as like Firefly. It carries battery charger, two radios, external storage and available interfaces and connectors for analogue or digital sensors [77].

4.2 Tools Utilized

All the technological details about the tools and technologies required for this chapter, IoT protocols and security mechanisms are thoroughly described in the previous chapters. Hence at this section 4.2, a brief description of the technical tools that were used to reach our motive of implementing secure communication between IoT devices are presented.

4.2.1 Operating Systems

Contiki: Contiki [78] is an open source operating system for low-powered, memory-constrained IoT devices that uses wireless sensor networks. Contiki is written in C language and provides the appropriate functions for the connectivity between IoT devices [27]. It supports full IP network stack IPv6 and IPv4 as well as 6LowPAN stack [77]. Contiki has a simulation environment Cooja Network Simulator that allows developers to see their applications is run in large-scale network or detail on the experimental hardware devices [78]. Contiki includes libraries for the applications like HTTP, UDP, CoAP and MQTT client [77].

RIOT: RIOT [79] is an open source, user friendly operating system for IoT. It was designed for real-time capabilities, high energy efficiency, multi-threading and to support wide range of low-power, low memory supportive IoT devices [77]. It provides tools and utilities like hash tables, system shell, SHA-256, bloom filters and cryptographic libraries [77] [79]. RIOT supports 6LowPAN stack, IPv6, UDP, RPL and CoAP [79].

4.2.2 6LowPAN Border Router

In order to implement MQTT system, CETIC-6LBR was used as a Border Router (BR) with a broker and Zolertia RE-Mote devices (WSN nodes) as clients running on Local Area Network (LAN). A Border Router is used to connect the 6LowPAN devices with the IPv6 network. Therefore, it is responsible for managing all traffics between 802.15.4 and IPv6 network. 6LBR [80] is an open source Border Router solution based on the Contiki Operating System. It can be used for low-cost, open source hardware platforms and Linux host [77]. 6LBR supports different network topologies and can be designed with variety of network architectures such as Smart Bridge, Router and Transparent Bridge [81]. Some additional features are network

autoconfiguration, multicast communication and synchronization of 6LowPAN WSNs with IP Network [81]. 6LBR border router architecture is presented in Figure 12.

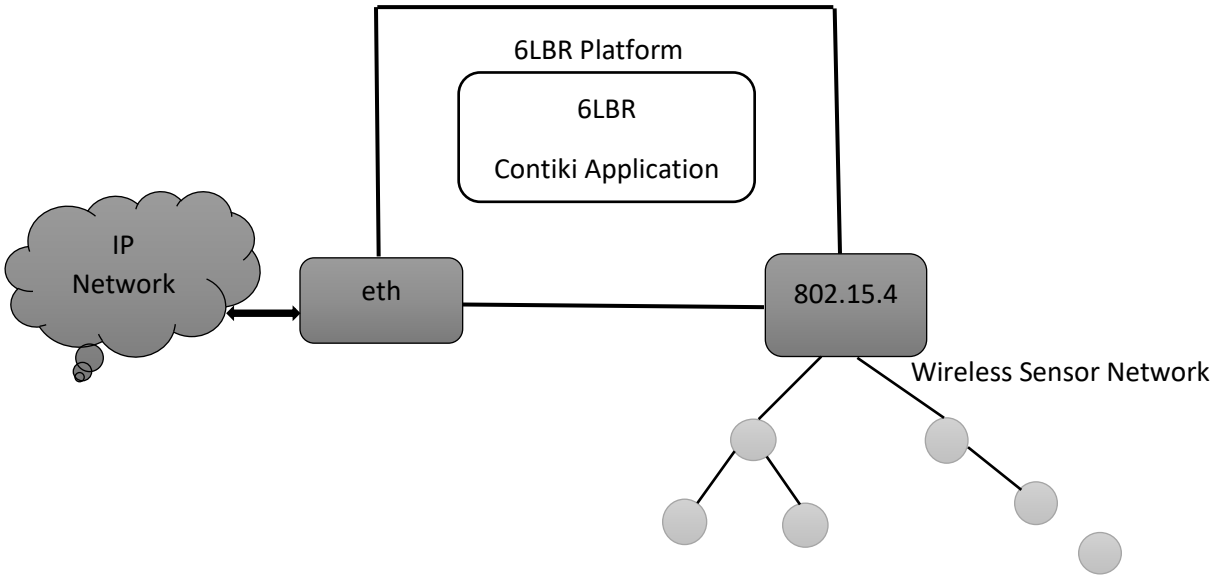

Figure 12: 6LBR Border Router

CETIC 6LBR in Figure 13 has two 2.4 GHz and SUB-G radio connections, 1 Mini to Standard USB cable connection and a RJ-45 cable connection port. It has a reset button which can be used to restart the device configuration.

Figure 13: CETIC 6LBR

4.2.3 Digital Grove Light Sensor

This Digital Grove Light Sensor module is based on the I2C light-to-digital converter TSL2561 to transform light intensity to a digital signal [82]. It is different from analog light sensor. Digital Grove Light Sensor has three different detection modes: infrared mode, human visible and full apectrum mode [82].

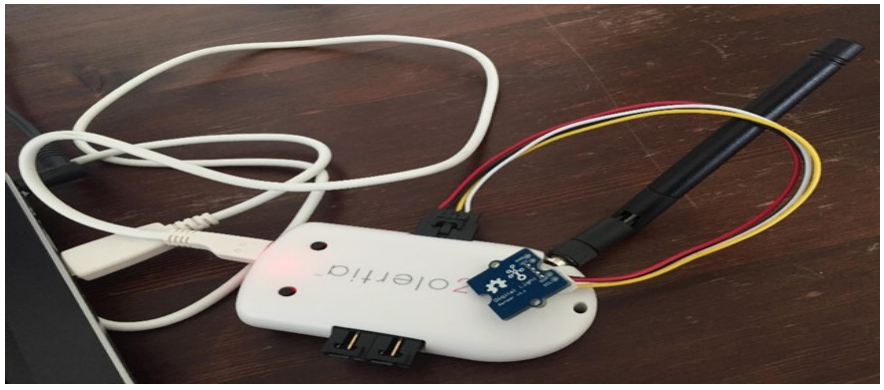

Figure 14: Digital Grove Light Sensor with Zolertia RE-Mote

Some key features of this light sensor are [82]:

- Detection modes are selectable.
- Dynamic range is 0.1 - 40,000 LUX.
- Sensor dimension is 24mm x20mm x9.8mm.
- Temperature range to operate is -40℃ to 85℃.
- High resolution 16-Bit digital output at 400 kHz I2C Fast-Mode.

4.2.4 Zolertia RE-Mote as a hardware platform

The Zolertia RE-Mote is a hardware development platform that allows the development of IoT applications. It includes powerful Texas Instruments CC2538 core ARM Cortex-M3 system on chip that works at up to 32 MHz with programmable 512 KB flash and 32 KB RAM [83]. RE-Mote has the following features [83]:

- Two radios (ISM 2.4-GHz IEEE 802.15.4 & Zigbee compliant radio and ISM 863-950-MHz ISM/SRD band IEEE 802.15.4 compliant radio) for both long range and indoor applications. The maximum communication range is 100 m to 20 km with highly configurable parameters like data rate, modulation and transmission power.

- It has built-in battery charger, uses rechargeable LiPo batteries, direct connection to solar panel and energy harvesting methods. USB cable connection can automatically recharge the battery.
- Ultra-low power consumption down to 150 nA using shutdown mode.
- External storage with a micro SD to store and retrieve data.

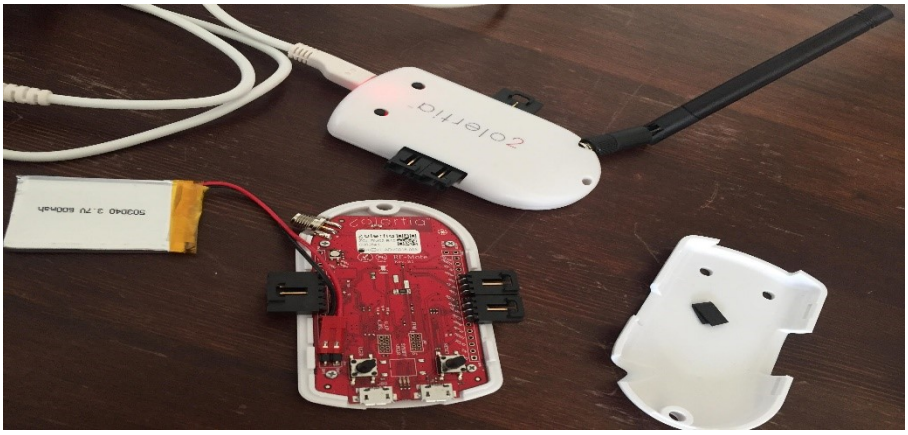

Figure 15: Zolertia RE-Mote

- Available connectors and interfaces for analog or digital sensor connection simultaneously.
- It has Real Time clock capabilities to develop applications based on real time information such as setting the device as wake up mode in every 10 minutes to send latest data.
- The RE-Mote devices are supported in Contiki and RIOT open source operative systems.

4.3 Test Implementation and Results

4.3.1 Printing text message

At first, we started our implementation round with printing simple text messages in a Zolertia RE-Mote device. For this testing, we used Contiki OS version 3.0 and a Zolertia RE-Mote as a hardware device with IPv6 address: 00:12:4B:00:18:E6:9C:62. We used an example program from Contiki for the testing purpose. In Figure 16 we can see the output of the program.

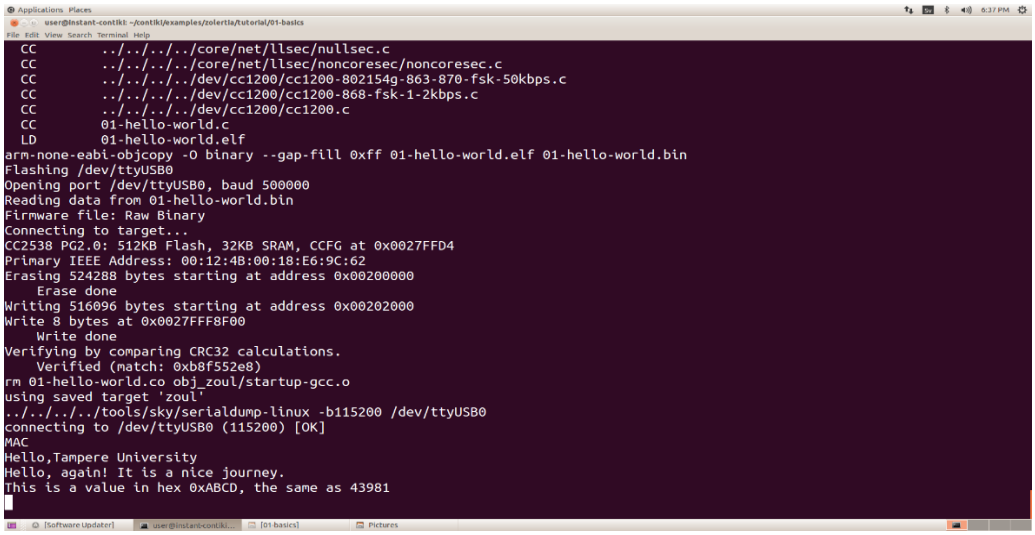

Figure 16: Hello message in a Zolertia RE-Mote

The main function of the testing program is given below:

```
PROCESS_THREAD(hello_world_process, ev, data)
{
  /* Process starts here*/
  PROCESS_BEGIN();

  static uint16_t num = 0xABCD;
  static const char *hello = "Hello, again! It is a nice journey.";

  printf("Hello,Tampere University\n");

  /* will print the defined string*/
  printf("%s\n", hello);

  /* Here then  mix numeric values with strings */
  printf("This is a value in hex 0x%02X, the same as %u\n", num, num);

  /* End of the process */
  PROCESS_END();
}
```

4.3.2 Data reading from a Digital Light Sensor

In this section 4.3.2 we used a Digital Grove Light Sensor to measure the light weight of a room. In addition, we used Contiki testbed and Zolertia RE-Mote to reach the desired goal. In Figure 17, we can see the variation of light value in between 0 to 85 (unsigned integer value as defined in program). We used a Contiki example program for this testing.

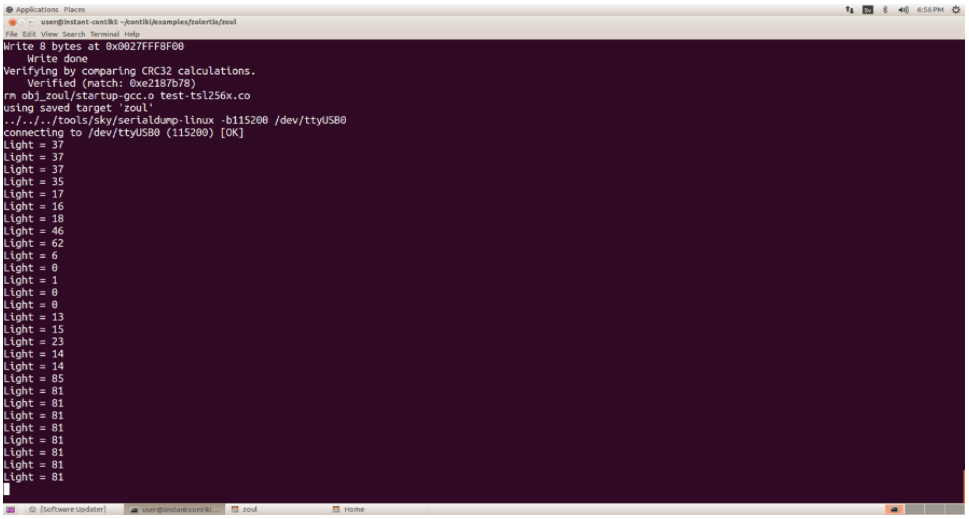

Figure 17: Testing with Digital Grove Light Sensor

Here is the main function of the Contiki example program for Digital Grove Light Sensor [84]:

```
PROCESS_THREAD(remote_tsl256x_process, ev, data)
{
  PROCESS_BEGIN();
  static uint16_t light;

  if(TSL256X_REF == TSL2561_SENSOR_REF) {
    printf("Light sensor test --> TSL2561\n");
  } else if(TSL256X_REF == TSL2563_SENSOR_REF) {
    printf("Light sensor test --> TSL2563\n");
  } else {
    printf("Unknown light sensor reference, aborting\n");
    PROCESS_EXIT();
  }

 SENSORS_ACTIVATE(tsl256x);

 TSL256X_REGISTER_INT(light_interrupt_callback);

 tsl256x.configure(TSL256X_INT_OVER, 0x15B8);

 while(1) {
    etimer_set(&et, SENSOR_READ_INTERVAL);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    light = tsl256x.value(TSL256X_VAL_READ);
    if(light != TSL256X_ERROR) {
      printf("Light = %u\n", (uint16_t)light);
    } else {
      printf("Error, enable the DEBUG flag in the tsl256x driver for info, ");
      printf("or check if the sensor is properly connected\n");
      PROCESS_EXIT();
    }
  }
  PROCESS_END();
}
```

4.3.3 Implementation of UDP Protocol

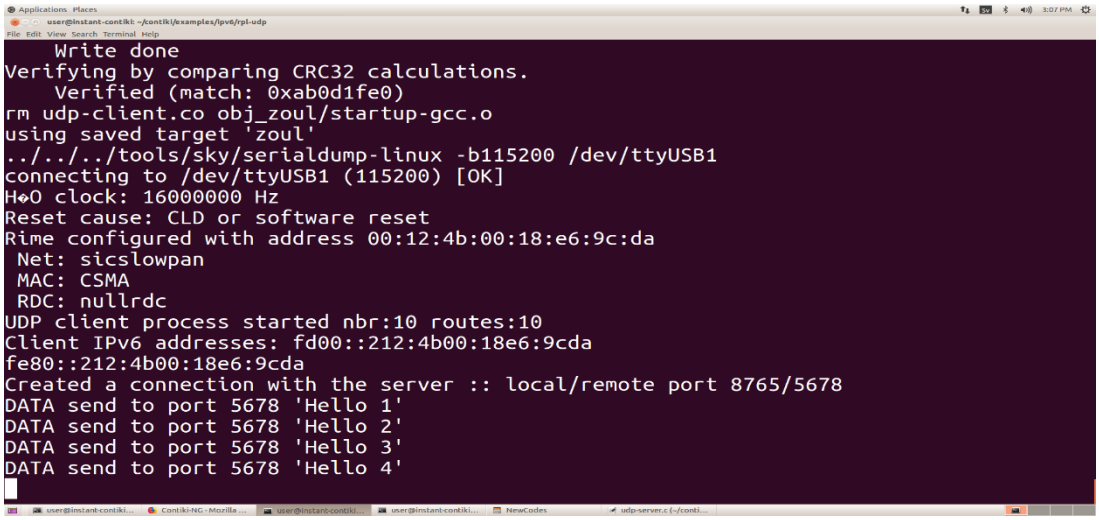

Figure 18: UDP Client

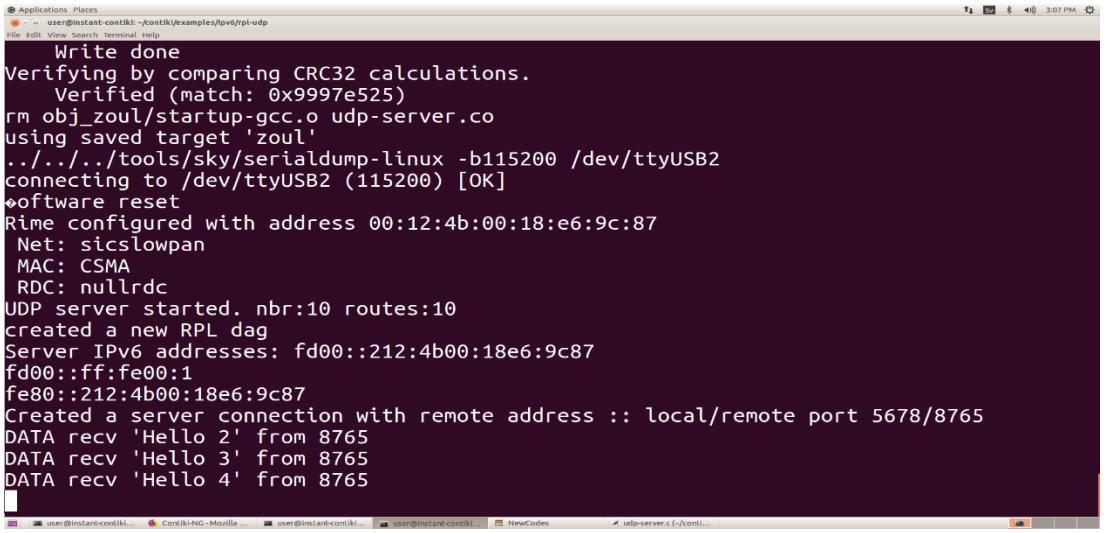

Figure 19: UDP Server

To establish the UDP communication we used two hardware RE-Mote device, one device acting as a UDP client and another one as a UDP server. We used a device as a border router, global and link local IPv6 address of the border router is in Figure 20. To check the device connection with the local network we activated the border router. UDP server port 5678 and Client port 8765 are defined in the example programs of Contiki. In Figure 18 we can see the UDP client is sending

the ‘Hello’ message to the UDP server and the server is receiving the same packet data from the client in Figure 19. As we know UDP does not recover the lost data, we can see that the UDP server missed the first message ‘Hello 1’ but after that it received all packets. Figure 21 is the Wireshark capture for UDP client and server communication.

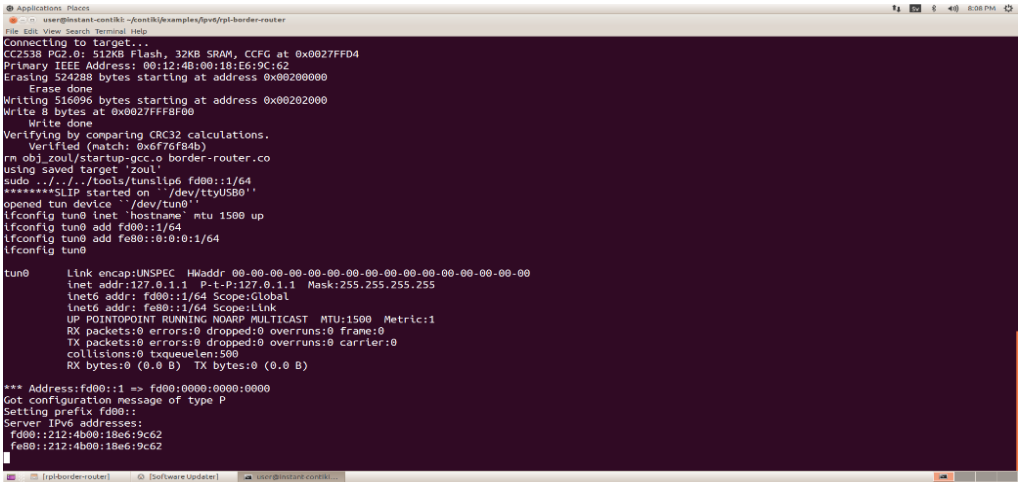

Figure 20: Border Router

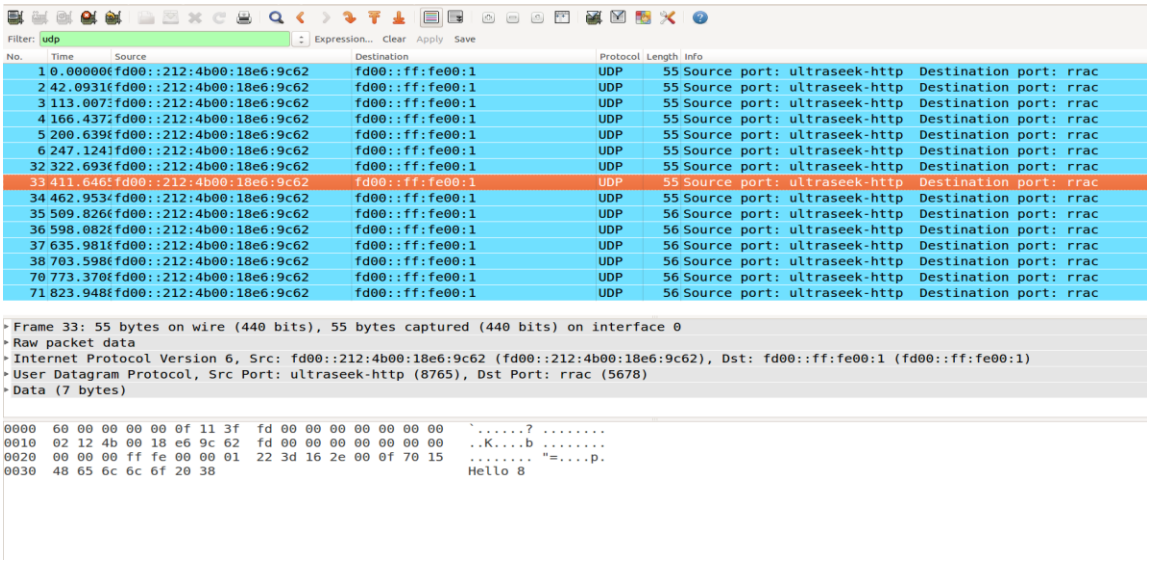

Figure 21: Wireshark capture for UDP protocol

4.3.4 Implementation of MQTT Protocol

For MQTT protocol implementation we used a CETIC-6LBR border router, three Zolertia RE-Mote devices and a temperature & humidity sensor. We used the IPv6 address of the border router as a MQTT broker. All MQTT clients are connected to the broker. Each MQTT client is responsible to publish data on a specific topic to the broker and as well as can subscribe data for a specific topic. Broker has the database to store the published topic data. Therefore, if any MQTT client subscribe to the broker for a topic it can provide the updated data of that topic to the client. MQTT clients can connect to the broker and also can disconnect at any time. In Figure 22 all activated MQTT clients are connected to the border router. Type is defined in the list according to the MAC address of the device and web or coap is the direct link to the HTTP server or CoAP server. Upstream and downstream packet reception rate can be added to the program. Last seen value is in seconds which means the time since any packet data received from the client. If everything is going well then, the status shows OK.

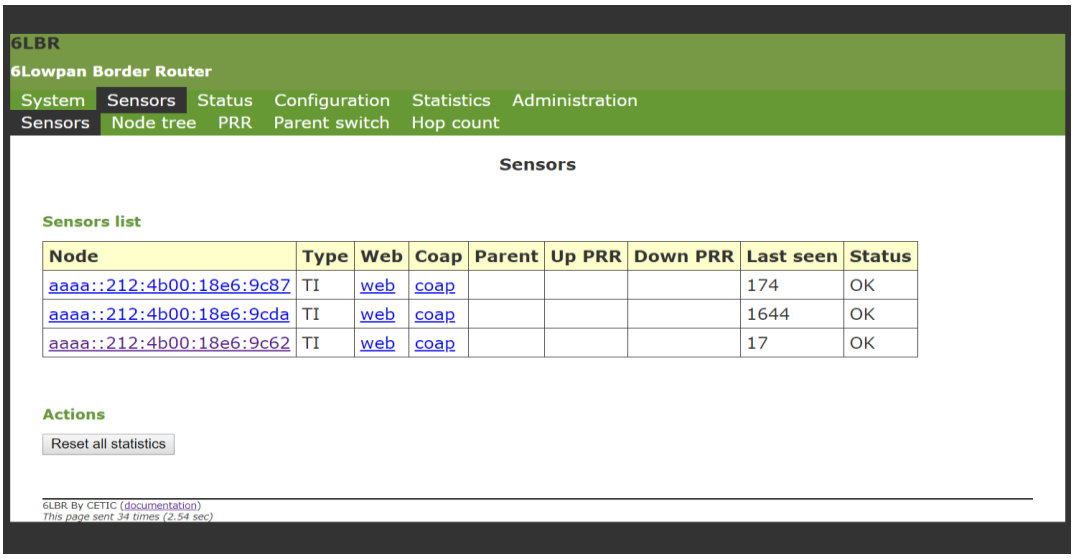

Figure 22: MQTT Clients connected to Border Router

In figure 23 we can see that a MQTT client publishing sensor data to the broker on a topic zolertia/evt/status. At first the client attempts to registered as a MQTT client. Then when the registration is done, client connected to the MQTT broker. Then it subscribes on a topic zolertia/cmd/leds which means that the client needs the updated data for the subscribed topic. Finally, MQTT client publishes packet data on a specific topic to the MQTT broker.

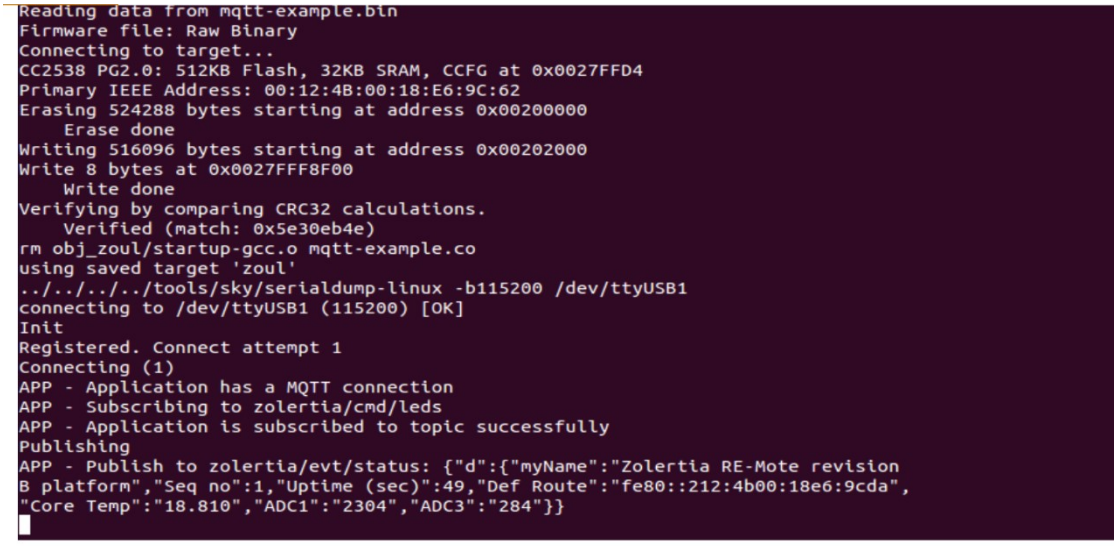

Figure 23: MQTT Client publishing data

4.3.5 Encrypted message

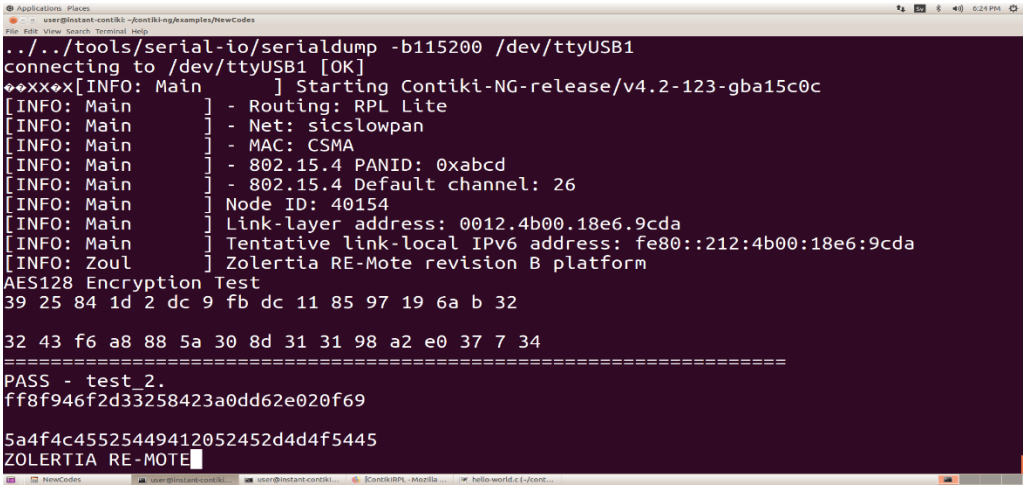

Figure 24: Message Encryption and Decryption

For this 4.3.5 we used AES-128 encryption method to encrypt and decrypt normal text messages to check the device response. In Figure 24 we can see the generated key first and then the ciphertext from the plaintext and finally the decryption of the ciphertext which is the normal given plain text in the program. In the below I have added the main function of the program and the other two functions test_2() and var_text_test() which are used in the program to get the desired output.

Main Function

```
  PROCESS_BEGIN();

  int result = TC_PASS;

  result = test_2();
  if (result == TC_FAIL) { /* terminate test */
    TC_ERROR("AES128 test #2 (NIST encryption test) failed.\n");
  }

  const char *msg = "ZOLERTIA RE-MOTE";

  struct tc_aes_key_sched_struct s;
  (void)tc_aes128_set_encrypt_key(&s, nist_key);

  var_text_test((const uint8_t *) msg, &s);

  PROCESS_END();
}
```

Other functions

```
int test_2(void)
 {
   int result = TC_PASS;
   int i;

   struct tc_aes_key_sched_struct s;
   uint8_t ciphertext[NUM_OF_NIST_KEYS];
   uint8_t plaintext[NUM_OF_NIST_KEYS];

   TC_PRINT("AES128 Encryption Test\n");

   (void)tc_aes128_set_encrypt_key(&s, nist_key);

   tc_aes_encrypt(ciphertext, nist_input, &s);

   for (i=0; i<NUM_OF_NIST_KEYS; i++)
   {
     printf("%x ", ciphertext[i]);
   }

   printf("\n\n");

   (void)tc_aes128_set_decrypt_key(&s, nist_key);

   tc_aes_decrypt(plaintext, ciphertext, &s);

   for (i=0; i<NUM_OF_NIST_KEYS; i++)
   {
     printf("%x ", plaintext[i]);
   }

   printf("\n");
```

```
int var_text_test(const uint8_t *in, TCAesKeySched_t s)
 {
   uint8_t ciphertext[NUM_OF_NIST_KEYS];
   uint8_t plaintext[NUM_OF_NIST_KEYS];
   int j;


   (void)tc_aes_encrypt(ciphertext, in, s);

   for (j=0; j<NUM_OF_NIST_KEYS; j++)
   {
     printf("%x", ciphertext[j]);
   }


   printf("\n\n");

   tc_aes_decrypt(plaintext, ciphertext, s);

   for (j=0; j<NUM_OF_NIST_KEYS; j++)
   {
     printf("%x", plaintext[j]);
   }

   printf("\n");

   for (j=0; j<NUM_OF_NIST_KEYS; j++)
   {
     printf("%c", plaintext[j]);
   }

   return 0;
 }
```

```
    result = check_result(2, expected, sizeof(expected),
ciphertext,
            sizeof(ciphertext));

    TC_END_RESULT(result);

    return result;
  }
```

4.3.6 Secure message Transmission

Here we used UDP communication between devices to transmit and receive secure messages. For this purpose, we used AES cryptographic method where both devices using a pre-shared key. We used two Zolertia RE-Mote devices as hardware platforms. In Figure 25 we can see the UDP client device first exchanging the session key and then establishing a secure connection by sending the encrypted response for text messages along with the key after receiving the request from the other device. How AES algorithm works is explained in the previous chapter in section 3.4.

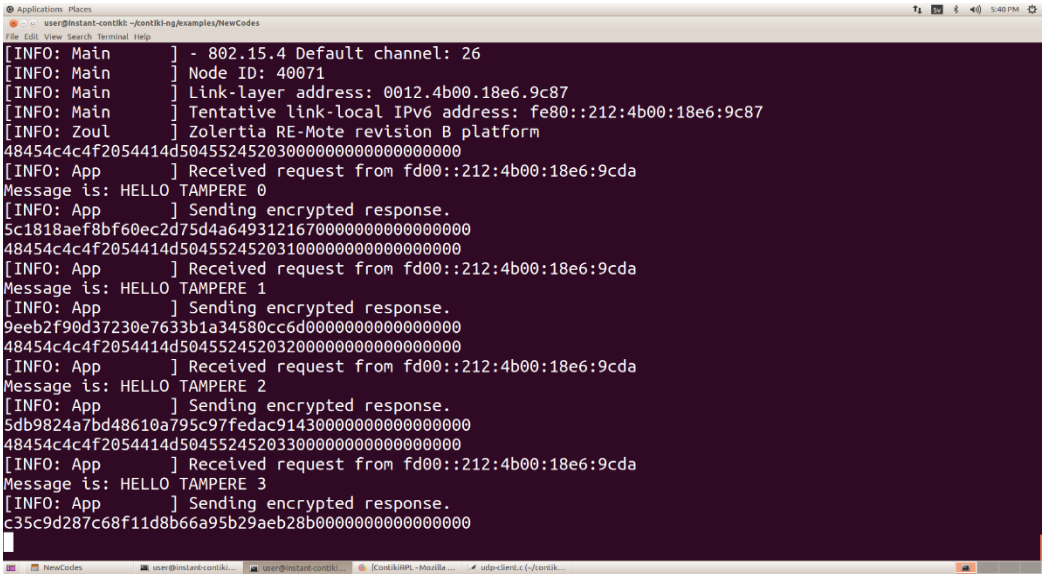

Figure 25: Device sending Encrypted Message

UDP server response is represented in Figure 26. After receiving the encrypted response from the client device, the server is decrypting the encrypted ciphertext to the plaintext. After that it is sending the request to the client device for the next message.

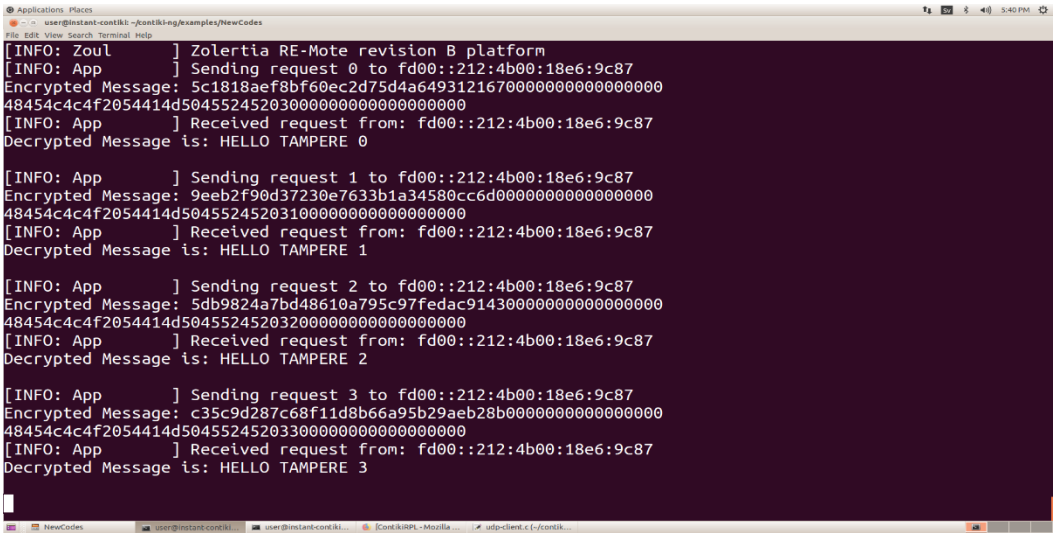

Figure 26: Device Decrypting received message

4.3.7 Message Encryption using ECC

Elliptic-Curve Cryptography (ECC) public key encryption method is used to secure the transmitted data. Our main goal was to secure the communication between two or more IoT devices. To this end, we used ECC in order to establish a symmetric encryption key that would be finally used as a session key for encrypting the communication channel between a set of IoT devices. For the encryption of the communication channel we used AES-128 message encryption– hence all the exchanged messages during a session were encrypted using this algorithm. We were able to deploy the complete ECC key generation process in a Zolertia RE-Mote device to see the performance and how the method works. The output result of the used ECC encryption process are presented in Figure 27. How the program is designed is explained below:

For two devices A and B first the ECC program computes two random 256 bits secret keys respectively for Secret A and Secret B. The functions ecc_setRandom(secretA) and ecc_setRandom(secretB) are used to compute the secret key. Then two Basepoints (X,Y) are defined in the program which are used along with the secret key to generate the public and private

key pair for each device. Then the devices sharing their keys with each other. After that the 256 bits shared private key for both devices generated using the key pairs and these keys should be same. To reduce the key size from 256 bits to 128 bits Hash function is used in the program to make it suitable for the IoT constrained device. Then the ciphertext and hash value for the plaintext are generated and the message decryption process added at the last part of the program. During the implementation we noticed that it took a bit longer time to generate the shared private key. To make the process clearer the main program is added below the Figure 27.

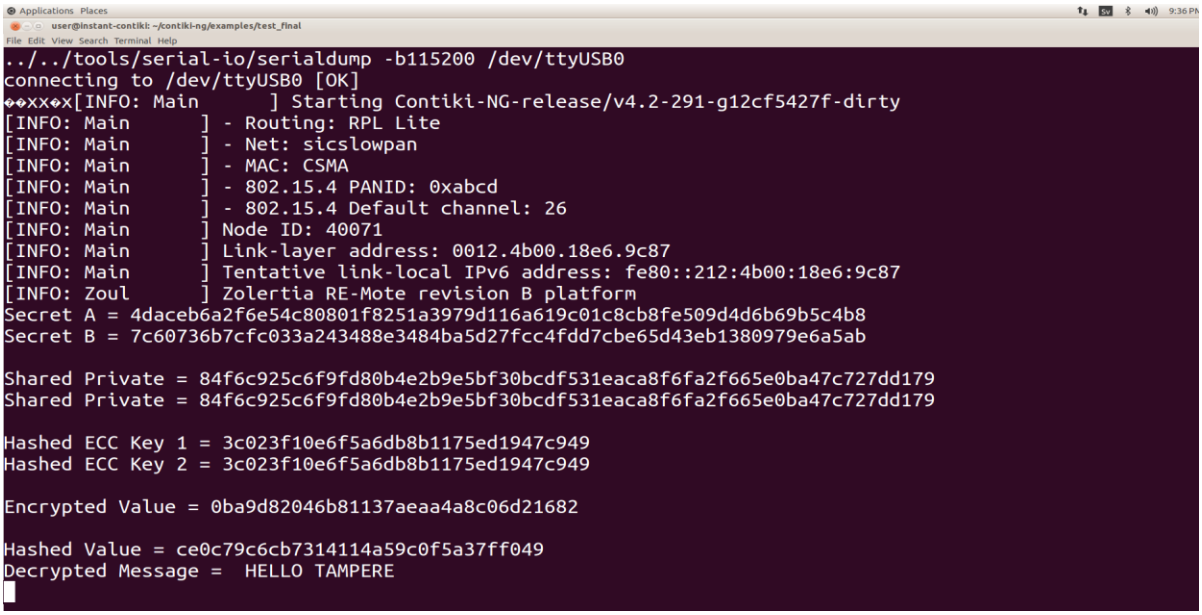

Figure 27: ECC key generation and message encryption

```
#define NUM_OF_NIST_KEYS 16
#define NUM_OF_FIXED_KEYS 128
struct kat_table {
  uint8_t in[NUM_OF_NIST_KEYS];
  uint8_t out[NUM_OF_NIST_KEYS];
};
//These are testvalues taken from the NIST P-256 definition
//6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296
```

```
uint32_t BasePointx[8] = {	0xd898c296, 0xf4a13945, 0x2deb33a0, 0x77037d81, 0x63a440f2, 0xf8bce6e5, 0xe12c4247, 0x6b17d1f2};

uint32_t BasePointy[8] = {	0x37bf51f5, 0xcbb64068, 0x6b315ece, 0x2bce3357, 0x7c0f9e16, 0x8ee7eb4a, 0xfe1a7f9b, 0x4fe342e2};

uint32_t Sx[8] = {	0x89da97c9, 0xb77cab39, 0x221a8fa0, 0x617519b3, 0x0f271508, 0x82edd27e, 0xbc8d36e6, 0xde2444be};

uint32_t Sy[8] = {	0x3042a256, 0xb6350b24, 0x53cec576, 0x702de80f, 0xd1e66659, 0xfc01a5aa, 0xf36e5380, 0xc093ae7f};

uint32_t Tx[8] = {	0x35e0986b, 0xbb8cf92e, 0x61c89575, 0x39540dc8, 0x5316212e, 0x62f6b3b2, 0x8da1d44e, 0x55a8b00f};

uint32_t Ty[8] = {	0xc8b24316, 0xb656e9d8, 0x598b9e7a, 0xf61a8a52, 0xc4c3dd90, 0x4835d82a, 0x9c2d6c70, 0x5421c320};


struct Hash_Msg

{

   uint8_t hashtag[NUM_OF_NIST_KEYS];

};

struct Encrypt_Msg

{

   uint8_t ciphertext[NUM_OF_NIST_KEYS];

   uint8_t plaintext[NUM_OF_NIST_KEYS];

};

struct Sha_Msg

{

   uint8_t digest[NUM_OF_NIST_KEYS];

};

struct Generate_ECC_Key

{

	uint32_t sharedK1[8];

	uint32_t sharedK2[8];

};

struct Generate_ECC_Key ecc_function()
```

```
{
        struct Generate_ECC_Key ecctest1;
        uint32_t tempx[8];
        uint32_t tempy[8];
        uint32_t tempAy2[8];
        uint32_t tempBx1[8];
        uint32_t tempBy1[8];
        uint32_t tempBy2[8];
        uint32_t secretA[8];
        uint32_t secretB[8];

        /* Compute the secret key randomly */
        ecc_setRandom(secretA);
        printNumber("Secret A", secretA, 8);
        ecc_setRandom(secretB);
        printNumber("Secret B", secretB, 8);

        /* Generate Private and Public Key Pair */
        ecc_ec_mult(BasePointx, BasePointy, secretA, tempx, tempy);
        ecc_ec_mult(BasePointx, BasePointy, secretB, tempBx1, tempBy1);

        //public key exchange
        ecc_ec_mult(tempBx1, tempBy1, secretA, ecctest1.sharedK1, tempAy2);
        ecc_ec_mult(tempx, tempy, secretB, ecctest1.sharedK2, tempBy2);
        return (ecctest1);
};

struct Encrypt_Msg encrypt_test(const uint8_t *in, TCAesKeySched_t s)
{
   struct Encrypt_Msg encrypt1;
```

```
    (void)tc_aes_encrypt(encrypt1.ciphertext, in, s);
    tc_aes_decrypt(encrypt1.plaintext, encrypt1.ciphertext, s);
    return (encrypt1);
};
struct Sha_Msg sha_function(const uint8_t *msg)
{
            struct tc_sha256_state_struct s;
            struct Sha_Msg sha12;
    (void)tc_sha256_init(&s);
    tc_sha256_update(&s, msg, sizeof(msg));
    (void)tc_sha256_final(sha12.digest, &s);

    return (sha12);
};

struct Hash_Msg hmac_function(const uint8_t *msg, const uint8_t *key)
  {
    struct tc_hmac_state_struct h;
    struct Hash_Msg Hash1;
    (void)memset(&h, 0x00, sizeof(h));
    (void)tc_hmac_set_key(&h, key, sizeof(key));
    (void)tc_hmac_init(&h);
    (void)tc_hmac_update(&h, msg, sizeof(msg));
    (void)tc_hmac_final(Hash1.hashtag, TC_SHA256_DIGEST_SIZE, &h);
    return (Hash1);
  }


/*---------------------------------------------------------------------------*/
PROCESS(final_program, "Final ECC Program");
```

```
AUTOSTART_PROCESSES(&final_program);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(final_program, ev, data)
{
        PROCESS_BEGIN();
        srand(5678);
        char *msg = " HELLO TAMPERE ";
        struct Generate_ECC_Key ecckey;
        struct Sha_Msg sha2;
        struct Sha_Msg sha3;
        struct Encrypt_Msg encrypt2;
        struct Hash_Msg hash2;

        struct tc_aes_key_sched_struct s;
        ecckey = ecc_function();
        printf("\n");
        printNumber("Shared Private", ecckey.sharedK1, 8);
        printNumber("Shared Private", ecckey.sharedK2, 8);
        printf("\n");
        sha2 = sha_function((const uint8_t *)ecckey.sharedK1);
        sha3 = sha_function((const uint8_t *)ecckey.sharedK2);
        (void)tc_aes128_set_encrypt_key(&s, sha2.digest);
        show_str("Hashed ECC Key 1", sha2.digest, sizeof(sha2.digest));
        show_str("Hashed ECC Key 2", sha3.digest, sizeof(sha3.digest));
        printf("\n");
        encrypt2 = encrypt_test((const uint8_t *)msg, &s);
        show_str("Encrypted Value", encrypt2.ciphertext, sizeof(encrypt2.ciphertext));
        printf("\n");
        hash2 = hmac_function(encrypt2.ciphertext, sha2.digest);
        show_str("Hashed Value", hash2.hashtag, sizeof(hash2.hashtag));
```

```
        unsigned int i;


    TC_PRINT("Decrypted Message = ");
    for (i = 0; i < (unsigned int) sizeof(encrypt2.plaintext); ++i)
    {
        TC_PRINT("%c", encrypt2.plaintext[i]);
    }
    TC_PRINT("\n");
        PROCESS_END();
}
```

5. CONCLUSIONS

5.1 Conclusion and Future Work

IoT technologies such as sensor and RFID make our life becomes more comfortable and easier. The Importance of security in the IoT sector is dramatically rising along with the rapid progression of the underlying industry. The vision of IoT is not only limited to make our lives easier but also to ensure safety benefits for the end-users. The presence of the IoT paradigm in the last few years has conducted to diverse security or privacy threats and attacks in IoT. In fact, in the IoT system model, there are many security issues which can be utilized by security threats to make the system weak. To protect IoT from those threats it is essential to completely enforce the security and trust management in the IoT world with the beginning of threats characterization for each layer of IoT networks as well as making secure the communication protocols.

The main target of this thesis has been to express the importance of IoT security and challenges to employ security in IoT. In this regard, this research focuses on the existing studies and analyzing them to describe the security issues, challenges in IoT and different IoT protocols. Several limitations to apply traditional security schemes in IoT are found from the literature study. It is found from thesis research that, due to the constrained resources of IoT devices, there are several challenges to implement security and threats to IoT which are discussed in section 3.1. In addition, some protocol implementations and secure message transmission using AES-128 encryption standard are added in Chapter 4. IoT protocols IEEE 802.15.4, 6LowPAN, RPL, MQTT and CoAP both application layer protocols, functions of each protocol and their security features as well as their limitations regarding security are discussed in section 3.2. DTLS security protocol which is required by the CoAP protocol as an additional protocol to establish secure communication and cryptographic algorithms Advanced Encryption Standard (AES) and Elliptic-curve cryptography (ECC) are also described in Chapter 3.

MQTT protocol implementation using Contiki OS is completed with a real IoT hardware platform Zolertia RE-Mote device. Other tools 6LowPAN border router 6LBR and the temperature and humidity sensor are used to deploy the MQTT communication. UDP protocol communication and secured UDP communication using AES-128 encryption standard established within this work. Along with the AES symmetric encryption method implementations, ECC public key algorithm process is discussed in Chapter 4. During the ECC implementation in section 4.3.7 we

noticed it takes time to generate the shared private key for devices. Since ECC generates smaller keys but provides the same level of cryptographic strength which is suitable for the limited processing power and limited memory devices, it is becoming the common security solution for IoT.

In this work, we could not cover the implementation of CoAP protocol and the DTLS security protocol in the Zolertia RE-Mote devices. It was not possible to establish the communication between devices using ECC encryption method, only the key generation process included in the thesis. In the future, we intend to complete the communication between devices using public key encryption standard ECC and to implement CoAP protocol including the DTLS security features.

5.2 Problems encountered

During the implementation phase of IoT protocols, some difficulties were encountered. CETIC 6LBR configuration was not properly done as per the given instructions in the Zolertia documentary. Some additional sensors like barometer sensor, temperature sensor and gas sensor were not working with the specific programs that are available in Contiki. In addition, DTLS security protocols could not be examined due to the bugs in the programs. Finally, there were a lot of bugs which prevented us from using the latest commit of Contiki and must be fixed. Due to the community-driven and open source nature of Contiki it is expected to be fixed as because everyone can contribute to it. For the MQTT implementation, a significant amount of time was spent to deploy the MQTT in real hardware devices and to connect the devices with the broker. After making some changes in 6LBR configurations it was possible to get the result. This problem could be solved easily if the 6LBR had complete documentation with the description of all supported modes of operations and how it must be configured in which situation.

REFERENCES

[1] "searchsecurity.techtarget.com," [Online]. Available: https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard.

[2] "searchsecurity.techtarget.com," [Online]. Available: https://searchsecurity.techtarget.com/definition/elliptical-curve-cryptography.

[3] S. N. Swamy, P. D. Jadhav and P. N. Kulkarni, "Security Threats in the Application layer in IOT Applications," *IEEE,* 2017.

[4] S. Haller, S. Karnouskos and C. Schroth, "The Internet of Things in an Enterprise Context," *FIS 2008,* vol. LNCS 5468, pp. 14-28, 2009.

[5] I. o. T. A. a. Glance, "www.cisco.com," [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf.

[6] A. Michalas and N. Komninos, " The Lord of the Sense: A Privacy Preserving Reputation System for Participatory Sensing Applications," in *Proceedings of the 19th IEEE International Conference on Communications (ISCC'14)*, Madeira, Portugal, 2014.

[7] "www.ietf.org," [Online]. Available: https://www.ietf.org/about/. [Accessed 2019].

[8] I. Ishaq, "IETF Standardization in the Field of the Internet of Things (IoT): A Survey," *Journal of Sensor and Actuator Networks,* 2013.

[9] "www.semanticscholar.org," [Online]. Available: https://www.semanticscholar.org/paper/Towards-a-definition-of-the-Internet-of-Things-(-)/e4ce63ff3e0120f63258e19c48b65feb4b3dd0a6/figure/28.

[10] R. A. Rahman and B. Shah, "Security analysis of IoT protocols: A focus in CoAP," *IEEE,* 2016.

[11] "mqtt.org," [Online].

[12] J. Han, M. Ha and D. Kim, "Practical security analysis for the constrained node networks: Focusing on the DTLS protocol," *IEEE,* 17 December, 2015.

[13] J. Vishwesh and M. B. Rajashekar, "Internet of Things (IoT): Security Analysis & Security Protocol CoAP," *International Journal of Recent Trends in Engineering & Research (IJRTER),* vol. 03, no. 03, March, 2017.

[14] "802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks," *IEEE,* 22 April 2016.

[15] "www.link-labs.com," November 2015. [Online]. Available: https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy.

[16] K. Y. Yigzaw, A. Michalas and J. G. Bellika, "Secure and Scalable Deduplication of Horizontally Partitioned Health Data for Privacy-Preserving Distributed Statistical Computation," *Journal of Medical Informatics and Decision Making (BMC),* 2017.

[17] Y. Y. Kassaye, A. Michalas and G. B. Johan, "Secure and scalable statistical computation of questionnaire data in R," *IEEE,* 2016.

[18] A. Michalas and N. Weingarten, "HealthShare: Using Attribute-Based Encryption for Secure Data Sharing Between Multiple Clouds," in *Proceedings of the 30th IEEE International Symposium on Computer-Based Medical Systems (CBMS'17)*, Thessaloniki, Greece, 2017.

[19] A. Michalas, N. Paladi and C. Gehrmann, "Security Aspects of e-Health Systems Migration to the Cloud," in *Proceedings of the 16th IEEE International Conference on E-health Networking, Application & Services (Healthcom)*, Natal, Brazil, October 15 - 18, 2014.

[20] "www.design-reuse.com," December 2009. [Online]. Available: https://www.design-reuse.com/news/22313/bluetooth-low-energy-wireless.html.

[21] "www.iec.ch," [Online]. Available: https://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf.

[22] M. Kocakulak and I. Butun, "An overview of Wireless Sensor Networks Towards Internet of Things," *IEEE,* 2017.

[23] "www.elprocus.com," [Online]. Available: https://www.elprocus.com/introduction-to-wireless-sensor-networks-types-and-applications/.

[24] C. A. Liñán, A. Vives, M. Zennaro, A. Bagula and E. Pietrosemoli, "Internet of Things In 5 Days".

[25] "resources.infosecinstitute.com," updated 2018. [Online]. Available: https://resources.infosecinstitute.com/ipv6-security-overview-a-small-view-of-the-future/#gref.

[26] C. Sharma and D. N. K. Gondhi, "Communication Protocol Stack for Constrained IoT Systems," *IEEE,* 2018.

[27] T. Paul and G. S. Kumar, "Safe Contiki OS: Type and Memory Safety for Contiki OS," *IEEE,* 2009.

[28] "www.techopedia.com," [Online]. Available: https://www.techopedia.com/definition/5773/transmission-control-protocol-tcp.

[29] T.Gopinath, A.S.Rathan Kumar, Rinki Sharma, "Performance Evaluation of TCP and UDP over Wireless Ad-hoc Networks with Varying Traffic Loads," *IEEE,* 2013.

[30] "www.howtogeek.com," [Online]. Available: https://www.howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp/.

[31] "www.diffen.com," [Online]. Available: https://www.diffen.com/difference/TCP_vs_UDP.

[32] T. Yokotani and Y. Sasaki, "Comprison with HTTP and MQTT on required Network Resources for IoT," *IEEE,* 2016.

[33] "www8.hp.com," 2014. [Online]. Available: https://www8.hp.com/us/en/hp-news/press-release.html?id=1744676.

[34] P. Kumar, R. S. Kunwar and A. Sachan3, "A Survey Report on: Security & Challenges in Internet of Things," in *IJSRD/Conf/NCICT /2016/10*, Ahmedabad, January 2016 .

[35] A. Pal, "www.cso.com.au," [Online]. Available: https://www.cso.com.au/article/575407/internet-things-iot-threats-countermeasures/.

[36] S. K. K, S. Sahoo, A. Mahapatra, A. K. Swain and K. Mahapatra, "Security Enhancements to System on Chip Devices for IoT Perception Layer," *IEEE,* 2017.

[37] A. Michalas and R. Murray, "Keep Pies Away from Kids: A Raspberry Pi Attacking Tool," in *Proceedings of the 1st ACM CCS International Workshop on Internet of Things Security and Privacy (IoT S&P'17) in Conjunction with ACM CCS*, Dallas, USA, 2017.

[38] A. Michalas and T. Dimitriou, "Multi-Party Trust Computation in Decentralized Environments in the Presence of Malicious Adversaries," *Ad Hoc Networks Journal, a special issue on "Smart Solutions for Mobility Supported Distributed and Embedded Systems,* 2014.

[39] A. Michalas and T. Dimitriou, "Multi-Party Trust Computation in Decentralized Environments," in *Proceedings of the 5th IFIP International Conference on New Technologies, Mobility & Security (NTMS'12)*, Istanbul, Turkey, 2012.

[40] A. Michalas, A. O. Vladimir, K. Nikos and R. P. Neeli, "Privacypreserving Trust Establishment scheme for Mobile Ad Hoc Networks," in *Proceedings of the 16th IEEE International Conference on Communications (ISCC'11)*, Corfu, Greece, 2011.

[41] A. Michalas, N. Komnino, N. R. Prasad and V. A. Oleshchuk, "New Client Puzzle Approach for DoS Resistance in Ad hoc Networks," in *Proceedings of the IEEE International Conference on Information Theory and Information Security (ICITIS'10)*, Beijing, China, 2010.

[42] A. Michalas, N. Komninos and N. R. Prasad, "Multiplayer Game for DDoS Attacks Resilience in Ad hoc Networks," in *Proceedings of the 2nd IEEE International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless Vitae 2011)*, Chennai, India, 2011.

[43] A. Michalas, N. Komninos and N. R. Prasad, "Mitigate DoS and DDoS attack in Ad Hoc Networks," *International Journal of Digital Crime and Forensics,* 2011.

[44] N. Paladi, C. Gehrmann and A. Michalas, "Providing End-User Security Guarantees in Public Infrastructure Clouds," *IEEE Transactions on Cloud Computing, a special issue on "Cloud Security Engineering",* 2016.

[45] Y. Verginadis, A. Michala, P. Gouvas, G. Schiefer, G. Hubsch and I. Paraskakis, "PaaSword: A Holistic Data Privacy and Security by Design Framework for Cloud Services," *Journal of Grid Computing, a special issue on "Cloud Computing and Services Science,* 2017.

[46] A. Michalas, "The Lord of the Shares: Combining Attribute-Based Encryption and Searchable Encryption for Flexible Data Sharing," in *Proceedings of the 34th ACM/SIGAPP Symposium On Applied Computing (SAC)*, Limassol, Cyprus, 2019.

[47] A. Michalas, "Sharing in the Rain: Secure and Efficient Data Sharing for the Cloud," in *Proceedings of the 11th IEEE International Conference for Internet Technology and Secured Transactions (ICITST'16)*, Barcelona, Spain, 2016.

[48] N. Paladi, A. Michalas and C. Gehrmann, "Domain Based Storage Protection with Secure Access Control for the Cloud," in *The 2014 International Workshop on Security in Cloud Computing, held in conjunction with the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Kyoto, Japan, 2014.

[49] M. M. Hossain, M. Fotouhi and R. Hasan, "Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things," *IEEE,* 2015.

[50] S. Rizvi, A. Kurtz, J. Pfeffer and M. Rizvi, "Securing the Internet of Things (IoT): A Security Taxonomy for IoT," *IEEE,* 2018.

[51] M. Burhan, R. A. Rehman, B. Khan and B.-S. Kim, "IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey," no. Sensors 2018, 18(9), 2796; https://doi.org/10.3390/s18092796.

[52] S. Deshmukh and S. S. Sonavane, "Security protocols for Internet of Things: A survey," *IEEE,* 2017.

[53] J. Granjal, E. Monteiro and J. S. Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," *IEEE,* vol. 17 , no. 3, pp. 1294 - 1312, 2015.

[54] D. Dragomir, L. Gheorghe, S. Costea and A. Radovici, "A Survey on Secure Communication Protocols for IoT Systems," *IEEE,* 2016.

[55] "www.researchgate.net," [Online]. Available: https://www.researchgate.net/figure/Outlines-the-principal-characteristics-of-the-RPL-protocol_tbl1_327868774. [Accessed 10 5 2019].

[56] G. Ma, X. Li, Q. Pei and Z. Li, "A Security Routing Protocol for Internet of Things Based on RPL," *IEEE,* 2018.

[57] Z. Shelby, K. Hartk, C. Bormann and B. Frank, "Constrained Application Protocol (CoAP)," [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-coap-13.

[58] B. C. Villaverde, D. Pesch, R. D. P. Alberola, S. Fedor and M. Boubekeur, "Constrained Application Protocol for Low Power Embedded Networks: A Survey," *IEEE,* 2012.

[59] J.-Y. Huang, P.-H. Tsai and I.-E. Liao, "Implementing publish/subscribe pattern for CoAP in fog computing environment," *IEEE,* 2017.

[60] D. Ugrenovic and G. Gardasevic, "CoAP protocol for Web-based monitoring in IoT healthcare applications," *IEEE,* 2016.

[61] "developer.artik.cloud," [Online]. Available: https://developer.artik.cloud/documentation/data-management/coap.html.

[62] S. Andy, B. Rahardjo and B. Hanindhito, "Attack scenarios and security analysis of MQTT communication protocol in IoT system," *IEEE,* 25 December 2017.

[63] "docs.oasis-open.org," [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.

[64] "www.steves-internet-guide.com," 8 November 2017. [Online]. Available: http://www.steves-internet-guide.com/mqtt-security-mechanisms/.

[65] [Online]. Available: https://iotbyhvm.ooo/coap-vs-mqtt/.

[66] G. L. d. Santos, V. T. Guimarães, G. d. C. Rodrigues, L. Z. Granville and L. M. R. Tarouco, "A DTLS-based Security Architecture for the Internet of Things," *IEEE,* 2015.

[67] A. Haroon, S. Akram, M. A. Shah and A. Wahid, "E-Lithe: A Lightweight Secure DTLS for IoT," *IEEE,* 2017.

[68] "www.ibm.com," [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.14/gtps7/s7symm.html. [Accessed 14 05 2019].

[69] D. Khambra and P. Dabas, "Secure Data Transmission using AES in IoT," *International Journal of Application or Innovation in Engineering & Management (IJAIEM),* vol. 6, no. 6, June 2017.

[70] "searchsecurity.techtarget.com," [Online]. Available: https://searchsecurity.techtarget.com/definition/elliptical-curve-cryptography.

[71] "www.ssl2buy.com," [Online]. Available: https://www.ssl2buy.com/wiki/ecc-algorithm-to-enhance-security-with-better-key-strength. [Accessed 15 5 2019].

[72] "dzone.com," [Online]. Available: https://dzone.com/articles/how-does-elliptic-curve-cryptography-work.

[73] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen and S. Shieh, "IoT Security: Ongoing Challenges and Research Opportunities," *IEEE,* 08 December 2014.

[74] M. M. Hossain, M. Fotouhi and R. Hasan, "Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things," *IEEE,* 17 August 2015.

[75] "www.zolertia.io," [Online]. Available: https://zolertia.io/. [Accessed 25 04 2019].

[76] [Online]. Available: https://www.startupbootcamp.org/startups/zolertia/.

[77] [Online]. Available: https://github.com/Zolertia.

[78] "http://www.contiki-os.org/," [Online].

[79] "www.riot-os.org," [Online]. [Accessed 26 04 2019].

[80] [Online]. Available: https://github.com/cetic/6lbr.

[81] [Online]. Available: https://github.com/cetic/6lbr/wiki/6LBR-in-a-Nutshell.

[82] "www.seeedstudio.com," [Online]. Available: https://www.seeedstudio.com/Grove-Digital-Light-Sensor.html. [Accessed 7 5 2019].

[83] [Online]. Available: https://github.com/Zolertia/Resources/wiki/RE-Mote.

[84] "github.com," [Online]. Available: https://github.com/contiki-os/contiki/blob/master/examples/zolertia/zoul/test-tsl256x.c.

[85] B. H. Çorak, F. Y. Okay, M. G. Ş. Murt2 and S. Ozdemir, "Comparative Analysis of IoT Communication Protocols," *IEEE,* 2018.

[86] J. Granjal, E. Monteiro and J. S. Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," *IEEE,* vol. 17, no. 03, pp. 1294 - 1312, January, 2015.

[87] A. Capossele, V. Cervo, G. D. Cicco and C. Petrioli, "Security as a CoAP resource: An optimized DTLS implementation for the IoT," *IEEE,* 2015.

[88] FirasAlbalas, M. Al-Soud, O. Almomani and A. Almomani, "Secure and Energy-effictive CoAP Application Layer Protocol for the Internet of Things," in *The International Arab Conference on Information Technology*, Yassmine Hammamet, Tunisia, December 22-24, 2017.

[89] M. Brachmann, O. Garcia-Morchon and M. Kirsche, "Security for Practical CoAP Applications: Issues and Solution Approaches," in *10th GI/ITG KuVS Fachgesprch Sensornetze (FGSN)*, 2011.

[90] "www.researchgate.net," [Online]. Available: https://www.researchgate.net/figure/IP-and-6LoWPAN-protocol-stack-in-reference-to-layers-of-the-TCP-IP-networking-model_fig1_281333084. [Accessed 10 5 2019].

[91] "www.tutorialspoint.com," [Online]. Available: https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm. [Accessed 15 05 2019].

[92] I. Setiadi, A. I. Kistijantoro and A. Miyaji, "Elliptic curve cryptography: Algorithms and implementation analysis over coordinate systems," *IEEE,* 23 November 2015.

[93] "www.ibm.com," [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.lxci/lxci_ecc_apis.html. [Accessed 16 05 2019].