Message Queuing Telemetry Transport – Secure Connection: A power-efficient secure communication

Abstract: The Internet of Things (IoT) consists of numerous connected devices that exchange vast amounts of data. The exchange is either discrete or continuous and is governed by different protocol implementations at the application layer. Message Queuing Telemetry Transport (MQTT) is a widely used protocol for message and data exchange. All such protocols are vulnerable to malicious attacks, which raises the integrity and trust issues of the sources and the received data. Trust needs to be established that ensures successful communication. The key to this trust lies in security advancements and enhancements in all IoT connectivity, implying that IoT must exhibit reliable, secure, and private behavior while keeping in check their power consumption. This paper highlights the security considerations for the MQTT protocol and leverages its four enhanced versions for a scenario-based setup. After applying the security measures in the base MQTT protocol, we provide a power consumption-based performance comparison, focusing on enhanced security for resource-constrained devices.

Keywords: IoT; internet of things; networks; MQTT; message queuing telemetry transport; security; performance management; communication protocols; power consumption; network monitoring and measurements.

1 Introduction

IoT and Cloud are emerging internet technologies that enable non-trivial IoT deployments Dizdarević et al. (2019). Legacy IT systems have advanced to constitute the future IoT systems Bojanova and Voas (2017) Stankovic (2014). The speed at which the data is exchanged by the autonomous objects within a network majorly drives the concept of IoT.

A constant threat of getting compromised because of numerous attacks and intrusions looms over the IoT networks. An end-to-end secure IoT solution development fuses essential security architecture features across multiple levels. Software-Defined Networking (SDN) is one of the concepts that manages these multiple levels to flexibly and efficiently protect the network. The SDN provides a dynamic, programmable, adaptable, and secure network Bhardwaj and Harit (2022) Raikar et al. (2020) and does not require any changes to the existing underlying IoT architecture, all this while being responsible for the network management orchestration (open and user-controlled access to communication hardware).

The application layer defines the extent of user interaction and consists of communication protocols, such as Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), Extensible Messaging and Presence Protocol (XMPP), MQTT, REST, and so many others Yassein et al. (2016) Mishra and Kertesz (2020), that enable different devices to interact over wired or wireless media. Wireless media has a broader range of applications for modern-day communication and hence becomes the utmost priority for security applications Burange et al. (2019). MQTT is the most widely used protocol for IoT communication systems because of its favorable publish/subscribe model.

The initial development of MQTT considers that a message exchange protocol must be bandwidth-efficient and uses as little power as possible. In 1999, "Andy Stanford-Clark (IBM) and Arlen Nipper (Eurotech; now Cirrus Link)" Team (2019), working on a project for monitoring of an oil pipeline, came up with this protocol keeping in mind that the devices were connected via satellite link, thus making it a costly connection for that time of the year.

MQTT has a publish/subscribe architecture Mishra and Kertesz (2020), enabling the broker to push messages Yassein et al. (2017) to the clients by being event-driven. The MQTT broker acts as a central communication point i.e.; it dispatches all messages among the senders and the registered receivers. Each client selects a topic, includes it in the message, and then publishes it to the broker. A topic combines simple strings with zero or more hierarchy levels. Hierarchies are represented with the help of a separator like a slash /. An example topic can be given as $home/guest\_room/temperature$, which sends the temperature data of the guest room. The broker may buffer Luzuriaga et al. (2016) the messages, referred to as a non-clean session, if no clients are connected or have subscribed to the topic. Thus, to have successful communication, the clients only require the identity the topic(s) of communication and subscribe to it. The other node´s identity remains the concern of the broker. MQTT supports several levels of Quality of Service (QoS) to ensure that all of the data published is received by using level QoS0 (at most once), QoS1 (at least once), and QoS2 (exactly once). The higher the QoS level, the higher complexity of the process Deschambault et al. (2017). Also, MQTT clients regularly send a *keepalive* message (usually 60 seconds) that notifies the broker about an actively connected client Steve (2017). Thus, maintaining the connected state even if they are not sending data. Such an architecture allows highly scalable data distribution mechanisms by removing the overhead of identity.

A web browser is a good and de-facto interface to display the information projected by MQTT clients/brokers. For this, WebSockets play a significant role with the help of a Javascript client. Modern MQTT implementations could use Nodejs, Reactjs, Tableau, or similar technologies for the logged data visualization, including the communication flow.

MQTT is the most widely used protocol for IoT communication systems because of its favorable publish/subscribe model. MQTT differs from HTTP as the latter uses a request/response model. Thus, for any update to the client information, the broker has to push a message, which in the case of HTTP, is a pull request from the client. For this research, MQTT is in focus.

1.1 Motivation

Establishing secure and power-efficient communication becomes challenging due to the high energy consumed by network nodes. A node's power consumption results from its components' cumulative power consumption contributions, which depend on the component's operation and state. The message exchange within the network and among the nodes explicitly determines the state and actions of a node (i.e. the radio operation mode, as in whether the node is in receive, transmit, or sleep mode) and the micro-controller operation mode. The network design and communication protocols influence the functioning of the nodes. Finally, an operating system provides a platform for application development, including a communication stack that implements network protocols. The count of messages injected within the network depends on the application requirements. In this research, the Average power consumption is calculated with reference to the node states.

1.2 Contribution

The research contributions of this paper are listed below:

1. We leveraged different encryption mechanisms to the basic MQTT protocol for secure communication among resource-constrained devices.
2. We evaluated the Average Power Consumption and Average Radio Duty Cycle of the discussed security implementations.
3. Further, we provide an average power consumption-based comparative analysis.

1.3 Organization

The organization of the paper is as follows: Section 2 presents the related work, highlights their major contributions, and their shortcomings. Section 3 describes the encryption mechanisms and the critical implementation parameters. Section 4 discusses the experimental setup, performance evaluation parameters, results obtained for all the implemented security techniques and their comparison. Section 5 concludes the paper, and section 6 opens up the scope for some future applications.

2 Related Work

Section 2.1 highlights the security considerations for the MQTT protocol and section 2.2 presents the existing security implementations in MQTT.

2.1 MQTT Protocol Security Considerations

As MQTT protocol is explicitly designed for devices with low processing power, it tries to cut down on factors that might affect power consumption, making way for serious security problems. The research on security considerations of MQTT protocol primarily revolves around authentication and authorization, as authentic and authorized users are secure enough to rope in integrity and confidentiality.

- **Authentication**: Authentication is not an inherent property of MQTT protocol Patel and Doshi (2020) but can be achieved easily by tweaking some fields in the messages. Un-authenticated usage can spoof any communicating participant's identity or distribution of unauthorized data. Broadly, to verify the identity of an MQTT client, the following three procedures are available in handy:

  1. Client ids: While configuring an MQTT client, assigning a unique name/id to the client is necessary. A non-unique naming convention results in a connection drop for existing connections, as dual-connection results in inconsistency. For each client's subscription to the topic(s), its ID gets linked to the topic in the network through the TCP connection. Also, a persistent connection ensures that the broker does not forget the client ID.
  2. Usernames and passwords: To establish a connection between a client and an MQTT broker, the protocol supports username and password fields in the CONNECT message at the client's end Hernández Ramos et al. (2018). Still, there remains the requirement for transport layer protection to transmit username/password combination as this exchange is not secure and takes place in the form of clear text. Steve (2017) This combination can also restrict access to the topics.
  3. Client Certificates: Here, the broker needs to maintain certificates for all the clients. While maintaining a small network that needs a higher level of security, this method is manageable, but as this network grows, the process becomes quite cumbersome.

- **Authorization**: An authentic client can publish topics and subscribe to any number of topics without being authorized to achieve the functions, which may pose a significant issue because if an authentic node (client) turns malicious, the attacker can flood the network with unauthorized activities. The protocol itself does not provide any authorization mechanism and, therefore, becomes the broker's responsibility, which is a straightforward implementation with permission validation on the topic(s).

- **Confidentiality**: MQTT depends on TCP as a transport protocol, implying that the communication is unencrypted by default. Therefore, any attacker listening in on the same network can spy on the data packets. Most MQTT brokers use TLS instead of plain TCP to encrypt the communication to mitigate this.
- **Integrity**: Malicious MQTT clients Dinculeană and Cheng (2019), when MQTT systems have untrusted or unidentified clients, can get access to the MQTT broker and topics. TLS/SSL Security and Payload Encryption techniques are in use to protect the contents of an MQTT message. The data integrity of sent messages must be verified, primarily when TLS is not used. MQTT supports Checksum, MAC, and Digital Signatures to provide integrity to exchanged packets Hernández Ramos et al. (2018) Ning et al. (2013).

2.2 *Existing Security Implementations in MQTT*

A framework development requires an architecture for its description and working. For this subsection, the discussion sheds light on some of the existing and deployed considerations for a secure MQTT Protocol.

2.2.1 *MQTT for Sensor Networks (MQTT-SN)*

MQTT-SN Steve (2017) uses UDP, instead of TCP (i.e., trade-off from a connection-oriented protocol to a connection-less protocol), for its working. Some operational characteristics are given below:

- **QoS Levels:** In addition to the current QoS levels of MQTT, it also supports a distinctive publish QoS of 3 or -1. The QoS flag is set as *binary* 11. The topic ID or short topic can be published. Although there is a virtual connection requirement, publishing messages with this QoS does not require an initial connection.
- **Working:** The gateway broker advertises message packets on a multicast address containing the advertising Gateway name, duration, and Gateway number. The client must listen to the multicast address or search for a gateway by sending a research packet to a multicast address. This search gets acknowledged by either another client or a Gateway. The client maintains a list of active gateways and may use the duration to identify an active gateway Steve (2017). A downside of this working criterion is channel flooding, which eventually results in network flooding unless mitigation mechanisms are employed to forestall broadcast storms and are covered within the specification.

2.2.2 *Secure-MQTT (SMQTT)*

Secure MQTT (SMQTT) uses lightweight attribute-based encryption Li et al. (2014) Bisne and Parmar (2017). The main advantage is the broadcast encryption feature, i.e., all nodes receive only encrypted messages. The algorithm consists of four main stages: setup, encryption, publish, and decryption Singh et al. (2015). Publishers and subscribers register themselves with the MQTT broker during the setup phase. The broker then assigns a secret key by employing a key generation algorithm. The publisher then encrypts the data and publishes it to the broker, who sends it to the subscribers. Subscribers decrypt it using the secret key.

While this implementation helps mitigate the security issues, the frequent channel access and message overhead drop their utility for resource-constrained devices as these processes are computationally intensive and drain the battery much faster.

2.2.3 *Direct Multicast-MQTT (DM-MQTT)*

For standard MQTT, the broker with a large number of connected IoT devices performs massive data exchange leading to traffic congestion and decreasing the edge network's capability. The broker, in turn, faces long transfer delays. MQTT with a multicast mechanism establishes the bidirectional SDN multicast trees between the publishers and the subscribers when multiple subscribers subscribe to the same topic by means of bypassing the centralized broker, hence reducing the transfer delays Park et al. (2018).

3 Implementation

IoT has trust-related issues that could be easily addressed if an all-agreed-to standardization existed. The focus of this study remains on the security of resource-constrained devices and their power consumption constraints. The implemented method tries to comprehend the industry towards a more standard security definition while checking that the solution does not favor any particular vendor or industry.

This section sheds light on the application of security techniques to secure the connection and the payload. For a brief idea, less resource-intensive AES Su et al. (2019) Sawataishi et al. (2020) and its variants are used to secure the protocol.

3.1 *Payload Encryption using barebone AES*

This implementation encrypts the payload of an MQTT packet using client-to-client (end-to-end) encryption and ensures the security of the transferred messages using AES. Typical key size of a 128-bit built-in AES implementation of Contiki OS encrypts messages of one-block-size length (16 bytes). Though, messages of smaller lengths must be manually padded. Larger payload sizes can also be encrypted using multiple encryption/decryption of consecutive 16 bytes in the ECB mode of operation.

3.2 *Payload Encryption using AES-CBC*

For messages with a length other than one block, the AES-CBC Tan et al. (2018) implementation from ContikiSec

Casado and Tsigas (2009) acts as an add-on extension to barebone AES. Although there is no defined limit on the message length to be encrypted because of RAM size limitations, the restriction limits to a size of four blocks, i.e., 64 bytes. This size is mutable.

3.3 AES-OCB Payload Authentication and Encryption

As the name suggests, this spin implements authentication, an essential property for payload encryption. AES-OCB Rogaway et al. (2003) Jang et al. (2004) Bogdanov et al. (2014) authenticated encryption mode ensures the integrity of the transferred messages. ContikiSec also implements AES-OCB Casado and Tsigas (2009). A tag appended with the message is transferred, validating this encrypted message on the receiving MQTT client. Because of RAM restrictions in Contiki, the message length was restricted to two block sizes, i.e., 32 bytes. Also, the size restrictions on the compiled program defer from incorporating the encryption and the decryption functions on a single mote. Thus, a mote publishes the messages in encrypted form while another mote receives and decrypts them.

3.4 CCM Encrypted Link-layer

Counter with CBC-MAC (CCM) Jonsson (2003) secure MQTT client implements Link-layer encryption (link-layer security –LLSec) through AES-CCM-128, which ensures intra-node security of all the sent packets (client id, topic, and others). Yet, this implementation has drawbacks:

- A group key allows access to the data, which should not be accessible by all the nodes, thus negating the root concept of end-to-end encryption.
- As the transmitted packet is encrypted, the receiving node must first decrypt the packet to identify the topic, and if the topic is not subscribed, the message must be dropped. This process is a significant overhead that consumes extra resources.

A workaround to provide confidentiality and authenticity could be using LLSec combined with the simple username/password authentication mechanism of MQTT.

4 Results and Discussion

4.1 Experimental Setup

Fig. 1 establishes a basic working model of the setup maintained with Raspberry Pi derivatives to make communication possible. The MQTT broker is a central entity that enables communication among all the MicroController Units (MCUs) with or without sensors.

Note that there is no specific description of the MQTT broker, as this is the sole entity that can either be incorporated as a separate entity or as an integration with any of the MCUs. Table 1 gives a brief description of all the tools in use for the implementation.

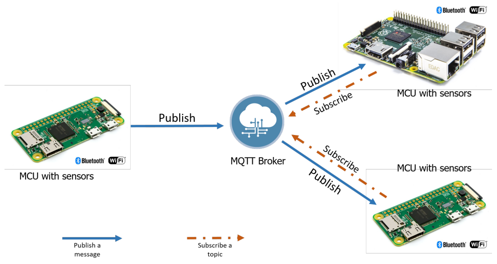

Figure 1 Publish subscribe model setup of MQTT using Raspberry Pi 3 Model B+ and Raspberry Pi Zero W

Table 1 Tools used in implementation phase

Tool	Version	Comment
Contiki OS	3.0	Stable Release: 2015-08-25
6lbr	CETIC 6LBR 1.5.0	Stable Release: December 2017
MOSQUITTO	1.5.0	update of 2018-05-02
MQTT Client	No version	In-built to Contiki Cooja
Development OS	Ubuntu 14.04-LTS	A virtual machine for Contiki Cooja
Special Hardware	Raspberry Pi 3 Model B+ and Raspberry Pi Zero W	N.A.

Fig. 2 represents a publishing cycle setup with payload encryption and Link Layer encryption. It should be noted that the MQTT broker could be any entity, i.e., from a System-on-Chip (SoC) like Raspberry Pi or BeagleBone, to a full-fledged NAS server.

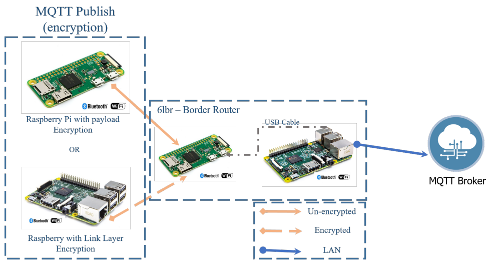
MQTT Publish (encryption)
Raspberry Pi with payload Encryption
OR
Raspberry with Link Layer Encryption
6lbr – Border Router
USB Cable
MQTT Broker
Un-encrypted
Encrypted
LAN

Figure 2 Publish setup of MQTT using listed hardware

- For a publish cycle with payload encryption, the message at the publisher's end is encrypted using either a single block AES, AES-CBC, or AES-OCB authenticated block.
- For a publish cycle with Link Layer encryption, packet encryption is done with the help of LLSec. At the same time, the authentication procedure is handled with a simple username and password.
- The MQTT broker saves the messages, for *Retain* = True, in the form of ciphertext when payload

encryption is in use and in the form of plaintext when LLSec is in use.

For all the scenarios listed above, the readings for the power consumption are obtained at the 6lbr-Border Router, which acts as a mediator between sensors and the MQTT server.

Fig. 3 represents a subscription cycle setup with payload decryption and Link Layer decryption.

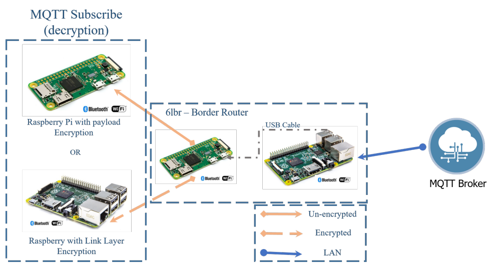
MQTT Subscribe (decryption)
Raspberry Pi with payload Encryption
OR
Raspberry with Link Layer Encryption
6lbr – Border Router
USB Cable
MQTT Broker
Un-encrypted
Encrypted
LAN

Figure 3 Subscriber setup of MQTT using listed hardware

- For a subscribe cycle, the receiver gets a payload encrypted message and decrypts it using the same technique.
- For a subscribe cycle with Link Layer encryption, packets are already encrypted with the help of LLSec. Thus there is no need for payload decryption, and the authentication procedure is basic i.e. username and password.
- The MQTT broker saves the messages, for *Retain* = True, in the form of ciphertext when payload encryption is in use and in the form of plaintext when LLSec is in use.

4.2 Performance criteria

To describe the configuration, the implementation uses two mote configurations. One acts as a publisher, and the other acts as a subscriber. For sustained loads to evaluate the performance and to ensure CPU cycle bursts, there is an exchange of 50 MQTT messages. In this research, Power consumption is the criteria for performance evaluation.

The study leverages the described functions to obtain all the parameters required for the estimation of power consumption.

- **Energest**, a built-in software tool of Contiki measures the CPU timer ticks for different power states (low mode (LPM) and high mode for CPU, transmission (Tx) and receiving (Rx) for motes) using wraparound macros.
- **Powertrace**, another built-in software tool of Contiki measures the power consumption of the mote, and the broker Katsikeas et al. (2017). A call to powertrace() prints a serial output with the collected data.

Power consumption depends on the calculation of the energy consumption and the radio duty cycle, which involves the use of two formulas:

$$Energy\ [mW] = \frac{energ\_val \times current\ [mA] \times volt\ [V]}{cpu\_tick\_sec \times time\_dev} \qquad (1)$$

Where, Energest periodically prints the *energ_val*, while *cpu_tick_sec* refers to the number of clock cycles for the CPU intrinsic timer (per second, obtained from another function called RTIMER), i.e. 32768 for this setup, and *time_dev* refers to the time deviation (in seconds) from the last recorded Energest values.

$$Radio\ Duty\ Cycle\ [\%] = \frac{energ\_Tx + energ\_Rx}{energ\_CPU + energ\_Lpm} \times 100 \qquad (2)$$

Where *energ_∗* is the value that corresponds for each call to the Energest function.

The above calculations depend on precise time interval analysis of publish and subscribe cycles. However, the communication channel introduces a certain time difference. The message latency Kenitar et al. (2018) refers to this time difference between a message publish, which includes the encryption (if done) and its processing on the receiving end (includes decryption, if required). This measurement technique is different for each test environment. For a simulated test bed, the calculation of latency in message response is easy and accurate because of the presence of timestamps on the simulator between two serial debug prints: first, after a successful publish by the publisher; second, after receiving and decryption (if encrypted) at the client´s end.

The message latency measurement for the real-world testing environment is difficult because of the absence of synchronized timestamps. Thus, to measure the message latency in such an environment, a slight workaround is possible, as depicted in Fig. 4:

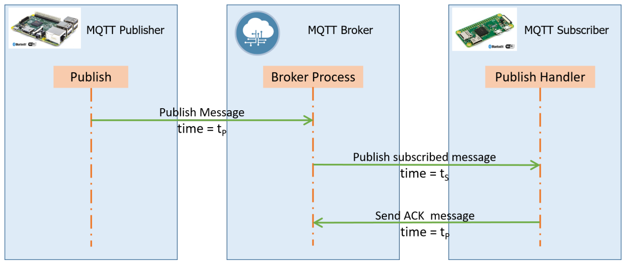

Figure 4 Message Latency measurement using ACK

The described strategy has a slight modification in that, in place of a debug serial message/value output from the subscriber, an acknowledgment (ACK) message is embodied into the message structure. This method corresponds to the fact that after the subscriber gets the message and unscrambles it (on the possible off-chance that it was scrambled), for that instance of time, it plays out a publish message on some topic (*clients/ack*, i.e., acknowledgment, for instance). Within the event where we accept that each publish (after scrambling/encoding the message) requires some energy e_P and every subscribed publish requires some energy e_S, one can promptly observe that the message inactivity is set by playing out the choice of e_P and e_S and afterward, including the past total, the time required for the encrypting the publishing bit. The combination of e_P and e_S could, without much of a stretch, be determined from the time distinction on the timestamps of the messages that associate with the respective node, while the time consumption to encode the publishing bit may be determined with the employment of an intrinsic run-time clock.

For the greatest conceivable precision, an MQTT broker was running on an analogous machine (like on another Raspberry Pi) alongside an MQTT client, which is subscribed to all the described messages, and afterward, the time distinction of their timestamps was calculated. The size of the running program routine cannot be sidelined, as it contributes to the power draw for storing it in the memory modules of the hardware in use. The scale of the aggregated program is a handy calculation, with the employment of the order of dimension that provides data about the occupancy of distinct segments of the ROM (called *text* and *data*) and the RAM (called *bss*).

A Raspberry Pi has an operating frequency of more than 1GHz, but there is no statement on the usage of the current draw at 16MHz. But it is a required parameter as Contiki runs at 8MHz. The datasheet of Raspberry Pi gives the current consumption per 1MHz, which eases the overall current calculation using the formula:

$$current, I(amp) = I(amp/1Mhz) \times freq(system)\ [1Mhz] \quad (3)$$

The rated typical current for a program run at 1MHz with flash memory and a voltage of 3V is 515 µA. Thus, from these values, the total rated current calculates to 4.12 mA for operation at 8MHz. Also, for the same operation frequency, the standard maximum rated current calculates to 4.51 mA. So, for ease of calculating the power consumption, an average of these values, i.e., 4.31 mA, runs throughout the observations.

4.3 Results

In this section, we evaluate the performance of the discussed security techniques in terms of the *Average Power Consumption* (APC), which is measured in *mW*, the *Average Radio Duty Cycle* (%) (ARDC), and the *Program Size* which is measured in *bytes*.

4.3.1 Observations for Base MQTT

Considering a payload length/size of 32 bytes, the simple/base MQTT client/ broker/ publisher setup gives the following results: Table 2 gives the sum of the average energy consumed in every cycle and is presented under Total. Table 3 gives the averaged sum percentage of energy consumed for each radio duty cycle. Table 4 gives the division of memory usage in terms of volatile memory (RAM) and persistent memory (ROM) for each execution cycle for both broker and client. This classification is true for all the observations.

Table 3 ARDC

	Tx	Rx	Total
Broker	0.26%	1.45%	1.71%
Client	0.29%	1.23	1.52%

Table 2 APC

	CPU	LPM	Tx	Rx	Total
Broker	0.241457	0.00125	0.093812	0.651287	0.987806
Client	0.238451	0.001316	0.082165	0.643145	0.965077

Table 4 Program Size

	ROM		RAM
	test	data	bss
Broker	51531	306	6584
Client	52632	306	6531

The APC readings are this low because soft LPM is active, resulting in power consumption optimization. With the calculation of ARDC, the average latency of the messages calculates to be 549.57 ms. The program size matters because it directly affects load and unload power consumption. The larger the executing program size, the more power consumption is because of a higher number of I/O (input/output) operations.

4.3.2 Observations for MQTT with AES Encrypted Payload

This mechanism supports a maximum of 16 bytes payload size because of the single block AES implementation limitations.

Table 5 APC

	CPU	LPM	Tx	Rx	Total
Broker	0.259571	0.001459	0.095281	0.653728	1.010039
Client	0.283386	0.001343	0.082785	0.654345	1.021859

Table 6 ARDC

	Tx	Rx	Total
Broker	0.28%	1.22%	1.50%
Client	0.19%	1.24%	1.43%

Table 7 Program Size

	ROM		RAM
	test	data	bss
Broker	53954	322	6657
Client	54742	322	6657

With the calculation of ARDC (Table 6), the average message latency calculates to be 728.57 ms. This is significant when the average encryption/decryption time is 10 ms.

4.3.3 Observations for MQTT with AES-OCB Encrypted and Authentication Payload

This is a designed method that comprises AES in CBC mode, supporting variable payload lengths. For evaluation purposes, a payload of 48 bytes was used.

Table 8 APC

	CPU	LPM	Tx	Rx	Total
Broker	0.327521	0.00125	0.159381	0.690125	1.178277
Client	0.448451	0.001366	0.198572	0.882156	1.530545

Table 9 ARDC

	Tx	Rx	Total
Broker	0.45%	1.33%	1.78%
Client	0.45%	1.36%	1.81%

Table 10 Program Size

	ROM		RAM
	test	data	bss
Broker	55326	322	7057
Client	55921	322	7023

With the calculation of ARDC (Table 9), the average latency of the messages calculates to be 749.57 ms. This increase in latency is not much significant over Base MQTT.

4.3.4 *Observations for MQTT with AES-OCB Encrypted and Authentication Payload*

This is the most advanced designed method that comprises AES in OCB mode, which supports variable payload lengths. For evaluation purposes, a payload of 48 bytes was used. An extra 16 bytes of payload appends (if) with the use of *Retain* Tag.

Table 11 APC

	CPU	LPM	Tx	Rx	Total
Broker	0.463597	0.001255	0.198321	0.782193	1.445366
Client	0.411568	0.001386	0.141562	0.824341	1.378857

Table 12 ARDC

	Tx	Rx	Total
Broker	0.43%	1.44%	1.87%
Client	0.38%	1.57%	1.95%

Table 13 Program Size

	ROM		RAM
	test	data	bss
Broker	54875	338	6941
Client	55719	338	6791

With the calculation of ARDC (Table 12), the average latency of the messages calculates to be 1537.91 ms. In comparison, this latency is very high (3 times of Base MQTT and more than twice of MQTT with AES Encrypted Payload and MQTT with AES-CBC Encrypted Payload).

4.3.5 *Observations for MQTT with CCM Encrypted Link Layer*

This implementation is validated on the Link Layer instead of the Application Layer using the Contiki OS functions. For evaluation, a payload of 48 bytes is used.

Table 14 APC

	CPU	LPM	Tx	Rx	Total
Broker	0.247521	0.00125	0.113812	0.651287	1.01387
Client	0.338451	0.001446	0.157854	0.735429	1.23318

Table 15 ARDC

	Tx	Rx	Total
Broker	0.30%	1.22%	1.52%
Client	0.35%	1.39%	1.74%

Table 16 Program Size

	ROM		RAM
	test	data	bss
Broker	52915	332	6948
Client	53961	332	6798

With the calculation of ARDC (Table 15), the messages' average latency is calculated as 1071.57 ms. This latency is much lower than AES-OCB because of the extent of encryption/decryption at the Link Layer.

4.4 *Comparative Analysis*

The comparison of discussed security mechanisms is purely based on the total power consumption for the publisher/broker and the receiver/client, with *Base MQTT* as a baseline consideration. All the security techniques are thoroughly tested, and the obtained values are an average of 50 emulations in identical conditions. This value of 50 ensures that any irregularity in the obtained values for any particular scenario does not affect the overall outcome. An increase or decrease in the number of emulations may affect the final results differently.

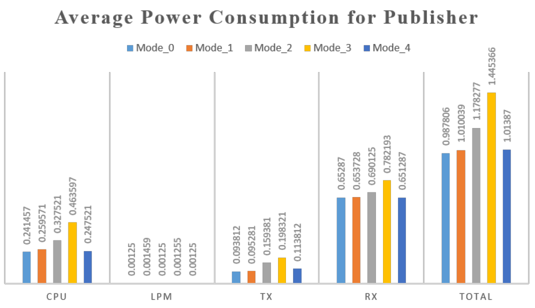

Figure 5 Comparison of average power consumption for Publisher

Fig. 5 depicts the comparison of the total power consumption for the publisher. It is clear that the MQTT with

AES-OCB Encrypted and Authentication Payload consumes much more power than its counterparts, making it unfit where the devices run on limited power sources, which may drain significantly over the period. Although theoretically, MQTT with AES-OCB Encrypted and Authentication Payload is the more potent encrypted payload mechanism, including authentication.

For the receiver end (Fig. 6), although MQTT with AES-CBC Encrypted Payload is one of the most secure implementations of AES spin, it gives the worst power consumption results, making it unsuitable for resource-constrained scenarios. Here, MQTT with CCM Encrypted Link Layer does not have a significant difference in power consumption and has better security than MQTT with AES Encrypted Payload.

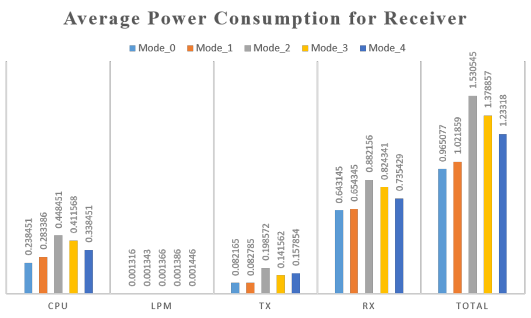

Figure 6 Comparison of average power consumption for Receiver

The plots above establish that transmission Tx power consumption is the least for Base MQTT implementation. Still, the power consumption difference for non-secure Base MQTT and other security implementations (except MQTT with AES-OCB Encrypted and Authentication Payload) is negligible. Thus, the network should use the secure version of MQTT as there is no significant difference in power consumption, making them useful for resource-constrained devices.

5 Conclusion

Secure communication is the key to an uninterrupted deployment of any message exchange protocol. While this is nonchalantly achievable in resource-abundant environments, resource-constrained environments (say, an agricultural field) suffer from the lack of unlimited, uninterrupted power supply. Through this article, we emphasize the power consumption of the MQTT protocol for connection security and payload encryption to make both the connection and the data safe when in transition. We also laid out a detailed power consumption-based comparison of the security enhancement techniques with base MQTT.

A higher message latency leads to the prolonged activity of the communication components, leading to higher power consumption. True to its form, larger message payload sizes require higher message latency because of both bigger encryption/decryption and packet trip times. Unanimously, AES-CBC had the most effective latency-to-payload size proportion. AES-OCB had the highest latency for all the calculations, except when utilizing AES-CBC operating with a larger payload of 64 bytes. Another perception expresses that the execution of AES in manual ECB mode with a single block (with unobstructed encryption/ decryption) expends the most assets for all the cases (that is, the payload size of either 32 bytes or 48 bytes) and has the more noteworthy message latency (because of the successive encryption/decryption) making it a less appropriate choice.

6 Future work and Lateral Enhancements

The scope of IoT security is like an open arena where everyone with the right thought process can enhance the existing technologies to make the future a better place to live. With MQTT security implementations in real-world applications such as smart-home communication, smart-vehicle management, and many more, it is possible to mitigate numerous threats.

As for future enhancements in the implemented methods, it may be possible to use a resource-friendly version of the blockchain algorithm to encrypt the payload. The current blockchain utilizes 256-bit encryption, which is too harsh for low-end devices such as sensor motes, healthcare devices, agricultural deployments, and many others. A chat server could be built using proposed implementations to enable secure private communication within an organization. Commercial messaging application development is also possible.

References

Bhardwaj, S. and Harit, S. (2022). Sdn-enabled secure iot architecture development: A review. In Ranganathan, G., Fernando, X., and Shi, F., editors, *Inventive Communication and Computational Technologies*, pages 599–619, Singapore. Springer Singapore.

Bisne, L. and Parmar, M. (2017). Composite secure mqtt for internet of things using abe and dynamic s-box aes. In *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–5.

Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., and Tischhauser, E. (2014). Ale: Aes-based lightweight authenticated encryption. In Moriai, S., editor, *Fast Software Encryption*, pages 447–466, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bojanova, I. and Voas, J. (2017). Trusting the internet of things. *IT Professional*, 19(5):16–19.

Burange, A., Misalkar, H., and Nikam, U. (2019). Security in mqtt and coap protocols of iot's application layer. In Verma, S., Tomar, R. S., Chaurasia, B. K., Singh, V., and Abawajy, J., editors, *Communication, Networks and Computing*, pages 273–285, Singapore. Springer Singapore.

Casado, L. and Tsigas, P. (2009). Contikisec: A secure network layer for wireless sensor networks under the contiki operating system. In *Nordic Conference on Secure IT Systems*, pages 133–147. Springer.

Deschambault, O., Gherbi, A., and Légaré, C. (2017). Efficient implementation of the mqtt protocol for embedded systems. *JIPS*, 13(1):26–39.

Dinculeană, D. and Cheng, X. (2019). Vulnerabilities and limitations of mqtt protocol used between iot devices. *Applied Sciences*, 9(5).

Dizdarević, J., Carpio, F., Jukan, A., and Masip-Bruin, X. (2019). A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Comput. Surv.*, 51(6).

Hernández Ramos, S., Villalba, M. T., and Lacuesta, R. (2018). Mqtt security: A novel fuzzing approach. *Wireless Communications and Mobile Computing*, 2018.

Jang, H. Y., Shim, J. H., Suk, J. H., Hwang, I. C., and Choi, J. R. (2004). Compatible design of ccmp and ocb aes cipher using separated encryptor and decryptor for ieee 802.11i. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 3, pages III–645.

Jonsson, J. (2003). On the security of ctr + cbc-mac. In Nyberg, K. and Heys, H., editors, *Selected Areas in Cryptography*, pages 76–93, Berlin, Heidelberg. Springer Berlin Heidelberg.

Katsikeas, S., Fysarakis, K., Miaoudakis, A., Van Bemten, A., Askoxylakis, I., Papaefstathiou, I., and Plemenos, A. (2017). Lightweight secure industrial iot communications via the mq telemetry transport protocol. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 1193–1200.

Kenitar, S. B., Marouane, S., Mounir, A., Younes, A., and Gonzalez, A. G. (2018). Evaluation of the mqtt protocol latency over different gateways. In *Proceedings of the 3rd International Conference on Smart City Applications*, SCA '18, New York, NY, USA. Association for Computing Machinery.

Li, D., Aung, Z., Williams, J., and Sanchez, A. (2014). P3: Privacy preservation protocol for automatic appliance control application in smart grid. *IEEE Internet of Things Journal*, 1(5):414–429.

Luzuriaga, J., Perez, M., Boronat, P., Cano, J.-C., Calafate, C., and Manzoni, P. (2016). Improving mqtt data delivery in mobile scenarios: Results from a realistic testbed. *Mobile Information Systems*, 2016:1–11.

Mishra, B. and Kertesz, A. (2020). The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, 8:201071–201086.

Ning, H., Liu, H., and Yang, L. T. (2013). Cyberentity security in the internet of things. *Computer*, 46(4):46–53.

Park, J.-H., Kim, H.-S., and Kim, W.-T. (2018). Dm-mqtt: An efficient mqtt based on sdn multicast for massive iot communications. *Sensors*, 18(9).

Patel, C. and Doshi, N. (2020). "a novel mqtt security framework in generic iot model". *Procedia Computer Science*, 171:1399–1408. Third International Conference on Computing and Network Communications (CoCoNet'19).

Raikar, M. M., S M, M., and Mulla, M. M. (2020). Software defined internet of things using lightweight protocol. *Procedia Computer Science*, 171:1409–1418. Third International Conference on Computing and Network Communications (CoCoNet'19).

Rogaway, P., Bellare, M., and Black, J. (2003). Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403.

Sawataishi, S., Ueno, R., and Homma, N. (2020). Unified hardware for high-throughput aes-based authenticated encryptions. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(9):1604–1608.

Singh, M., Rajan, M. A., Shivraj, V. L., and Balamuralidhar, P. (2015). Secure mqtt for internet of things (iot). In *2015 Fifth International Conference on Communication Systems and Network Technologies*, pages 746–751.

Stankovic, J. A. (2014). Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3–9.

Steve (2017). Introduction to mqtt security mechanisms.

Su, N., Zhang, Y., and Li, M. (2019). Research on data encryption standard based on aes algorithm in internet of things environment. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 2071–2075.

Tan, C., Deng, X., and Zhang, L. (2018). Identification of block ciphers under cbc mode. *Procedia Comput. Sci.*, 131(C):65–71.

Team, T. H. (2019). Getting started with mqtt.

Yassein, M. B., Shatnawi, M. Q., Aljwarneh, S., and Al-Hatmi, R. (2017). Internet of things: Survey and open issues of mqtt protocol. In *2017 International Conference on Engineering MIS (ICEMIS)*, pages 1–6.

Yassein, M. B., Shatnawi, M. Q., et al. (2016). Application layer protocols for the internet of things: A survey. In *Engineering & MIS (ICEMIS), International Conference on*, pages 1–4. IEEE.