Application Layer Security: MQTT perspective with TLS Implementation and Analysis

Saud S. Alharbi
Department of Computer Science
Brunel University
London, United Kingdom.
saud.alharbi@brunel.ac.uk

David Bell
Department of Computer Science
Brunel University
London, United Kingdom
david.bell@brunel.ac.uk

Wasan Awad
College of Information Technology
Ahlia University
Kingdom of Bahrain
wawad@ahlia.edu.bh

***Abstract* - The Internet of Things (IoT) has become ingrained in our daily lives, transforming the way we interact with technology. From smart homes to wearable devices, IoT enhances convenience and connectivity. However, this widespread adoptionraises security concerns. Recent years have witnessed a surge in cyberattacks exploiting vulnerabilities in IoT devices. Security lapses in device development and the sheer volume of interconnected devices contribute to the challenges. Data breaches and privacy infringements also loom large, highlighting the need for a balanced approach to technological advancement and robustcybersecurity measures to safeguard personal information and ensure the positive impact of IoT on daily life. In this study, we reviewed IoT encryption algorithms with focus on the integration between the IoT application layer and some encryption algorithms with focus on the MQTT protocol.Additionally, we conducted a comparative performance analysis of MQTT with and without TLS, highlighting the impact of encryption on IoT communication in terms of different performance metrics like CPU cycles, memory consumption, latency and throughput.**

Keywords - IoT data security, secure MQTT protocol, TLS, MQTT, lightweight secure communication, IoT application layer.

I. Introduction

In the current landscape, the Internet of Things (IoT) has [1]seamlessly integrated into our daily lives, revolutionizing theway we interact with technology and the physical world. Fromsmart homes to wearable devices, IoT has become an indispensable part of our routines, enhancing convenience, efficiency, and connectivity. IoT's influence is prominently visible in the realm of smart homes, where interconnected devices like thermostats, lights, and security cameras collaborate to create an intelligent living environment. Home automation allows individuals to control various aspects of their homes remotely, optimizing energy consumption and ensuring security. Beyond the confines of homes, wearable devices have also become prevalent. From fitness trackers monitoring our physical activity to smartwatches seamlessly connecting with our smartphones, these devices provide real-time data that empowers individuals to make informed decisions about their health and well-being. In the industrial sector, IoT plays a pivotal role in enabling the concept of Industry 4.0, where interconnected devices and sensors enhance manufacturing processes, improve efficiency, and enable predictive maintenance. This integration of IoT in industries contributes to the evolution of smart cities, intelligent transportation systems, and efficient energy management. However, the widespread adoption of IoT comes [2] with itsown set of challenges, most notably in the realm of security. As IoT devices proliferate, the attack surface for malicious actors expands, leading to an increase in security threats. Recent years have seen a surge in cyberattacks targeting IoT devices, exploiting vulnerabilities to gain unauthorized access, manipulate data, or launch large-scale distributed denial-of-service (DDoS) attacks. One significant security concern is the insufficient attention given to cybersecurity measures in the development of IoT devices. Many devices lack robust security protocols, making them susceptible to exploitation. Additionally, the sheer volume and diversity of IoT devices make it challenging to implement standardized security practices across the ecosystem.
As the Internet continues to penetrate further [2] into various aspects of human life and the physical environment, the potential for cyber threats expands alongside. For instance, even a smart coffee machine may possess vulnerabilities that, if exploited, could grant an attacker unauthorized access to manipulate all connected devices within a home network [3]. This includes critical components such as sensors and life-supporting medical devices like insulin pumps or heart pacemakers directly connected to the human body.

II. Theoretical Background

In the realm of IoT, [4] cryptography seeks to fulfill two fundamental objectives of information security: maintaining the confidentiality and integrity of data. Encrypting the data is essential to ensure its confidentiality as it traverses a communication channel. Cryptography algorithms can be categorized into two primary types: symmetric key and asymmetric key ciphers illustrated in Fig. 1.

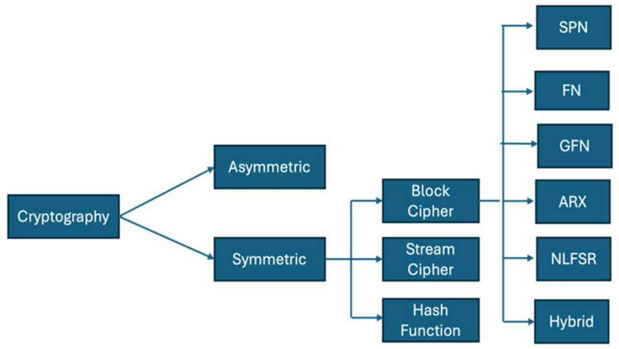


Fig. 1. Cryptography algorithms [5]

As discussed in [5], in resource-constrained IoT devices, block ciphers are favored over stream ciphers. Table I shows that the most well-known algorithms are block ciphers.

TABLE I. IOT ENCRYPTION ALGORITHMS [5]

Structure Type	Algorithms
SPN	AES, Present, GIFT, SKINNY, Rectangle, Midori, mCrypton, Noekeon, Iceberg, Puffin-2, Prince, Pride, Print, Klein, Led, Picaro, Zorro, I-Present, EPCBC
FN	DESL / DESXL, TEA / XTEA/ XXTEA, Camellia, Simon, SEA, KASUMI, MIBS, LBlock, ITUbee, FeW, GOST, Robin, Fantomas
GFN	CLEFIA, Piccolo, Twis, Twine, HISEC
ARX	Speck, IDEA, HIGHT, BEST-1, LEA
NLFSR	KeeLoq, KATAN/KTANTAN, Halka
Hybrid	Hummingbird, Hummingbird-2, Present-GRP

In this section, we present a survey of lightweight encryption algorithms tailored for the Internet of Things (IoT). Additionally, we identify certain gaps in the existing literature. The survey encompasses a compilation of recent, high-quality publications pertaining to lightweight encryption algorithms for IoT. Most of the selected research papers were published after the year 2020. The Google Scholar search engine served as the primary tool for sourcingthese papers. It is noteworthy that a substantial proportion ofthe reviewed papers garnered notable citation counts, although it is observed that papers published subsequent to theyear 2023 may exhibit fewer citations, underscoring thepossibility of high-quality contributions with less citations.

The Internet of Things (IoT) [1] refers to the network of interconnected devices or things, enabling them to communicate, exchange data, and operate seamlessly to enhance efficiency and convenience in various domains. Kevin Ashton introduced the IoT paradigm in 1998 [1]. Within an IoT network, diverse and heterogeneous devices, along with varied communication protocols, facilitate the collection and exchange of data among different nodes within the network.

Several methods [6] can be employed to secure MQTT communication, focusing on either securing the MQTT brokeror the data being transferred. Securing the MQTT broker involves several techniques. One technique is restricting access based on unique client IDs. Another technique is authenticating clients with valid credentials using a usernameand password, although these are transmitted in plain text. Additionally, installing and maintaining x509 client certificates on client applications and devices enhances security. Securing data transfer methods include utilizing theTLS protocol to establish an encrypted tunnel for MQTT message transfer, though TLS support may be limited on someclient devices. Another method is implementing encryption and decryption routines at the client level to secure data endpoints within the MQTT communication protocol. The three-layer architecture, the most common model in IoT networks [1], comprises the application layer, network layer,and perception layer. In the four-layer architecture, comparedto the three-layer model, an additional layer, the data processing layer, is incorporated. Expanding on the three- layer model, the five-layer architecture introduces two extra layers: the business layer and the data processing layer. Sincethe application layer is the most common communication model on IoT, it will be explained in further detail.

1) *Perception Layer:* The perception layer [2] in IoT is a pivotal component responsible for interfacing with the physical environment. It incorporates various sensors and devices to collect data,enabling the system to make informed decisions. Devices in this layer, such as sensors and RFID tags, gather information on environmental conditions and object presence. The collected data is processed locally or transmitted for further analysis, facilitating quick decision-making and efficient system operation. In essence, the perception layer bridges thegap between the physical world and the digital realm of the IoT system, facilitating data acquisition and laying the foundation for intelligent decision-making.

2) *Network Layer :* The IoT network layer [7] is crucial for managing communication among IoT devices. It ensures seamless connectivity, utilizes protocols for standardized data exchange, handles scalability challenges, implements security measures, and maintains reliability in data transmission. Essentially, it forms the backbone of the IoT ecosystem, enabling efficient and secure interactions between interconnected devices. In the context of IoT network layer protocols, Zigbee and 6LoWPAN are indeed more fitting examples. Zigbee is a low-power, short-range wireless communication protocol suitable for IoT devices, promoting efficient connectivity in home automation and industrial settings. 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) enables the transmission of IPv6 packets over low-power and lossy networks, optimizing communication for resource-constrained IoT devices. These network protocols enhance the functionality of the IoT

network layer by providing reliable, standardized communication mechanisms for interconnected devices.

3) Application Layer: The IoT application layer [1] is the top layer in the architecture, facilitating user interaction, processing and analyzing data, and implementing application-specific functionalities. It integrates with external systems, ensuring security, and access control, making the IoT system user- friendly and adaptable to various domains. Two major protocols exist in the application layer of IoT explained in[1].

- **Message Queuing Telemetry Transport (MQTT)** is a lightweight message passing protocol developed to facilitate data exchange among numerous devices in a network. Employing a publish/subscribe mechanism with acentral server or broker, MQTT enables reliable messagepublication in low-bandwidth networks. Initially proprietary in the oil and gas industries for SCADA systems, MQTT hasevolved into an open-source standard for connecting millionsof IoT and industrial IoT devices across various applicationslike remote monitoring, health parameter tracking, and motion detection. The MQTT protocol offers diverse authentication mechanisms and encryption techniques based on TLS.
- **Constrained Application Protocol (CoAP)** is specifically designed for use with constrained nodes,such as IoT devices, and networks, including building automation. Operating as a client-server protocol, CoAP enables one node (client) to command another by transmittingCoAP packets. A notable advantage of CoAP is its capabilityto integrate resource-constrained devices into IoT networks, even in environments with limited resources like low bandwidth and network availability. CoAP finds predominantadoption in Machine-to-Machine (M2M) scenarios, includingapplications in smart homes, smart energy, and buildingautomation.

Other IoT application layer protocols with their security weaknesses were discussed in [1], like XMPP, mDNS, SSDP.

A. Edge Computing in The Context of IoT

Edge computing in the context of IoT [3] involves processing data closer to the source of generation, typically near or on the IoT devices themselves, rather than relying solely on centralized cloud servers. This approach reduces latency, enhances real-time processing capabilities, and alleviates theburden on network bandwidth. By distributing computing power to the edge of the network, edge computing improves the efficiency and responsiveness of IoT applications, makingit particularly valuable for time-sensitive tasks, data privacy, and overall system performance. According to [3] and [1], edge computing architecture can be utilized in order to outsource some cryptographic functions to reduce the overhead on IoT resource-constrained devices. Edge computing addresses challenges related to device mobility and latency by strategically positioning storage and processing resources in an intermediate layer between smart devices and fog/cloud computing platforms. This approach optimizes data processing and analysis by bringing computational capabilities closer to the source of data generation.

However, as explained in [3] and [1], introducing edge computing in IoT encryption may expose other types ofsecurity threats by potentially decentralizing encryption processes, leading to increased vulnerability. Distributingencryption tasks across edge devices could create points of attack, necessitating robust measures to safeguard against unauthorized access and potential breaches as they operate in more malicious environments.

B. IoT Encryption Algorithms

The traditional encryption protocols face challenges in IoT environments due to the unique characteristics of IoT devices [3],[1],[2],[7],[5]. IoT devices often have constrained computational resources, making traditional encryption impractical. High energy overhead and large data size in encrypted payloads can deplete limited power sources and hinder network efficiency. Complex key distribution and management are challenging in dynamic IoT ecosystems. Traditional encryption may introduce unacceptable delays in applications with stringent latency requirements. Implementing traditional encryption may require additional hardware, increasing device costs. Scalability becomes difficult when adapting traditional encryption to the vast and diverse IoT ecosystem. To address these issues, lightweight encryption protocols are preferred in IoT, striking a balance between security and resource efficiency.

The following list explains some lightweight IoT encryption algorithms from various aspects based on the reviewed research papers. The papers discussed the IoT encryption onapplication layer are excluded from the below list as it will bediscussed in another section of this paper, i.e. section D. below.

1) Traditional encryption techniques [8] like RSA or AES demand significant computational resources and extensive memory, potentially leading to a detrimental impact on device performance. Simplerencryption methods are susceptible to compromise,while cryptographic algorithms are consistently improved for server and desktop environments, deploying these advancements on resource-limited devices poses significant challenges. Moreover, a new algorithm was introduced in [8], then the authors utilized FELICS (Fair Evaluation of Lightweight Cryptographic Systems) to compare theperformance and security of the new algorithm including Advanced Encryption Standard (AES), PRESENT cryptography algorithm, HIGHT cryptography algorithm, SIMON&SPECK encryption algorithm, and A Lightweight EncryptionAlgorithm for Secure Internet of Things (SIT). The FELICS offers a comprehensive comparison of key scheduling, encryption, and decryption performance across multiple lightweight algorithms. The proposed algorithm demonstrates exceptional efficiency, notably requiring far fewer cycles for keyscheduling (1641 cycles), encryption (803 cycles), and decryption (800 cycles) compared to its peers. This outstanding performance underscores its speed and resource optimization, positioning it as a promising solution in lightweight cryptography.

2) As discussed in [7], there are different attacks on IoTper IoT layer, mainly on network and application layers: The network layer faces routing attacks such as sinkhole and selective forwarding in its multi-hop environment. DDoS attacks exploit this layerthrough spoofing and routing path modifications. Eavesdropping poses a significant threat, compromising system security and privacy by providing attackers access to crucial information about network nodes, impacting Quality of Service (QoS). Insecure networks become prime targets for eavesdroppers, posing risks of identity theft and financial loss. The application layer is susceptible todirect end-user access, creating vulnerabilities for unauthorized users. Protocols like MQTT, XAMPP,and COAP aim to safeguard against attacks butremain vulnerable. Security threats such as DDoS, sniffing, malicious node injection, and phishing directly impact application functionality, posing significant risks. More attacks on 6LoWPAN were discussed in [9], with additional references provided for potential solutions aimed at safeguarding 6LoWPAN like [10], [11]. In [11], The author addressesthe unique security challenges posed by IoT devices,stressing the need for tailored encryption protocols and authentication processes distinct from traditionalinternet security. In their proposed method, the authors conduct a comparative analysis between two encryption mechanisms: AES (Advanced Encryption Standard with a 128-bit key) and PRESENT with an 80-bit key. The authors believe that the PRESENT algorithm is lightweight andmore suitable for IoT.

3) As stated in [4], four cryptographic algorithms are currently in the process of establishing themselves as the standard for Lightweight Cryptography (LWC). These algorithms were put forward for inclusion in the ISO/IEC 29192 standard for LWC, developed in2012. The algorithms under consideration areSIMON, SPECK, PRESENT, and CLEFIA. The authors proposed the creation of a novel LightweightCryptography (LWC) solution by incorporating the most favorable features from each of the four algorithms. This fresh cryptographic solution should exhibit flexibility concerning block and key sizes, demonstrate efficient performance in both hardwareand software implementations, and possess resilience against diverse cryptanalytic and brute- force attacks. The development of the new algorithm could involve integrating elements from each of the four algorithms:

- Simon, introduced by the National Security Agency's research directorate (NSA) in June 2013, is a block cipher designed for use in constrained environments withlimited computational power and energy resources. Leveraging the NSA's extensive cryptography expertise, Simon is deemed secure and belongs to the Feistel network family. This family provides advantages inhardware due to the similarity between encryption and decryption operations, resulting in a reduced hardware requirement and a more compact circuit. Simon employs a straightforward round function incorporating basic operations such as XOR, AND, and left circular shifts.
- In contrast to Simon's focus on hardware optimization, SPECK is tailored for software-based implementations, especially on microcontroller processors.Noteworthy for its speed and low memory consumption, SPECK performs only 32operations on the input and does not rely ona lookup table. Like Simon, SPECK isbased on the Feistel structure, involving bitwise XOR, modular addition, and circular shifts in each Feistel round.
- PRESENT, a hardware-oriented block cipher, follows the Substitution- Permutation network (SPN) design [4],[12]. Accepting a 64-bit block size and offeringkey sizes of 80 and 128 bits, PRESENT is suitable for applications prioritizing performance over high security. The cipher assumes data to be encrypted is of small tomedium size, aligning with its focus on space and performance optimization fordevices with average power consumption.
- CLEFIA, a 128-bit block cipher, supports various key sizes, including 128, 192, and 256 bits. Structured on the Feistel network with four data lines and two 32-bit f functions per round, CLEFIA incorporates a diffusion switching mechanism (DSM) inits f-functions to enhance resistance againstspecific attacks. Additionally, CLEFIA employs a compact key scheduling approach and a doubleswap function for efficient round key generation, making it suitable for both software and hardware implementations. The algorithm supports multiple key sizes and utilizes two distinct8-bit S-boxes [4],[13].

4) As discussed in [5], organizations and research groups actively contributing to advancements in cryptography for enhancing lightweight standards in resource-constrained devices include:

- National Institute of Standards and Technology, USA (NIST)
- International Organization of Standardization and the International Electrotechnical Commission (ISO/IEC)
- Cryptography Research and Evaluation Committees, Japan (Cryptrec)
- European Network of Excellence in Cryptology (Ecrypt)
- National Security Agency of the USA (NSA)
- CryptoLUX (University of Luxembourg)

Among the cryptographic algorithms, only PRESENT and CLEFIA have received approval from the ISO/IEC 29192 standard. Meanwhile,Cryptrec targets AES, CLEFIA, TDES, Camellia, PRESENT, PRINCE, Piccolo, LED, TWINE, SIMON & SPECK, and Midori.

5) PRESENT [14] is a well-known lightweight block cipher. Developed in 2007 by A. Bogdanov et al., this cipher is based on a substitution-permutation structure. It features a 64-bit block size, keys of lengths 80 bits (PRESENT80) and 128 bits (PRESENT128) and comprises 31 rounds. It was certified as a standard lightweight block cipher by the International Organization for Standardization (ISO / IEC 29192-2) in 2012.

- **Problem Statement :** Renowned for its lightweight and efficient publish-subscribe messaging model, the MQTT protocol serves as the backbone for numerous IoT deployments [5]. Nevertheless, its vulnerability to various threats arises from the absence of built-in security mechanisms. There are many studies that have been carried out to address data security issues in the MQTT protocol using cryptography. By default, [15] transmissions through MQTT on port 1883 remain unencrypted. However, for safeguarding sensitive information within messages, the MQTT specification suggests employing the TLS protocol on port 8883. While many brokers and MQTT platforms support TLS, it comes with significant overhead. Upon message publication, the TLS protocol necessitates a handshake process. Despite IoT payloads being small, they are transmitted frequently. Frequent use of TLS protocol mandates repeated reconnections and handshake processes due to the intermittent nature of IoT connections, leading to substantial power and computation consumption. Additionally, the TLS session persists until the MQTT client completes its tasks, which is not advantageous for short-lived connections. This research demonstrates the impact of using TLS with the MQTT protocol, particularly highlighting the challenges posed by TLS in terms of CPU cycles, memory consumption, latency and throughput.

- **Related Work**: In this section, research papers that explore IoT encryption protocols are investigated, and the incorporation of these algorithms into the MQTT communication protocol is examined. Message Queuing Telemetry Transport (MQTT) is anticipated to become the predominant messaging standard for the Internet of Things (IoT). It is imperative for MQTT to establish efficient security measures. However, a notable drawback of MQTT is its absence of inherent protection mechanisms. Existing approaches have introduced processing overhead to devices and remain susceptible to various attacks. Originally developed by IBM as an affordable Machine-to- Machine (M2M) interaction protocol operating atop TCP, MQTT serves as a data communication protocol within the IoT context and has gained recognition from OASIS. Hintaw et al. proposed a modified AES protocol called RSS, comparing it with the original AES and addressing the need for enhanced security without significantly increasing implementation costs [16].

Rahman et al. introduced a new protocol called S-MQTT by integrating the original MQTT with the Elliptic Curve cryptosystem (ECC) [17]. ECC, a public key cryptosystem similar to RSA, operates with one key for encryption and another for decryption. It relies on the mathematics of elliptic curves, utilizing the coordinates of points on the curve for these operations. The key advantage of ECC lies in its ability to achieve an equivalent level of security with much shorter key lengths compared to other public key cryptosystems. Furthermore, ECC's security level increases more rapidly with key size than integer-based discrete logarithm or RSA-based systems. ECC also boasts faster implementation, requiring less bandwidth and power, making it particularly valuable for resource-constrained systems with limitations in computing power, memory, and battery life. Nevertheless, no performance analysis was presented in comparison to other encryption methods (Rahman et al., 2018). More-over the IoT, yet they remain vulnerable to multiple attacks. The authors introduced a dynamic adaptation of the AES algorithm (D-AES) tailored for MQTT security by reconfiguring key expansion, ShiftRows, and SubBytes transformations of the AES algorithm. Publishers encrypt the payload of MQTT packets using the D-AES algorithm. Additionally, the symmetric key of the D-AES algorithm undergoes encryption using the KeyPolicy Attribute-Based Encryption (KP-ABE) scheme. Subsequently, the encrypted payload and key are transmitted to the broker, which then relays them to subscribers. Subscribers initiate a request for the key to an external authority, which executes the key generation algorithm and sends the decryption key to the subscribers. Experimental results suggest that the proposed method marginally increases execution and processing times while introducing significant traffic overhead (Hintaw et al., 2023).

Iqbal et al. enhanced the security of MQTT by utilizing ARIA as an encryption algorithm and MbedTLS to establish secure communication channels [18]. The usage of MbedTLS facilitated the establishment of secure connections, allowing for the encryption and decryption of MQTT messages. As a result, the sensitive data transmitted within the IoT network is safeguarded. ARIA, the Advanced Encryption Standard (AES) designed for the Republic of Korea, stands as the foundational cryptographic algorithm. Originating from the Korean Agency for Technology and Standards (KATS), ARIA is widely acknowledged for its strong encryption features and seamless compatibility with AES. This study incorporates the ARIA cryptography algorithm into MQTT, facilitating secure data transmission within IoT networks using a symmetric key encryption approach to ensure the confidentiality and integrity of data. MbedTLS, an open- source cryptographic library created by Arm Limited, provides a comprehensive array of cryptographic functions and protocols, establishing itself as a reliable option for secure communication in IoT deployments. However, no performance evaluation was conducted on the new protocol in comparison to other similar protocols (Iqbal et al., 2023).

Wijayanto et al. evaluated the effectiveness of AES, Grain V1, and RC4 algorithms against passive sniffing attacks and their performance in data processing time. Results demonstrate that all three algorithms successfully thwart passive sniffing attacks. Furthermore, the study assessed the resistance of these algorithms to cryptanalysis by comparing the time required to break their keys. The findings reveal that AES exhibits the highest resistance, making it the safest choice compared to Grain V1 and RC4. Additionally, performance testing was conducted to assess encryption and decryption processing times for each algorithm. RC4 emerged as the fastest algorithm, with an average encryption time of 20.4 microseconds, followed by Grain V1 with 763.4 microseconds, and AES with 796.84 microseconds. Similarly, RC4 demonstrated superior decryption performance, with an average time of 0.13 microseconds, followed by Grain V1 with 0.18 microseconds, and AES with 1.16 microseconds (Wijayanto et al., 2023) [19].

Al-Ani et al. conducted a comprehensive examination of the MQTT protocol, incorporating experimentation on an MQTTsystem employing diverse cryptographic techniques, namelyAES-CBC, RSA, and ECC AES Hybrid Scheme, to evaluate processing time and message size. Results suggest that encrypting payloads increases both processing time and message size. Among the cryptographic techniques tested, RSA demonstrates the longest processing time, followed by ECC AES Hybrid Scheme and AES-CBC.

Additionally, the study assesses the effectiveness of attack prevention between standard MQTT and secured MQTT setups by simulating various IoT attacks, including black-box penetration, identityspoofing, DoS, and MITM attacks. The findings and ensuing discussion shed light on the cryptographic algorithms that impose the most overhead on standard MQTT implementation and their resilience against common attacks, addressing the research question at hand (Al-Ani et al., 2023).

Jayan and Harini explored the advantages of employing cryptographic algorithms to mitigate the effects of spoofing on MQTT networks. Their approach involves encrypting the payload of MQTT packets before publishing, utilizing Rivest-Shamir-Adleman (RSA) and ECC algorithms. Their experimental findings indicate that ECC exhibits lower time complexity compared to RSA, rendering it a superior option for IoT devices (Jayan and Harini, 2018) [20].

De Rango et al. devised a dynamic IoT security system aimed at thwarting attacks on the MQTT protocol. Their approach involves employing elliptic curves to encrypt MQTTpayloads, thereby preventing eavesdropping and data tampering attacks. Additionally, they utilize timestamps and wake-up patterns to mitigate replay attacks. Enhancements in energy efficiency were achieved by adjusting the key strength of the ECC algorithm, albeit leading to heightened system overhead (De Rango et al., 2019).

Hamad et al. provided a model called SEEMQTT for IoT systems to ensure secure end-to-end MQTT communications. This model is designed to address vulnerabilities in MQTT communications and improve security through a combination of secret sharing and trust delegation mechanisms (Hamad et al., 2022).

Varma and UniKrishnan explored MQTT, a lightweight messaging protocol where the payload serves as the conduit for the transmitted data, for its susceptibility to cyber-attacks, specifically the Man-in-the-Middle (MITM) attack. The paper investigates the susceptibility of the payloadto cyber-attacks by subjecting it to two different security measures. Initially, channel-based security using the Transport Layer Security (TLS) is applied to the payload, followed by object-based security using the Advanced Encryption Scheme (AES). Both encryption techniques for securing MQTT protocol payloads are thoroughly examined and discussed (Varma and UniKrishnan, 2021).

Dewanta et al. introduced the SibProMQTT methodology designed to safeguard MQTT communication against Sybil attacks. The SibProMQTT approach achieves attack mitigation through mechanisms such as message lifetime verification, session key validation, and concealment of private information. Experimental findings demonstrate that SibProMQTT can be effectively implemented on the ESP32,a low-computational-resources hardware commonly utilizedin IoT devices, incurring computational costs of 8.30 ms and 8.44 ms for SHAI and SHA256 hash function usage, respectively. Furthermore, post-installation of SibProMQTTon the ESP32, there remains ample space available for additional computations. Additionally, it was discussed that both TLS and authenticated key exchange (AKE) approaches are too expensive for MQTT IoT environments with few resources (Dewanta et al., 2023).

Sadio et al. advocated for the adoption of ChaCha20-Poly1305 AEAD as a means to ensure secure communication among constrained nodes over MQTT/MQTT-SN. ChaCha20 and Poly1305 are recognized as lightweight stream cipher and one-time authenticator, gaining prominence within the crypto community. They developed a prototype of this solution implemented on constrained nodes such as Arduino UNO. The paper primarily focuses on presenting results concerning memory usage and execution time. Findings demonstrate thatthe proposed scheme demands minimal memory footprint andboasts low processing time (Sadio et al., 2019).

Fan et al. proposed payload encryption using hierarchical identity-based encryption (HIBE) for MQTT. The proposed scheme exhibits superior performance in terms of time complexity and offers protection against replay attacks. In the event of an attacker attempting to resend a message, the system remains unaffected, and the subscriber simply receivesthe same message again. Furthermore, they formally demonstrated the security of the proposed MQHIBE schemewithin the standard model (Fan et al., 2023).

Ahmad et al. examined a technique for enhancing the securityof the MQTT protocol through the implementation of TLS and evaluated its performance. By employing TLS in MQTT, messages transmitted from MQTT clients to servers are encrypted. Analysis conducted using Wireshark demonstrated that integrating TLS into MQTT effectively encrypts messages within network traffic. The performance analysis encompasses both the CPU utilization of the applicationengaged in the MQTT protocol and the packet size of the transported messages (Ahmad et al., 2023).

Sahmi et al. utilized the PRESENT encryption algorithm to encrypt MQTT messages (Sahmi et al., 2021). Hkiri et al. discussed the implementation of four lightweight cryptographic algorithms—PRESENT, LED, Piccolo, and SPARX—on a Contiki-based IoT operating system designedfor IoT platforms [21].

The evaluation includes an assessment of RAM and ROM usage, power and energy consumption, and CPU cycle count. The Cooja network simulator is employed in this study to identify the most suitable lightweight algorithms for IoT applications utilizing wireless sensor network technology (Hkiri et al., 2022).

The literature on securing MQTT within IoT environments reveals a consistent focus on balancing security with the limitations inherent to resource-constrained devices. While numerous encryption protocols have been proposed and tested, each comes with its own set of trade-offs between enhanced security, processing time, and system overhead.

A key insight from this review is the critical need for lightweight encryption techniques that do not compromise the efficiency of MQTT. The AES-based adaptations and ECC implementations emerge as prominent solutions, offering strong security while maintaining reasonable performance metrics. However, these solutions often introduce complexities that may not be suitable for all IoT use cases, particularly in environments with extremely limited computational resources. In summary, while significant strides have been made in enhancing the security of MQTT, the quest for an optimal solution—one that seamlessly integrates strong encryption with minimal performance trade-offs—continues to drive research in this critical area of IoT security.
Table 2 presents a summary of various research works focusing on encryption protocols and their performance when integrated with MQTT. The studies evaluate different encryption techniques and their effectiveness in providing security while maintaining performance efficiency. This comparative analysis highlights the strengths, limitations and testbed of each approach.

TABLE II. COMPARISONS OF DIFFERENT MQTT ENCRYPTION METHODS

Research Work	Year of Publication	Encryption Protocols	Performance Tool	Testbed	Strengths	Limitations
[18]	2023	-ARIA chipper -TLS	Wireshark	Hardware: -Raspberry Pi 3B+ -Raspberry Sense HAT -Artix-7 CMOD A7 Software: -Modified PahoMQTT -mbedTLS	Combination of ARIA and TLS provided the necessary encryption, authentication, and data integrity mechanisms	-TLS might be not suitable for many IoT environments as it is heavy protocol. -No comparison with other protocols. -Using Wireshark -Complex Testbed
[17]	2018	ECC		Hardware: -ATmega328P Software: -Node Js Server -MongoDB -Eclipse Mosquitto -MQTT V3.1.1	Using ECC is better choice when compared to RSA and more lightweight	-Lack of comparison with other protocols -Using symmetric encryption is more suitable for IoT when compared to symmetric encryption.
[22]	2023	-AES-CBC -RSA -ECC AES Hybrid Scheme		Hardware: ESP32 DEVKITV1 Raspberry Pi 3B+ Software: Eclipse Paho MQTT	-Good evaluation of the three protocols against four attack implementations: black box penetration, entity spoofing, DoS and MITM.	-Limited to Performance evaluation and four attacks.
[23]	2018	-ECC -RSA	Wireshark	Hardware: Not mentioned Software: Not Mentioned	-Focus on spoofing attacks in IoT environments.	-No details about Testbed. - No detailed and clear analysis
[20]	2019	-ECC	Not Mentioned	Hardware: -ESP8266 -ATmega328P -Sensor DHT11 Software: -bcprov-jdk15on-160 library for ECC.	-Prevents reply attacks -IDS for the validation of packets timestamps -Analysis for power consumption	-Lack of comparison with other protocols -Using symmetric encryption is more suitable for IoT when compared to symmetric encryption.
[24]	2022	-open-source Arduino platform - IBE -AES		Hardware: -Espressif ESP32-WROOM-32D as publisher -Seven Raspberry Pi 4 Model B (RPI4) one was used as a broker and one as subscriber and rest for key stores.	-Novel work by introducing SEEMQTT which provide MQTT main functions. Other papers use existing MQTT implementations and integrate them encryption protocols.	-Complex setup

[25]	2021	-TLS		Hardware: -Different Laptops -Software -Kali Linux -Paho MQTT client	-MITM attack was analyzed on MQTT with and without TLS and AES in different scenarios.	-Testbed is not simulating real IoT environment. -No performance evaluation. -TLS and AES is expensive solutions for IoT with low computational resources.
[26]	2023	-SHA1 -SHA256	Wireshark	Hardware: -Laptop -Raspberry Pi 4b+ Software: -MQTT (HiveMQ)	-Provided approach to protect Sybil attacks such as reply back attacks. -Provided security and performance evaluation under some scenarios.	- Despite the usage of hashing algorithms, encryption protocol is not mentioned in the analysis. -No comparison with other encryption protocols in terms of security and performance analysis.
[27]	2019	-ChaCha20 - Poly1305	Not mentioned	Hardware: - Arduino Uno - Raspberry PiSoftware: - Arduino Cryptography Library -Eclipse Paho MQTT-SN library -Mosquitto /RSMB as MQTT broker	-Provided results related to memory footprint and execution time. -Provided solution suitable for constrained resources in terms of memory and processing time.	-No comparison with other encryption protocols in terms of security and performance analysis. -No power consumption analysis.
[15]	2023	ID-based encryption	Python libraries on a Ubuntu 18.04.4 LTS Linux system	Hardware: -Laptop with Ubunto Linux	-Provided security proof as well as comparison with TLS and a comparison to the work in [28].	-No comparison with other protocols. - Decryption cost is high compared with MQTT with TLS.
[29]	2023	TLS	Wireshark	Hardware: -Raspberry Pi -ESP32 Software: Mosquitto MQTT	-Provided practical way of integrating TLS with MQTT.	-TLS is not suitable for IoT devices with low resources.
[30]	2021	PRESENT	Not Mentioned	Not Mentioned	-Provided an approach to integrate PRESENT with MQTT. PRESENT is lightweight protocol with a low resources consumption compared to other protocols.	-No security nor performance evaluation. -More details is needed to prove the effectiveness of the protocol.

The studies provided in Table 2 cover a wide range of encryption protocols, including ARIA, ECC, AES, RSA, ChaCha20, Poly1305, and PRESENT. Each protocol offers different strengths and limitations, highlighting theimportance of selecting the appropriate encryption technique based on specific IoT requirements. Also, various performance tools such as Wireshark and different hardware and software configurations are used across the studies. This diversity in testing environments underscores the need for standardized evaluation methods to ensure consistent and comparable results. A recurring theme in the research is the trade-off between security and performance. While protocolslike TLS and AES provide strong security, they are often deemed too heavy for low-resource IoT devices. Conversely, lightweight protocols like ECC and PRESENT offer better performance but may not provide the same level of security.

III. Implementation and Performance Analysis of TLS in MQTT Protocol

This section discusses the implementation and analysis of TLS in MQTT using self-signed certificate. The publisher and subscriber were implemented in Python and integrated with TLS protocol. The self-signed certificate was generatedusing Ubuntu OpenSSL tool. This section provides performance analysis of MQTT with TLS compared to MQTTwithout any encryption. Fig 2 shows the hardware and software setup used to implement the TLS and analyze its performance against non-TLS implementation of MQTT.

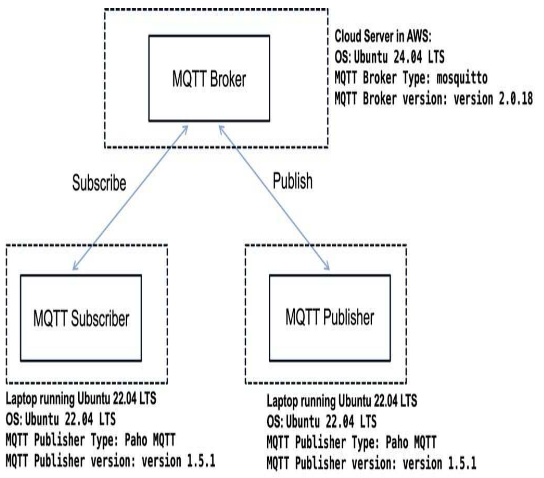

MQTT Broker
Cloud Server in AWS:
OS: Ubuntu 24.04 LTS
MQTT Broker Type: mosquitto
MQTT Broker version: version 2.0.18
Subscribe
Publish
MQTT Subscriber
MQTT Publisher
Laptop running Ubuntu 22.04 LTS
OS: Ubuntu 22.04 LTS
MQTT Publisher Type: Paho MQTT
MQTT Publisher version: version 1.5.1
Laptop running Ubuntu 22.04 LTS
OS: Ubuntu 22.04 LTS
MQTT Publisher Type: Paho MQTT
MQTT Publisher version: version 1.5.1

Fig. 2. Testbed Hardware and Software Setup

The performance that will be conducted in this study is relatedto comparing TLS and non-TLS implementations of MQTT using the following performance metrics from the publisher side: average CPU utilization in terms of CPU cycles, peak memory usage and throughput. The following metric is calculated from subscriber side: average latency. The publisher represents the IoT resource-constraint device whilethe subscriber represents the end user device to manage and control the IoT device.

A. Hardware Setup

The hardware used in this study includes a MacBook Pro with16GB RAM and an Apple M1 Pro chip featuring an 8-core CPU, which hosts a virtual server acting as a subscriber. Thepublisher runs on a laptop with 16GB RAM and an Intel i7 CPU. Additionally, the MQTT broker is hosted on a cloud server in AWS, which has specifications of 1 vCPU and 2GB of RAM.

B. Software Setup

The software used in this study is based on Mosquitto MQTTversion 2.0.18 running on the AWS server. Also, the softwarepart contains Paho MQTT client of version 1.5.1 running as publisher and subscriber. The publish and subscriber arerunning Ubuntu 22.04 LTS while the broker is running Ubuntu 24.04 LTS. The analysis is performed by utilizing Ubuntu tools: perf and psutil.

C. Methodology

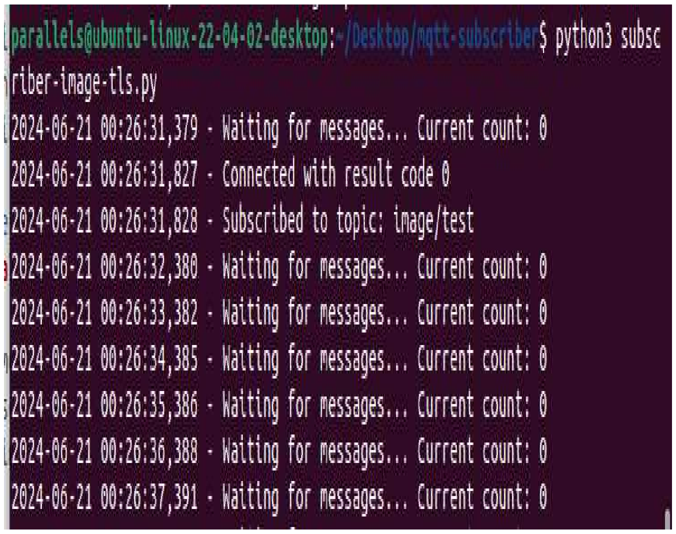
The publisher component is responsible for connecting to theMQTT broker, reading the image files, and encrypting them (if using TLS). Each encrypted image file is then published along with a timestamp. Throughout the publishing process, the CPU and memory usage are monitored and logged in thepublisher, i.e. representing the IoT device. The subscriber component connects to the MQTT broker andsubscribes to the topic. It receives the messages, decrypts them (if using TLS), and measures the latency. The latency ismeasured by comparing the current time with the timestamp sent with the message. This approach encountered synchronization issues between the publisher and subscriber systems, leading to inaccurate latency measurements.

Toresolve this, the subscriber was configured as a time server for the publisher to ensure accurate timestamp between the publisher and subscriber. The CPU and memory usage in the publisher are monitored using the 'psutil' and 'perf' Linux libraries. Data points are collected at regular intervals, and the average CPU and memory usage over the duration of the experiment are computed. The Latency is measured by calculating the time difference between the current time and the timestamp sent with each message. The latency values are stored, and the average latency is computed at the end of the experiment.

Thethroughput is calculated in terms of actual network load usingthe size of the messages sent over the elapsed time. This givesa more accurate representation of the performance impact onthe network. As shown in Fig 3, the subscriber is initiated first, establishing a connection with the MQTT broker and entering a state of readiness to receive messages. This proactive start ensures that the subscriber is fully prepared to handle incoming messages as soon as they are published.

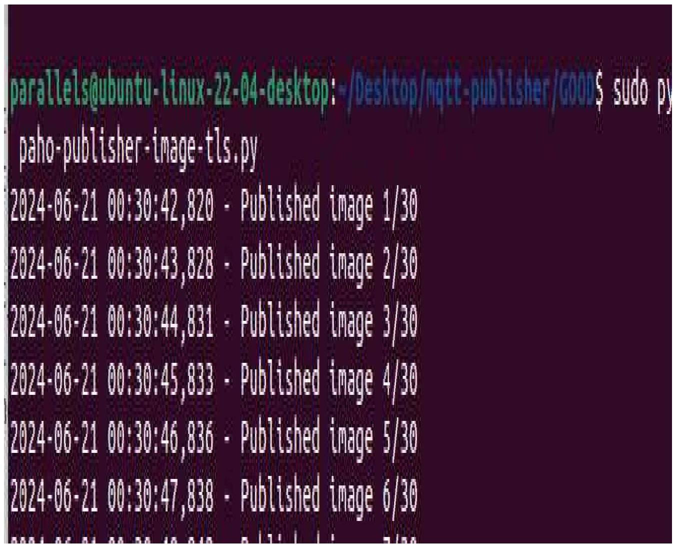
Once the publisher begins its operation, as illustrated in Fig 4, it connects to the same MQTT broker and starts publishing the pre-determined number of messages (100, 500, or 1000) to the designated topic. Each message, consisting of a 10KB file, is encrypted (in the case of TLS) and timestamped before being sent. As illustrated in Fig 5, the subscriber began processing messages after the publisher was initiated.

The test is conducted four times for each scenario, with the publisher sending 10, 50, 100 and 500 messages, respectively. As the publisher starts transmitting these messages, the subscriber, already in a listening state, promptly receives and processes each message. This immediate processing by the subscriber minimizes latency and ensures that messages are handled efficiently. The synchronization between the start of the subscriber and the subsequent message flow from the publisher is critical for the accurate measurement of performance metrics, such as latency, throughput, CPU, and memory usage.



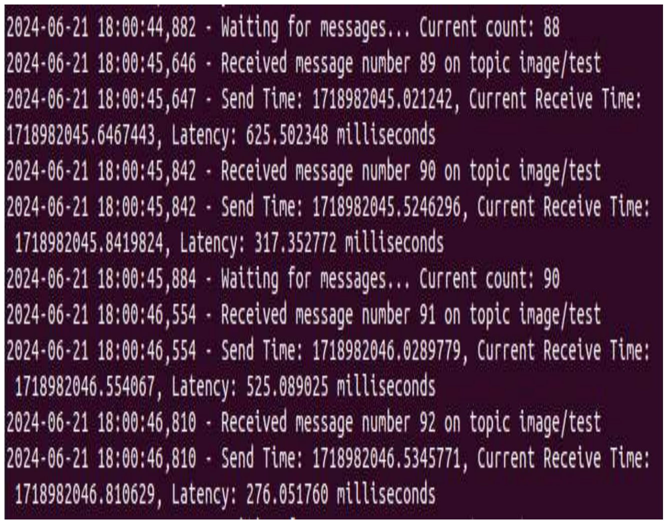
```
parallels@ubuntu-linux-22-04-02-desktop:~/Desktop/mqtt-subscriber$ python3 subsc
riber-image-tls.py
2024-06-21 00:26:31,379 - Waiting for messages... Current count: 0
2024-06-21 00:26:31,827 - Connected with result code 0
2024-06-21 00:26:31,828 - Subscribed to topic: image/test
2024-06-21 00:26:32,380 - Waiting for messages... Current count: 0
2024-06-21 00:26:33,382 - Waiting for messages... Current count: 0
2024-06-21 00:26:34,385 - Waiting for messages... Current count: 0
2024-06-21 00:26:35,386 - Waiting for messages... Current count: 0
2024-06-21 00:26:36,388 - Waiting for messages... Current count: 0
2024-06-21 00:26:37,391 - Waiting for messages... Current count: 0
```

Fig. 3. Subscriber waiting for messages



```
parallels@ubuntu-linux-22-04-desktop:~/Desktop/mqtt-publisher/GOOD$ sudo py
 paho-publisher-image-tls.py
2024-06-21 00:30:42,820 - Published image 1/30
2024-06-21 00:30:43,828 - Published image 2/30
2024-06-21 00:30:44,831 - Published image 3/30
2024-06-21 00:30:45,833 - Published image 4/30
2024-06-21 00:30:46,836 - Published image 5/30
2024-06-21 00:30:47,838 - Published image 6/30
```

Fig. 4. Publisher start publishing messages



```
2024-06-21 18:00:44,882 - Waiting for messages... Current count: 88
2024-06-21 18:00:45,646 - Received message number 89 on topic image/test
2024-06-21 18:00:45,647 - Send Time: 1718982045.021242, Current Receive Time:
1718982045.6467443, Latency: 625.502348 milliseconds
2024-06-21 18:00:45,842 - Received message number 90 on topic image/test
2024-06-21 18:00:45,842 - Send Time: 1718982045.5246296, Current Receive Time:
 1718982045.8419824, Latency: 317.352772 milliseconds
2024-06-21 18:00:45,884 - Waiting for messages... Current count: 90
2024-06-21 18:00:46,554 - Received message number 91 on topic image/test
2024-06-21 18:00:46,554 - Send Time: 1718982046.0289779, Current Receive Time:
 1718982046.554067, Latency: 525.089025 milliseconds
2024-06-21 18:00:46,810 - Received message number 92 on topic image/test
2024-06-21 18:00:46,810 - Send Time: 1718982046.5345771, Current Receive Time:
 1718982046.810629, Latency: 276.051760 milliseconds
```

Fig. 5. Subscriber processing messages and latency

To switch between TLS and non-TLS onfigurations in the broker, subscriber, and publisher, the use_tls parameter is utilized. When use_tls is set to True, the broker is configured to listen on port 8883 and use the specified TLS certificates. The subscriber and publisher are also set to connect to port 8883 and use the appropriate CA certificate for secure communication. Conversely, when use_tls is set to False, the broker listens on port 1883 without any TLS settings, and the subscriber and publisher connect to this non-secure port. For example, in the broker configuration file (mosquitto.conf), enabling TLS would involve setting the listener to port 8883 and specifying the paths to the CA certificate, server certificate, and server key:

```
listener 8883
allow_anonymous true
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/mosquitto.crt
keyfile /etc/mosquitto/certs/mosquitto.key
```

For non-TLS, the configuration would simply include:

```
listener 1883
allow_anonymous true
```

In the subscriber and publisher scripts, the use_tls parameter controls the configuration. When use_tls is True, the scripts set the port to 8883 and configure TLS settings. When use_tls is False, they connect to port 1883 without TLS settings.

```
use_tls = True # or False for non-TLS
port = 8883 if use_tls else 1883
ca_cert = "/path/to/ca.crt" if use_tls else None
```

This setup ensures that the system can flexibly switch between secure and non-secure communication modes by adjusting the use_tls parameter and the corresponding configuration settings in the broker, subscriber, and publisher.

IV. Discussion

A. Results

The results of the experiment include the average latency (milliseconds), throughput (bytes per second), CPU cycles and Peak RSS (Resident Set Size) for each scenario (10, 50,100 and 500 files each one 10KB of size). These metrics provide a comprehensive view of the efficiency and overhead introduced by TLS encryption compared to non-TLS communication for varying message volumes.

In Fig 6, the comparison of CPU cycles from publisher side is shown for each scenario (10, 50 , 100 and 500 files). The CPU cycles are generally higher for the TLS configuration, which reflects the increased computational load due to encryption. As the number of messages increases, the difference in CPU cycles between TLS and non-TLS configurations becomes more pronounced. Higher CPU cycles observed for TLS indicate more processing power is required for secure communication. This can lead to higher power consumption, which is a critical factor for battery-powered IoT devices.

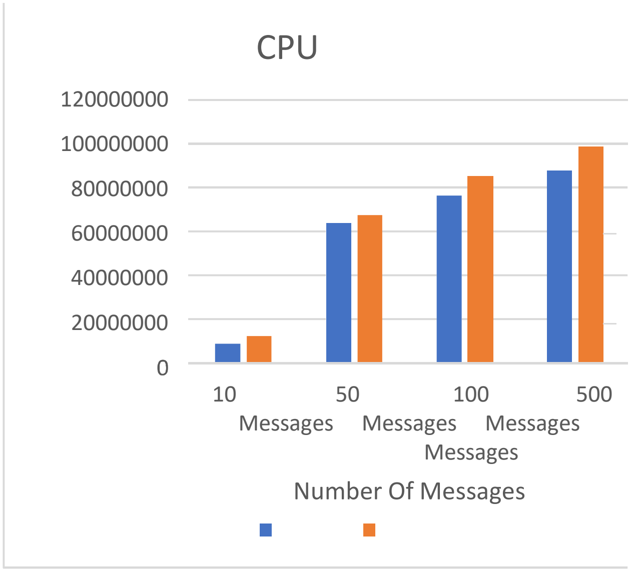


Fig. 6. Publisher CPU cycles comparisons

When evaluating the performance of IoT devices, Peak RSS (Resident Set Size) is a critical metric. It indicates the maximum amount of memory a process occupies during its execution. Given the constrained resources of IoT devices, managing memory usage effectively is essential to ensure reliable and efficient operation. In Fig7, the Peak RSS values are presented for non-TLS and TLS protocols over varying message counts. TLS introduces a clear increase in Peak RSS compared to non-TLS. The overall memory usage increase is approximately 1 MB. This difference, although modest on a laptop, can be significant for IoT devices with very limited memory.

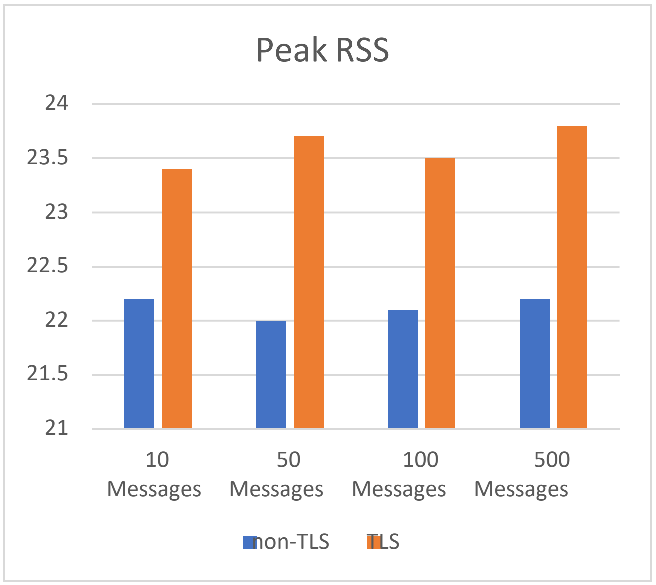


Fig. 7. Peak RSS comparison

For each message count (10, 50, 100, 500), the average latency is calculated by taking the average of five readings per message count. The results are illustrated in Fig.8, for lower message counts, the latency is observed to be higher. This can be attributed to the fact that when fewer messages are sent, each individual message may experience higher overhead relative to the overall operation time. The initial setup, connection time, and other fixed overheads have a more significant impact when spread across fewer messages. The average latency is generally higher for TLS compared to non-TLS. This is expected as TLS involves additional encryption and decryption processes which introduce more overhead.

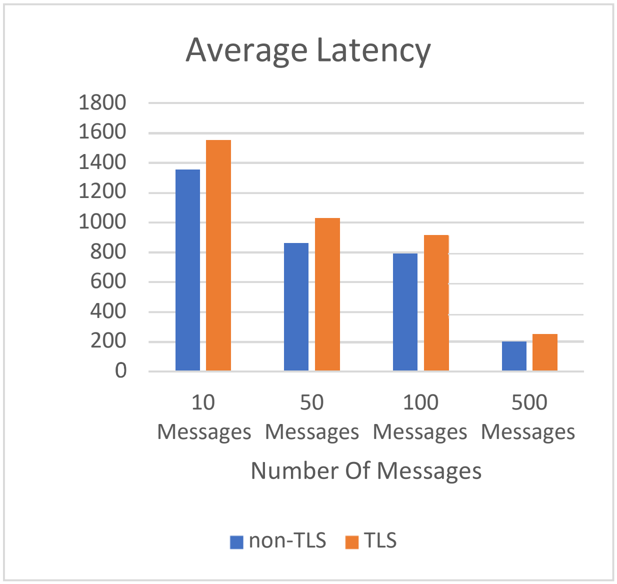


Fig. 8. Average latency comparison

For each message count (10, 50, 100,500), the average throughput is calculated by taking the average of five readings per message count. The results are illustrated in Fig.9, both TLS and non-TLS show a consistent increase in throughput as the number of messages increases. This suggests that the overhead of encryption in TLS does not significantly impact throughput as the message count increases. The throughput analysis indicates that the performance of TLS is very close to non-TLS, with only slight variations observed at higher message counts.

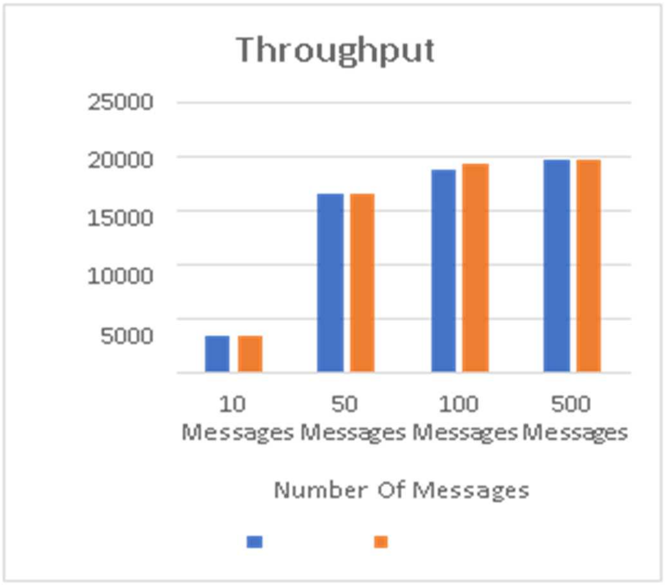


Fig. 9. Throughput comparison

B. *Limitations*

As previously mentioned, the absence of encryption in standard MQTT implementations poses significant risks to IoT systems. Attackers can exploit this vulnerability to manipulate MQTT messages, inserting false data or monitoring communications to compromise confidentiality and integrity. Such attacks in MQTT environments could lead to economic losses or unintended consequences, such as raising room temperatures, generating false alarms by altering application messages like door statuses, or exposing sensitive information like client and message identities that are not encrypted. As discussed in this paper, efforts are underway to develop lightweight and efficient encryption algorithms specifically tailored for securing the MQTT protocol. These algorithms aim to balance security, performance, and resource constraints inherent in IoT environments. However, as mentioned in Table 2, these efforts have many limitations:

- Not considering IoT-related factors such as power consumption, latency, and bandwidth.
- No end-to-end encryption solutions, such as protecting only the payload.
- No evaluation and comparison with other similar solutions.

C. *Future Work*

Future research efforts should prioritize the development of lightweight, cost-effective, fine-tuned, and high performing security solutions to enhance MQTT security. It is crucial to thoroughly evaluate these proposed solutions for scalability, compatibility, and adaptability to the resources available in IoT deployments, ensuring effective management of MQTT security challenges.

V. Conclusion

This paper details recent research efforts aimed at safeguarding data security through various encryption algorithms, with a particular emphasis on MQTT. It includes a comparative analysis of these algorithms, their respective testbeds, strengths, and weaknesses. Further research is essential to refine existing protocols to meet the evolving security and performance demands of IoT systems. Additionally, the paper presents an implementation of TLS in MQTT along with a performance analysis compared to standard MQTT in terms of CPU cycles, peak RSS, latency, and throughput. The use of TLS for secure communication introduces additional overheads in latency, CPU cycles, and memory usage. These overheads are significant for constrained IoT devices, as they can affect the performance and energy consumption of the devices. While non-TLS communication is faster and less resource-intensive, it does not provide the necessary security for sensitive data transmission.

References

[1] M. Abbasi, M. Plaza-Hernandez, J. Prieto, and J. M. Corchado, "Security in the Internet of Things Application Layer: Requirements, Threats, and Solutions," IEEE Access, vol. 10, pp. 97197–97216, 2022, doi: https://doi.org/10.1109/access.2022.3205351.

[2] Y. Harbi, Z. Aliouat, A. Refoufi, and S. Harous, "Recent Security Trends in Internet of Things: A Comprehensive Survey," IEEE Access, vol. 9, pp.113292–113314, 2021, doi: https://doi.org/10.1109/access.2021.3103725.

[3] M. N. Khan, A. Rao, and S. Camtepe, "Lightweight Cryptographic Protocols for IoT Constrained Devices: A Survey," IEEE Internet of Things Journal, vol. 8, no. 6, pp. 1–1, 2020, doi: https://doi.org/10.1109/jiot.2020.3026493.

[4] Hasan, H., Ali, G., Elmedany, W. and Balakrishna, C., 2022, November. Lightweight encryption algorithms for internet of things: a review on security and performance aspects. In 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT) (pp. 239-244). IEEE.

[5] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker, "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," IEEE Access, vol. 9, pp. 28177–28193, 2021, doi: https://doi.org/10.1109/access.2021.3052867.

[6] Iordan Stoev, Snezhinka Zaharieva, A. Borodzhieva, and Gergana Staevska, "An Approach for Securing MQTT Protocol in ESP8266 WiFi Module," Jul. 2020, doi: https://doi.org/10.1109/electronica50406.2020.9305164.

[7] Jahangeer, A., Bazai, S. U., Aslam, S., Marjan, S., Anas, M., & Hashemi, S. H. A review on the security of IoT networks: From network layer's perspective. IEEE Access, 11, 71073-71087, 2023.

[8] Alluhaidan, A. S. and Prabu, P., 'End to End encryption in resource-constrained IoT device', IEEE Access, 2023.

[9] A. Behal, "Comparison Of MQTT And COAP Protocol Using Contiki OS For Low Power And Lossy Networks In IoT," Jul. 2023, doi: https://doi.org/10.1109/icccnt56998.2023.10307381.

[10] G. Glissa and A. Meddeb, "6LowPSec: An end-to-end security protocol for 6LoWPAN," Ad Hoc Networks, vol. 82, pp. 100–112, Jan. 2019, doi: https://doi.org/10.1016/j.adhoc.2018.01.013.

[11] Aruna Gawde, Nishit Sakariya, A. Shah, and Dishith Poojary, "Lightweight Authentication and Encryption Mechanism in Routing Protocol for Low Power and Lossy Networks (RPL)," Jun. 2018, doi: https://doi.org/10.1109/iccons.2018.8663178.

[12] S. Katsikeas et al., "Lightweight amp; secure industrial IoT communications via the MQ telemetry transport protocol," IEEE Xplore, Jul. 01, 2017. https://ieeexplore.ieee.org/abstract/document/8024687.

[13] L. Bisne and M. Parmar, "Composite secure MQTT for Internet of Things using ABE and dynamic S-box AES," 2017 Innovations in Power and Advanced Computing Technologies (i-PACT), Apr. 2017, doi: https://doi.org/10.1109/ipact.2017.8245126.

[14] Nayancy, S. Dutta, and S. Chakraborty, "A survey on implementation of lightweight block ciphers for resource constraints devices," Journal of Discrete Mathematical Sciences and Cryptography, pp. 1–22, Jun. 2020, doi: https://doi.org/10.1080/09720502.2020.1766764.

[15] Fan, C. et al. (2023) 'An Efficient Data Protection Scheme Based on Hierarchical ID-Based Encryption for MQTT', ACM Transactions on Sensor Networks (TOSN), 19(3), pp. 21.

[16] A. J. Hintaw, S. Manickam, S. Karuppayah, M. A. Aladaileh, M. F. Aboalmaaly, and S. U. A. Laghari, "A Robust Security Scheme Based on Enhanced Symmetric Algorithm for MQTT in the Internet of Things," IEEE Access, vol. 11, pp. 43019–43040, 2023, doi: https://doi.org/10.1109/ACCESS.2023.3267718.

[17] A. Rahman, S. Roy, M. S. Kaiser, and Md. S. Islam, "A Lightweight Multi-tier S-MQTT Framework to Secure Communication between low-end IoT Nodes," in Proceedings of the 5th International Conference on Networking, Systems and Security (NSysS), pp. 1-6, 2018,IEEE.

[18] M. Iqbal, Agus, Aji Teguh Prihatno, Derry Pratama, B. Jeong, and H. Kim, "Enhancing IoT Security: Integrating MQTT with ARIA Cipher 256 Algorithm Cryptography and mbedTLS," Aug. 2023, doi: https://doi.org/10.1109/platcon60102.2023.10255171 .

[19] Ardhi Wijayanto, Saffira Syafa Nugrahani, Dewi Wisnu Wardani, Hasan Dwi Cahyono, and Haryono Setiadi, "Performance Comparison of AES, Grain V1, and RC4 Algorithms on the MQTT Protocol," Aug. 2023, doi: https://doi.org/10.1109/icitacee58587.2023.10277160.

[20] F. De Rango, G. Potrino, M. Tropea, and P. Fazio, "Energy-aware dynamic Internet of Things security system based on Elliptic Curve Cryptography and Message Queue Telemetry Transport protocol for mitigating Replay attacks," Pervasive and Mobile Computing, vol. 61, p. 101105, Jan. 2020, doi: https://doi.org/10.1016/j.pmcj.2019.101105.

[21] A. Hkiri, Mouna Karmani, and Mohsen Machhout, "Implementation and Performance Analysis of Lightweight Block Ciphers for IoT applications using the Contiki Operating system," May 2022, doi: https://doi.org/10.1109/setit54465.2022.9875503.

[22] Ayman Al-Ani, Wong Kang Shen, A. K. Al-Ani, S. A. Laghari, and O. E. Elejla, "Evaluating Security of MQTT Protocol in Internet of Things," Sep. 2023, doi: https://doi.org/10.1109/ccece58730.2023.10288857.

[23] Jayan, A. P. and Harini, N, 'A Scheme to Enhance the Security and Efficiency of MQTT Protocol', International Journal of Pure and Applied Mathematics, 119(12), 2018, pp. 13975–13982.

[24] M. Hamad, A. Finkenzeller, H. Liu, J. Lauinger, V. Prevelakis, and S. Steinhorst, "SEEMQTT: Secure End-to-End MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation," IEEE Internet of Things Journal, pp. 1–1, 2022, doi: https://doi.org/10.1109/jiot.2022.3221857.

[25] Varma, A. and UniKrishnan, S. (2021) 'Effect of payload security in MQTT protocol over transport and application layer', in IOP Conference Series: Materials Science and Engineering, 1166(1), p. 012019.

[26] Favian Dewanta, I. Wahidah, Sofia Naning Hertiana, Danu Dwi Sanjoyo, and H. Syed, "SibProMQTT: Protection of the MQTT Communication Protocol Against Sybil Attacks Applied for IoT Devices," Sep. 2023, doi: https://doi.org/10.1109/icicdt59917.2023.10332423.

[27] O. Sadio, I. Ngom, and C. Lishou, "Lightweight Security Scheme for MQTT/MQTT-SN Protocol," 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Oct. 2019, doi: https://doi.org/10.1109/iotsms48152.2019.8939177.

[28] Singh, M., Rajan, M. A., Shivraj, V. L. and Balamuralidhar, P. 'Secure MQTT for Internet of Things (IoT)', Proceedings of the 5th International Conference on Communication Systems and Networks Technologies, 2015, pp. 746-751.

[29] Ahmad, M. Z. et al. 'Performance Analysis of Secure MQTT Communication Protocol', Proceedings of the 2023 19th IEEE International Colloquium on Signal Processing & Its Applications (CSPA), Kedah, Malaysia, 2023,pp. 225-229.

[30] I. Sahmi, A. Abdellaoui, T. Mazri, and N. Hmina, "MQTT-PRESENT: Approach to secure internet of things applications using MQTT protocol," International Journal of Electrical and Computer Engineering (IJECE), vol. 11, no. 5, p. 4577, Oct. 2021, doi: https://doi.org/10.11591/ijece.v11i5.pp4577-4586.