Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/190402

Lightweight Security Scheme for Internet of Things Encryption

Saud S. Alharbi[1,*], *David Bell* [1] *and Wasan Awad* [2]

[1]Department of Computer Science, College of Engineering, Design and Physical Sciences, Brunel University, London, United Kingdom
[2]College of Information Technology, Ahlia University, Manama, Kingdom of Bahrain

Received: 28 Jan. 2025, Revised: 10 Feb. 2025, Accepted: 28 Feb. 2025
Published online: 1 Jul. 2025

Abstract: In this paper, we propose a lightweight security scheme for IoT communications, optimized for resource-constrained devices and integrated with the MQTT protocol to enhance secure data transmission. Nowadays, the encryption of Internet of Things (IoT) traffic demands thorough performance analysis, driven by the inherent security challenges arising from the resource limitations of IoT devices. Traditional security protocols are often too resource-intensive for these constrained nodes. To ensure end-to-end and lightweight encryption, this paper proposes a lightweight security scheme using the Salsa20 encryption protocol to provide confidentiality and the modern hashing algorithm Blake2b to ensure message integrity of the payload. The new scheme was implemented and integrated with the MQTT protocol, which is considered the de facto standard for IoT communications. The publisher represents a resource-constrained IoT device, which was the focus of all analyses conducted. The study presents a comprehensive analysis of different implementations using 256-bit key and 128-bit key, if supported by the protocol. The following protocols were included in the study: AES-EAX, AES-GCM, AES-CCM8, TinyAES-HMAC, ASCON, SIMON-HMAC, SPECK-HMAC, and ChaCha20-Poly1305. The new scheme was compared against the included protocols in terms of CPU cycles, encryption time, and throughput. The results demonstrate that the proposed scheme consumes fewer CPU cycles, indicating lower power usage, requires less encryption time, reflecting faster processing, and achieves higher throughput, further highlighting its superior speed compared to other protocols

Keywords: IoT data security, secure MQTT protocol, TLS, MQTT encryption, lightweight IoT encryption, IoT application layer

1 Introduction

The Internet of Things (IoT) ecosystems have increasingly adopted the MQTT (Message Queue Telemetry Transport) protocol as the de facto standard for communication due to its lightweight nature and efficiency in resource-constrained environments. MQTT operates on a publisher-subscriber model, enabling asynchronous communication between devices and systems. While the MQTT standard includes support for Transport Layer Security (TLS) to provide secure communication, numerous research studies have demonstrated that TLS is not well-suited for IoT applications, primarily due to the significant overhead it introduces to MQTT traffic.

For instance, Alharbi et al. [1] conducted a comparative analysis of MQTT with TLS and MQTT without TLS, highlighting that the addition of TLS imposes considerable computational and memory burdens on resource-constrained IoT nodes, thereby degrading performance. These findings underscore the need for alternative security mechanisms that maintain a balance between robust encryption and resource efficiency.

In this context, this paper proposes a novel security scheme designed to enhance the security of MQTT communications while addressing the limitations associated with IoT device resources constraints. The proposed scheme leverages the lightweight and efficient Salsa20 encryption algorithm to ensure data confidentiality, complemented by the Blake2b hashing

* Corresponding author e-mail: saud.alharbi@brunel.ac.uk

algorithm to provide data integrity and authentication. By integrating these cryptographic techniques, the proposed solution aims to achieve a secure, resource-optimized approach suitable for IoT devices with constrained computational capacities.

The main contributions of this article include the implementation of an experimental setup to evaluate the performance of different encryption protocols with MQTT as the underlying M2M protocol. The article also analyzes the performance overhead introduced by these protocols in MQTT-based systems. Additionally, various encryption protocols were implemented and integrated into the publisher and subscriber components of MQTT. Finally, a novel security scheme was presented, designed to ensure data confidentiality and integrity while being suitable for resource-constrained publishers.

The paper is organized as follows. Section II presents the background of MQTT security, highlighting its vulnerabilities and the Salsa20-Blake2b security scheme. Section III reviews existing literature on security schemes for MQTT in resource-constrained IoT environments. Section IV discusses the research methodology, including the testbed setup and performance evaluation of various encryption protocols. Section V details the implementation of the MQTT publisher and the proposed Salsa20-Blake2b security scheme. Finally, Section VI concludes the paper and suggests future research directions for enhancing IoT security.

2 Theoretical Background

2.1 MQTT Security

MQTT (Message Queue Telemetry Transport) is a lightweight messaging protocol widely [3] used in IoT ecosystems for its efficiency and low resource requirements. It operates on a publish-subscribe model, the publisher send data to subscriber through a broker on a particular topic, making it good fit for IoT environments with limited resources. Its small footprint and simplicity have made it a de facto standard for IoT communication. However, MQTT has no security by default. By design, it is a plain-text protocol [3], [4] that does not enforce any encryption or authentication mechanisms, leaving security implementation to vendors. While the MQTT standard recommends using Transport Layer Security (TLS) to protect data in transit, TLS introduces significant overhead [1], making it unsuitable for many resource-constrained IoT devices. Additionally, TLS does not provide end-to-end encryption, as data is decrypted at the broker, creating potential vulnerabilities. These limitations make MQTT deployments susceptible to various threats, including eavesdropping, data tampering, and unauthorized access, underscoring the need for lightweight and efficient security solutions tailored to IoT's unique constraints.

2.2 Security Scheme

Salsa20 is a symmetric key stream cipher designed by Daniel J. Bernstein, known for its simplicity and efficiency [2]. It is widely regarded as a fast and secure algorithm suitable for environments with constrained resources, such as IoT devices. Salsa20 operates on 256-bit keys (or 128-bit keys in the case of Salsa20/12) and uses a 64-bit nonce for each encryption, ensuring the uniqueness of the output stream for each encryption operation. Salsa20 takes the input consisting of the key, nonce, and a counter, and mixes these values using a series of XOR operations, rotations, and modular additions to generate the keystream. It has been extensively analyzed and is considered secure against differential and linear cryptanalysis. No significant cryptographic weaknesses have been discovered, making it an attractive option for secure communications.

Blake2b is a cryptographic hash function designed [21] as a faster and more secure alternative to MD5 and SHA-2. It belongs to the family of Blake cryptographic functions, which were finalists in the SHA-3 competition. Blake2b was designed to be fast in both hardware and software, offering a high level of security with a straightforward design.

3 Literature Review

Encryption is a critical component of IoT communication, ensuring the protection of confidential data from unauthorized access or tampering. As a result, numerous research efforts have focused on addressing this challenge and developing practical solutions for resource-constrained IoT devices. A commonly used method to secure MQTT-based IoT traffic is through the TLS protocol. However, the resource limitations of IoT MQTT publishers often lead to significant overhead when using TLS. Additionally, the broker must decrypt incoming messages, re-encrypt them, and forward them to subscribers, which could introduce a potential security risk if the broker is compromised [5]. To achieve the goal of end-to-end encryption, many proposed new solutions to encrypt the payload of the MQTT mainly using symmetric protocols.

Numerous lightweight algorithms have been proposed by researchers while accommodating the resource constraints of IoT devices. Singh et al. [6] introduced the SMQTT and SMQTT-SN security schemes, which leverage Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE) combined with lightweight Elliptic Curve Cryptography. Their study included a security analysis, demonstrating resilience against various types of attacks. However, the performance evaluation fell short, as it did not provide a comparison with other protocols or a practical analysis on constrained nodes. Consequently, the suitability of the proposed security

scheme for resource-constrained IoT devices remains unproven.

Similarly, Sadio et al. [7] proposed the use of the ChaCha20-Poly1305 AEAD to secure communication over MQTT/MQTT-SN in constrained environments. ChaCha20, a lightweight stream cipher, and Poly1305, a one-time authenticator, formed the basis of their security scheme. The proposed solution was prototyped on constrained devices like the Arduino UNO, and its performance was primarily evaluated in terms of memory footprint and execution time. The results indicated low memory requirements and minimal processing overhead. However, the evaluation was limited to only two performance metrics: memory and payload size. This limited scope, along with the absence of comparisons with similar solutions, makes it difficult to fully assess the scheme's practicality and effectiveness for constrained nodes. Pal et al. [8] and Wang et al. [9] proposed encryption solutions based on Attribute-Based Encryption (ABE) to ensure data confidentiality. However, these approaches introduce significant overhead due to the use of the ABE protocol, potentially rendering them unsuitable for resource-constrained publishers.

Iqbal et al. [10] proposed a novel security scheme for MQTT, employing the ARIA encryption algorithm for securing the MQTT payload and MbedTLS for network tunnel encryption. While the solution introduces a unique approach, the study lacked any comparative analysis with existing methods, leaving its relative performance and applicability to IoT environments unexplored.

Hintaw et al [11] presented a new protocol derived from the standard AES. RSS introduces a new design architecture of the symmetric AES algorithm to encrypt the MQTT payload called D-AES. Results revealed that the proposed D-AES is more promising with improvements than the standard AES algorithm. Other researchers proposed similar solution derived from AES that were integrated utilized to secure MQTT payload. Another similar solution was presented by Bisne et al. [13] Attribute-based Encryption (ABE) and Dynamic S-Box Advanced Encryption Standard (AES) were applied for payload encryption in MQTT. The various modes of AES (GCM, EAX, CCM8) were compared with the proposed security scheme, demonstrating that the new scheme is more suitable for resource-constrained publishers.

Wijayanto et al. [12] evaluated the effectiveness of AES, Grain V1, and RC4 algorithms in countering passive sniffing attacks and their performance in terms of data processing time. The results demonstrated that all three algorithms effectively mitigated passive sniffing attacks. Additionally, the study examined the resistance of these algorithms to cryptanalysis by comparing the time required to break their keys. The findings indicated that AES provided the highest level of resistance, making it the most secure option compared to Grain V1 and RC4. Performance testing was also conducted to measure the encryption and decryption processing times of each algorithm. RC4 emerged as the fastest for encryption, with an average time of 20.4 microseconds, followed by Grain V1 at 763.4 microseconds, and AES at 796.84 microseconds. Similarly, for decryption, RC4 achieved the best performance with an average time of 0.13 microseconds, followed by Grain V1 at 0.18 microseconds, and AES at 1.16 microseconds.

Table 1 provides a comparative analysis of other various security protocols designed for MQTT-based Pub/Sub communication. The comparison highlights key features, protocols used, drawbacks, primary services offered, and their suitability for IoT-constrained nodes. The proposed protocol using Salsa20 and Blake2B demonstrates significant advantages for edge environments, particularly in controlled settings. The proposed protocol employs the lightweight encryption algorithm Salsa20 for confidentiality and the hashing algorithm Blake2B for integrity. It is specifically designed for deployment in edge environments with physically secure local brokers. Unlike computationally intensive protocols such as CP-ABE or IBE, the proposed solution minimizes computational overhead, making it highly suitable for resource-constrained IoT devices. The protocol avoids reliance on trusted third parties, such as PKG or CA, and simplifies key management by eliminating the need for complex schemes like secret sharing or proxy re-encryption.

A review of the current literature reveals a wide array of encryption solutions aimed at securing MQTT-based IoT communication. Existing approaches typically employ TLS for securing traffic or adopt various symmetric encryption methods, such as lightweight versions of AES, ChaCha20-Poly1305, ARIA, and even attribute-based schemes like ABE. However, several recurring challenges have emerged. Traditional TLS introduces substantial computational and memory overhead, particularly for IoT publishers with constrained resources. Similarly, while approaches using ABE or re-encryption at the broker (e.g., [5], [8], [9]) enhance security, they often incur heavy processing costs and complicate key management.

By leveraging Salsa20 and Blake2B, the protocol avoids the computational intensity of schemes like CP-ABE and IBE. This not only ensures better performance on devices with limited processing power but also simplifies key management by eliminating the need for third-party authorities or complex re-encryption schemes.

Table 1: Comparative Analysis Of Different Protocols

Reference	Description	Protocols Used	Drawbacks	Main Services	Suitability for IoT Constraint Nodes
Lee et al. [14]	Proposed MQTLS, an extension of TLS, for E2E security via an untrusted broker.	TLS, Diffie–Hellman (DH)	High overhead for mobile IoT devices; requires per-subscriber authentication and key exchange.	E2E confidentiality, secure key exchange.	Limited due to high computational and communication overhead.
Peng et al. [15]	Proposed IBE-based Pub/Sub system using a third-party PKG.	Identity-Based Encryption (IBE)	Does not ensure E2E encryption; requires trusted third-party PKG.	Confidentiality via IBE.	Limited due to reliance on third-party PKG.
Chien et al. [16]	Proposed a trusted broker for secure group communication using topic keys.	Symmetric Encryption (Topic Keys)	Relies on a trusted broker; lacks message integrity.	Group confidentiality.	Moderate, but depends on broker trustworthiness.
Mektoubi et al. [17]	Used a CA to create topic certificates and private keys for communication.	Certificate Authority (CA), Symmetric Encryption	Manual distribution of keys and certificates; difficult to update periodically.	Confidentiality, topic-based access control.	Limited due to manual key management.
Dahlmanns et al. [20]	Used a key server for pairwise PSK setup and session key generation.	Symmetric Encryption, PSK	Relies on a single key server; prone to single point of failure.	E2E confidentiality.	Moderate, depending on server reliability.
Borcea et al. [18]	Proposed proxy re-encryption (PRE) for E2E security without prenegotiation.	Asymmetric Encryption, Proxy Re-encryption	Broker requires re-encryption capabilities; lacks message integrity.	E2E confidentiality.	Limited due to asymmetric encryption overhead.
Hamad et al. [19]	Used secret splitting and RSA for distributing topic keys via KeyStores.	Secret Splitting, RSA	Overhead from certificate verification; requires all KeyStores for key reconstruction.	Confidentiality, access control.	Limited due to overhead and KeyStore reliance.
Singh et al. [6]	Proposed CP-ABE and KP-ABE for secure MQTT communication.	Ciphertext Policy ABE (CP-ABE), Key Policy ABE (KP-ABE)	High computational overhead due to pairing operations.	Confidentiality, access control.	Limited due to high computational requirements.
Pal et al. [8]	Used CP-ABE for content-based Pub/Sub encryption.	Ciphertext Policy ABE (CP-ABE)	Massive overhead from ABE operations.	Confidentiality, access control.	Limited due to computational complexity.
Wang et al. [9]	Proposed encrypting symmetric keys with ABE for secure data encryption.	Attribute-Based Encryption (ABE), Symmetric Encryption	High overhead from pairing operations in ABE.	Confidentiality, access control.	Limited due to ABE complexity.
Our new protocol	Proposed a new MQTT payload lightweight encryption scheme using Salsa20 and Blake2B	Salsa20 and Blake2B	Limited to installations at the edge with secure and controlled environment hosting local broker server	Confidentiality, Integrity	Suitable due to low computational overhead.

4 Research Methodology

4.1 Introduction

The methodology section is structured to comprehensively evaluate the performance of various encryption protocols for securing IoT communications using MQTT. It encompasses software and hardware components designed to replicate real-world IoT scenarios, along with a detailed description of the publisher's role, which simulates the behavior of resource-constrained IoT devices. The aim is to assess the computational efficiency of the protocols under test.

The software layer integrates a custom-developed MQTT publisher, which is implemented in Python and augmented with various encryption algorithms such as AES-GCM, ChaCha20-Poly1305, SPECK-HMAC, and others. The publisher encrypts image payloads of varying sizes (e.g., 1KB, 5KB, and 10KB) and transmits them to an MQTT broker. Performance metrics such as CPU cycles, encryption time, memory usage, and throughput are captured using tools like perf and psutil.

The hardware testbed is designed to emulate IoT environments. The publisher is deployed on a laptop running Ubuntu OS, utilizing accurate measurements of CPU cycles, RSS, and throughput, and encryption time directly measured in the code, to reflect the computational performance and energy constraints typical of IoT nodes.

The MQTT broker, which acts as the intermediary for message distribution, is hosted on a cloud platform. The subscriber, representing the end-user or application, is implemented on a mobile device or a PC for testing the decryption and performance at the receiving end.

At the heart of the system, the publisher plays a crucial role as the IoT data generator. It represents a resource-constrained device capable of encrypting and sending data to the broker. By simulating an IoT device, the publisher provides a realistic benchmark for evaluating the efficiency of encryption protocols under limited resources. The publisher encrypts image files using the selected protocol, with the process repeated for a specified number of iterations (e.g., 100 messages per protocol). It then logs performance metrics and securely sends the encrypted payloads to the broker, enabling end-to-end testing of the encryption schemes. This integrated approach ensures a robust evaluation of encryption protocols, addressing both the performance and security aspects critical for IoT applications. The methodology establishes a foundation for comparing various protocols and identifying the most suitable options for secure and efficient IoT communication.

4.2 Testbed

We have deployed and evaluated our implementation using the testbed that is shown in Fig. 1. The testbed is an essential component of the experimental framework, designed to emulate a real-world IoT environment for evaluating encryption protocols. In this setup, the publisher represents an IoT resource-constrained device which is responsible for generating and encrypting data before transmitting it to the cloud or other devices through an MQTT broker. The testbed is structured to simulate conditions that IoT devices typically face, including limited computational power, restricted memory, and constrained energy resources.

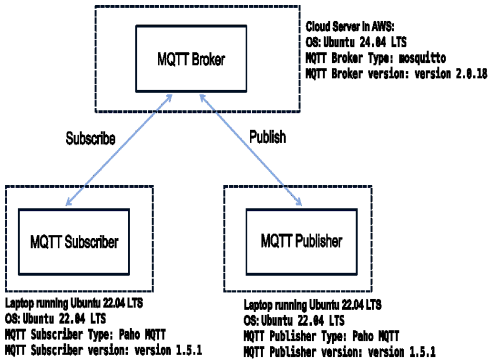

Fig. 1: Testbed Hardware And Software Components

We have deployed and evaluated our implementation using the testbed illustrated in Fig. 1. This testbed is a cornerstone of our experimental framework, designed to emulate a real-world IoT environment for assessing encryption protocols under resource-constrained conditions and realistic environmental scenarios.

4.2.1 -Hardware Specifications:

–Laptops:
Two laptops, each running Ubuntu 22.04 LTS, serve as the MQTT Publisher and MQTT Subscriber. Their hardware configurations include:

- –Processor: Intel Core i5 (quad-core, approximately 2.6GHz)
- –Memory: 8GB DDR4 RAM
- –Storage: 256GB SSD
- –Network Interface: Dual-band Wi-Fi adapters supporting both 2.4GHz and 5GHz bands to ensure stable and high-speed wireless connectivity.

–Cloud Server:
The MQTT broker is hosted on an AWS cloud server running Ubuntu 24.04 LTS with Mosquitto version 2.8.16. The server's specifications are:

- –Processor: Intel Xeon (4 cores at 2.3GHz)
- –Memory: 16GB RAM
- –Storage: Enterprise-grade SSD
  This configuration ensures that the broker can handle high message throughput and maintain reliable performance under load.

The MQTT broker is hosted on the AWS cloud server, it centrally manages all message exchanges, ensuring that both publishing and subscribing processes can be evaluated under consistent conditions. The MQTT Publisher (one of the laptops) utilizes the Paho MQTT client library (version 1.5.1) to generate and encrypt messages before sending them to the broker. The MQTT Subscriber (the second laptop) receives and processes these messages, enabling end-to-end performance and security analysis.

All experiments were conducted in a controlled lab setting where the ambient temperature was maintained at $22\pm2^{\circ}$C, and relative humidity levels were held within the 45–55% range. This control minimizes environmental variability and potential interference that could affect wireless communication. Both laptops and the simulated IoT device were connected to the same dedicated Wi-Fi network.

5 MQTT Publisher Implementation

The publisher serves as the central entity for encryption and message publishing. Its role is to process image data (used as payloads) and apply various encryption protocols to secure the data before sending it to the MQTT broker. By simulating an IoT device, the publisher implementation mirrors the

performance limitations and challenges associated with real-world IoT deployments.

The choice of image files in this testbed, with sizes such as 1KB, 5KB, and 10KB, represents typical payload sizes in IoT scenarios, ranging from small telemetry data to larger payloads like real images. By iterating over these different file sizes, the testbed assesses how protocol performance scales with increasing payload complexity. This scalability analysis is crucial, as IoT devices often need to balance security requirements with resource constraints.

In this setup, the publisher's primary functions include:

Data Encryption: Using various protocols, such as AES-GCM, ChaCha20-Poly1305, and SPECK-HMAC, the publisher encrypts image data to ensure confidentiality. Each protocol is tested iteratively to gather performance metrics.

Performance Measurement: Through integration with tools like perf and psutil, the publisher monitors and logs critical performance metrics, including encryption time, CPU cycles, memory usage, and throughput. These metrics are crucial for determining the suitability of encryption protocols for IoT use cases.

Message Publishing: Encrypted data is packaged and sent to the MQTT broker. The use of MQTT, a lightweight messaging protocol, aligns with the energy-efficient and low-overhead communication requirements of IoT networks.

The testbed's flexibility also allows for additional configurations, such as using TLS for secure transport or testing non-TLS scenarios. By enabling different configurations, the testbed evaluates the impact of transport layer security on overall performance and energy consumption.

In summary, the publisher simulates a constrained IoT device within the testbed, encapsulating the challenges and trade-offs faced in real-world deployments. Its role as the data generator, encrypter, and transmitter is pivotal in benchmarking encryption protocols and assessing their feasibility for secure IoT communications. This setup provides insights into protocol efficiency and helps determine the optimal choice of encryption for IoT applications, considering the constraints of computational resources, memory, and energy.

The publisher code is designed to encrypt and publish image data over MQTT while dynamically supporting multiple encryption protocols. Its primary focus is on evaluating the performance of these protocols under IoT conditions. The code starts by accepting configuration inputs, such as the encryption method and whether TLS is used. These parameters are passed as command-line arguments, enabling flexibility in choosing the protocol dynamically. For example, protocols like AES-GCM, ChaCha20-Poly1305, Salsa20, and SPECK can be selected at runtime, and each protocol has a specific implementation logic in the encrypt_payload function.

The script reads image data in binary format from a specified file, which serves as the payload for encryption. The selected encryption protocol determines how the payload is processed. For instance, AES-GCM and ChaCha20-Poly1305 use authenticated encryption with associated data (AEAD) for both confidentiality and integrity. Block ciphers like SIMON and SPECK encrypt data in chunks, with optional HMAC appended to ensure integrity.

The encrypted payload is packaged into a JSON object along with metadata like timestamps. This object is published to the MQTT broker under a predefined topic. The publishing process is iterative, with a specified number of messages being sent per protocol, allowing for a robust performance evaluation. Each iteration measures metrics such as encryption time, throughput, and memory usage. These are gathered using the Linux perf tool for CPU and cache metrics and the psutil library for memory monitoring. The script integrates performance monitoring by invoking perf during execution. It tracks task execution time, CPU cycles, cache references, and cache misses, while also logging average and peak memory usage during the encryption process. These metrics are saved to a CSV file for further analysis, enabling a detailed comparison of the protocols.

To ensure consistent and statistically significant results, the code is embedded in a shell automation script that iterates through all supported protocols. Each protocol is tested multiple times with different file sizes, such as 1KB, 5KB, and 10KB. This iterative testing ensures that variations due to transient system conditions are mitigated, and reliable averages for performance metrics can be derived.

Overall, the publisher modified Python code is a comprehensive tool for evaluating the efficiency of various encryption protocols in IoT settings. It dynamically supports multiple protocols, integrates with system performance tools, and automates iterative testing across different file sizes, making it ideal for large-scale performance analyses.

6 Results And Analysis

6.1 Introduction

The results section presents analyzing the performance of different cryptographic protocols in the context of IoT (Internet of Things) communication. Given the constraints of IoT devices, which often suffer from limited resources (e.g., processing power, memory, energy), this research investigates the efficiency of various encryption protocols when applied to IoT traffic. These protocols are tested to ensure confidentiality and integrity while considering the limited resources available on IoT devices.

The study proposes a lightweight security scheme, integrating the Salsa20 encryption algorithm and the Blake2b hashing algorithm, offering a balance between speed and security. The focus is primarily on the publisher (i.e., the resource-constrained IoT device), assessing how well the encryption performs under varying conditions. The study evaluates the following cryptographic protocols, each with its distinct advantages and trade-offs:

–AES-EAX
–AES-GCM
–AES-CCM8
–TinyAES-HMAC
–ASCON
–SIMON-HMAC
–SPECK-HMAC
–ChaCha20-Poly1305
–Salsa20-Blake2b (Proposed Scheme)

These protocols were chosen for their varied computational complexities, key sizes, and security properties. The goal was to assess how each protocol performs in terms of CPU cycles, encryption time, and throughput, which are critical for resource-constrained IoT devices.

6.2 Performance Analysis And Discussion

The dataset included in the performance analysis contains the following columns:

- –TimeStamp: The timestamp of the performance measurement.
- –Elapsed Time: The total time taken to encrypt the data.
- –CPU Cycles: The total number of CPU cycles consumed during encryption.
- –Throughput: The rate at which data is processed (typically in KB/s or similar units).
- –Number Of Messages (Images): The number of images processed during the encryption process.
- –Protocol: The cryptographic protocol used for the encryption (e.g., AES-GCM, Salsa20, etc.).
- –Image Size: The size of the image being encrypted (1KB, 5KB, or 10KB).

The dataset tracks performance across three image sizes (1KB, 5KB, and 10KB) to assess how each protocol scales with increasing data sizes, which is critical for understanding the practical performance of these protocols in real-world IoT applications. The analysis aimed to evaluate and compare cryptographic protocols by following the below steps:

- –Data Aggregation: The analysis began by aggregating the raw data based on protocol and image size. This enabled a fair comparison of the protocols under consistent conditions. Appendix A contains the complete dataset, which includes the average results from ten runs for each protocol. Each run involved encrypting 100 messages, with performance measured throughout.

- –Metric Calculation: We focused on three performance metrics:
  - –CPU Cycles: Measures the computational load for each protocol.
  - –Elapsed Time: Represents the time taken for encryption, where lower values indicate faster performance.
  - –Throughput: Indicates the amount of data processed per unit of time, where higher throughput signifies better efficiency.

- –Visualization: After calculating the average for each metric (CPU Cycles, Elapsed Time, and Throughput) across the different image sizes, we created visualizations (charts) to make the comparisons more intuitive. The charts allow us to observe trends and differences between protocols and how they perform with different image sizes. Table 2 and Table 3. showed the final results by comparing different protocols using 128-bit key and 256-bit key with the new scheme.

- –Insights: The analysis provided insights into the relative performance of each protocol, allowing for conclusions about the most efficient cryptographic schemes for IoT devices with limited resources. In particular, the proposed Salsa20-Blake2b scheme demonstrated superior performance in terms of faster encryption times, lower CPU cycles, and higher throughput compared to the other protocols.

6.3 Rationale for Dual Key Sizes

In our experimental framework, we evaluated encryption protocols using both 128-bit and 256-bit key sizes. This dual-key approach serves two main objectives:

- –Security Margin:
  - –128-bit Keys: Widely regarded as secure for most practical applications, 128-bit keys offer robust protection against brute-force attacks while keeping computational overhead low—a critical factor for resource-constrained devices such as IoT endpoints.

  - –256-bit Keys: Providing a higher security margin, 256-bit keys are particularly relevant in contexts with elevated threat models or where data must remain secure over long periods (e.g., against future quantum adversaries). Although the increased key length may introduce slightly higher computational demands, it offers additional protection where needed.

- –Performance Trade-offs: The experimental results reveal that for many encryption algorithms (e.g., Salsa20, GCM, and TinyAES), the differences in throughput, CPU cycles, and encryption time between 128-bit and 256-bit keys are marginal. This suggests that, in many cases, the extra security provided by a 256-bit key does not incur a significant performance penalty—even on devices with limited computational resources. However, even minor differences may be important in extremely energy-constrained environments.

6.4 Security vs performance trade-off

Lightweight encryption protocols are particularly well-suited for edge devices, where computational resources, memory, and energy are constrained. At the edge, these protocols efficiently secure data transmissions without overwhelming the device, making them an excellent choice for real-time processing and local communication. However, when data moves beyond the edge—specifically when it is transmitted to or stored in the cloud—the security requirements typically become more stringent. In cloud environments, where computational overhead is less of a concern, employing more robust encryption schemes is advisable. This is because cloud systems are exposed to a broader threat landscape, including sophisticated attacks that lightweight protocols might not sufficiently mitigate.

Table 2: Comparing the new scheme with other protocols (AES- GCM, AES- EAX, ChaCha20-Poly1305, AES-CCM8, TinyAES) with 256-bit key

Image Size	Lowest CPU Cycles	Highest Throughput	Lowest Encryption Time
1KB	Our New Scheme	Our New Scheme	TinyAES (Our scheme is second and near to TinyAES)
5KB	Our New Scheme	Our New Scheme	Our New Scheme (TinyAES is very near to our scheme)
10KB	Our New Scheme	Our New Scheme	Our New Scheme

Table 3: Comparing the new scheme with other protocols (AES-EAX, AES-GCM,AES-CCM8, TinyAES-HMAC, SPECK-HMAC, SIMON-HMAC, ASCON) with 128-bit key.

Image Size	Lowest CPU Cycles	Highest Throughput	Lowest Encryption Time
1KB	Our New Scheme	Our New Scheme (SPECK And SIMON protocols are very near to our scheme)	TinyAES (Our scheme is second and near to TinyAES)
5KB	Our New Scheme	Our New Scheme	Our New Scheme (TinyAES is very near to our scheme)
10KB	Our New Scheme	Our New Scheme	Our New Scheme

7 Conclusion And Future Research

The raw data and subsequent analysis highlight the practical trade-offs involved in selecting encryption protocols for IoT applications. While more traditional encryption methods, such as AES-based schemes, offer strong security, they often come with significant computational overhead, making them less suitable for resource-constrained IoT devices. The Salsa20-Blake2b scheme, being lightweight, showed promising results, offering lower CPU cycles, faster encryption, and higher throughput, making it an ideal choice for IoT environments. However, the condition of using secure physical environments may not be directly relevant in cloud-based MQTT brokers, where different security considerations are at play. This difference requires further study to better understand the security dynamics in MQTT cloud brokers and their implications on the choice of encryption schemes. Additionally, this opens future research lines related to using the Salsa20-Blake2b scheme at the edge, where IoT devices operate in resource-constrained environments.

This study emphasizes that encryption strategies must be carefully aligned with the deployment context. At the edge, where devices face limited resources, lightweight encryption not only ensures acceptable performance but also meets the immediate security needs. However, as data transitions from these localized environments to cloud infrastructures, the security requirements evolve. Cloud-based systems operate under a broader threat landscape, necessitating more robust encryption measures to protect against sophisticated attacks. Future research directions might invovle hybrid encryption architectures that integrate lightweight encryption at the edge with stronger, more comprehensive encryption protocols for cloud communications, ensuring end-to-end data security. Another direction is related to extensive real-world studies to assess the performance and security of the Salsa20-Blake2b scheme in diverse operational environments, particularly in scenarios with variable physical security conditions.

References

[1] S. Alharbi, D. Bell, W. Awad, “Application Layer Security: MQTT perspective with TLS Implementation and Analysis,” 2025, accepted paper.

[2] J. Bernstein, “The Salsa20 family of stream ciphers,” in New Stream Cipher Designs, vol. 4986, Springer Berlin Heidelberg, 2008, pp. 84–97.

[3] B. Mishra and A. Kertesz, “The use of MQTT in M2M and IoT systems: A survey,” IEEE Access, vol. 8, pp. 201071–201086, 2020.

[4] C.-S. Park and H.-M. Nam, “Security architecture and protocols for secure MQTT-SN,” IEEE Access, vol. 8, pp. 226422–226436, 2020.

[5] H. Hidayat, P. Sukarno, and A. A. Wardana, “Overhead analysis on the use of digital signature in MQTT protocol,” in Proc. Int. Conf. Electr. Eng. Informat. (ICEEI), 2019, pp. 87–92.

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

[7] O. Sadio, I. Ngom, and C. Lishou, “Lightweight Security Scheme for MQTT/MQTT-SN Protocol,” in 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Oct. 2019, doi: https://doi.org/10.1109/IOTSMS48152.2019.8939177.

[8] P. Pal, G. Lauer, J. Khoury, N. Hoff, and J. Loyall, “P3S: A privacy preserving publish-subscribe middleware,” in Proc. ACM/IFIP/USENIX Int. Conf. Distrib. Syst. Platforms Open Distrib. Process., Dec. 2012, pp. 476–495.

[9] J. Wang, J. Zhang, E. M. Schooler, and M. Ion, “Performance evaluation of attribute-based encryption: Toward data privacy in the IoT,” in Proc. IEEE Int. Conf. Commun. (ICC), 2014, pp. 725–730.

[10] M. Iqbal, A. M. Ari Laksmono, A. T. Prihatno, D. Pratama, B. Jeong, and H. Kim, “Enhancing IoT security: Integrating MQTT with ARIA Cipher 256 algorithm cryptography and mbedTLS,” 2023 International Conference on Platform Technology and Service (PlatCon), Busan, Korea, Republic of, 2023, pp. 91-96, doi: 10.1109/PlatCon60102.2023.10255171.

[11] 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.

[12] Wijayanto, S. S. Nugrahani, D. W. Wardani, H. D. Cahyono, and H. 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.
[13] L. Bisne and M. Parmar, "Composite secure MQTT for Internet of Things using ABE and dynamic S-box AES," in 2017 Innovations in Power and Advanced Computing Technologies (IPACT), 2017, pp. 1–5, doi: 10.1109/IPACT.2017.8245126.
[14] H. Lee, J. Lim, and T. T. Kwon, "MQTLS: Toward secure MQTT communication with an untrusted broker," in Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC), 2019, pp. 53–58.
[15] W. Peng, S. Liu, K. Peng, J. Wang, and J. Liang, "A secure publish/subscribe protocol for Internet of Things using identity-based cryptography," in Proc. 5th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT), Dec. 2016, pp. 628–634.
[16] H.-Y. Chien, P.-C. Lin, and M.-L. Chiang, "Efficient MQTT platform facilitating secure group communication," J. Internet Technol., vol. 21, no. 7, pp. 1929–1940, 2020.
[17] Mektoubi, H. L. Hassani, H. Belhadaoui, M. Rifi, and A. Zakari, "New approach for securing communication over MQTT protocol: A comparison between RSA and elliptic curve," in Proc. 3rd Int. Conf. Syst. Collaboration (SysCo), 2016, pp. 1–6.
[18] Borcea, Y. Polyakov, K. Rohloff, G. Ryan, and A. B. Deb Gupta, "PICADOR: End-to-end encrypted publish–subscribe information distribution with proxy re-encryption," Future Gener. Comput. Syst., vol. 71, pp. 177–191, Jun. 2017.
[19] M. Hamad, E. Regnath, J. Lauinger, V. Prevelakis, and S. Steinhorst, "SPPS: Secure policy-based publish/subscribe system for V2C communication," in Proc. Design, Autom. Test Europe Conf. Exhibition (DATE), 2021, pp. 529–534.
[20] M. Dahlmanns, J. Pennekamp, I. B. Fink, B. Schoolmann, K. Wehrle, and M. Henze, "Transparent end-to-end security for publish/subscribe communication in cyber-physical systems," in Proc. ACM Workshop Secure Trustworthy Cyber-Phys. Syst., 2021, pp. 78–87.
[21] J.-P. Aumasson and D. J. Bernstein, "Blake2: Simple, fast, secure hash functions," [Online]. Available: https://blake2.net, 2025.

Saud Alharbi graduated from Birmingham University in (2010) with MSc in computer security. He has a diverse experience across different technology and information security domains. He is interested in IoT security, encryption algorithms and cloud computing security.

David Bell is a Reader in Computer Science at Brunel University of London. Dr. Bell completed his PhD degree in Computer Science at Brunel University of London in 2006, utilising semantic web techniques to discover grid-based software services. From 2006 he held a teaching and research appointment at Brunel University of London. Dr. Bell has authored over 100 papers in refereed international conferences and journals. His research interests include modelling and simulating healthcare systems (including cybersecurity and using machine learning). Dr. Bell's funded research projects include emotion analytics, AI-driven health economic modelling, digital productivity and digital personhood. He is also Chief Technology Officer at HecoAnalytics Limited where his previous research on evidence visualisation has been commercialised into a suite of cloud-based health AI tools.

Wasan Awad is Professor of Computer Sciences. She is currently working as a Dean of College of Information Technology College and Chair of ITIKD conference at Ahlia University. She Published a number of papers in different international journals and conferences in information security, artificial intelligence, coding theory, and generative AI. She is the founder and chair of IEEE SSIT Bahrain chapter.

Table 4: Appendix 1

Protocol	Image Size	Average Throughput	Average CPU Cycles	Average Encryption Time	Number Of Runs	Number Of Messages Per Run	Key Size
Salsa20	10KB	26149.91	94615315.2	95.676	10	100	256-bit
Salsa20	10KB	26151.057	94649266.2	95.651	10	100	128-bit
CCM8	10KB	26010.281	99483501	105.952	10	100	256-bit
CCM8	10KB	26009.129	100950239.1	108.025	10	100	128-bit
ChaCha20-Poly1305	10KB	26032.733	101475664.1	100.631	10	100	256-bit
GCM	10KB	26041.045	102810133.7	109.703	10	100	128-bit
GCM	10KB	26040.961	104287283.6	111.259	10	100	256-bit
EAX	10KB	26041.29cm2	151693341.6	138.104	10	100	128-bit
EAX	10KB	26038.991	151889348.1	135.843	10	100	256-bit
TinyAES	10KB	25963.324	229554006.7	109.229	10	100	128-bit
TinyAES	10KB	25963.619	254852205.5	115.03	10	100	256-bit
ascon	10KB	43.504	16487610258	2435.564	10	100	128-bit
SPECK	10KB	23545.237	18487802953	3287.174	10	100	128-bit
SIMON	10KB	22924.193	25533917382	4055.224	10	100	128-bit
Salsa20	1KB	2827.624	29783764.3	82.791	10	100	128-bit
Salsa20	1KB	2827.314	30621037.7	78.912	10	100	256-bit
TinyAES	1KB	2639.649	40078871.1	76.21	10	100	128-bit
ChaCha20-Poly1305	1KB	2709.877	40090915.2	85.42	10	100	256-bit
TinyAES	1KB	2639.339	41323512	73.655	10	100	256-bit
CCM8	1KB	2429.2	47089176.1	100.063	10	100	128-bit
CCM8	1KB	2686.294	47097224.7	92.273	10	100	256-bit
GCM	1KB	2717.599	53618149.7	98.533	10	100	256-bit
GCM	1KB	2718.008	54018473.3	106.628	10	100	128-bit
EAX	1KB	2717.53	99860519.2	125.227	10	100	256-bit
EAX	1KB	2451.484	100498533.9	135.028	10	100	128-bit
SPECK	1KB	2819.263	1461976906	404.767	10	100	128-bit
ascon	1KB	46.989	1948811951	465.429	10	100	128-bit
SIMON	1KB	2819.215	2471199223	555.588	10	100	128-bit
Salsa20	5KB	13049.017	59372831.8	94.54	10	100	128-bit
Salsa20	5KB	13047.631	61438767.3	87.035	10	100	256-bit
ChaCha20-Poly1305	5KB	12938.185	66666744.9	91.453	10	100	256-bit
CCM8	5KB	12915.906	69077373	108.351	10	100	128-bit
CCM8	5KB	11673.362	70072261.9	101.108	10	100	256-bit
GCM	5KB	12244.41	74000954.6	101.406	10	100	256-bit
GCM	5KB	11671.593	76274281.7	111.761	10	100	128-bit
TinyAES	5KB	12861.428	121269896.7	95.493	10	100	128-bit
EAX	5KB	12947.208	122832191.3	138.758	10	100	128-bit
EAX	5KB	12945.722	126049102.1	129.406	10	100	256-bit
TinyAES	5KB	12859.947	133649650.8	90.552	10	100	256-bit
SPECK	5KB	11316.471	7632311131	1389.327	10	100	128-bit
ascon	5KB	45.193	8550672327	1395.252	10	100	128-bit
SIMON	5KB	12302.669	12279359186	2003.317	10	100	128-bit