Experimental Evaluation of End-to-End Security Protocols for the Internet of Everything

Experimental Evaluation of End-to-End Security Protocols for the Internet of Everything

Marco Esposito
Department of Information Engineering
Università Politecnica delle Marche
Ancona, Italy
m.esposito@pm.univpm.it

Monica Marconi Sciarroni
Department of Information Engineering
Università Politecnica delle Marche
Ancona, Italy
monica.marconi@staff.univpm.it

Tommaso Fava
Department of Information Engineering
Università Politecnica delle Marche
Ancona, Italy
s1113145@studenti.univpm.it

Alberto Belli
Department of Information Engineering
Università Politecnica delle Marche
Ancona, Italy
a.belli@.univpm.it

Lorenzo Palma
Department of Information Engineering
Università Politecnica delle Marche
Ancona, Italy
l.palma@univpm.it

Emanuele Storti
Department of Information Engineering
Università Politecnica delle Marche
Ancona, Italy
e.storti@univpm.it

Paola Pierleoni
Department of Information Engineering
Università Politecnica delle Marche
Ancona, Italy
p.pierleoni@univpm.it

***Abstract*—The Internet of Everything (IoE) includes interconnected devices, processes, networks, and people. Within this framework, smart objects collect environmental data and transmit it across the IoE architecture. Smart objects are often resource-constrained, with limited bandwidth, battery, and processing capabilities. IoE applications require some degree of security, which will introduce communication overhead. This paper presents an evaluation of communication protocols for end-to-end security for smart devices in an IoE framework. The JOSE and COSE formats are evaluated and compared to non-secure serialization. Experiments carried out on a Raspberry Pi Zero W and showed that COSE has low overhead, nearly comparable to the non-secure formats. However, COSE is about 3 ms slower than JOSE for the smallest packet size at the transmitter, while the protocols have closer performances at the receiver. For larger packets, COSE becomes faster at both the receiver (>2 kB) and the transmitter (>20 kB). The experiments suggest that COSE, due to its small overhead, may be considered as a potential lightweight solution for end-to-end security in IoE applications where slight latency is acceptable, such as periodic sensor data collection in Industry 5.0 applications. It also demonstrates better scalability than JOSE for increasing payload sizes.**

***Index Terms*—IoE, Industry 5.0, lightweight communication, end-to-end security, MQTT**

This work has been supported by the PRIN 2022 Project "HOMEY: a Human-centric IoE-based Framework for Supporting the Transition Towards Industry 5.0" (code: 2022NX7WKE, CUP: F53D23004340006) funded by the European Union - Next Generation EU.

DOI: 10.1109/RTSI61910.2024.10761793

I. Introduction

Sensor networks and smart objects play a crucial role in many Internet of Everything (IoE) architectures. Within this broad framework, in which things, people, and processes are interconnected, sensors and smart objects serve as essential components, embedding processors, transmission units, sensors to measure data from the physical environment, and often also actuators. In these scenarios, interconnected devices are often low-cost, low-energy devices with limited communication and processing capabilities [1], [2]. Further challenges arise when these sensors generate data at high speed or in large quantities. Lightweight alternatives to message exchange protocols such as HTTP have therefore become essential for Internet of Things (IoT) ecosystems [3], especially when framing them in the larger IoE context, where a greater amount of data may be generated. Protocols such as the Message Queuing Telemetry Transport (MQTT) protocol [4] have emerged as a valuable solution for many IoT applications that require a small payload size and low energy consumption. More generally, application layer protocol design is very important for building IoT applications, as it directly impacts system flexibility and overall performance [3]. In particular, data serialization plays an important role in optimizing the payload message size [5]. MQTT is domain and data agnostic [6], making it extremely flexible as it is able to transport differently serialized payloads.

Security also plays a major role in all modern IoT and IoE applications. MQTT has native security mechanisms, including identification with username and password, and transport layer security (TLS). While transport layer security ensures authentication and encrypted communication towards the Broker, the latter might become a security bottleneck. Furthermore, TLS is not always suitable for resource-constrained IoT environments [7]. Lightweight end-to-end security is a viable alternative option toestablish a highly secure communication channel [8]. In an end-to-end secure system, while the MQTT Broker uses unencrypted packet metadata for topic routing, the application data remains encrypted throughout transmission from the publisher to the subscriber [9]. Spina et al. [10] proposed a mechanism for end-to-end security in MQTT with reduced overhead compared to classical TLS. While perfectly viable, Tschofenig and Bacelli [11] suggest the use of object security rather than channel security for infrequent, asynchronous communication and one-shot payloads. Such situations are common in sensor networks, e.g. during firmware updates or sporadic sensor data collection, and few solutions for end-to-end object security have emerged. JOSE (JSON Object Signing and Encryption) [12] and COSE (CBOR Object Signing and Encryption) [13] are protocols that provide methods to sign and/or encrypt application layer data. They encrypt payloads and serialize them (plus some additional headers) using JSON (JavaScript Object Notation) [14] and CBOR (Concise Binary Object Representation) [15], respectively, two widely used serialization formats. Nevertheless, object security adds another layer of complexity to application layer design. It inevitably results in an overhead which must be assessed on a case-by-case basis, especially for devices with constrained capabilities [16]. COSE and JOSE have already found use in some application scenarios. OSCORE (Object Security for Constrained RESTful Environments) [17] is a COSE-based solution to enable application-layer protection for the Constrained Application Protocol (CoAP) [18]. The work by Tjäder [19] uses COSE for object-security in constrained IoT platform. However, a direct comparison between COSE and JOSE is lacking. Consequently, the objective of this work is to evaluate the overhead in terms of packet size and processing time introduced by the JOSE and COSE protocols on resource-constrained devices, which may serve as components of IoE architectures comprising of interconnected things, people, and processes.

The paper is divided as follows. Section II introduces some theoretical concepts about serialization and security formats that are suitable for MQTT payloads. Section III details the experiments and methods used to compare the protocols under examination. Section IV provides the results of the evaluations, and conclusions are drawn in Section V.

II. Theoretical Background

A. MQTT

MQTT is an OASIS standard messaging protocol for the Internet of Things [4]. With a limited code footprint and bandwidth usage, it is an extremely lightweight solution for resource-constrained devices, while also offering reliable communication thanks to the underlying TCP/IP model and three quality of service (QoS) levels [3]. Moreover, it is highly scalable because it uses a decoupled publish-subscribe pattern. The sender (Publisher) and the receiver (Subscriber) exchange data within Topics, and do not need to know each other identity or communicate directly. The connection between them is handled by the MQTT Broker, which filters all incoming messages and dispatches them correctly to the Subscribers. It has found applications in multiple IoT fields, representing a valuable and fairly simple solutions for many scenarios, ranging from home automation [20], to smart cities [21] and smart farming [22], structural health monitoring [23], and also for early warning systems [24] and environmental monitoring [25].

B. Data Representation

MQTT does not impose any specific data representation for the payload. However, different formats offer advantages and disadvantages. The use of raw data is lightweight but requires parsing on the receiving end, while text data is easier to interpret but larger and less efficient.

One popular solution for message serialization and encoding is JSON [14]. JSON is a lightweight data-interchange format structured as a collection of key-value pairs. It is easy for humans to read and write, and it is also easy for machines to parse and generate. It enables rapid interpretation of the message, and it is supported by all modern programming languages. Those reasons make JSON attractive for many IoT applications, but it may not be suitable for resource-constrained environments [26]. Alternative solutions to text-based serialization exist, wherein data is sent in binary format. This approach reduces the size of the data at the cost of human readability.

There exist many binary serialization formats. Besides language-specific formats, such as pickle [27], popular binary serialization formats are MessagePack [28] and CBOR [15]. Both claim smaller payloads than JSON, while ensuring compatibility with all data types allowed by JSON, requiring minimal changes to the application layer design. They share similar conciseness, but have different design goals [15]. CBOR aims for compact encoder and decoder code, self-describing data, and full JSON conversion support, making it ideal for IoT and sensor data streaming applications [29]. On the other hand, MessagePack evolution is partly hindered by the need for backward compatibility, limiting new features and data formats [30].

CBOR encoded data is seen as a stream of data items. Each item begins with a 1-byte header with 3 bits that define the data type, while the rest directly represents the element or indicates the number of subsequent bytes to represent it. The example below shows a conversion from JSON to CBOR,

```
JSON: "message"
CBOR: 67               # text type
        6D657373616765 # "message"
```

C. Security

The growing interest towards IoT and IoE applications has required the development of different security measures. MQTT has different security mechanisms, including transport layer security with TLS, which may be challenging on resource-constrained devices [7]. Furthermore, transport layer security has limitations as it does not prevent the Broker from reading application layer messages. This is a critical security bottleneck in MQTT.

A solution to effectively secure infrequent, asynchronous communication and one-shot payloads is represented by end-to-end object security [11]. Object security is a concept where the Application layer packet or sensitive parts of the packet are encrypted. With this mechanism, a compromised intermediate node in the network, such as an MQTT broker, will not be able to access encrypted data, thus maintaining full end-to-end security [19].

When dealing with data in JSON, one option to achieve end-to-end security is to use JOSE. JOSE is a framework designed to provide methods to securely transfer a set of key-value pairs. In particular, JSON Web Encryption (JWE) [12] allows to represent encrypted content using JSON-based data structures and Base64URL encoding. The structure of a JOSE message is the following (adapted from [12]):

```
{
  "header":
   {
     "alg" : key management alg,
     "enc" : encryption alg
   },
   "encrypted_key" : key,
   "iv" : IV,
   "ciphertext" : ciphertext,
   "tag" : auth-tag
}
```

To achieve compactness, the JWE-Compact Serialization is defined. It represents the above structure as a string consisting of the concatenation of the various fields encoded in Base64URL and separated by a dot [12].

COSE [13] is the CBOR-based equivalent of JOSE. It provides the possibility to create and process signatures, message authentication codes and encryption using CBOR rather than Base64URL serialization. There are different message structures for the various security options, but all of them are CBOR arrays and share the first three fields, named `protected`, `unprotected`, and `content`, respectively. The content field is either plaintext or ciphertext. Focusing on encryption, [13] defines the `cose-encrypt0` type for situations with a single recipient that already knows the key, a situation that corresponds to the use of JWE with `"alg"` set to `dir` and the absence of the `"encrypted_key"` field. For the rest, its structure is similar to the JWE one, as can be seen in the following (adapted from [13]):

```
COSE_Encrypt0(
[
  {
  "protected": {
    "alg": encryption alg
  },
  "unprotected": {
    "iv": IV
  },
  "ciphertext": ciphertext
  }
]
)
```

III. Materials and Methods

This Section details the tests that were carried out and the performance metrics that were used to evaluate the performance of different serialization and encryption methods.

A. Setup

The experimental evaluation of the protocols is performed on a Raspberry Pi Zero W, a low-cost, widely available device which can be seen as a generic IoT node. It has a 1GHz, single-core CPU with 512MB RAM, and it runs Raspberry Pi OS 64-bit loaded on a 8GB microSD card. The device is controlled via SSH connection.

On the software side, Python 3.9.2 is used. JSON functionalities are given by the *ultrajson* library, which is faster than *json* [31]. For CBOR, the actively maintained, high quality *cbor2* implementation [32] is chosen. JOSE and COSE are implemented with *python-jose* [33] and *pycose* [34], respectively. While those libraries include several cryptographic functions, the focus has been on data confidentiality. Encryption is done using the AES-GCM algorithm, as it is the only algorithm supported by both *python-jose* and *pycose*. The key is a 128-bit hard-coded string. The size helps minimise memory usage and energy consumption [35], while a static key allows the key exchange to be ignored, simplifying the evaluation. The choice of a static key does not impact the generality of the experiment, as it is common practice to deploy IoT devices with a static key [36].

B. Sample payload and Protocols under examination

The evaluation proposed in this article aims to identify the best lightweight solution that may guarantee end-to-end security for a broad category of IoE applications, including smart objects and sensors data collection. To ensure an approach that is as general as possible, a neutral, variable-length payload has been chosen. The tests carried out concern a dictionary-like payload with a single item, which may be used in different situations, e.g., for a firmware update, for transmitting sensor data to an IoT hub, or, in the larger IoE context, where devices, processes, and people are interconnecting, for the exchange of files and reports of varying size. Its structure is the following:

```
{ "msg" :  secret }
```

where `secret` is a variable-length string of 2^7 to 2^{16} characters. Each char is represented with a single byte, but a

49-bytes overhead is added due to the Python string encoding. However, here this overhead will be ignored because it does not play a significant role in the comparison.

The payload is to be transmitted according to JOSE and COSE standards. These protocols require the payload to be serialized before it is used as plaintext. The following combinations of payload serialization and packet encryption format have therefore been evaluated.

- COSE packet with CBOR-serialized payload
- JOSE packet with JSON-serialized payload (*JOSE1* format)
- JOSE packet with CBOR-serialized payload (*JOSE2* format)

In addition to the above, and to better assess the trade-off between security and performance, the baseline (non secure) JSON and CBOR serializations are also tested in terms of packet size and serialization/deserialization times.

C. Performance Metrics

The performance metrics of the evaluation have been the final size of the encrypted (and serialized) packet, alongside the percentage overhead, calculated using the following formula:

$$\text{Percentage Overhead} = \left(\frac{\text{Final Size} - \text{Secret Length}}{\text{Secret Length}}\right) \times 100\%$$

The processing time at both the transmitter and receiver ends were also computed. This is critical for devices performing cryptographic operations, which may have a significant execution time and therefore energy cost [37]. Processing time at the transmitter includes the following contributes:

1) payload serialization
2) packetization and encryption
3) packet serialization

The processing time at the receiver follows the same steps but in reverse order:

1) packet deserialization
2) decryption
3) packetization
4) payload deserialization

For the baseline CBOR and JSON formats, only serialization and de-serialization times contribute to the overall processing time.

The average elapsed time at the transmitter is defined as the average time it takes to go from the unserialized, non-secure, dictionary-like data structure to the final payload. Time is measured using Python's `time.process_time()` function, which only considers the system and CPU time during the process. The elapsed time $\bar{T}_{\text{el}}$ has therefore been computed as:

$$\bar{T}_{\text{el}} = \frac{1}{N}\sum_{i=1}^{N}(t_{\text{end}_i} - t_{\text{start}_i})$$

where t_{end_i} is the timestamp taken at the end of the serialization and encryption process for the i-th payload, t_{start_i} is the timestamp at the start of the process for that same payload, and N is the number of repetitions of the experiment. N has been set to 1000. Similarly, at the receiver, the average elapsed time is defined as the average time it takes to go back from the final payload to the original dictionary-like data structure, following the same equation used for $\bar{T}_{\text{el}}$.

IV. Results

Table I shows the overhead introduced by the various protocols under test for different lengths of the secret (expressed in bytes). As explained in Section III, *secret length* is defined as the number of characters of the value in the key-value pair used as payload, and it does not include the overhead introduced by Python. Looking at the serialization formats (JSON and CBOR columns), it can be observed how CBOR in general provides a smaller size increment than JSON, even though the difference in overhead is almost negligible, especially as the packet gets larger. The *JOSE1* format affects overhead considerably, providing a notable increase in size compared to plain JSON serialization, although the overhead due to security becomes less pronounced as the payload increases. JOSE applied on binary data (*JOSE2* format) is still costly in terms of payload increase, but slightly less compared to JOSE1. The table shows that COSE is a far more performing solution in terms of overhead, introducing a far smaller increment in size compared to JOSE1 and JOSE2. Its percentage overhead becomes comparable with the overhead introduced by plain serialization as the secret length increases. Since the overhead it introduces is small, especially for larger secret lengths, this secure format enables end-to-end object security at an almost negligible cost compared to both JSON and CBOR.

TABLE I
PERCENTAGE OVERHEAD INTRODUCED BY DIFFERENT SERIALIZATION AND/OR ENCODING SCHEMES FOR DIFFERENT SECRET LENGTHS.

Secret Length	JSON	JOSE1	CBOR	JOSE2	COSE
128	26.6%	125%	18.8%	121.9%	49.2%
256	13.7%	79.3%	9.0%	78.9%	25.4%
512	7.0%	56.3%	4.9%	55.9%	12.1%
1024	3.2%	44.9%	2.5%	44.5%	6.4%
2048	0.2%	38.4%	0.2%	38.1%	3.2%
4096	0.2%	35.8%	0.1%	35.5%	1.5%
8192	0.1%	34.9%	0.1%	34.9%	0.4%
16384	0.1%	34.3%	0.1%	34.3%	0.3%
32768	0.1%	33.5%	0.1%	33.5%	0.2%
65536	0.1%	33.3%	0.1%	33.3%	0.1%

Figures 1 and 2 show the time needed to encrypt and decrypt the variable-length payload applying JOSE and COSE standards. As before, a comparison against plain JSON and CBOR encoding is also included. In this case, *secret length* represents the payload before serialization and the elapsed time only includes the time taken to de/serialize at the receiver/transmitter, respectively.

As expected, operations without encryption are significantly faster both at the transmitter and receiver sides. CBOR performs better than JSON for secrets above 8 kB at the transmitter and above 2 kB at the receiver, but their performances

are similar for smaller sizes. When it comes to the secure formats, on the transmitter side COSE becomes faster than JOSE for larger payloads, between 16 kB and 32 kB. On the other hand, for the smallest secret size of 128 bytes, it is around 3 ms slower than JOSE1. On the receiver end, COSE is quite consistently fast even for smaller packet sizes, and it becomes faster than JOSE already for secrets of 2 kB of size. *JOSE2* is faster than *JOSE1* for larger packets at both ends, most likely due to the CBOR serialization.

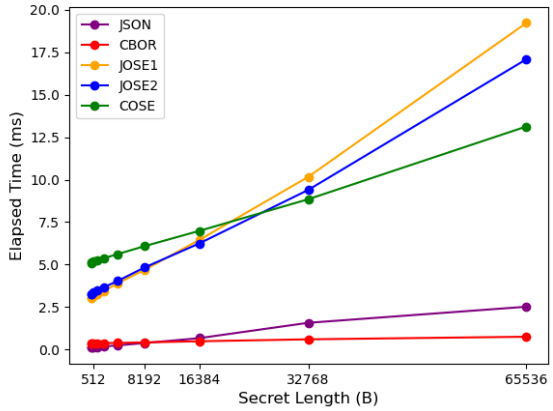

Fig. 1. Encoding and encryption time for different secret lengths. Operations without encryption (JSON and CBOR) are significantly faster. CBOR performs better than JSON for secrets above 8 kB. For secure formats, COSE becomes faster than JOSE1 and JOSE2 for larger payloads (32 kB and above), although it is slower than JOSE1 for the smallest secret size.

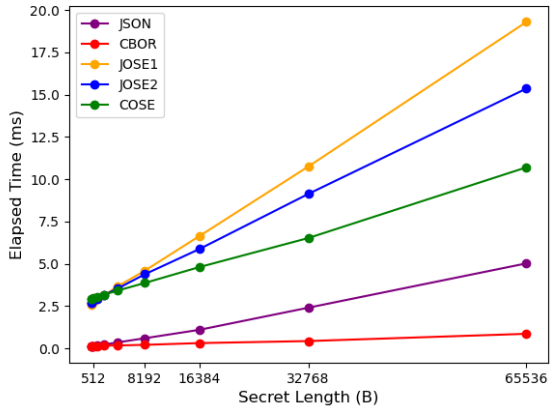

Fig. 2. Decoding and decryption time for different secret lengths. Similar to encoding, operations without encryption are faster, with CBOR outperforming JSON for secrets above 2 kB. COSE shows consistent performance and becomes faster than JOSE1 and JOSE2 for secrets of 8 kB and above. JOSE2 is faster than JOSE1 for larger packets, likely due to CBOR serialization.

V. Conclusion

In this paper, an experimental evaluation of popular communication protocols for end-to-end security in IoE environments is carried out. The evaluation considered the JOSE and COSE protocols, designed to encrypt and serialize packets in JSON and CBOR respectively, two widely used data formats for the Internet of Things. Their performance was measured in terms of overhead introduced by the serialization and encryption processes, and the time required to perform encryption and encoding at the transmitter, as well as decryption and deserialization at the receiver. The tests were carried out on a Raspberry Pi Zero W and covered different combinations of payload serialization and secure packet formats. In order to assess the trade-off between security and performance, non-secure JSON- and CBOR-only encoded packets were also included in the experiments. The results show that COSE offers advantages over JOSE, particularly in terms of packet compactness and processing times for large packets. In particular, COSE overhead settles at approximately exactly the same values as the plain (non-secure) JSON serialization format, with a negligible overhead as the packet grows larger. Nonetheless, adding encryption on top of serialization introduces a time overhead which may not be suitable for all sensor networks and IoE applications. COSE encryption is about 3 ms slower than JOSE for the smallest pack size. Conversely, for larger packets it becomes consistently faster, both at the transmitter and receiver. Accounting for both metrics, COSE outperforms JOSE in terms of average overhead and scalability, with a more stable performance as payload size increases. It shows promise as a feasible solution for end-to-end security in IoE applications that may afford some latency, such as periodic sensor readings in Industry 4.0/5.0 applications.

References

[1] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end iot devices," *IEEE Access*, vol. 6, pp. 70 528–70 554, 2018.

[2] R. Hassan, F. Qamar, M. K. Hasan, A. H. M. Aman, and A. S. Ahmed, "Internet of things and its applications: A comprehensive survey," *Symmetry*, vol. 12, no. 10, p. 1674, 2020.

[3] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, and C. Yan, "Investigating messaging protocols for the internet of things (iot)," *IEEE Access*, vol. 8, pp. 94 880–94 911, 2020.

[4] MQTT: The standard for IoT messaging. [Online]. Available: https://mqtt.org/

[5] D. P. Proos and N. Carlsson, "Performance comparison of messaging protocols and serialization formats for digital twins in iov," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 10–18.

[6] H. Raddatz, E. Mahmoud, F. Hölzke, P. Danielis, D. Timmermann, and F. Golatowski, "Evaluation and extension of opc ua publish/subscribe mqtt binding," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1. IEEE, 2020, pp. 543–548.

[7] F. Chen, Y. Huo, J. Zhu, and D. Fan, "A review on the study on mqtt security challenge," in *2020 IEEE International Conference on Smart Cloud (SmartCloud)*, Washington, DC, USA, 2020, pp. 128–133.

[8] N. A. Khan, A. Awang, and S. A. A. Karim, "Security in internet of things: A review," *IEEE Access*, vol. 10, pp. 104 649–104 670, 2022.

[9] OASIS Message Queuing Telemetry Transport (MQTT) Technical Committee, "MQTT Version 3.1.1 Plus Errata 01," OASIS, OASIS Standard Incorporating Approved Errata 01, 2015, section 5.4.5: Privacy of Application Messages and Control Packets; Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. [Online]. Available: http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html

[10] M. G. Spina, F. D. Rango, and G. M. Marotta, "Lightweight dynamic topic-centric end-to-end security mechanism for mqtt," in *2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Valencia, Spain, 2021, pp. 1–7.

[11] H. Tschofenig and E. Baccelli, "Cyberphysical security for the masses: A survey of the internet protocol suite for internet of things security," *IEEE Security & Privacy*, vol. 17, no. 5, pp. 47–57, Sept.-Oct. 2019.

[12] M. Jones and J. Hildebrand. (2015, May) Json web encryption (JWE). RFC 7516. IETF. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7516

[13] J. Schaad. (2017, July) Cbor object signing and encryption (COSE). RFC 8152. IETF. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8152

[14] T. Bray. (2017, December) The JavaScript object notation (JSON) data interchange format. RFC 8259. IETF. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8259

[15] C. Bormann. (2020, December) Concise binary object representation (CBOR). RFC 8949. IETF. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8949

[16] G. Nebbione and M. C. Calzarossa, "Security of iot application layer protocols: Challenges and findings," *Future Internet*, vol. 12, no. 3, p. 55, 2020.

[17] G. Selander, J. P. Mattsson, F. Palombini, and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)," Internet Engineering Task Force, Request for Comments RFC 8613, Jul. 2019. [Online]. Available: https://datatracker.ietf.org/doc/rfc8613/

[18] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," Internet Engineering Task Force, Request for Comments RFC 7252, Jun. 2014. [Online]. Available: https://datatracker.ietf.org/doc/rfc7252/

[19] H. Tjäder, "End-to-end security enhancement of an iot platform using object security," Master's thesis, Department of Electrical Engineering, Information Coding, Linköping University, 2017.

[20] P. Macheso, T. D. Manda, S. Chisale, N. Dzupire, J. Mlatho, and D. Mukanyiligira, "Design of esp8266 smart home using mqtt and node-red," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. IEEE, 2021, pp. 502–505.

[21] C. D'Ortona, D. Tarchi, and C. Raffaelli, "Open-source mqtt-based end-to-end iot system for smart city scenarios," *Future Internet*, vol. 14, no. 2, p. 57, 2022.

[22] G. Vitali, M. Francia, M. Golfarelli, and M. Canavari, "Crop management with the iot: An interdisciplinary survey," *Agronomy*, vol. 11, no. 1, p. 181, 2021.

[23] P. Pierleoni, M. Conti, A. Belli, L. Palma, L. Incipini, L. Sabbatini, S. Valenti, M. Mercuri, and R. Concetti, "Iot solution based on mqtt protocol for real-time building monitoring," in *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*. IEEE, 2019, pp. 57–62.

[24] P. Pierleoni, R. Concetti, A. Belli, L. Palma, S. Marzorati, and M. Esposito, "A cloud-iot architecture for latency-aware localization in earthquake early warning," *Sensors*, vol. 23, no. 20, p. 8431, 2023.

[25] A. N. Rosli, R. Mohamad, Y. W. Mohamad Yusof, S. Shahbudin, and F. Y. Abdul Rahman, "Implementation of mqtt and lorawan system for real-time environmental monitoring application," in *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 2020, pp. 287–291.

[26] C. Ortega Corral, L. Palafox, J. García-Macías, L. Aguilar, J. Sanchez Garcia, A. Valenzuela-León, and I. Chon-Aguiar, "A "lighter" json for message exchange in highly resource constrained wireless sensor network applications," 10 2012.

[27] M. Pilgrim, "Serializing python objects," in *Dive Into Python 3*. Springer, 2009, pp. 205–223.

[28] Messagepack. [Online]. Available: https://msgpack.org/

[29] S. M. Araque, I. Martinez, G. Z. Papadopoulos, N. Montavont, and L. Toutain, "Toward a standard time series representation for iot based on cbor templates," in *2022 Global Information Infrastructure and Networking Symposium (GIIS)*, 2022, pp. 13–19.

[30] C. Bormann and P. Hoffman, "RFC 7049 - comparison of other binary formats to CBOR's design objectives: Messagepack and conciseness on the wire," RFC 7049, 2013, [Online]. Available: https://www.rfc-editor.org/rfc/rfc7049.html#section-Appendix.E.2, https://www.rfc-editor.org/rfc/rfc7049.html#section-Appendix.E.6.

[31] H. van Kemenade. Ultrajson. [Online]. Available: https://github.com/ultrajson/ultrajson

[32] A. Grönholm. Cbor2. [Online]. Available: https://cbor2.readthedocs.io/en/latest/

[33] M. Davis. python-jose. [Online]. Available: https://python-jose.readthedocs.io/en/latest/index.html

[34] T. Claeys. pycose. [Online]. Available: https://pycose.readthedocs.io/en/latest/index.html

[35] P. S. Munoz, N. Tran, B. Craig, B. Dezfouli, and Y. Liu, "Analyzing the Resource Utilization of AES Encryption on IoT Devices," pp. 1200–1207, Nov. 2018.

[36] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-38D, Nov. 2007. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/38/d/final

[37] S. S. Dhanda, B. Singh, and P. Jindal, "Lightweight Cryptography: A Solution to Secure IoT," *Wireless Personal Communications*, vol. 112, no. 3, pp. 1947–1980, 2020.