Information Systems Frontiers
https://doi.org/10.1007/s10796-023-10383-9

Analysis of the Cryptographic Algorithms in IoT Communications

Catarina Silva[1,2] · Vitor A. Cunha[1,2] · João P. Barraca[1,2] · Rui L. Aguiar[1,2]

Accepted: 28 February 2023

Abstract
The advances on recent communication paradigms brings new security and privacy challenges, mainly about personal data collection by smart devices. Resource constrained devices face serious issues to run complex cryptographic algorithms. Trying to evaluate the performance impact of those algorithms in usual and common devices used in smart homes, in this paper we tested the impact of different cryptographic algorithms in low computational devices, typically used in smart devices applied in smart homes, testing different security configurations and using the two most used transport protocols (HTTP and MQTT). The experiment measures their effects on six commonly used embedded devices in IoT WSNs: ESP8622, ESP32, and Raspberry Pi (RPi) from 1 to 4. The experiment measured the power consumption, message delay, and additional message length (bytes). Moreover, the analysis was also used to model security algorithms. The experimental results from long runs (72 hours) reveal the cryptographic solution choice is significant for the message delay and additional message length.

Keywords Cryptography · Security · Privacy · Smart home · IoT · WSN

1 Introduction

The vast amount of available smart devices brings new security and privacy concerns. These devices are connected to our daily life in multiple ways and facilitate daily activities in different sectors. In fact, Internet of Things (IoT) is one of the most promising technology to be applied in mobility, healthcare, energy, and Location Based Services (LBS) scenarios, creating a massive worldwide network of interconnected physical objects [3].

Vitor A. Cunha, João P. Barraca and Rui L. Aguiar contributed equally to this work.

✉ Catarina Silva
c.alexandracorreia@av.it.pt

Vitor A. Cunha
vitorcunha@av.it.pt

João P. Barraca
jpbarraca@ua.pt

Rui L. Aguiar
ruilaa@ua.pt

1 Instituto de Telecomunicações, Universidade de Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal

2 Departamento de Electrónica, Telecomunicações e Informática, Universidade de Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal

Smart homes are equipped with several smart devices with diverse functionalities and collect data from different sources, from simple cases such as environmental temperature to video surveillance. Thus, this is a critical scenario where highly sensitive data is collected, sometimes without the individual's knowledge.

The IoT devices access and process humans' personal data highlighting the need to ensure secure and private communications to preserve data integrity and avoid unauthorized access to personal data. However, some standard devices in that kind of architecture have limited computational resources and reduced capacity to process high computational security algorithms. Typically, these kinds of devices are called resource-constrained devices, both computationally and with power constraints. In this way, lightweight cryptographic algorithms are being considered in the scientific literature [15, 36, 41, 42] to reduce the computational impact of security algorithms. These more appropriate algorithms try to ensure humans' security and privacy with trade-offs between cost and performance.

Privacy and security level may be estimated by considering different cases of study since the application interface to the network infrastructure is highly relevant to humans taking advantage of new technologies safely and privately. Although other previous research studies focused on exploring the delay added by the security algorithms, such as [1, 9, 35], a gap in privacy and security analysis was identified: power consumption analysis and computational efficiency to examine algorithms and protocols in usual and common boards applied in typical smart homes. In such a manner, the paper intends to understand the current impact of cryptographic algorithms to evaluate the security and privacy offered using a combination of algorithms and specific boards.

The experiment in this paper, and its main contribution, intends to explore the impact of security and privacy mechanisms on top of conventional Wireless Sensor Networks (WSN) use cases. The experiment measures three fundamental metrics (power consumption, message delay, and additional bytes) using state-of-the-art communication protocols (Hypertext Transfer Protocol (HTTP) and Message Queuing Telemetry Transport (MQTT)) and representative embedded devices (the ESP8622, ESP32, and RPi from 1 to 4). There is a huge number of smart devices with multiple characteristics. Although Arduino devices are well-known development boards, several commercial products use ESP microcontrollers instead (IoTaWatt[1], Tasmota[2], amongst others). RPi devices were selected since they are heavily used for IoT projects and prototyping while being several times more capable than simple microcontrollers in computational power. The research also considers two different cryptographic algorithms. ChaCha20 [45] is the best lightweight cryptographic algorithm to be used in IoT devices by considering memory size and energy. Advanced Encryption Standard (AES) is a popular cryptographic algorithm that offers robust and platform-independent implementation [11]. Generally, ChaCha20 is faster than AES due to the mathematical operations (multiplication, rotation, and XOR) used to encrypt and decrypt the messages compared to binary digits in AES that are used for encryption to secure the messages [43].

By considering this selection, the main contributions of the paper are:

- Overview of what is essential in a WSN regarding security and privacy.
- Description of the most relevant cryptographic algorithms in IoT architectures.
- Analysis of protocols and IoT boards to improve security and privacy (computational and power consumption analyses).
- Reference for comparison amongst the very constrained devices and the opposite.
- A simplified model that can be used to estimate the delays measured in the experiment for the considered embedded devices.

[1] https://iotawatt.com/

[2] https://tasmota.github.io/docs/

The document is organized as follows: Section 2 describes the relevant background about the current paradigm of IoT communications; Section 3 describes the privacy and data protection in IoT communications briefly; Section 4 details security algorithms in IoT devices; the experimental setup and implementation details are outlined in Section 5; the main results are presented and discussed in Sections 6 and 7 presents the relevant State-of-the-Art (SoA); and, finally, the main conclusions are discussed in Section 8.

2 IoT Communications

The IoT concept refers to a complex network of interconnected computing components such as sensors, actuators, or any other device on the Internet. The IoT paradigm resides in having a wide variety of devices connecting and sharing data over the network, including embedded devices capable of interacting with users and other devices even without user interaction ([4, 26, 44]). Since most mobile devices are equipped with WiFi to connect to the Internet, it is relevant to analyze the impact of this technology on enabling Internet Protocol (IP) connectivity of battery-powered devices. In fact, [44] demonstrates the feasibility of low-power WiFi technology in those devices.

Entirely related to the concept of the IoT paradigm is the purpose of WSN. A WSN is a network of nodes, linked via wireless media, that work cooperatively to sense and control the surrounding environment. Typically, this type of architecture has three components: sensor nodes, gateway, and observer (user) [19]. Moreover, it is applied to quotidian scenarios such as smart homes, smart agriculture, security control, and medical applications. Generally, in WSN, some aspects should be considered: reliability (the ability of a sensor to maintain its network functionality without any interruption); nodes' density (coverage area, reliability, and accuracy); network latency characteristics (latency, capacity, and robustness); and the complexity of routing data which depends on the network topology. In addition, since sensor nodes are mainly battery-powered, each node's lifetime depends on the battery's lifetime and the power consumption of the work performed in that device. There is a need to develop a distributed security approach for WSNs, highlighting the necessity to have WSNs where nodes are self-organized to handle the failure of nodes. [37] proposed a dynamic and autonomous scheme for a node to select another node (in case of failure) and also to ensure the individuals' privacy considering a delegation server based on individual consent.

Different protocols and network implementations are being used in WSNs to interconnect nodes, such as Radio Frequency Identification (RFID), Zig- Bee, Wireless Personal Area Networks (WPAN), Digital subscriber line (DSL), Universal Mobile Telecommunications System (UMTS), General Packet Radio Service (GPRS), WiFi, Worldwide Interoperability for Microwave Access (WiMax), Local Area Network (LAN), or Wide Area Network (WAN).

The scenarios like smart homes, agriculture, eHealth, or mobility, are probably the most considered in the literature where IoT devices are used. This observation has a simple explanation. IoT devices are successful solutions to simplify humans' daily activities due to their environment monitoring and actuation abilities. [21] stated that users need to allocate an extraordinary amount of time for their routine work with little distraction taking high advantage of smart devices in their smart homes. The smart home combines smaller systems linked to the home network, containing several communication technologies that can support comfort, safety, security, and convenience to enhance residents' lifestyles [4]. In addition, [17] reviewed the WSN-based agricultural applications and concluded that the main advantage of using WSN in agriculture is to improve quality and production.

Since these devices are connected to the Internet, transmit data over communication channels, communicate with third parties, and collect and process personal data, it is highly relevant to consider this architecture's privacy and security aspects. Moreover, energy preservation in battery-powered devices is also important to increase sensor availability.

3 Privacy in IoT Environments

Independently of the vast advantages of deploying IoT environments, human privacy is severely affected by these environments. Most users do not understand how systems access, process, and transmit their personal data; in other cases, the users do not take care of their privacy rights [32].

The lack of users' knowledge about how recent IoT environments monitor their life affects the users' perception of what happens with their personal data. Researchers and Data Protection Authority (DPA)s have been working in data protection frameworks in trying to address this issue. Across the world, different frameworks are continuously being updated to make it possible to follow recent technological improvements.

Some data protection frameworks have common concepts, which is the case, for example, of General Data Protection Regulation (GDPR) in the European Union (EU) and California Consumer Privacy Act (CCPA) in California. For the scope of this work, some points may be highlighted:

- Increase the transparency about how data is accessed, processed, and communicated with third parties.
- Users should be able to provide informed consent to personal data collection and processing.
- Users should be notified about what happens with their personal data.
- Systems should provide features to users to control their personal data (for example, requesting data deletion).

Since privacy is a fundamental human right, users of the recent systems are at the top of application development concerns following data protection principles. Although, discussions about how systems should behave and how interfaces should display the content and influence users are out of the scope of this work. However, discussions about how data is processed at the communication level are considered in this work.

Encrypted communications in IoT devices are essential to prevent privacy breaches and user identification at the WSN level [31]. Understanding how to improve the computational impact of the cryptographic algorithms depending on the combination of boards and algorithms helps prevent privacy issues and ensure the good work of IoT devices. Moreover, knowing the impact on power consumption is also helpful to ensure good working order and WSN survivability since this is another parameter that prevents device failure (e.g., running out of battery).

4 Security Algorithms for IoT Devices

Security is one of the most relevant challenges in IoT communications due to the limited resources available in common devices. Thus, the problem is ensuring confidentiality, data integrity, and authentication to IoT devices even though most security mechanisms require heavy computation loads and large memory requirements.

Cryptography is essential to ensure secure data communication over the network by encrypting the data using encryption algorithms that hide the information. Some of the most relevant cryptographic algorithms available are vital in information systems security, such as Rivest-Shamir-Adleman (RSA), AES, or Data Encryption Standard (DES). However, similarly to other security schemes, cryptographic algorithms consume a significant amount of computing resources which is not compatible with the low computation power of most usual IoT devices. In addition, these devices are also quite vulnerable to hardware attacks since they are typically more accessible to an attacker than other general-purpose computing devices [34].

Lightweight cryptographic algorithms are being actively developed trying to address these challenges. There are strictly defined criteria for lightweight cryptography algorithms. However, the idea is to have a minimum size required for hardware implementation suited to the low computational power of microprocessors or microcontrollers, favoring low implementation costs and a high level of security [33, 41]. The literature indicates an increasing interest in highlighting the need for these algorithms, including block ciphers, stream ciphers, and hash functions, as described by [34] and [38].

Traditionally, secure encrypted communication between two parties requires the secure exchange of cryptographic keys. Diffie-Hellman (DH) is a established method to securely exchange cryptographic keys over a public channel. This method allows two parties, without prior common knowledge, to jointly establish a shared secret key over an insecure channel. Initially, this algorithm was based on the modulo of a prime number and the security was ensured due to the difficulty associated with Discrete Logarithm Problem (DLP) [30].

Recently, Elliptic Curve Cryptography (ECC) has been widely used as a faster and more secure alternative to algorithms based on DLP. ECC has its security based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP), which is considered more challenging to solve than the Integer Factorization Problem (IFP) used by RSA or the DLP, which is the basis of the ElGamal encryption scheme [24]. The advantage of elliptic curves is that they ensure a level of security equivalent to that of existing public-key systems but with shorter key lengths [7, 18, 25], as shown in Table 1.

Using ECC for the AES-256 session, a 512-bit ECC key is required. In turn, a 15360-bit RSA key would be required to achieve comparable security, which is computationally impracticable in the current systems, both for key transmission and encryption/description operations. This huge difference makes ECC dearer and a potential algorithm for the current embedded system.

[28] showed that ECC has a great performance and makes it possible to generate smaller, faster, and more reliable cryptography keys. In addition, ECC increases the memory requirements and the execution encryption and decryption time. Also, [23] used ECC to establish a secure session key between IoT devices and the remote server, to address existing limitations related to key management and multicast security in Constraint Application Protocol (CoAP), proposing the ECC-CoAP, a lightweight and secure proposed protocol.

Due to the resource constraint of most IoT devices, it is not easy to execute or implement complex mathematical operations. This difficulty is why [2] proposes an effective, secure symmetric cryptographic mechanism based on the certificate authority management and Elliptic Curve Diffie-Hellman (ECDH) to share a digital certificate among IoT devices. In this way, they can provide an adequate lightweight digital certificate management and cryptographic scheme to help detect and prevent several cyber attacks.

Table 1 Key length for public-key and symmetric-key cryptography

Security (bits)	DSA/RSA/DH	ECC
128	3072	256
192	7680	384
256	15360	521

[27] proposed a key management module based on trust graphs for IoT. This module allows the IoT objects to have a key pair that will be used to generate their symmetric key. It allows these IoT objects to generate and store their secret keys in a distributed manner without resorting to a central authority. This way, they can minimize the number of keys because each IoT object only shares a single symmetric cryptographic key with the objects it trusts in the network. The results show that their proposed scheme offers reduced processing, storage, and communication costs, encryption and decryption time, and minimizes energy consumption.

5 Experimental Setup

In this section, we detail how the experiment was designed. The experiment was based on a typical smart home (similar in topology to the one described here [40]), as depicted in Fig. 1. Although the topology is the same, our experiment considers two significant changes. First, it considers HTTP as a communication protocol (not only MQTT). This protocol represents typical usage in legacy systems (see Sections 5.2 and 5.3). Second, we consider the existence of a key server in the network responsible for sharing the public keys used in the scenario. The experiment is based on WiFi since it is one of the most common radio technologies used in smart home deployments (next to Zigbee and Bluetooth). The experiment was based on a physical deployment instead of a software-based simulation because one crucial objective was measuring the power consumption accurately.

The gateway is responsible for the communication between the observer and the sensor. The key server is responsible for securely sharing the session's public keys. The experiment measures key agreement, transport delay, and encryption/decryption delay. These delays will characterize the efficiency of different security configurations within typical hardware.

When a key exchange uses Ephemeral DH, a temporary DH key is generated for every connection; thus, the same key is unlikely to be used twice. On the other hand, in a static DH, the shared key is generated once and may then be reused across sessions. The synchronous request-response nature of HTTP makes it ideal for the ephemeral experimentation of DH, while the asynchronous nature of MQTT makes it better suited for static DH experimentation. Therefore, our experimental setup is designed to evaluate ephemeral DH on top of HTTP, while the static DH will be deployed on top of

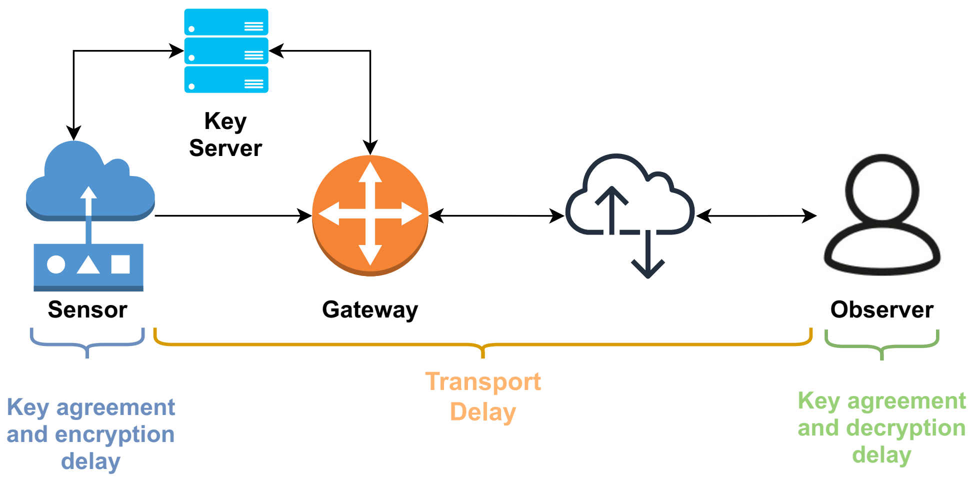

Key Server
Sensor
Gateway
Observer
Key agreement and encryption delay
Transport Delay
Key agreement and decryption delay

Fig. 1 Experimental setup for the proposed evaluation. The measured delays are represented with brackets

MQTT. It is important to mention that even in a static DH implementation, the keys should be updated regularly (every *n* message). However, that is not explored within this work.

Due to the constrained resources on the embedded devices, the key server was introduced as the method to share the keys between entities and share the data when the HTTP was used. While running an HTTP server on the embedded device or the client is possible, the additional server's overheads could have impact on the security performance evaluation.

All the connections between the embedded device (sensor) and the servers (either the HTTP key exchange server or the MQTT server) were unencrypted. The Arduino vanilla libraries for MQTT communication did not support Transport Layer Security (TLS), it is possible to use it with ESP-specific code, but that could also impact the security performance measurement.

In our experiment, the MQTT protocol uses static DH, meaning that the session key is computed only once. For this reason, we did not vary the ECDH since it has minimal impact on the experiment.

In the following section (Section 5.1), the hardware used for the experiment is detailed and discussed. Sections 5.2 and 5.3 described the typical message flow between the entities when the HTTP and MQTT gateways were used, respectively.

5.1 Hardware

Six different embedded solutions were selected for this work: ESP8266, ESP32, and Raspberry Pi versions from 1 to 4. It is important to mention that all boards, except for Raspberry Pi 1 and 2, have native wireless support. These two boards used the official Wireless USB dongle[3] to have wireless capabilities. The main characteristics of these devices can be found in Table 2.

All embedded devices were connected using WiFi 802.11n on a dedicated Access Point (AP), a tp-link TL-WR802N[4]. On this AP were only connected embedded devices and the remaining entities.

The remaining entities (key server, gateway, and observer) were regular computers (conventional laptops running Linux) connected to the same network. The sensors shared three environmental phenomenons: temperature, relative humidity, and atmospheric pressure. These values were measured in real-time using a BME280[5] sensing device. The BME280 was selected because it is highly used in smart home devices (for example, Xiaomi Aqara). This device is an environmental sensor with temperature, barometric pressure, and humidity.

Whenever possible, the code was written in Python. The primary reasons for this choice were the dependencies isolation through virtual environments, vast library collection (through the pip package manager), and code portability. Nevertheless, the code for the ESP boards was written in C using the Arduino IDE environment[6]. Arduino is an open-source platform and a valid option for anyone working on interactive hardware and software projects. The studies mention an increasing usage of the Arduino platform [20].

For this work, measuring the power consumption of the devices is necessary. We have used a USB meter for that task, the *AVHzY CT-3*[7].

3 https://thepihut.com/products/raspberry-pi-wipi-wireless-adapter

4 https://www.tp-link.com/en/home-networking/wifi-router/tl-wr802n/

5 https://www.adafruit.com/product/2652

6 https://www.arduino.cc/

7 https://store.avhzy.com/index.php?route=product/product&product_id=51

Table 2 Specifications for the devices used in the experiment

Device	Cores	Clock CPU	RAM	Wireless
ESP8266	1	160 Mhz	64 KB	802.11 b/g/n
ESP32	2	240 MHz	520 KB	802.11 b/g/n; BLE
Raspberry Pi 1	1	700 MHz	512 MB	WiPi dongle (b/g/n)
Raspberry Pi 2	4	900 MHz	1 GB	WiPi dongle (b/g/n)
Raspberry Pi 3	4	1.2 GHz	1 GB	802.11n; BLE
Raspberry Pi 4	4	1.5 GHz	2 GB	802.11ac; BLE

5.2 HTTP

HTTP is predominantly a web messaging protocol. HTTP supports request/response RESTful Web architecture. Analogous to CoAP, HTTP uses Universal Resource Identifier (URI) instead of topics. The server sends data through the URI, and the client receives data through a particular URI. HTTP is a text-based protocol that does not define the size of header and message payloads; rather, it may depend on the web server or the programming technology. HTTP uses Transmission Control Protocol (TCP) as a default transport protocol and TLS/SSL for security. Thus, communication between the client and server is connection-oriented. It does not explicitly define Quality of Service (QoS) and requires additional support for it. HTTP, a globally accepted web messaging standard, offers several features, such as persistent connections, request pipelining, and chunked transfer encoding.

There is a single HTTP server. The same server has methods to store/retrieve the public keys (ECDH key exchange) and store/retrieve data (JavaScript Object Notation (JSON document) (as seen in Fig. 2).

The typical sequence of messages is depicted in Fig. 3. Consider that the observer and the sensor do not operate at the same speed. Some retries should exist in a regular execution (the message retries are not depicted within the diagram).

5.3 MQTT

MQTT is a standardized publish/subscribe Machine to machine (M2M) communication protocol [39]. This protocol has low network overhead and can be implemented on low-power devices such as microcontrollers that might be used in IoT scenarios as a communication protocol for sensors [22]. It is lightweight M2M communication for constrained networks.

MQTT client publishes messages to an MQTT broker, which are subscribed by other clients or may be retained for future subscriptions. Every message is published to an address known as a topic [29]. Clients can subscribe to multiple topics and receive every message published on each topic. It uses TCP as a transport protocol and TLS for security. Thus, communication between the client and broker is connection-oriented. Another great feature of MQTT is its three levels of QoS for the reliable delivery of messages [5]. It is a very basic messaging protocol offering only a few control options.

There are two servers: the key server (the same HTTP from the previous execution) and the MQTT server. The key exchange is performed at the beginning of the session. After that, the observer subscribes to the topic **sensor/bme280**, and the sensor publishes to that topic (as seen in Fig. 4).

The typical sequence of messages is depicted in Fig. 5. Considering that the observer and the sensor do not operate at the same speed, some retries should exist during the key exchange operation.

6 Results and Discussion

As previously stated, we want to evaluate the impact of security mechanisms in WSN. It is unfeasible to select all possible combinations, but we tried to select a representative approach. It is essential to mention that the selection of secure algorithms was guided by the recommendations present within the official Arduino Cryptography Library[8]. ECDH [14] was selected as the method to exchange the secret key for that session securely, as suggested by the guidelines of the previously mentioned library. The library offers two different ECDHs: curve25519 and P-521. ESP devices support some hardware accelerator cryptographic operations, but there are more limited than the cryptographic library offered by Arduino. This library is also available on a more extensive set of embedded devices, not only Arduino and ESP.

Each embedded device will be explored considering the HTTP and MQTT protocols, the ECDH with curve25519 and P-521, and the AES[128, 192, 256] and Chacha20 as cipher algorithms.

ChaCha [6] is a stream cipher, while the AES [10] is a block cipher. The main difference between a Block cipher and a Stream cipher is that a block cipher converts the plain text into cipher text by taking plain text one well-defined data

[8] https://rweather.github.io/arduinolibs/crypto.html

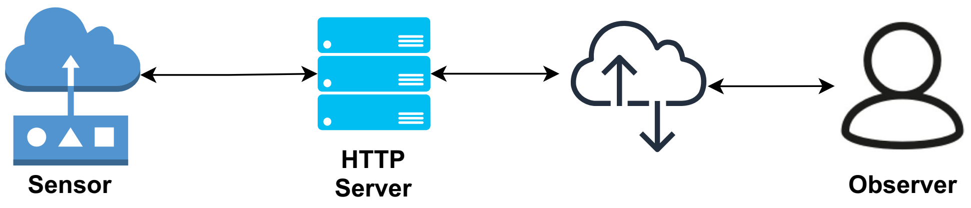


Fig. 2 WSN architecture for HTTP

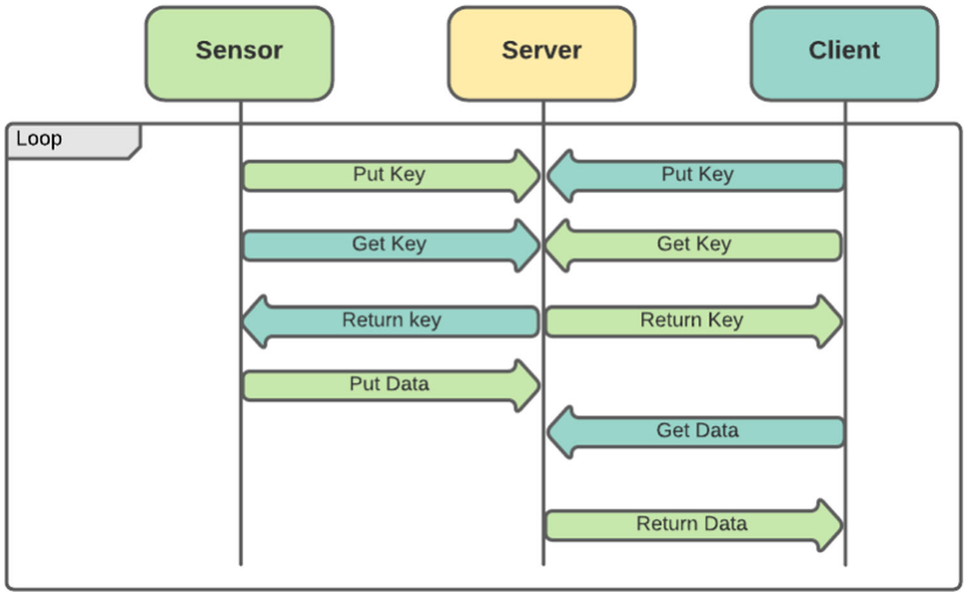


Fig. 3 HTTP messages exchanged between the entities

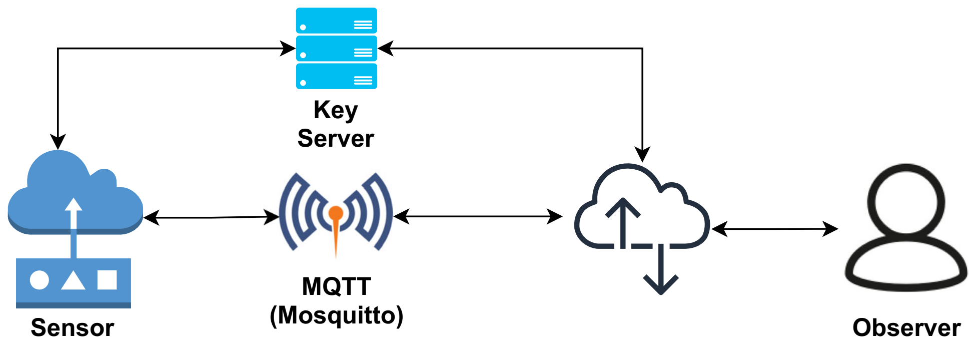


Fig. 4 WSN architecture for MQTT

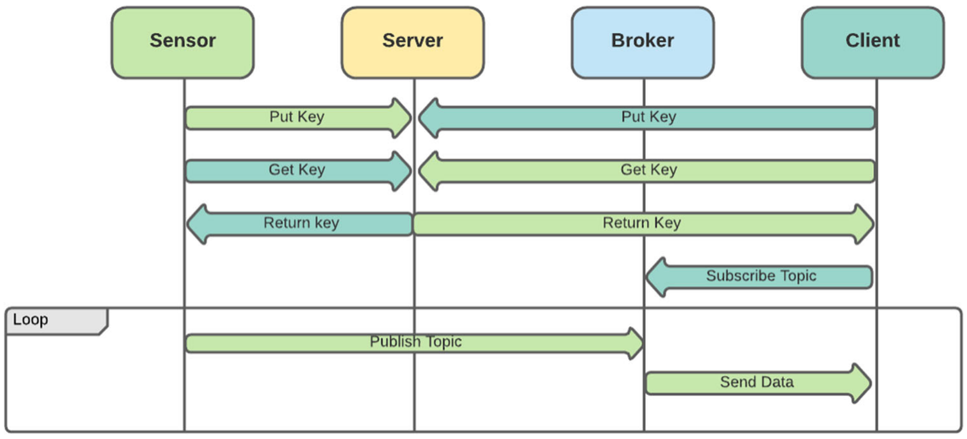


Fig. 5 MQTT messages exchanged between the entities

chunk at a time. In contrast, a stream cipher converts the plain text into cipher text by taking 1 byte of plain text at a time.

Each configuration runs for about one hour, totaling 72 hours for the entire experiment (48 hours for HTTP and 24 for MQTT). For each configuration, we measured three different latency (delay) values (as described in Section 5): the key agreement (in both parties), the transport latency, and the encryption/decryption latency (encryption on the sensor and decryption on the client). The timestamp differences measured the transport latency by taking advantage of Network Time Protocol (NTP) and the synchronization clocks between the devices.

After running the experiments described in Section 5, we gathered the power consumption and the delay added by the different security mechanisms. The results can be found in Tables 5 and 4.

The following subsections discuss the results from three perspectives: the delay added by the security mechanisms, the measured overhead in bytes added, and the recorded power consumption.

Without considering the transport delay between the HTTP and MQTT protocols, there are no major differences between their key agreement and encryption delays. For this reason, we will focus on the HTTP protocol delays. Furthermore, we are only focusing on the delays of the sensor device and not the client since the client in this experiment was executed in a conventional computer.

6.1 Delay Analysis

Before discussing the delay of the security mechanisms, it is important to mention that the transport delay in the HTTP mechanism is not as reliable as with the MQTT protocols. Due to the Request/Reply nature of the HTTP protocol, both the sensor and the observer must perform polling on the GET operations. The server is coded to return 400 BAD REQUEST when the data is unavailable, and the sensor/observer retries the request after some time. These polling operations lead to inconsistencies in the transport delay. The measured delay is the sum of the actual delay and the synchronization made by the entities during the polling operations.

These retries do not occur with the Publish/Subscribe nature of the MQTT protocol. After the initial key agreement algorithm, the observer subscribes to the correct topic and awaits the messages forwarded by the broker.

In Fig. 6, we can find the delays of the key agreement protocols for each board (the time axis is in the logarithm scale). As expected, the usage of different cipher algorithms does not impact on the delay of the key agreement protocol. The fastest boards (RPi) can compute the shared key with a smaller delay. Finally, the delay of P-521 is several times higher than that of Curve-25519.

In Fig. 7, we can find the delays from the cipher algorithm for each board. As expected, the stream cipher ChaCha20 is the fastest overall. The block cipher AES has a delay

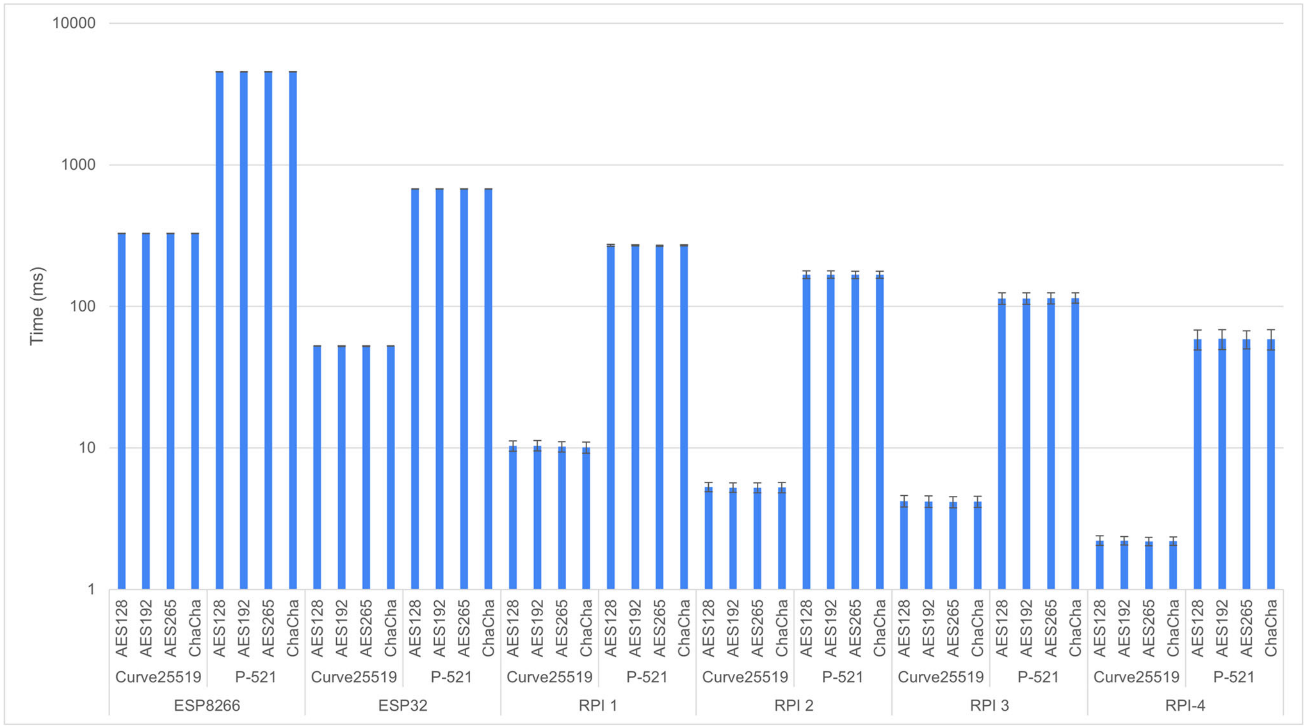


Fig. 6 ECDH delay for the different embedded devices. The time axis is in the logarithm scale

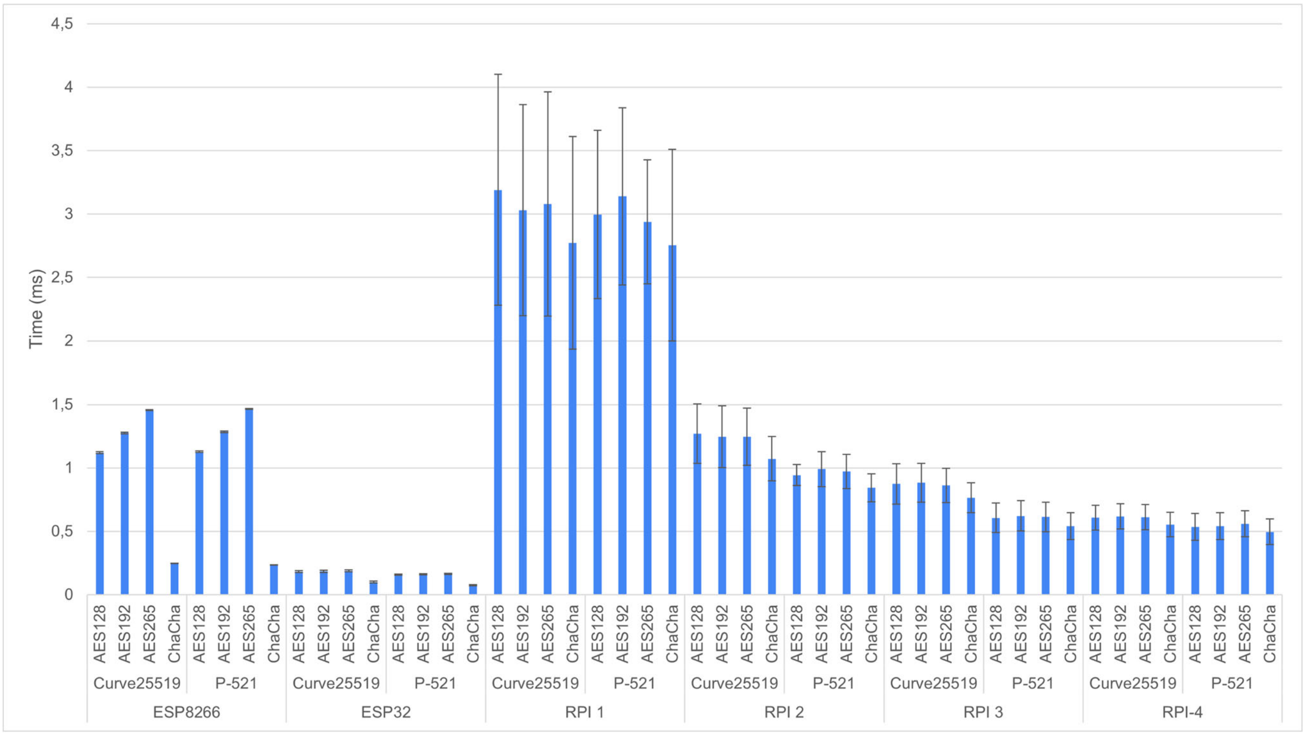


Fig. 7 Cipher delay for the different embedded devices

proportional to its key size. Contrary to the results with ECDH, the fastest boards (RPis) have the worst times for the encryption operation. There are some possible explanations for this. First, the code for the ESP devices is a highly optimized library developed especially for small boards. In contrast, the Python library on Raspberry Pi devices uses OpenSSL as the backend for cryptographic computations. To the best of our knowledge, the OpenSSL shipped with the Operative System (OS) is not optimized for embedded devices. Second, while the ESP devices only run a single program (do not have an OS), the Raspberry Pi has a full-fledged OS running with other processes.

6.2 Size Analysis

Another way to measure the impact of security methods is to compute the ratio of extra bytes shared between the entities. Let us assume that the normal message, without any cipher applied, is N bytes. All the cipher algorithm evaluated requires an initialization vector (IV) to be sent to the other entity. Lets us define the size of IV as I. For the ephemeral DH, changing the public key between the two entities is also necessary. Let us define the size of the public keys as P.

The ratio for ephemeral DH is given by equation 1, where N is the size of the original message (plain text), P is the size of the public key, and I is the size of the IV.

$$\frac{N + 2P + I}{N} \tag{1}$$

Both P and I are constants during the session. We can simplify the ratio by replacing the $2P + I$ with k (a constant for the session). This simplification generalizes the equation, making it usable for the static DH scenario. In this scenario, the value k would be only the size of the IV: $k = I$. The public keys are shared only once at the beginning of the session.

The ratio equation simplifies to equation 2 where N is the size of the original message (plain text), and k is the constant value that depends on the type of ECDH (static or ephemeral).

$$\frac{N + k}{N} \tag{2}$$

One crucial property of equation 2 is that the limit of the ratio when the size of the data increases ($N \to \infty$) the ratio tends to 1 (as seen in equation 3).

$$\lim_{N\to\infty} \frac{N + k}{N} = 1 \tag{3}$$

The main of this property is that, since the security bytes are finite in size, as the data part of the message increases, the extra bytes added by the security methods are negligible. Table 3 lists the ratios for the experiment done in this work.

Table 3 Ratios for the experiment

Protocol	ECDH	Ratio
HTTP	Curve25519	163%
	P-521	322%
MQTT	Curve25519	113%
	P-521	113%

The size of the raw message is 128 bytes, and the IV is 16 bytes. The size of the public keys is 32 and 132 bytes for Curve25519 and P-521, respectively. As stated previously, for the static ECDH, we do not consider the size of the public keys. The impact of security measures is significant when considering ephemeral DH and is minimized with the static DH.

6.3 Modeling Security Algorithms

The authors of [13] present a simple linear model to estimate the time a security operation lasts based on the text size and the CPU frequency, and bus size (see equation 4).

$$t = \frac{a + b\lceil\frac{text\_length}{block\_size}\rceil}{cpu\_frequency \times bus\_witdh} \quad (4)$$

We adopted this model as it appeared to work well on the boards evaluated in the paper. The ESP boards are quite similar to the Arduinos used by the authors. The RPi devices are more complex and may not be correctly modeled with a simple linear equation. In this subsection, we explore if a simple linear equation is sufficient to model the typical cryptographic algorithms used in WSN.

The variable $text\_length/block\_size$ is constant within our simulation, all messages have a fixed length (128 bytes), and the block size for all cipher algorithms evaluated is 16 bytes. With a simple reorganization, it can be written as seen in equation 5, where p is $text\_length/block\_size$ and c is $cpu\_frequency \times bus\_witdh$.

$$t = a \times \frac{1}{c} + b \times \frac{p}{c} \quad (5)$$

We now have a linear equation where t is the dependent variable, c and p are the independent variables, and a and b are the weights of the linear system. Since it is a linear equation, we can use the least square optimization to find the ideal values of a and b. We generated a dataset using an Excel spreadsheet and a Matlab script to perform the linear regression (the script uses the **regress** command).

Using the values from our simulation (for the cipher algorithms), we obtained the following values: $a = 1.2869E09$ and $b = -3.2171E08$. However, the coefficient of determination (R^2) was close to zero, meaning that our data could not be fitted with a linear function. This result is expected. To illustrate this point, in Fig. 8, we plotted the relationship between $\frac{1}{c}$ and the encryption duration (t). As we can see, the boards with less computational capabilities (represented by the red line) achieved faster encryption time (represented by the blue bars).

There are two possible explanations for this. First, the code for the ESP devices is a highly optimized library developed especially for small boards. In contrast, the Python library on Raspberry Pi devices uses OpenSSL as the backend for cryptographic computations. To the best of our knowledge, the OpenSSL shipped with the OS is not optimized for the devices. Second, while the ESP devices only run a single program (do not have an OS), the Raspberry Pi has a full-fledged OS running with other processes.

The next logical step is to divide the dataset into two, one for ESP-based devices and another for RPi-based devices.

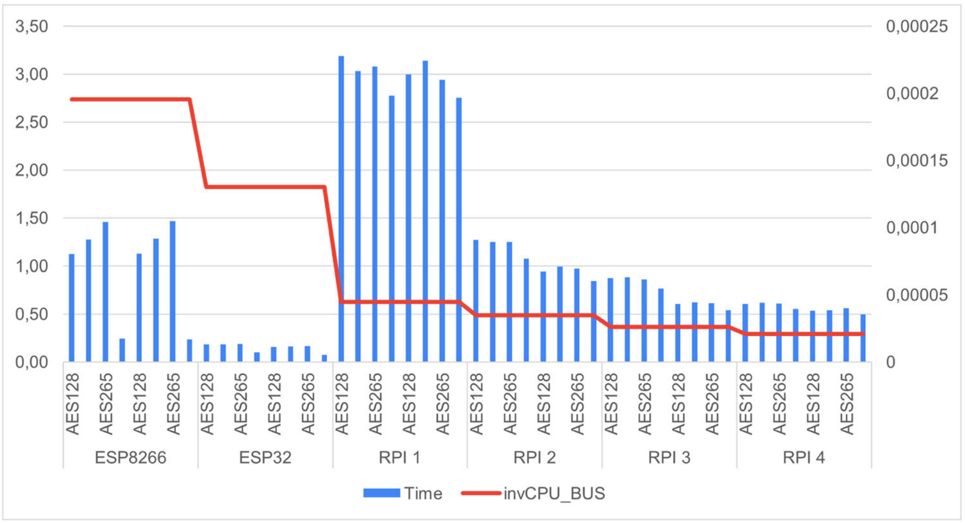


Fig. 8 Relation between computational capabilities ($\frac{1}{c}$) and the encryption duration (t ms). The boards with less computational power achieved faster encryption times

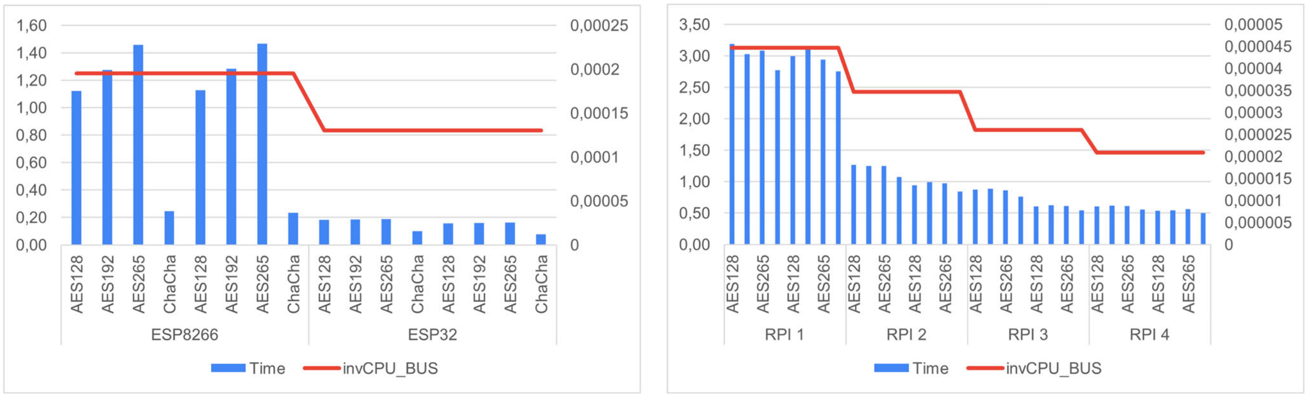

(a) ESP based devices (b) RPi based devices

Fig. 9 Relation between $\frac{1}{c}$ and the cipher duration (t)

Given the significant differences between these types of boards, it is difficult to assume that a linear model could capture the underlying model. We again plotted the relationship between the variables in Fig. 9. The relation for the RPi devices appears to follow a linear approach. On the other hand, ESP devices are not as well defined. The key size appears to have greater importance for the ESP devices (the AES algorithm uses three key sizes [128, 192, 256] and the ChaCha also uses 256). And the type of cipher algorithms: block (AES) or stream (ChaCha) cipher.

We applied the previous linear model to both datasets (ESP and RPi). The RPi dataset, the model reached a 0.98 of R^2 with the following values: $a = 7.6329E09$ and $b = -1.9082E09$.

For the ESP dataset, the model reached a 0.633 of R^2 with the following values: $a = 9.1344E8$ and $b = -2.2836E8$. The linear model is a good fit for the RPi devices. Although the performance of the ESP devices increased, the value of 0.633 is not close enough to be a relevant model.

We expanded the linear model to have two other variables, as seen in equation 6, where p is $text\_length/block\_size$, c is $cpu\_frequency \times bus\_witdh$, k represents the key size [128, 192, 256], t represents the type of cipher (0 for block and 1 for stream) and $[a, b, d, e]$: are the weights of the previous mentioned independent variables.

$$t = a \times \frac{1}{c} + b \times \frac{p}{c} + d \times k + e \times t \tag{6}$$

After adding these two values to the dataset, the performance increased to 0.844 R^2 with the following values: $a = 9.7978E8$, $b = -2.4495E8$, $d = 1.3425E - 3$ and $e = -6.5253e - 1$. The last mentioned models could capture the linear pattern from the underlying model and can be useful for the modulation and estimation of security overhead on embedded devices.

Finally, we adapted the linear equation from [13] to ECDH time estimation. Similarly to the previous analysis, we plotted

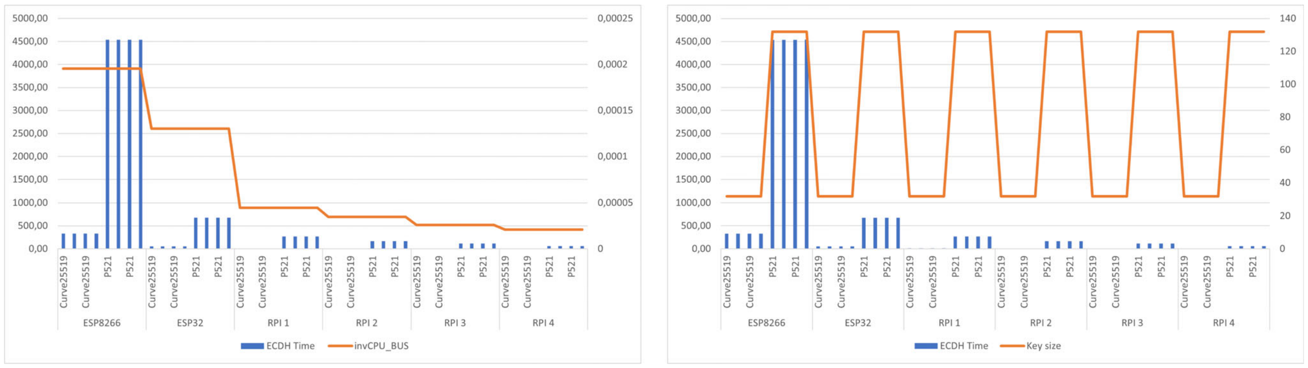

(a) Relation between the ECDH duration with the $\frac{1}{c}$. (b) Relation between the ECDH duration with public key size.

Fig. 10 Relation between the ECDH duration with the $\frac{1}{c}$ and the ECDH duration with the public key size

the relationship between the ECDH duration with the $\frac{1}{c}$ and the ECDH duration with the public key size (see Fig. 10).

Although the $\frac{1}{c}$ does have a perfect linear relation with the duration of the ECDH algorithm. The linear model becomes a better fit by adding the public key size (second plot). The adapted linear equation is given by equation 7, where c, is $cpu\_frequency \times bus\_witdh$, k is the size of the public key used by the algorithm and $[a, b]$: are the weights of the previous mentioned independent variables.

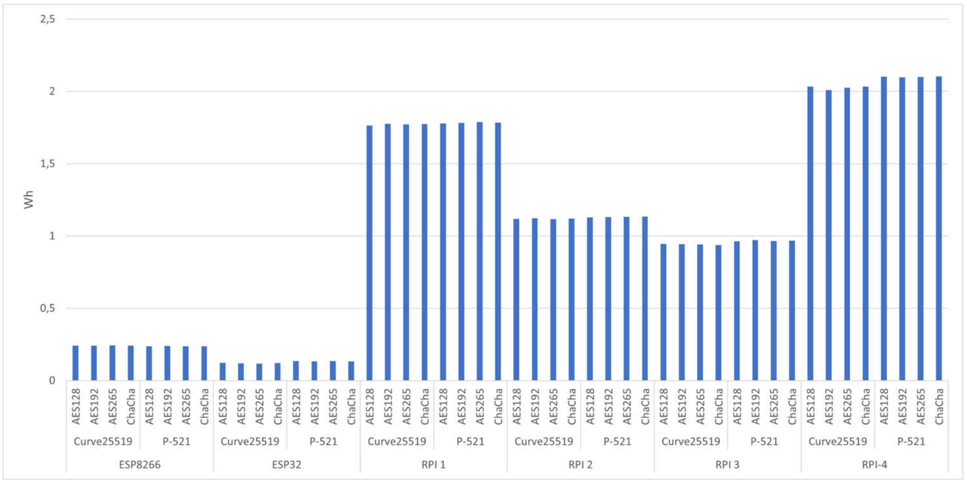
$$t = a \times \frac{1}{c} + b \times \frac{k}{c} \tag{7}$$

By replacing the values and using the previously mentioned least square optimization, we get a value of 0.751 R^2 with the following values: $a = -3.7727E6$ and $b = 1.5566E5$. Although it is the lowest value obtained (from valid models), this model combines ESP and RPi devices, making this model generic. It also points to another interesting result, the internal implementation of ECDH is not/could not be appropriately accelerated by hardware.

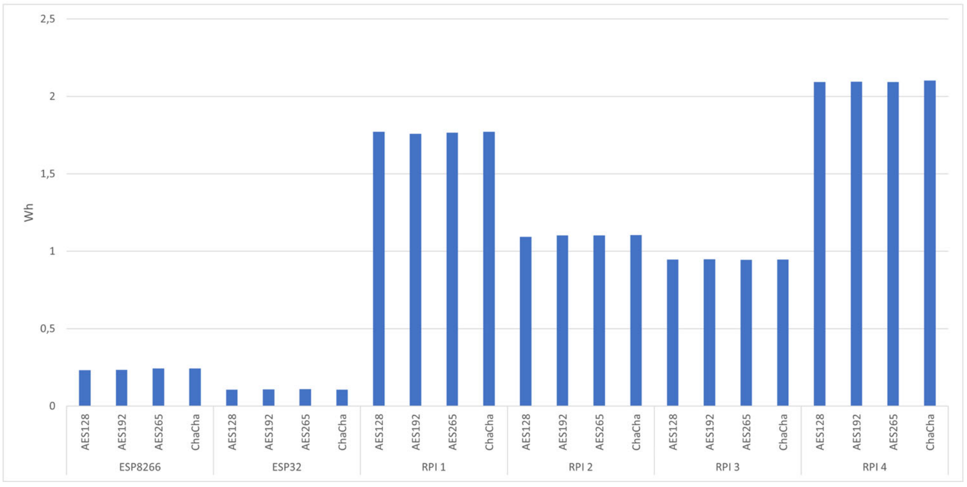
6.4 Power Consumption Analysis

We observed in Fig. 11 (as in Tables 5 and 4) that the power consumption within embedded devices does not change significantly with each security configuration. It considerably varies when taking into account the communication protocol, with HTTP consuming slightly more (as it implements the Ephemeral DH). A possible reason for this behavior is that the sensor only publishes a message per second, causing the CPU to be idle most of the time.

Although power consumption does not vary greatly in the same embedded device, the most potent devices consume less power during the experiment, except for the RPi 4. This statement appears counter-intuitive but is explained by the



(a) Power consumption on the HTTP protocol.



(b) Power consumption on the MQTT protocol.

Fig. 11 Power consumption on the booth protocols

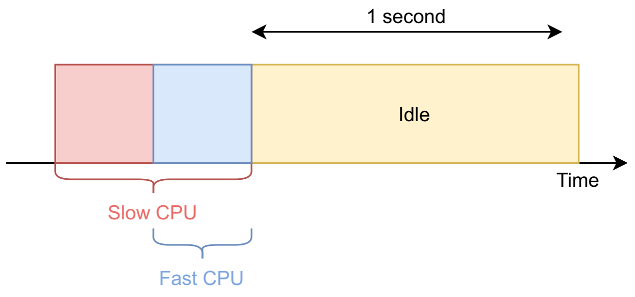


Fig. 12 CPU processing time for a single message

previous observation. Most of the time, the CPU is idle. The fastest CPU spends more time idle, while the slower CPU spends more time in load, as seen in Fig. 12.

Accounting for this imbalance means that, for this scenario, the ESP32 consumes less power than the ESP8266. Similarly, RPi 3 consumes less power than RPi 1 and 2. The direct comparison between ESP and Raspberry Pi is not interesting. The ESP-embedded device is rather simple and can only run a single program. While the Raspberry Pi is a small computer that runs a full-fledged OS and can execute multiple processes.

RPi 4 is an exception to this observation. Although the CPU speed only increased 300Mhz from the previous iteration, several other changes were made to the boards, such as adding a novel power delivery system based on USB-C, the addition of Wireless AX and gigabit Ethernet, and finally, the addition of a USB3 chip and ports. All of these changes may explain the high power consumption of this board.

7 Related Work

The impact of cryptographic algorithms in WSN has been studied in the literature. It is not a recent problem, but different lightweight algorithms and devices are considered in various studies. In addition, different applicability scenarios are contemplated.

Dhanda et al. [8] provided a comprehensive survey where 21 lightweight block ciphers, 19 lightweight stream ciphers, 9 lightweight hash functions, and 5 variants of ECC were compared in terms of chip area, energy, power, hardware and software efficiency, throughput, latency, and Figure of Merit (FoM). The conclusion resulted from comparing the remaining practical studies in the literature where the performed simulations outputted results to compare the performance of different algorithms. The results showed that AES remains the preferred choice for security, and it is the most competitive cipher among block ciphers; ECC also remains the option that provides authentication and non-repudiation in addition to confidentiality. However, this study does not contemplate a selection of devices and respective analyses of the algorithms.

Haque et al. [16] focused on performance analysis of cryptographic algorithms (AES, RC4, Blowfish, CAST, 3DES, Twofish as symmetric key encryption and Digital Signature Algorithm (DSA) and ElGamal as asymmetric encryption algorithms) in resource-constrained devices and examined the key size, data blocks, data types, and encryption/decryption speed. However, this study did not examine the impact on different devices and provided no comparison between devices, only between algorithms. We considered it relevant to experiment with how different devices and algorithms impact these parameters to provide a reference for future researchers or developers simply identify the best combination of device and algorithm according to therequirements.

Fotovvat et al. [12] proposed a more practical analysis considering devices and algorithms, similar to the idea of the current paper, and compared the performance of 32 lightweight algorithms from different categories permutation, block cipher, stream cipher, and tweakable block cipher). These algorithms were tested in well-known devices (RPi 3, RPi Zero W, and iMX233, in terms of power consumption, random access memory usage, and execution time. One of the outcomes of this study is to assist researchers in weighing the pros and cons of different design topologies. The idea of this study is very aligned with the current paper, and both consider the test on RPi 3. However, both papers consider other different devices and other lightweight algorithms. [12] focused on recent algorithms that could be used in future applications, while the current paper focused on existing and severely used algorithms.

8 Conclusion

In this paper, we considered the security impact on systems performance as a scientific contribution in terms that researchers or service providers can directly understand how systems may be affected by cryptographic algorithms. In this case, we focused on WSN since these environments traditionally handle constrained devices with less computational power to run security algorithms. We considered this analysis important due to the high applicability of these network architectures in Smart Homes. The lack of systems mechanisms to prevent abusing the users' privacy affects the users' safety severely. Security mechanisms are important to prevent users' privacy abuse and to avoid malicious entities accessing unauthorized data. In this way, this paper focuses on a specific case where boards used on smart devices, typically applied in smart homes, were analyzed to help users, developers, or service providers choose the most appropriate cryptographic algorithm for the corresponding board in order to optimize the computational power and energy-efficient performance of the system, to ensure the existence of security.

This paper intends to address important identified gaps: security mechanisms' overheads within small sensing devices are still not completely explored; previous studies focused on measuring the delay added by the security layer but were still missing studies considering the power consumption. For those reasons, the described study considers the power consumption analysis within representative WSN use-cases, using SoA communications protocols (HTTP and MQTT) and representative embedded devices (used in usual and common smart home scenarios): ESP8622, ESP32, RPi1, RPi2, RPi3, and RPi4.

We found that the security algorithms on WSN considerably impact the size of the messages and times of execution. Although the power consumption within the same embedded device does not vary with the multiple security configurations, the boards with a faster CPU tend to use less power. The literature review in Section 7 did not clearly indicate that the power consumption would follow the pattern identified in this work. As the size of the raw message increases, the extra bytes added by the security algorithms becomes less relevant.

A possible limitation is the number of brands of boards selected for the study. In fact, it was impractical to choose and implement all the existing boards. For this reason, we selected the most significant ones.

Acknowledgements This work is funded by FCT/MCTES through national funds and when applicable co-funded EU funds under the project UIDB/50008/2020-UIDP/50008/2020.

Funding Open access funding provided by FCT|FCCN (b-on).

Appendix A: Experimental Results.

Table 4 Results from the experiment MQTT

Device	Cipher	Encrypt (μs)	Transport (μs)	Decrypt (μs)	Wh	Ah
ESP8266	AES128	1099,02 ± 15,69	588079,85 ± 447649,23	175,54 ± 11,72	0,233	0,051
	AES192	1265,59 ± 15,66	497408,47 ± 471691,53	170,90 ± 15,19	0,235	0,051
	AES256	1454,65 ± 16,31	666822,48 ± 436653,25	174,74 ± 12,27	0,243	0,053
	ChaCha20	261,96 ± 4,30	770039,26 ± 386658,71	157,80 ± 13,05	0,243	0,053
ESP32	AES128	141,34 ± 6,80	471233,51 ± 490600,43	176,90 ± 19,10	0,107	0,023
	AES192	144,82 ± 7,74	816345,09 ± 370775,30	178,18 ± 16,32	0,109	0,024
	AES256	147,23 ± 7,68	797144,29 ± 368438,65	178,16 ± 23,00	0,110	0,024
	ChaCha20	69,15 ± 5,81	965871,42 ± 97463,14	160,02 ± 16,10	0,107	0,024
RPi 1	AES128	2642,76 ± 231,96	335339,04 ± 465086,92	174,43 ± 14,28	1,772	0,379
	AES192	2669,88 ± 204,42	563632,35 ± 477824,09	170,77 ± 13,68	1,759	0,376
	AES256	2686,75 ± 199,34	21188,47 ± 3214,50	169,40 ± 12,55	1,766	0,378
	ChaCha20	2410,02 ± 198,65	18266,81 ± 3698,13	156,93 ± 11,19	1,772	0,379
RPi 2	AES128	1150,27 ± 50,30	13507,98 ± 57757,64	171,13 ± 12,67	1,093	0,234
	AES192	1180,06 ± 48,32	16632,30 ± 28281,20	172,57 ± 12,71	1,103	0,236
	AES256	1169,76 ± 46,22	34044,58 ± 161167,64	174,06 ± 12,98	1,103	0,236
	ChaCha20	1055,62 ± 66,21	11193,80 ± 28012,58	156,55 ± 10,34	1,104	0,237
RPi 3	AES128	876,62 ± 17,13	21084,11 ± 24397,87	169,12 ± 11,28	0,946	0,203
	AES192	882,09 ± 17,74	20193,67 ± 14495,71	172,12 ± 9,24	0,949	0,203
	AES256	895,93 ± 17,76	16769,77 ± 5991,42	167,77 ± 12,88	0,944	0,202
	ChaCha20	794,51 ± 17,28	19890,03 ± 32270,03	154,18 ± 11,96	0,946	0,203
RPi 4	AES128	565,74 ± 27,67	192736,05 ± 127523,86	336,30 ± 47,30	2,094	0,409
	AES192	584,02 ± 27,59	219108,65 ± 71331,81	334,91 ± 43,18	2,095	0,410
	AES256	28,06 ± 222431,83	232822,39 ± 340,03	46,56 ± 12,88	2,093	0,408
	ChaCha20	510,95 ± 24,47	324192,93 ± 349209,21	317,27 ± 43,45	2,103	0,411

Table 5 Results from the experiment HTTP

Device	ECDH	Cipher	Sensor Key (μs)	Encrypt (μs)	Transport (μs)	Client Key (μs)	Decrypt (μs)	Wh	Ah
ESP8266	Curve25519	AES128	328562,84 ± 33,98	1120,59 ± 7,03	1021216,14 ± 563814,05	366,54 ± 1439,17	145,54 ± 10,97	0,241	0,052
		AES192	328571,03 ± 30,24	1275,68 ± 5,33	1012832,75 ± 565148,42	366,98 ± 1442,93	146,95 ± 9,27	0,241	0,052
		AES256	328570,79 ± 31,11	1456,80 ± 2,17	1022357,36 ± 557677,59	372,20 ± 1476,97	149,53 ± 19,47	0,244	0,053
		ChaCha20	328562,14 ± 31,95	245,76 ± 1,78	992376,79 ± 567520,10	362,68 ± 1371,23	134,37 ± 10,40	0,241	0,052
	P-521	AES128	4538532,35 ± 720,12	1127,87 ± 5,99	1017741,15 ± 593570,26	1614,42 ± 2242,46	165,41 ± 12,58	0,238	0,052
		AES192	4538642,86 ± 785,22	1284,90 ± 5,52	1012657,58 ± 561245,13	1682,99 ± 3479,68	163,40 ± 8,24	0,240	0,052
		AES256	4538579,54 ± 703,47	1466,50 ± 2,99	1012370,59 ± 562113,09	1544,99 ± 1618,86	154,17 ± 8,51	0,238	0,052
		ChaCha20	4538672,72 ± 657,60	235,12 ± 0,76	1000746,16 ± 551824,87	1640,99 ± 3442,16	141,56 ± 8,35	0,238	0,052
ESP32	Curve25519	AES128	52469,85 ± 185,16	181,69 ± 7,93	1021560,08 ± 552853,00	355,90 ± 1115,86	151,17 ± 7,86	0,122	0,026
		AES192	52473,80 ± 303,66	185,20 ± 8,95	1013823,71 ± 560445,54	380,09 ± 1419,94	155,85 ± 9,89	0,119	0,026
		AES256	52472,28 ± 302,33	188,19 ± 7,92	1058468,20 ± 853065,40	377,14 ± 1408,45	152,67 ± 9,54	0,116	0,026
		ChaCha20	52459,38 ± 236,85	99,66 ± 7,03	1022617,20 ± 560080,61	376,90 ± 1394,94	137,71 ± 12,70	0,121	0,027
	P-521	AES128	676896,17 ± 283,98	157,65 ± 5,62	1032991,16 ± 549276,57	1548,22 ± 1544,35	157,77 ± 18,02	0,135	0,030
		AES192	676901,10 ± 349,02	160,59 ± 4,78	1024080,45 ± 557776,50	1542,37 ± 1481,72	156,05 ± 11,56	0,134	0,029
		AES256	676897,11 ± 324,67	163,66 ± 4,69	1045246,19 ± 841745,54	1533,76 ± 1425,47	156,80 ± 11,35	0,136	0,029
		ChaCha20	676918,75 ± 273,26	76,13 ± 4,00	1021810,64 ± 557622,40	1558,35 ± 1544,31	148,77 ± 11,81	0,133	0,029
RPi 1	Curve25519	AES128	10330,28 ± 864,52	3189,96 ± 910,34	1090885,59 ± 517115,84	392,37 ± 1909,78	155,34 ± 8,99	1,765	0,378
		AES192	10373,21 ± 871,76	3029,81 ± 833,22	1097316,15 ± 514032,84	350,36 ± 1311,35	156,87 ± 9,12	1,776	0,381
		AES256	10205,97 ± 849,14	3080,56 ± 884,30	1100755,83 ± 508087,65	349,83 ± 1348,58	155,06 ± 9,20	1,773	0,379
		ChaCha20	10073,53 ± 890,71	2773,06 ± 837,46	1091973,55 ± 512154,12	369,61 ± 1347,99	144,49 ± 13,54	1,774	0,379
	P-521	AES128	270039,54 ± 4224,22	2996,39 ± 665,06	1035805,84 ± 521418,35	1585,33 ± 1270,63	155,99 ± 15,92	1,778	0,383
		AES192	269208,55 ± 2699,69	3139,63 ± 698,39	1041563,19 ± 525724,66	1590,42 ± 1280,90	154,05 ± 12,94	1,782	0,383
		AES256	268270,95 ± 2837,70	2939,13 ± 489,26	1032561,03 ± 525718,10	1563,79 ± 1260,25	156,11 ± 14,53	1,788	0,383
		ChaCha20	269403,18 ± 2615,00	2754,02 ± 755,38	1029914,41 ± 521869,06	1600,68 ± 2029,12	143,38 ± 13,38	1,785	0,383
RPi 2	Curve25519	AES128	5308,18 ± 408,47	1269,44 ± 234,67	1027858,12 ± 554080,67	364,06 ± 1835,65	157,67 ± 9,95	1,118	0,239
		AES192	5261,29 ± 407,49	1246,49 ± 243,91	1038463,43 ± 567412,46	349,92 ± 1283,08	155,68 ± 9,39	1,123	0,240
		AES256	5244,20 ± 415,64	1246,87 ± 225,10	1028524,72 ± 573442,20	350,77 ± 1403,18	155,82 ± 12,76	1,117	0,240
		ChaCha20	5271,91 ± 435,08	1072,85 ± 174,18	1034310,65 ± 570468,54	349,47 ± 1281,43	143,09 ± 15,17	1,121	0,240
	P-521	AES128	167443,44 ± 10162,30	942,94 ± 82,79	990104,56 ± 569967,35	1553,42 ± 1286,20	156,63 ± 20,27	1,128	0,242
		AES192	167841,92 ± 10125,03	990,49 ± 137,74	1003211,73 ± 569932,40	1584,25 ± 1340,26	154,63 ± 17,81	1,131	0,241
		AES256	167461,08 ± 9895,18	972,11 ± 134,25	994980,04 ± 561239,06	1592,87 ± 1350,18	153,18 ± 12,45	1,132	0,242
		ChaCha20	167520,59 ± 9743,76	843,94 ± 111,02	996866,58 ± 567634,20	1583,58 ± 2151,70	143,31 ± 15,01	1,134	0,244

Table 5 continued

Device	ECDH	Cipher	Sensor Key (μs)	Encrypt (μs)	Transport (μs)	Client Key (μs)	Decrypt (μs)	Wh	Ah
RPi 3	Curve25519	AES128	4212,93 ± 390,06	874,43 ± 160,00	1043848,93 ± 571232,40	367,42 ± 1382,54	166,91 ± 13,02	0,946	0,202
		AES192	4200,14 ± 385,97	883,54 ± 152,65	1030993,07 ± 565133,03	373,24 ± 1423,73	172,06 ± 27,40	0,943	0,201
		AES256	4162,91 ± 372,77	862,46 ± 134,56	1042588,78 ± 576387,43	356,93 ± 1389,07	164,56 ± 15,75	0,942	0,201
		ChaCha20	4182,29 ± 382,24	764,38 ± 117,99	1034179,29 ± 573235,91	340,63 ± 924,46	150,34 ± 11,06	0,938	0,200
	P-521	AES128	114047,24 ± 10663,40	606,61 ± 116,23	1050853,72 ± 571558,64	1622,43 ± 1480,62	167,12 ± 22,20	0,963	0,206
		AES192	114289,52 ± 10691,42	622,13 ± 120,41	1056427,11 ± 569460,77	1616,82 ± 1412,84	165,56 ± 13,09	0,971	0,206
		AES256	114365,56 ± 10037,19	613,38 ± 116,22	1062689,38 ± 563629,00	1615,12 ± 1438,33	165,96 ± 16,81	0,966	0,207
		ChaCha20	114915,53 ± 9528,70	541,87 ± 105,37	1067784,71 ± 564839,31	1612,49 ± 1485,99	150,01 ± 13,64	0,967	0,207
RPi 4	Curve25519	AES128	2222,54 ± 174,04	607,08 ± 98,19	1045067,81 ± 563731,33	626,42 ± 516,33	331,80 ± 80,75	2,033	0,397
		AES192	2213,14 ± 147,91	617,84 ± 100,88	1037054,47 ± 528603,32	376,89 ± 1170,76	183,36 ± 98,01	2,01	0,394
		AES256	2190,75 ± 147,30	611,25 ± 100,27	1065369,01 ± 518942,79	365,97 ± 505,11	184,11 ± 98,28	2,026	3,96
		ChaCha20	2197,98 ± 151,61	553,99 ± 96,99	1025877,48 ± 569577,46	373,51 ± 545,21	172,06 ± 90,76	2,034	0,398
	P-521	AES128	58629,21 ± 9434,25	534,67 ± 105,05	1115914,05 ± 534340,76	1576,50 ± 1094,37	184,70 ± 99,58	2,102	0,411
		AES192	59042,43 ± 9527,55	540,78 ± 105,19	1078371,55 ± 536770,04	1591,91 ± 1032,89	186,56 ± 101.16	2,099	0,411
		AES256	58659,64 ± 8407,56	559,50 ± 101,64	1075373,69 ± 562454,01	1612,81 ± 1061,51	185,20 ± 98,76	2,101	0,411
		ChaCha20	58731,05 ± 9627,96	496,43 ± 102,15	1067970,57 ± 582711,46	1583,49 ± 1040,67	171,30 ± 91,04	2,104	0,411

References

1. Abu-Tair, M., Djahel, S., Perry, P., et al. (2020). Towards secure and privacy-preserving IoT enabled smart home: architecture and experimental study. *Sensors, 20*(21), 6131.
2. Ahmed, A. A. (2021). Lightweight digital certificate management and efficacious symmetric cryptographic mechanism over industrial internet of things. *Sensors, 21*(8), 2810.
3. Alavi, A. H., Jiao, P., Buttlar, W. G., et al. (2018). Internet of things-enabled smart cities: State-of-the-art and future trends. *Measurement, 129*, 589–606.
4. AlHammadi, A., AlZaabi, A., AlMarzooqi, B., et al. (2019) Survey of iot-based smart home approaches. In: ASET, pp. 1–6
5. Bandyopadhyay, S., & Bhattacharyya, A. (2013). Lightweight internet protocols for web enablement of sensors using constrained gateway devices. *2013 International Conference on Computing* (pp. 334–340). IEEE: Networking and Communications (ICNC).
6. Bernstein, D.J., et al. (2008) Chacha, a variant of salsa20. In: Workshop record of SASC, pp. 3–5
7. Chandel, S., Cao, W., Sun, Z., et al. (2019) A multi-dimensional adversary analysis of rsa and ecc in blockchain encryption. In: Future of information and communication conference. Springer, pp. 988–1003
8. Dhanda, S. S., Singh, B., & Jindal, P. (2020). Lightweight cryptography: a solution to secure iot. *Wireless Personal Communications, 112*(3), 1947–1980.
9. Diro, A., Reda, H., Chilamkurti, H., et al. (2020). Lightweight authenticated-encryption scheme for internet of things based on publish-subscribe communication. *IEEE Access, 8*, 60,539-60,551.
10. Dworkin, M. J., Barker, E. B., Nechvatal, J. R., et al. (2001). *Advanced encryption standard (AES)*. NIST Pubs: Tech. rep.
11. Farooq, U., Mushtaq, M., Bhatti, M. (2020) Efficient aes implementation for better resource usage and performance of iots. In: CYBER 2020
12. Fotovvat, A., Rahman, G. M., Vedaei, S. S., et al. (2020). Comparative performance analysis of lightweight cryptography algorithms for iot sensor nodes. *IEEE Internet of Things Journal, 8*(10), 8279–8290.
13. Ganesan, P., Venugopalan, R., Peddabachagari, P., et al. (2003) Analyzing and modeling encryption overhead for sensor network nodes. In: International conference on Wireless sensor networks and applications, pp. 151–159
14. Goyal, T.K., Sahula, V. (2016) Lightweight security algorithm for low power IoT devices. In: 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE
15. Hameed, A., Alomary, A. (2019) Security issues in iot: A survey. In: 3ICT, pp. 1–5
16. Haque, M.E., Zobaed, S., Islam, M.U., et al. (2018) Performance analysis of cryptographic algorithms for selecting better utilization on resource constraint devices. In: 2018 21st International Conference of Computer and Information Technology (ICCIT), pp. 1–6. https://doi.org/10.1109/ICCITECHN.2018.8631957
17. Jawad, H., Nordin, R., Gharghan, S., et al. (2017). Energy-efficient wireless sensor networks for precision agriculture: a review. *Sensors, 17*(8), 1781.
18. Kamal, A., Dahshan, H., Rohiem, A. (2013) An elliptic curve threshold group signature scheme. In: International conference on aerospace sciences and aviation technology, pp. 1–9
19. Kocakulak, M., Butun, I. (2017) An overview of wireless sensor networks towards internet of things. In: 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), pp. 1–6
20. Kondaveeti, H. K., Kumaravelu, N. K., Vanambathina, S. D., et al. (2021). A systematic literature review on prototyping with arduino: applications, challenges, advantages, and limitations. *Computer Science Review, 40*(100), 364.
21. Latif, R.M.A., Imran, L., Farhan, M., et al. (2019) Real-time simulation of iot based smart home live mirror using wsn. In: 2019 International conference on frontiers of information Technology (FIT), pp. 49–495
22. Light, R. A. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software, 2*(13), 265.
23. Majumder, S., Ray, S., Sadhukhan, D., et al. (2021). Ecc-coap: Elliptic curve cryptography based constraint application protocol for internet of things. *Wireless Personal Communications, 116*(3), 1867–1896.
24. Martínez, V.G., González-Manzano, L., Muñoz, A.M. (2017) Secure elliptic curves in cryptography. In: Computer and network security essentials. Springer International Publishing, pp. 283–298
25. Mehibel, N., Hamadouche, M. (2017) A new approach of elliptic curve diffie-hellman key exchange. In: ICEE-B, pp. 1–6
26. Mendez Mena, D., Papapanagiotou, I., & Yang, B. (2018). Internet of things: survey on security. *Information Security Journal: A Global Perspective, 27*(3), 162–182.
27. Mohammedi, M., Omar, M., Zamouche, D., et al. (2021) Energy-aware key management and access control for the internet of things. World Wide Web, pp. 1–32
28. Mousavi, S. K., Ghaffari, A., Besharat, S., et al. (2021). Security of internet of things based on cryptographic algorithms: a survey. *Wireless Networks, 27*(2), 1515–1555.
29. Naik, N. (2017) Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: 2017 IEEE International Systems Engineering Symposium (ISSE). IEEE
30. Nan, L.i. (2010) Research on diffie-hellman key exchange protocol. In: International conference on computer engineering and technology, pp. V4–634–V4–637
31. Riazi, M.S., Laine, K., Pelton, B., et al. (2020) Heax: an architecture for computing on encrypted data. In: Proceedings of the twenty-fifth international conference on architectural support for programming languages and operating systems, pp. 1295–1309
32. Rosner, G., Kenneally, E. (2018) Clearly opaque: Privacy risks of the internet of things. In: Gilad, R., Erin, K. Clearly opaque: privacy risks of the internet of things (May 1, 2018). IoT Privacy Forum
33. Sadkhan, S.B., Salman, A.O. (2018) A survey on lightweight-cryptography status and future challenges. In: ICASEA, pp. 105–108
34. Sallam, S., Beheshti, B.D. (2018) A survey on lightweight cryptographic algorithms. In: TENCON, pp. 1784–1789
35. Saraiva, D. A. F., Leithardt, V. R. Q., de Paula, D., et al. (2019). PRISEC: Comparison of symmetric key algorithms for IoT devices. *Sensors, 19*(19), 4312.
36. Shah, A., Engineer, M. (2019) A survey of lightweight cryptographic algorithms for iot-based applications. In: Smart innovations in communication and computational sciences. Springer, pp. 283–293
37. Silva, C., Barraca, J.P. (2022) Dynamic delegation-based privacy preserving in iot architectures. In: 2022 9th International Conference on Future Internet of Things and Cloud (FiCloud), IEEE, pp. 46–54
38. Singh, S., Sharma, P.K., Moon, S.Y., et al. (2017) Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions. Journal of Ambient Intelligence and Humanized Computing, pp. 1–18
39. Soni, D., Makwana, A. (2017) A survey on mqtt: a protocol of internet of things (iot). In: International conference on telecommunication, power analysis and computing techniques
40. Stolojescu-Crisan, C., Crisan, C., & Butunoi, B. P. (2021). An iot-based smart home automation system. *Sensors, 21*(11), 3784.

41. Surendran, S., Nassef, A., Beheshti, B.D. (2018) A survey of cryptographic algorithms for iot devices. In: LISAT, pp. 1–8
42. Thakor, V. .A., Razzaque, M. .A., & Khandaker, M. .R. .A. (2021). Lightweight cryptography algorithms for resource-constrained iot devices: A review, comparison and research opportunities. *IEEE Access, 9*, 28,177-28,193.
43. Thara, K.S. Vallala, P.G. (2020) A survey of encryption algorithms in IoT. 17th SC@ RUG 2019-2020, p. 9
44. Tozlu, S., Senel, M., Mao, W., et al. (2012). Wi-fi enabled sensors for internet of things: a practical approach. *IEEE Communications Magazine, 50*(6), 134–143.
45. Zahoor, A., Kaur, S. (2021) A review of algorithms for secure data transmission in iot devices. In: Data driven approach towards disruptive technologies: proceedings of MIDAS 2020. Springer Singapore, pp. 83–95

Catarina Silva is a Ph.D. Candidate at the University of Aveiro and she is a researcher at Instituto de Telecomunicações in Aveiro. Her main research area focuses on Digital Privacy and Privacy Engineering approaches. She works on privacy-enhancing technologies development and proposes techniques to evaluate privacy in dynamic scenarios. Her work was already applied in several international and national projects in the context of smart environments, eHealth applications, and network scenarios.

Vitor Cunha received his Ph.D. in Computer Engineering from the Univ. of Aveiro in 2022 and is a researcher at Instituto de Telecomunicações. He is currently working on dynamic security mechanisms for softwarized and virtualized networks. Interests include network security, SDN, NFV, and pervasive computing.

João P. Barraca is an Assistant Professor at the University of Aveiro (UA) and Vice-Director of the Masters in Cybersecurity program. He is also a researcher at the Institute of Telecommunications and a member of the Scientific Committee of the DETI Doctoral Program, as well as the Executive Board of UNAVE. Dr. Barraca has published over 100 articles in peer-reviewed journals and conferences on networking, software engineering, and cybersecurity.

Rui L. Aguiar is currently a Full Professor at Universidade de Aveiro. He was the founder of the ATNOG research group, an advanced telecommunication research group at the Universidade de Aveiro and is currently co-coordinating a research line in Instituto de Telecomunicações, on the area of Networks and Services. He has been an advisory for the portuguese Secretaria de Estado das Comunicações and member of the task force for 5G cybersecurity. He is a Chartered Engineer, a Senior Member of IEEE, and a member of ACM. He has served as the Portugal Chapter Chair of IEEE Communications Society and has been serving as Steering Board Chair of Networld Europe, the European ETP representing the telecommunications community, engaged in the discussions of the future European R&D workprogrames for telecommunications. As further community engagement, he has served as Technical and General (co) Chair of several conferences (ICNS, ICT, ISCC, Mobiarch, Monami, NTMS, etc). He is a regular keynote speaker in the future of mobile communications and digital society, with dozens of talks across the whole world. He is an associated editor of Wiley's Emerging Telecommunication Technologies and Springer's Wireless Networks.