Wireless Networks
https://doi.org/10.1007/s11276-022-02999-7

ORIGINAL PAPER

Analysis on functionalities and security features of Internet of Things related protocols

Alessandra Rizzardi[1] · Sabrina Sicari[1] · Alberto Coen-Porisini[1]

Accepted: 10 May 2022

Abstract
The Internet of Things (IoT) paradigm is characterized by the adoption of different protocols and standards to enable communications among heterogeneous and, often, resource-constrained devices. The risk of violation is high due to the wireless nature of the communication protocols usually involved in the IoT environments (e.g., e-health, smart agriculture, industry 4.0, military scenarios). For such a reason, proper security countermeasures must be undertaken, in order to prevent and react to malicious attacks, which could hinder the data reliability. In particular, the following requirements should be addressed: authentication, confidentiality, integrity, and authorization. This paper aims at investigating such security features, which are often combined with native functionalities, in the most known IoT-related protocols: MQTT, CoAP, LoRaWAN, AMQP, RFID, ZigBee, and Sigfox. The advantages and weaknesses of each one will be revealed, in order to point out open issues and best practices in the design of efficient and robust IoT network infrastructure.

Keywords Internet of things · Communication protocols · Security · Authentication · Confidentiality · Integrity · Authorization

1 Introduction

In recent years, technological progress has undergone a significant increase in the diffusion of the Internet of Things (IoT) devices [1]. This was mainly caused by the decrease in hardware costs. In 2021, the total turnover was estimated to reach 124 billion dollars and the number of IoT devices will reach a threshold of 35 billion. An IoT device is an electronic component with limited computational and energy capabilities, able to communicate via radio frequencies within a network system. Its main purpose is acquiring data of interest in a certain scenario. An example is a sensor that carries out measurements regarding the environment surrounding it and transmits the result throughout a distributed network infrastructure. In order to regulate communications within an IoT system, different protocols are used, offering different features, such as the transmission's distance, the number of sent packets, the quality of the offered service, the packets' transmission speed, the energy impact on devices and, finally, the reliability guaranteed by the protocol. Such a requirement becomes more severe in certain scenarios, such as e-health or finance, where errors or violations could cause serious harm [2, 3].

The complexity of managing security within IoT networks is not limited to its implementation. Still, it extends to the need to find the right balance between the level of protection guaranteed and the performance achieved. There are currently several methods to ensure one or more security requirements, but many of them are not applicable in all IoT scenarios. For example, not all IoT devices are able to perform some types of cryptographic-mathematical computation, or they are not able to complete them in acceptable times. Furthermore, devices with limited energy capabilities in IoT systems are often located in critical or inaccessible positions, making it difficult or impossible to replace their batteries. However, the goal of security within

✉ Sabrina Sicari
sabrina.sicari@uninsubria.it

Alessandra Rizzardi
alessandra.rizzardi@uninsubria.it

Alberto Coen-Porisini
alberto.coenporisini@uninsubria.it

[1] Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, via O. Rossi 9, 21100 Varese, Italy

Published online: 04 June 2022

IoT systems is not only to avoid the violation of confidential information or prevent access to malicious entities: an intruder may be simply interested in taking control over the device for very different purposes. Hence, the importance of ensuring security within IoT systems, from the physical to the application level, becomes evident.

The purpose of this paper is to analyze the state of art related to the functionalities and security features of some of the main protocols, which are usually adopted in the IoT field, paying particular attention to authentication, confidentiality, integrity, and authorization requirements. The investigated protocols are the following: MQTT, CoAP, LoRaWAN, AMQP, RFID, ZigBee, and Sigfox.

The rest of the paper is organized as follows. Section 2 presents the existing surveys and tutorials on IoT-related protocols, pointing out the current state of the art. Still, it extends this field, both in general and with a focus on security aspects and on the role of blockchain. Section 3 details the features and the security-related functionalities of the analyzed protocols. Section 4 provides a discussion about the outcomes of the conducted analysis, revealing open challenges. Section 5 ends the paper and draws some hints for future research directions.

2 Motivations and related works

The proposed work tries to fill a gap in the literature by clarifying the role played by different IoT-related protocols, from a practical perspective (i.e., starting from the actual functionalities offered by each one) focusing on security requirements. Other papers survey IoT architectures and provide taxonomies on communication protocols, without primarily focusing on their intrinsic specifications.

For example, the authors of [4] categorize and classify IoT architectures and devise a taxonomy based on important parameters such as: applications, enabling technologies, business objectives, architectural requirements, network topologies, and IoT platform architecture types. A study on IoT protocols and communication is also provided in [5], where existing approaches are compared with respect to mobility functionalities. A review on IoT communication protocols, for enabling the connection of smart devices, which run on Internet Protocol Version 6 (IPv6), to Low power Wireless Personal Area Networks (6LoWPAN), is presented in [6]. In particular, short-range standard network protocols, such as ZigBee, Bluetooth Low Energy (BLE), Z-Wave, and Near Field Communication (NFC), are considered, as well as SigFox and Cellular, which, instead, are LPWAN standard protocols. Furthermore, the paper presented in [7] focuses on IoT application layer protocols, without considering the security-related requirements.

Concerning security, the work in [8] outlines the related issues at the perceptual, network, support, and application layers, which are widely recognized to be part of typical IoT network infrastructures. Instead, the works in [9], and [10] present a taxonomy of IoT security protocols, which different authors have proposed in the literature. Hence, the focus is not on well-known native IoT-related protocols, but on custom solutions, which are integrated with some security mechanisms. Moreover, in [9] an empirical analysis is also conducted on the estimated performance of the envisioned approaches; while the authors in [10] investigate some key distribution schemes. The survey in [11] analyzes existing protocols and mechanisms to secure communications in the IoT, by focusing on the CoAP protocol, thus without considering other ones, and on AES as a security mechanism.

In this paper, attention is paid to authentication, confidentiality, integrity, and authorization as secure-aware methods natively owned by specific IoT communication protocols, with the scope of identifying the current state of the art about the available solutions and shed light on what still lacks to obtain a robust and reliable system.

2.1 The role of blockchain

As it will emerge from the discussion, blockchain is playing an increasing role in guaranteeing the security of IoT protocols in data transmission at various levels [12]. IoT environments can be powered by blockchain since it provides a much more robust level of encryption that makes it virtually impossible to overwrite existing data records (i.e., blocks). The blockchain is essentially a distributed ledger shared among the network nodes, which produces a set of transactions to be approved and stored into the ledger itself. If all the nodes belonging/joining to a certain IoT network participate in maintaining the blockchain, the blockchain is public, also known as an open or permissionless blockchain. Because of their open nature, these blockchains must be secured with cryptography and a consensus system (e.g., Proof-of-Work) in charge of establishing the validity of each transaction. Otherwise, if only certain nodes are enabled to participate in maintaining the blockchain, we have a private or permissioned blockchain. In this case, each node must be approved before joining and, after approval, it is considered trusted. An example of a public blockchain is Ethereum[1], while a permissioned blockchain is Hyperledger Fabric[2]

Further insight into the blockchain functionalities is represented by smart contracts [13]. A smart contract is a computer code that can be built into the blockchain to

[1] https://ethereum.org/en/

[2] https://www.hyperledger.org/use/fabric

facilitate, verify, or negotiate a contract agreement. Smart contracts are typically used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without an intermediary's involvement or time loss.

Summarizing, data are kept by nodes into immutable blocks, which are linked to the previously filled block, forming a chain of data (i.e., the blockchain). Hence, blockchain (either public or private) can be adopted to protect the information managed inside the IoT network or also to guarantee the authorization of devices participating in the IoT network. Blockchain is sometimes coupled with *TLS (Transport Layer Security)* or *DTLS (Datagram Transport Layer Security)*. In such situations, TLS/DTLS protocol is used to exchange ciphering keys among the network nodes, while blockchain blocks are signed with such keys, previously transmitted through the secure TLS/DTLS channel. Hence, an authentication mechanism is built on the top of TLS/DTLS protocols, by means of blockchain, to cope with resource-constrained IoT devices, as presented in [14]. In Sect. 3, we will describe various works adopting blockchain for different purposes to ensure access control to the IoT network and data protection.

3 Internet of Things communications protocols

In this section, IoT communication protocols (i.e., MQTT, CoAP, LoRaWAN, AMQP, RFID, ZigBee, and Sigfox) are detailed, along with the related security functionalities. More in detail, the following section will deeply discuss, respectively, about: (i) authentication and access control; (ii) confidentiality; (iii) integrity; (iv) authorization. Each protocol must not always be intended as an alternative to the others, since they could be adopted for different scopes in various application scenarios. Moreover, they belong to different levels of IoT network architecture, as shown in Fig. 1. More in detail, MQTT, CoAP, and AMQP protocols act, as the well-known HTTP protocol, at the application layer, which is responsible for directly interacting with the users' applications. Such protocols mainly run on top of TCP or UPD protocols, above which TLS or DTLS layers

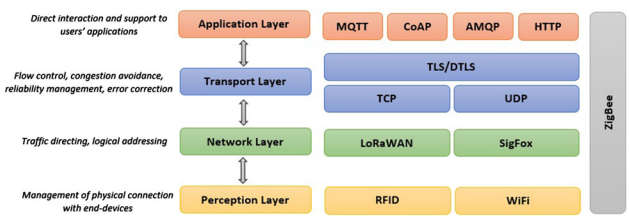

Fig. 1 IoT layers architecture

can be added to introduce security features. The transport layer aims to control the data flow, avoid congestion states, manage reliability, and correct transmission errors. Below, the network layer directs the traffic to perform logical addressing operations to handle data transmissions. LoRaWAN and Sigfox mainly belong to this layer, due to their functionalities inherent to network management. Finally, the perception layer has to handle connections with IoT end devices. RFID is mainly linked to this layer due to its sensing capabilities, while ZigBee owns a full protocol stack, which spans over the four layers that make up the IoT stack.

3.1 MQTT

MQTT[3] (Message Queue Telemetry Transport) is an application-level protocol, which is based on the TCP protocol and was introduced for the first time in 1999. Subsequently, a different version called *MQTT-SN*[4] *(MQTT for Sensor Networks)* was devised, as an alternative which is specifically designed for sensor networks. It does not use the TCP protocol for sending information. In this way, MQTT-SN is lighter than the standard MQTT, since it does not perform the opening and closing operations on the connection, which are usually provided by the TCP protocol. However, MQTT-SN needs to introduce a gateway able to translate MQTT-SN packets into MQTT ones.

In general, MQTT is targeted to situations where a low computational-energy impact is required on the individual devices, and where the usable bandwidth for communications is limited. It is based on a publish &subscribe communication system, which is able to connect multiple client nodes (i.e., publishers and subscribers) through a server node, named broker (or dispatcher). The publisher is responsible for publishing information regarding a specific topic to the broker node, while the subscriber subscribes to a topic made available by the broker, receiving new publications during the time. The broker gets the information from the publishers and sends a notification with updated information to the various subscribers of a specific topic (see Fig. 2).

Note that, in the case of a large number of clients interacting with the broker, it is possible to take advantage of a distributed brokers' network. Such a mode allows for balancing the workload on several broker nodes, avoiding the possibility of bottleneck formation, as described in [15] and [16]. More in detail, a network of brokers, combined with the distributed devices involved within the IoT platform, forms a fog layer, capable of decentralizing as much

[3] https://mqtt.org/

[4] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt-sn

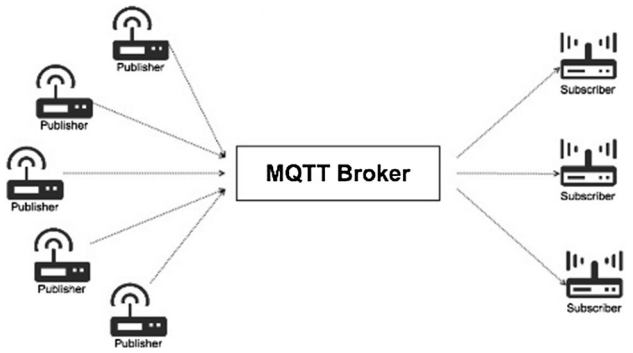

Fig. 2 MQTT scheme

as possible the tasks to be performed to manage the data sent by IoT devices.

It is worth remarking that the roles of publisher and subscriber can be covered by any device (from a simple micro-controller to a complex server). Moreover, a device can be both a subscriber of one topic and a publisher of another. Given the asynchronous nature of the publish &subscribe communication system, a subscriber node and a publisher never communicate directly, but are totally independent. Regarding the broker, different implementations are available: *Mosquitto, GridServer, Mqttools, Moquette* and *KMQTT*. Some of them provide both client and broker functionality, such as *Mqttools* and *Mosquitto*, while not all of them support the latest version of MQTT 5.0, as in the case of *Moquette*.

Furthermore, one of the key aspects of the MQTT protocol is Quality of Service (QoS), which contributes to regulating the communication's reliability levels among the involved entities. Three QoS levels are available:

- *QoS 0*: represents the lowest quality level and is characterized by the absence of checks and confirmation responses from the receivers. Hence, the message will be sent only once, without being sure of its actual delivery.
- *QoS 1*: ensures that the message is delivered to the recipient. To achieve such a goal, the sender stores a copy of the sent message, waiting to receive a confirmation message from the receiver. To ensure the re-sending of the news, a timer is started and, when it expires, in case of lack of confirmation by the receiver, the sender will take care of sending the initial message again. Compared to QoS 0, this one offers higher reliability, besides an increase in the resources exploited by each device.
- *QoS 2*: guarantees the highest standards of service quality, but also has the most increased cost in terms of resources. This level ensures that the receiver obtains exactly one copy of the message through a double confirmation mechanism. Four steps are used to carry out this flow: (i) the first step consists in sending the message; (ii) in the second step, the receiver will send a *PUBREC* packet to notify the receipt of the message; if the sender does not receive this packet, it will re-send the message sent at step 1, with the value of the *DUP* flag set to 1; (iii) the sender replies to the receiver with a *PUBREL* message; if the receiver does not receive this message, it will send a new *PUBREC*; (iv) once the *PUBREL* packet has been obtained from the receiver, a *PUBCOMP* message will be sent to the sender; such a step will mark the end of the flow.

The choice of the QoS level depends on several factors, such as (i) network reliability, (ii) importance of message's content, (iii) frequency of sent messages. Furthermore, it is possible to find different QoS values within the same network since the QoS parameter can be set both between the publisher and the broker, and between the broker and the subscriber; therefore, a broker can receive information from a publisher and send the same information to a subscriber by exploiting two different QoS configurations.

3.1.1 Authentication

Concerning authentication, MQTT provides a simple native implementation. It allows managing authentication based on the pair [*username, password*], which are provided to the broker at the time of connection. Both fields are optional, but a *password* cannot be specified if the *username* has not been previously specified as well.

However, the transmitted information is not encrypted by default, thus enabling potential attackers to listen for communications by intercepting the *CONNECT* message, thus reading the credentials in clear [17]. In [18], the authentication scheme via One Time Password (OTP) is proposed and combined with the Ethereum blockchain as an external communication channel. The use of such a mechanism requires the addition of some steps during the authentication phase, which, fortunately, do not considerably increase the energy- computational requirements of the IoT devices. Furthermore, the TLS protocol is not used in the treated solution. As demonstrated in [18], the envisioned authentication system satisfies three properties: (i) *privacy*, which ensures that user's information privacy is preserved throughout the communication flow, exchanging only an Ethereum address with the broker; (ii) *not impersonation*, which represents the impossibility, by a malicious entity, to impersonate a legitimate one; (iii) *accountability*, which guarantees the possibility of assigning the responsibility for each action to whom actually carried it out. Hence, such an approach is better than the standard authentication mechanism provided by MQTT and lighter than the TLS-based system, as stated in [18].

However, further analysis should be carried out to measure the actual overhead introduced and, consequently, the gain, in terms of energy and time on a real system.

In [19], the possibility of guaranteeing the authentication requirement, in addition to confidentiality, is presented and analyzed through the adoption of cryptographic systems, based on elliptic curves and hash functions. More in detail, authentication is guaranteed through the use of a combination of: (i) a random number; (ii) the hash value of the credentials provided by the clients; (iii) the broker's secret key. Furthermore, besides ensuring confidentiality and authentication, such an approach solves potential attacks such as replay and man-in-the-middle. In conclusion, the reduced size of the transmitted packets, and the fewer handshakes performed compared to TLS, reduce the resource consumption.

In [20], the authors propose a solution which makes the use of: (i) cryptographic schemes such as AES-GCM and Rabin; (ii) Schnorr's algorithm for generating short signatures. Three security levels can be configured:

- *SL*1: such a level has the minimum computational cost, and it is able to guarantee non-repudiation, integrity and authentication.
- *SL*2: in addition to ensuring when provided by SL1, such a level improves privacy and adds confidentiality to the guaranteed requirements.
- *SL*3: such a level guarantees the highest security, but it also requires huge computational resources.

From a computational overhead point of view, the system proposed in [19] adds some operations, which are reported in Table 1. As shown in Table 1, in level *SL*3, for a communication that involves all three actors, the following steps are required: (i) 12 scalar multiplications (*SM*); (ii) 2 modular multiplications (*M*); (iii) 4 modular additions (*A*); (iv) 4 hash functions (*H*); (v) 6 encryption operations using AES. The overall management must be separated between publisher and subscriber concerning the authentication requirement. More in detail, authentication is guaranteed for the publisher in all three levels, by sending the signature and checking the credentials on the broker using Schnorr's digital signature algorithm and a RAD (RAbin Decryption); instead, the subscriber must apply its digital signature only in the third security level *SL*3.

3.1.2 Confidentiality

In the MQTT protocol, confidentiality is one of the most interesting points for researchers, since no data encryption mechanism is natively provided. For such a reason, one of the main, and simplest, passive attacks, applicable to IoT systems that exploit the MQTT protocol, is packet sniffing. A malicious entity could listen to the communication and read the data transmitted between the clients and the broker, even if an authentication system is adopted [17]. Since MQTT runs through the TCP protocol, it is possible to mitigate such a security issue by using the TLS protocol, or DTLS, in case of adoption of the version based on the UDP protocol [17]. Such an approach is feasible as long as the involved IoT devices are not too limited in terms of resources. Note that, by default, TCP connections do not use encrypted communication. Still, to encrypt the whole MQTT communication, many MQTT brokers (such as *HiveMQ*[5]) allow the use of TLS instead of plain TCP.

In [20] and [21], confidentiality can be applied in two ways:

- $client - to - broker$: such a mode implies that the broker must be able to decrypt incoming messages and, if necessary, re-encrypt them upon a request by a subscriber; Fig. 3 shows an example of a flow that implements $client - to - broker$ encryption;
- $client - to - client$: in this case, the broker always manages encrypted information and only forwards them to the registered subscribers. Such a mode is recommended when the broker is not trusted. Figure 4 shows an example of a flow which implements $client - to - client$ encryption, which a sort of end-to-end approach.

As pointed out in [21], the second mode means that less computational effort is required by the broker, relieving it on the encryption and decryption tasks. In particular, in systems including a huge number of publishers and subscribers, expecting intense interactions with the broker, such an operating mode saves a lot of the broker's computational resources. Also, as described in Sect. 3.1.1, the approach envisioned in [20] ensures data confidentiality, since the messages exchanged by brokers, publishers, and subscribers are encrypted according to the chosen security level, using one of the two available systems: (i) *SL*2 uses the Rabin encryption algorithm; (ii) *SL*3 uses the AES-GCM encryption algorithm. Furthermore, in *SL*3 operating mode, the broker, to save computational resources, can decide to multicast with the subscribers of the same topic, encrypting the message once with a key shared by the subscriber group.

Coupling a modified version of the famous Diffie-Hellman key exchange protocol, named *AugPAKE* algorithm, with a lightweight block cipher encryption mechanism, named *PRESENT*, in the approach proposed in [22], the confidentiality of the published message is protected twice: first when it is transferred to the broker, by using the secure session generated by the *AugPAKE* algorithm (i.e., only the client who has the session key can decrypt the message) and, in the second time, in the side of the broker,

[5] HiveMQ MQTT broker, https://www.hivemq.com/

Table 1 Operations per level

Level	Publisher	Broker	Subscriber
SL1—(subscribe)	No operation	No operation	No operation
SL1—(publish)	1SM+1M+1A+1H	2SM+A1+1H	2SM+1A+1H
SL2—(subscribe)	No operation	1RAD	1M
SL2—(publish)	1SM+2M+1A+1H	1RAD+2SM+1A+1H+1AES	1AES
SL3—(subscribe)	No operation	3SM+1A+1H+1AES	3SM+1M+1A +1H+1AES
SL3—(publish)	3SM+1M+1A+1H+1AES	1SM+2AES	2SM+1A+1H+1AES

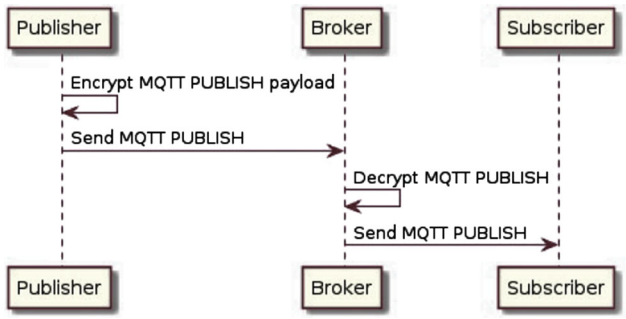

Fig. 3 client-to-broker encryption

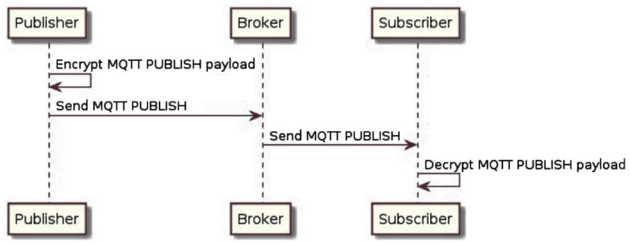

Fig. 4 client-to-client encryption

the message is not stored in plaintext due to the *PRESENT* encryption. Also, this solution provides mutual authentication between the broker and their clients (i.e., publishers and subscribers), the integrity, and non-repudiation of MQTT messages which are protected during transmissions.

3.1.3 Integrity

Integrity is guaranteed if a malicious entity is prevented from modifying the data transmitted among devices and broker. As explained in [20], MQTT, in its standard version, does not apply mechanisms to guarantee data integrity, but, despite this, if the device is not extremely limited in energy and computational terms, it is possible to calculate an integrity value of the payload to be concatenated with the payload itself. Since *MAC (Media Access Control)* algorithms and the solutions for checksum's calculation do not introduce excessive overhead, their integration to communications is recommended; however, it must be taken into consideration that the checksum is easily re-calculated if the payload is not encrypted before sending a message. Concerning digital signatures, given the structure based on private and public keys, the overhead introduced could be excessive for devices with limited energy and computational resources. Adopting such mechanisms has a double effect: (i) pointing out any communication errors that occurred during the information transmission; (ii) pointing out possible tampering within the message content, operated by third parties.

In [17], the authors define the most relevant and particular scenario caused by the absence of mechanisms that ensure integrity, as follows: links for downloading firmware updates are sent via MQTT. More in detail, a hacker could modify such a link, replacing it with a malicious one. Malicious firmware could disable the device, send information to external systems or send compromised information, or alter the functionality of the device itself, making it a bot. A system, claiming to guarantee integrity, would check the validity of the signature carried out by the device. If a malicious entity replaced the download link contained in the payload, the verification procedure would produce a negative result, and the message will be dropped.

In [20], through the use of the Schnorr algorithm, a scheme, able to process short digital signatures, produces a signature of the messages' content. In addition to the devices' authentication, such a signature guarantees data integrity. Even if a malicious entity modifies the message, the broker will discard it because, by re-calculating the signature, it would notice that the value of the signature does not correspond to the expected one. Subsequently, the broker will forward all the published messages, which have also been verified, to the interested subscribers. They will take care of re-validating the signature, once the message has been obtained. Such a flow ensures the end-to-end integrity of the data transmitted from the publisher to the subscriber.

Because TLS is unfeasible for resource-constrained devices, the authors of [23] propose to secure MQTT protocol in two stages. First, the payload is encrypted using a lightweight symmetric block cipher, to limit the overhead

with respect to asymmetric encryption. Post encryption, a lightweight hash function is used to ensure both message authentication and data integrity. Then the encrypted message is published along with its hashed output to the concerned subscribers.

3.1.4 Authorization

As reported in [20], the management of permissions associated with a device is in charge of the broker. More in detail, it is possible to divide the permissions associated with a device (or group of devices) into the following categories: (i) by topic allowed: all topics or only specific ones; (ii) by allowed operations: publish, subscribe, or both; (ii) by QoS. The first two categories, if properly configured, avoid uncontrolled access by external entities. In the first case, it is possible to restrict access to topics only to specific devices, and not to all devices; while, in the second case, it is guaranteed that only devices considered trusted can actually publish information, or receive it. Combining the two categories means that an untrusted device cannot exploit the same permissions as a trusted device. The authorization control can be applied with respect to the *username* or an *id*; both fields are present in the *CONNECT* message. As explained in [21] and [24], the broker can manage an Access Control List (ACL), which associates an allowed action to an entity or a group. For example, it is possible to specify whether a device, or a group of devices, can perform a specific action on a topic, according to the three categories of permissions listed above. Figure 5 shows an example of an ACL configured for a Mosquitto[6] broker.

The authors of [21] and [24] suggest to adopt a closed configured ACL. In this way, the system will exclude from communications all devices that do not respect the authorization rules defined in the ACL. In terms of overhead, it must be considered that, often, the broker is an unrestricted device. Hence, it is supposed that the overhead introduced by the controls necessary to implement the authorization mechanisms is negligible.

In [25], the use of *UMA (User Managed Access)*, a protocol based on *OAuth2*[7], is proposed as an alternative solution to authorization issues. The use of UMA involves the following changes: (i) alteration of the topics flows' management; topics will be considered as resources; (ii) introduction of an authorization server and extension of the broker's functions with those of UMA. Such an alternative allows the resources' owner to define his/her own permissions; if they are not specified, the resource would be inaccessible to everyone. A closed system is thus created which, by default, limits access to resources. Such a solution is not targeted to a specific platform or system. The results, obtained in [25], show that the use of this authorization mechanism, compared with the standard authorization mechanism provided by MQTT, slightly increases the consumption on devices. Still, this addition is considered negligible in terms of delay and computational required resources. More in detail, regarding the overhead in the test environment, a delay of 0.23 milliseconds has been introduced, which is considered an acceptable value with respect to the whole communication flow. To conclude, the overhead introduced by the permissions' check to each individual message has been analyzed, revealing an average of 15% of the time taken for this task.

The native MQTT protocol has been further extended with AUPS (AUthenticated Publish &Subscribe system) in [26]. In this work, a policy enforcement framework is coupled with a key management one to effectively manage publications and subscriptions through MQTT interactions. Data is encrypted with proper keys, which are associated with the topics within the IoT system.

3.2 CoAP

CoAP[8] is an application layer protocol widely adopted in IoT environments, thanks to its low computational impact on IoT devices. The structure of a CoAP package has been designed to be easily convertible into an HTTP message, so as to ease the integration with systems based on the HTTP protocol. Compared to HTTP, CoAP drastically reduces the packet size and the amount of required energy [27]. To demonstrate this, Table 2 shows the values obtained from the analysis conducted in [28], where a CoAP server is implemented by means of a *Tmote Sky* sensor running *Contiki OS* with *6LoWPAN/RPL* on the network layer and CoAP on the application layer, while the HTTP server is obtained with the same *Tmote Sky* platform and Contiki OS loaded with the HTTP server instead of the CoAP server. Note that the power consumption has been calculated by means of *Energest*, a tool able to estimate the power consumption of *Tmote Sky* motes.

There are different implementations of the protocol CoAP. Each of them mainly differs in the programming language used, the functionality of the CoAP protocol offered and the type of device (i.e., client and/or server). Some examples are the followings:

- *CoAPthon*: implemented in Python and able to operate as a client, server, forward proxy or reverse proxy. This implementation integrates the CoAP functionalities

[6] https://mosquitto.org/man/mosquitto-conf-5.html

[7] https://oauth.net/2/

[8] https://coap.technology/

```
# This affects access control for clients with no
username.
topic read $SYS/#

# This only affects clients with username
"roger".
user roger
topic foo/bar

# This affects all clients.
pattern write $SYS/broker/connection/%c/state
```

Fig. 5 Example of ACL

Table 2 HTTP and CoAP overhead

Parameter	HTTP	CoAP
Bytes per transmission	1451	154
Energy consumption (mW)	1333	151
Lifetime (days)	0.744	84

named *Observe, Multicast server discovery, CoRE Link Format parsing, Blockwise Transfers*

- *Californium*: implemented in Java and able to act as client or server. Also, it implements CoAP features: *Observe, Blockwise Transfers, DTLS*
- *FreeCoAP*: implemented in C and able to operate as a client, Server, or HTTP/CoAP Proxy. In addition, it implements the CoAP features: *CoRE, DTLS, Blockwise Transfers*

CoAP only provides two types of messages, namely request and response. Furthermore, it is based on a client/server system, thus operating asynchronously. CoAP runs over the UDP protocol, hence decreasing the computational-energy impact on devices; however, unlike the TCP protocol, UDP does not provide the same QoS standards. As a consequence, CoAP implements a simple mechanism to ensure reliability, which consists of the following steps: (i) the sender transmits a packet marked as "to be confirmed"; (ii) if a certain timer expires and the confirmation message has not yet been received, the sender will transmit the message again, as described in [11]. Moreover, CoAP nodes may cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests[9] It is worth remarking that, unlike HTTP, the cacheability of CoAP responses does not depend on the request method, but it depends on the response code.

Concerning security, since CoAP is based on UDP, it is possible to take advantage of the DTLS protocol. The use of DTLS ensures confidentiality, integrity, authentication, and non-repudiation [11]. However, DTLS introduces some further steps in the communication process (i.e., six messages in the initial handshake phase) and adds some information in the messages (i.e., 13 bytes). Combining such two factors increases the computational requirements of devices, which may not be satisfied by constrained ones. Currently, some studies aim at reducing the size of the DTLS header and the number of steps required to perform the handshake, without affecting the security offered by the protocol, as described in [27]. Moreover, CoAP introduces four security modes that can be used within the network, which mainly define how authentication and key negotiation must be performed:

- *NoSec*: basic (and efficient) mode that does not apply any security mechanism to messages.
- *PreSharedKey*: mode that uses one or more security keys defined in the configuration phase of the single device. It is useful when devices are not able to handle public-key encryption. A key can be uniquely shared with a device or a group of devices.
- *RawPublicKey*: a mode similar to the previous one, but based on a public key encryption mechanism; the devices are configured with an asymmetric key pair, an identity calculated from the public key and with a set of [*dentity pairs, public key*], which specifies the legitimate devices for communication.
- *Certificates*: a mode that works through X.509 certificates. In this mode, the devices are able to manage certificate chains and communicate with a trusted entity, which is part of a pre-configured list stored in the devices themselves, and used for certificate validation.

It is worth remarking that CoAP is also heavily used with *NarrowBand-IoT (NB-IoT)* [29] and *Long Term Evolution - enhanced Machine type communications (LTE-M/eMTC)* [30] radios, which are reliant on licensed spectrum. As mobile IoT networks use dedicated spectrum bands, interference from other radio technologies is kept to a minimum, thus representing a clear advantage of such technologies. In terms of security, mobile operators employ *Subscriber Identity Modules (SIMs)*, which contain highly secure integrated circuits, to authenticate the devices accessing networks and services. Data is encrypted while traveling across the network infrastructure. Moreover, communications from the end-IoT devices and the network are mediated by an *Evolved NodeB (eNB)*, which executes control functionality, and, subsequently, a *Mobility Management Entity (MME)*, which is responsible for security key management, checking the authorization of user devices and enforcing roaming restrictions.

Instead, as presented in [31], a *Lightweight Machine to Machine (LwM2M)* protocol can be run at the top of CoAP,

[9] IETF CoAP RFC, https://datatracker.ietf.org/doc/html/draft-ietf-ace-wg-coap-eap

thus defining a client-server architecture where an LwM2M client (i.e., the CoAP server) registers itself to the LwM2M server (i.e., the CoAP client) as an endpoint. Hence, it represents a higher-level application technology with respect to CoAP. Concerning security, the LwM2M protocol provides APIs for bootstrapping, registration, data access, and eventing. Coupled with *Object Security for Constrained RESTful Environments (OSCORE)* [32], it also helps to overcome one of the limitations of CoAP combined with TLS/DTLS, which is the end-to-end encryption. More in detail, TLS and DTLS encrypt from the IoT end-device until the next gateway, at which point the data is unencrypted, re-encrypted, and forwarded on. In this hop-by-hop scenario, information is only as secure as the network. Otherwise, OSCORE encrypts only the payload, and only a pre-authorized end-point may unencrypt the data. Hence, the network could be compromised, but data would still be secure. OSCORE is also more flexible than TLS/DTLS, since, unlike TLS and DTLS, it works at the applicative layer and with different transport protocols, so in addition to CoAP, a developer can use OSCORE with *Non-IP Data Delivery (NIDD)*, *Short Message Service (SMS)*, TCP, and others. One of the current disadvantages of OSCORE is that it does not have a native key exchange mechanism.

3.2.1 Authentication

CoAP, as just introduced, natively provides four operating modes, three of which (i.e., *PreSharedKey*, *RawPublicKey*, *Certificates*) provided security functionalities. In [11], DTLS is indicated as a method for sharing keys, but it is also pointed out that this protocol could excessively degrade the performance of the IoT devices (e.g., due to packet fragmentation). Moreover, DTLS does not support group key management, which is an important requirement in IoT environments. A solution to the overhead introduced by DTLS is provided in [33], which shows how the use of DTLS header compression mechanisms can reduce the information transmitted and, consequently, the energy consumption of devices. [33] also presents an alternative to the DTLS protocol. Such a solution adopts a Kerberos server connected with the CoAP server. During the access phase, unique information is provided to the Kerberos server, which compares the values stored in the CoAP server to assign an identification token to the device; note that it is essential to perform the registration phase the first time the device connects with the Kerberos server, in order to provide the unique information, produced in such a phase, in the subsequent authentication phases. The token has a limited time validity and can only be used to request access to a specific service: when a service is requested to the CoAP server, the token must be presented. Based on the received token, the server will decide whether to decline the request or provide the service access token, which will be different from the authentication token. In this way, device authentication is guaranteed.

A Kerberos server, along with the *RADIUS* (Remote Authentication Dial-In User Service) protocol, is also adopted in [34]; *RADIUS* is used to guarantee authentication and authorization. Furthermore, confidentiality is guaranteed by the DTLS and the IPSec protocol. A CoAP-NAS server manages the access permissions, which can be extended to the membership group. The authentication phase has been structured as a REST service that can work in three ways: (i) the first consists in sending the information in clear; such a mode is recommended only for testing purposes; (ii) the second consists in sending the hash value of the transmitted information; (iii) the last mode allows to send a *RADIUS* packet within the payload of the CoAP packet. Another approach is offered in [35], where the presented system ensures the authentication requirement through the use of hash values calculated from the *clientId* and a random value, by means of an intermediary authentication server. A malicious entity would not be able to generate the same hash value, since it does not know the *clientId* and the random value.

The authors of [36] show how the use of $Auth - Lite$ as a means of authentication can reduce consumption compared to DTLS. This solution requires that client and server share a pre-shared secret; moreover, it is necessary to modify the CoAP header with two new fields: (i) *AUTH*, which contains a Boolean value to indicate if the packet is used in the authentication phase; (ii) $AUTH\_MSG_TYPE$, which serves to identify the various authentication's phases. The mechanism introduced in [36] uses a nonce value and the AES encryption algorithm. This allows the system to face potential attacks such as man-in-the-middle, chosen-plaintext attacks, and replay. To demonstrate that $Auth - Lite$ is lighter than DTLS, a simulation campaign has been conducted, also taking into account the occurrence of packet loss. The number of bytes transmitted throughout the stream is reduced during the authentication phase of $Auth - Lite$ compared to the DTLS protocol. Hence, $Auth - Lite$ can be adopted as an authentication mechanism for DTLS, in order to obtain a lower energy impact during the handshake phase, combining the advantages of both protocols.

In [37], the possibility of delegating the handshake phase of the DTLS protocol to the *Secure Service Manager (SSM)* component, is proposed. The only restriction imposed by such a system is that the SSM component and the Constrained CoAP Sensor (CCS) devices must reside in a low-power and lossy network, which is an inefficient network that has a non-negligible packet loss rate [38]. Such an approach solves different vulnerabilities:

- *SSM Spoofing Attack*: the possibility that a malicious entity impersonates the SSM component is solved thanks to the bootstrapping phase. In this initial phase, CCS and SSM exchange a pre-shared key; since no other component or malicious entity is aware of such a key, and no one will be able to decrypt the messages sent by the CCS.
- *Semi End-to-end Security*: also, in this case, this property is guaranteed thanks to the encryption and decryption applied by the end devices. The SSM node does not store any session information and limits itself to forwarding the information received from the CCS.
- *Denial of Service*: since the DTLS handshake requires six messages to complete the flow, sending multiple requests simultaneously would reduce the life of the CCS component and, although legitimate, could be recognized as a DoS attack. To solve this issue the task is delegated to the *delegator*.
- *Single Point of Failure*: the SSM could be compromised. As a result, all connected devices would not be able to establish a secure DTLS session. For such a reason, defining a trusted manager can solve the issue, since the trusted manager could elect a new unrestricted device as SSM.
- *Fragmentation Attacks*: very limited devices in terms of memory are often exploited within LLN networks. For such a reason, packet fragmentation could be a severe issue (i.e., DTLS could send 27 different fragments). Such a problem is solved by using the *delegator*; since the devices belonging to the LLN do not directly perform the handshake, fragmentation does not occur.

To conclude, the system proposed in [37] relieves the IoT devices from performing the most demanding phases of the DTLS protocol, leaving them only the message encryption task.

3.2.2 Confidentiality

As described in Sect. 3.2, DTLS, along with authentication, ensures confidentiality, but introduces a non-negligible overhead. To cope with such an issue, the authors of [27] propose the use of elliptic cryptography (ECC) and the algebra of elliptic curves, demonstrating the effective gain of their $CoAP-ECC$ solution compared to the *RSA*-based version. CoAP-ECC has the advantage of using keys whose length is shorter than many other cryptographic systems. Table 3 shows how the key length varies according to the type of cryptographic system implemented. More in detail, in Table 3, the key length in the case of ECC and RSA is compared progressively, increasing the key length in the case of a symmetric algorithm. As demonstrated in [27], the computational complexity and overhead introduced by elliptical cryptography are lower than the RSA counterpart; the simulation revealed that the CoAP-ECC system consumes 47% less energy than the version based on RSA. This data implies an extension of the battery life of devices, while satisfying the confidentiality requirements.

Table 3 Key length (in bits) with respect to encryption mechanism

Symmetric algorithms	ECC	DH/DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

3.2.3 Integrity

DTLS protocol can also guarantee data integrity, although this approach would cause an excessive increase in overhead towards IoT devices. A solution is proposed in [39] through the use of hash functions. The hash value of the data is calculated and concatenated with the data itself within the payload. The flow includes: (i) a *sequenceId* to avoid replay attacks; (ii) a symmetric key shared between client and server; (iii) the LZW (Lempel-Ziv-Welch) lossless data compression algorithm, defined by Abraham Lempel, Jacob Ziv, and Terry Welch [40]; (iv) an algorithm for processing the hash value. Three algorithms have been evaluated, namely SHA224, SHA1, and SHA256. The obtained results show that SHA224 is the best candidate (64 rounds), in terms of efficiency and resource-saving, with respect to SHA1 and SHA512 (80 rounds).

A Secure Hybrid RSA (SHRSA) cipher is envisioned in [41] to provide end-to-end security on the top of DTLS. Seamlessly, end-to-end encryption is proposed in [42] at the application layer, leveraging the 6LoWPAN standard.

3.2.4 Authorization

CoAP does not natively provide a mechanism for managing permissions. Hence, the ad hoc method should be implemented. The work in [27] proposes a CoAP-ECC system providing a phase of key pairs generation [*public_key, private_key*], which allow to manage authorizations within the network; if a malicious entity tries to communicate with the CoAP system, its messages would be discarded since the system would not recognize the device among those registered. As a result, unauthorized devices are automatically excluded from the network.

The solution proposed in [34] satisfies the authorization requirement, since all requests made by the IoT devices must contain a ticket issued by the server during authentication. The validity check is not limited to just the ticket.

If the ticket is compliant, the server will also check the other information contained in the received message, such as the IP address, the *messageID*, the name of the requested service or the *URI (Uniform Resource Identifier)*. Thanks to such information, it is possible to assess which device requested the service and the associated permissions. At the end of the checks, the device itself communicates the result.

In [43], OAuth mechanism, together with CoAP, adopts a unique token provided by an authorization server (AS), thus allowing devices to access the protected information present in the resource server (RS), based on a set of permissions defined by the owner of the resource. Since OAuth leaves all interactions involving the RS undefined, the authors defined the following flow:

- The device must send an authorization request to the AS, where specifies some of its unique information
- The AS sends a response containing a key and the *PoP (Proof-of-Possession)* token
- The device registers itself to the RS by sending the PoP token; if the received token is valid, the RS will proceed to store the token along with the linked permission set, otherwise it is discarded
- The authentication between the device and the RS is performed; in this phase, the device sends a message containing the PoP token and the requested resource. Such a device, thanks to the key obtained from the AS, uses DTLS to secure the information exchange
- The RS authenticates the devices on the basis of the information received in the previous phase and, subsequently, checks that the requested resource is present in the set of authorized information, stored on the RS itself. If successful, the RS responds with the resource requested by the device.

The authors of [44] suggest adopting a hybrid system, consisting of an OSCAR server and a blockchain. The OSCAR server is used as a manager and distributor of keys to other servers, through the use of secure channels established by the DTLS protocol. On the other hand, the blockchain is used for guaranteeing the authorization of devices. Furthermore, it is assumed that all servers own certificates validated by trusted authorities. The system is made up of: (i) *Resource Servers*, which stores information; (ii) *Resource Owners*, which consist of the legitimate owners of the resource; (iii) *Clients*, which requires access to protected resources; (iv) *Proxy Servers*, which are in charge of storing encrypted resources when the resource servers are limited; (v) *Key Servers*, which are the servers used to generate the keys, used to encrypt and decrypt information; (vi) *Access token*, which is information necessary for the description of the access rights of the clients towards the resources; (vii) *Authorization Servers*, which are the servers used to generate access tokens. Once the system has been defined, the authorization flow is the following:

- The *Resource Owners* creates a smart contract and publishes it into the blockchain. Smart contracts are compiled programs capable of generating tokens for clients who want to access the private resources; the tokens also contain the permissions granted to the resource itself
- The client who wants to access the resource must first send a transaction to the blockchain by activating the contract; within the transaction, the client must enter the information necessary for validation in the data field, such as the public key. This type of request is broadcast, so all listening devices will receive the transaction and proceed with validation. Once the validation is completed, the smart contract is executed, which will generate a token for the client
- The client needs to own the decryption key to read the resource. To obtain this key, he/she must send a request to the keys server, which will initiate a challenge-response process in order to verify the identity of the requester
- The client proceeds to download the resource and decrypt it with the key just obtained. [44] points out that no authentication mechanism has been exploited, since only the authorized entity can actually access the resource.

As demonstrated in [44], such a solution solves the following vulnerabilities: (i) it is resistant to attacks concerning the security of the token; (ii) it ensures the privacy of the client, even if a different key must be used for each transaction; (iii) the communications are secure, since all the keys are exchanged using the DTLS protocol, and the transactions entered in the blockchain are signed; (iv) *Bootstrapping Key Server*, before an insecure node can communicate with the key server, it must contact a *bootnode* in order to synchronize with the blockchain. The *bootnode*, like other servers, must own a valid certificate. Simulations to assess the effectiveness of the envisioned approach are conducted by adopting Ethereum blockchain and Proof-of-Work mechanism, while the authors claim that also Casper blockchain [45] should be tested, since it assumes the less expensive (i.e., with respect to Proof-of-Work) Proof-of-Stake mechanism.

3.3 LoRaWAN

The LoRaWAN[10] protocol is widely used in IoT systems due to its low computational impact and its large transmission range, which can go up to 10 km away. Furthermore, it has transmission speeds ranging from 250bps to

[10] https://lora-alliance.org/

50kbps and the packet size can be a maximum of 255 Bytes. The structure of a LoRaWAN sensor node includes two layers: the LoRaWAN MAC layer and the LoRaWAN application layer. The first one is responsible for computing the *MIC (Message Integrity Code)* value considering the header and payload of the packet; while the second layer deals with the encryption of the information read by the sensor. The architecture of a LoRaWAN system provides, in addition to the end nodes, also the presence of a *Network Server*, an *Application Server* and a *Join Server*, as shown in Fig. 6. The servers are responsible for activating the nodes and storing the data. If a node is not activated, the server will ignore its messages. A node is considered active when it has a valid copy of the following values: (i) *Device Address (DevAddr)*, which consists of the device address; (ii) *Network Session Key (NwkSKey)*, or the key used to compute the integrity value of MAC packets; (iii) *Application Session Key (AppSKey)*, which indicates the key used to encrypt messages. The LoRaWAN protocol provides two mechanisms for activating available nodes:

- *Activation By Personalization (ABP)*, indicates that a node is considered active even if only the session keys are pre-configured in the node: *AppSKey* and *NwkSKey*.
- *Over-The-Air Activation (OTAA)*, describes the most complex activation scenario. A node, pre-configured with the *AppKey*, must necessarily communicate with the *Network Server* by sending a join message, where it specifies some of its unique parameters. Once the received values have been validated, the server proceeds to generate the keys and transmit the join accept message [46].

Currently, the LoRaWAN protocol version is 1.1. Some security vulnerabilities in version 1.0.3 have been fixed in this release. Additionally, key management has been changed to make it even more robust. LoRaWAN natively ensures the requirements of confidentiality, data integrity, and authentication. To deal with them, it uses a set of secret keys and $AES - CMAC$ algorithms to calculate the MIC (Message Integrity Code) value and $AES - 128 - CTR$ to encrypt the text. The management of secret keys has changed with the latest 1.1 version of the protocol. In the previous version, the protocol only used a pre-configured key, and two other keys derived from the first one; while, in the new version, the sensor node is initialized with two keys, which further generate a pool of keys to be used depending on the specific task carried on at a time (e.g., session establishment, join phase, forwarding messages, communication with the server). More details are available in [47]. The split into distinct keys has led to obtain a higher level of confidentiality. Despite this, the new version cannot be considered free from vulnerabilities, because we must consider the extreme case of physical

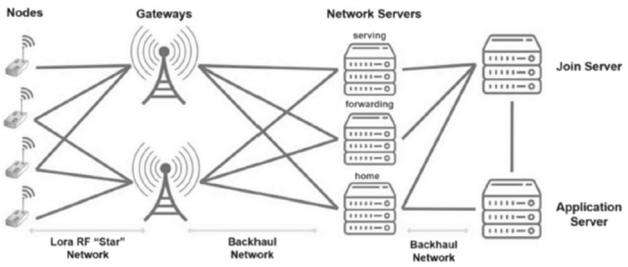

Fig. 6 LoRaWAN network scheme

tampering of the device. In this scenario, a malicious entity could obtain the static keys directly from the device. The first solution is to save the two static keys in a secure key manager on the device itself. This solution would slow down the malicious entity, even if it would not guarantee sufficient long-term security [48].

3.3.1 Authentication

The authentication requirement is guaranteed through the use of secret keys by the IoT devices towards the server. When a device computes the MIC value of the MAC packet, in addition to preventing the modification of the packet, it provides proof of authentication: only a device that owns the session key will be able to produce this result. The authors in [49] implement and test an improved version of the *Join* request, which leverages an Ethereum blockchain. The goal is to combine the peculiarities of the current *Join* mechanism with those offered by a blockchain, such as data immutability, resistance to data tampering, decentralized transaction validation. Compared to a LoRaWAN system, the following components are introduced: a blockchain network consisting of agent nodes and an *Agent Server*. In addition, the *Join* phase is modified as follows:

- Initialization of the blockchain network: in this first phase a smart contract is registered, and its address is replicated within the blockchain. The network server uses this address to gather the device's information from the blockchain.
- Initial registration and authentication: the second phase consists in registering a LoRaWAN end device in the blockchain network. This device sends a *Join* request to the gateway. Since the *block_id* is not specified in the request, the gateway will forward this message to the network server along with the smart contract. The network server will check the received message and, in the meantime, the blockchain network will mine the entered transaction. Once such a task is completed, the blockchain will notify the network server which will further send a join to accept message to the end node.

Hence, the device is considered to be registered as its information is contained in the blockchain.

- Two-factor authentication: this last step is necessary when a device has to rejoin. The device will send a join request encrypted with a proper key to the gateway which will only forward it to the network server. At this point, the server will extract the *block_id* from the message and will read the information previously stored in the blockchain. If this information matches the received one, the device and the request are considered authenticated and the server proceeds to send the join to accept the message. Hence, the device will be again connected to the network.

The security introduced by this mechanism mainly consists of the *block_id*, which will be unique for each LoRaWAN end device. As specified in [49], the main limitation of this solution is the first registration. Since the transaction must be mined, a huge amount of time is required to complete the whole registration. Finally, a system simulation is carried out to evaluate the latency introduced during the join phase and the number of join operations that the network server can perform in one second. The result showed an increase in the time needed to carry out the operation, while the number of operations that could be carried out shows a slight decrease along with an increase in the performed requests.

The authors of [50], to ensure authentication, propose the use of the 3*GPP* security mechanism based on the *AKA (Authentication and Key Agreement)* procedure, and a *USIM (Universal Subscriber Identity Module)*, installed on the LoRaWAN device. To this end, it is necessary to modify the end device by adding the USIM slot and to introduce a 3*GPPSecurityServer* that will communicate with the application server to: (i) modify the saved preconfigured key; (ii) read the messages sent by the end devices; (iii) send messages to the end devices. The $3GPP - AKA$ procedure consists of four different phases:

- LoRaWAN initialization: this phase involves the standard initialization offered by the LoRaWAN protocol; the device sends an *attach request* and receives an *Authentication request* message containing a 128-bit long random value as a response from the *Secure Server*.
- Wait for *AUTN*: in this phase, the end device sends a *Wait For Authentication Request* message, waiting to receive an *Authentication request* message from the server containing the *AUTN* parameter.
- Synchronization check: the device checks that its *USIM* is synchronized with the server thanks to a parameter included in the *AUTN* value. In case of success, the device proceeds to send an *Authentication response* message, further waiting for an ACK message; while in the event of a negative result, the device notifies the server that it will send a new *AUTN* value in order to repeat the synchronization check.
- Authentication closure: regardless of the positive or negative outcome of the procedure, the authentication ends. If the device is not authenticated, the procedure must be entirely repeated.

The discussion concerning the energy impact of the proposed solution still remains open: different messages are required to authenticate a device. Moreover, the use of an additional component, such as the *USIM*, increases energy consumption, albeit slightly. Finally, as pointed out in [50], it would be interesting to implement the 5G-based authentication mechanism; this would guarantee the device's resistance to impersonation and cloning attacks.

The authors of [51] define an improved key management system that, through the use of a *permissioned* blockchain, is able to guarantee the authentication requirement, in addition to solving the single point of failure vulnerability of the join server, which is used for the execution of the join procedures and storing of devices' keys. A permissioned blockchain, compared to a public blockchain, such as Ethereum, is faster in validating information and allows access only to authenticated nodes. In [51], Hyperledger Fabric is used as permissioned blockchain, consisting of: (i) peers, which are devices in charge of storing part of the blockchain data; (ii) *Orderer*, which is a device in charge of ordering transactions in blocks and performing access control operations on the blockchain register; (iii) *MSP (Membership Service Provider)*, which is a component that defines the identity validation mechanisms of the other components, using a Certificate Authority, to revoke or create certificates in the blockchain; (iv) clients, which are authenticated applications that send transaction proposals to the blockchain. The blockchain aims to manage the secret keys of the devices, entailing several advantages: (i) higher availability of replicated data on different peers; (ii) improved verifiability since all join requests are stored on the blockchain. Furthermore, the blockchain works through smart contracts, which implement four functions to support key management: (i) *RegisterDeviceKeys*, which is an operation necessary for registering a new end device, which allows storing a new pair of keys identified by the *DevEUI* value; (ii) *GetDeviceKeys*, which is an operation used to read the keys from the blockchain; (iii) *UpdateDeviceKeys*, which is an operation used to update a set of keys; (iv) *RevokeDeviceKeys*, which is a function that permanently drops the keys from the blockchain. Note that the devices' keys are encrypted before being stored in the blockchain, in order to improve the security level. In [51], a test environment has been set up to simulate a real operating scenario. The test result confirmed that, from an

authentication point of view, peers must have a valid certificate to operate on the blockchain. While, from a performance analysis point of view, the system presents an increase in the responses' latency from the server, which, however, decreases with the increase in the number of simultaneous requests, stabilizing at a value just above the average.

3.3.2 Confidentiality

To ensure confidentiality, LoRaWAN implements the AES data encryption mechanism with 128-bit keys operating in counter mode (AES-128-CTR) and the secret key *AppSKey* [52], as shown in Fig. 7. *AppKey* and *NwkKey* keys never change, representing a potential vulnerability. In [52], a further solution to the issue of static key management is proposed, following these techniques:

- The first to be proposed is DTLS. This protocol is generally used as a security guarantor for the application layer, as demonstrated in Sect. 3.2; but, in the solution proposed in [52], only the handshake phase of the protocol is used in order to agree on a new key between the node and *Network Server*. However, as just said in Sect. 3.2, the handshake phase of the DTLS protocol is too expensive for devices that have limited energy.
- The second scenario proposed makes use of the *IKEv2* (*Internet Key Exchange*) protocol, exploiting it as a service for the management of keys, or their generation and refresh. The overhead introduced by *IKEv2* depends on the chosen cryptographic system and, as reported in [52], the use of elliptical cryptography would be an excellent compromise between overhead and execution time. However, *IKEv2* requires four additional messages and uses *IPSec* to ensure the robustness of the messages, which, on the other hand, entails an increase in the size of the packets, thus causing their fragmentation. Combining such two factors dramatically increases the final overhead on sensor nodes, resulting in the impossibility of adoption for constrained devices.
- The latest proposal is the *EDHOC* protocol (*Ephemeral Diffie-Hellman Over COSE*), which is used to exchange symmetric keys between the sensor node and the *Network Server*. This protocol requires three messages to establish both keys and, as demonstrated in [52], although the *EDHOC* packet is first encapsulated in a CoAP packet and then in a LoRaWAN packet, the length of the messages does not exceed 255 Bytes, avoiding fragmentation, as shown in Fig. 8.

Hence, the LoRaWAN protocol natively provides an efficient data encryption mechanism, thanks to the use of the AES-128 protocol operating in counter mode. More in detail, when a packet is sent from an end device to the server, its payload is encrypted using the *AppSKey* [53]. Although the mechanism used is resistant and fast simultaneously, the use of the counter mode, combined with the absence of an efficient key refresh mechanism, can lead to vulnerabilities. As demonstrated in [53], $AES-128-CTR$ uses an input counter and, since the counter cannot increase indefinitely, it will reach a limit value, causing its restart from zero. The vulnerability is exposed at the moment of the reset task: since the *AppSKey* encryption key does not have a high refresh rate, two different messages could be encrypted with the same key. Figure 9 shows the properties of the XOR operator, pointing out that two encrypted messages C1 and C2, if encrypted with the same key, can easily lead back to the two clear messages, P1 and P2. This vulnerability highlights the need to exploit a key refresh system, such as the one proposed in [52].

The authors of [54] propose the use of *ECCDH* elliptical encryption, as an alternative to $AES-128-CTR$. This mechanism requires the server and the IoT device to share two pre-configured secret parameters. During the initialization phase, the device and server must exploit such two secret parameters to generate the encryption keys. In order to also guarantee data integrity, in [54], the elliptic curves are also used for the calculation of the *MIC*. After generating the necessary keys, the device must encrypt the packet and then sign it. On the other hand, the server must validate the signature and decrypt the messages if the signature matches. The results obtained from the simulation of the proposed elliptic cipher are compared with other solutions; the outcome reveals a drastic decrease in the encryption and decryption times of the proposed solution. Furthermore, as specified in [54], such an approach solves the potential $bit-flipping$ attack. The proposed system

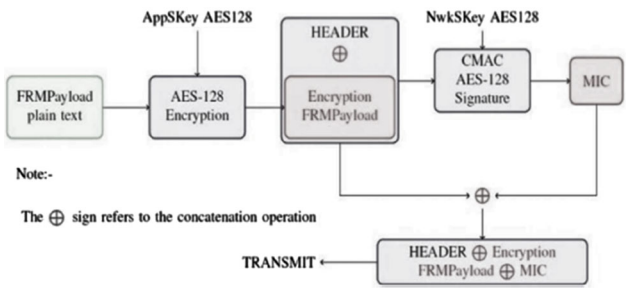

Fig. 7 AES encryption scheme in LoRaWAN

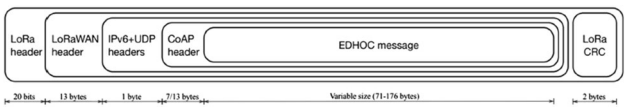

Fig. 8 LoRaWAN packet structure in the EDHOC protocol

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K)$$
$$= P_1 \oplus P_2 \oplus \underbrace{(K \oplus K)}_{\text{cancels out}}$$
$$= P_1 \oplus P_2.$$

Fig. 9 XOR property

does not improve confidentiality at the protocol level during data encryption, but improves key management. More in detail, by moving the keys from a central server to a distributed network, it is guaranteed that, if the server is hacked, the malicious entity would not have access to the keys. Furthermore, storing the encrypted version of the keys ensures that their violation does not generate an immediate vulnerability. This is due to the fact that the end devices would have all the time necessary to update their encryption keys, before the malicious entity can actually obtain their decrypted version.

3.3.3 Integrity

As regards integrity, the LoRaWAN protocol natively satisfies such a requirement, without the need to introduce any additional external mechanisms. As highlighted in [53], LoRaWAN, before transmitting a message, generates the MIC value, using the *NwkSKey* and the $AES - CMAC$ algorithm. The calculation of this value takes into account the entire header of the MAC packet and its payload; Fig. 10 shows the final structure of the package. However, in case the *NwkSKey* has not yet been generated, the *Join* phase must be carried out again. To solve the issue, the MIC value is computed by means of the *AppKey* [53]. When the server receives a message, it checks that the MIC matches the value transmitted in the message before decrypting it. As described in [54] through ECCDH, devices produce a MIC value capable of signing the message and guaranteeing the data integrity requirement. The goal of the MIC value is to avoid receiving tampered information from malicious entities.

Even not supported in the current LoRaWAN standard, the study conducted in [55] enables device-to-device (D2D) communication. Direct communication among devices can reduce battery consumption and can be protected by a secure link establishment scheme. D2D nodes securely share cryptographic keys guaranteeing data integrity, mutual authentication, and confidentiality.

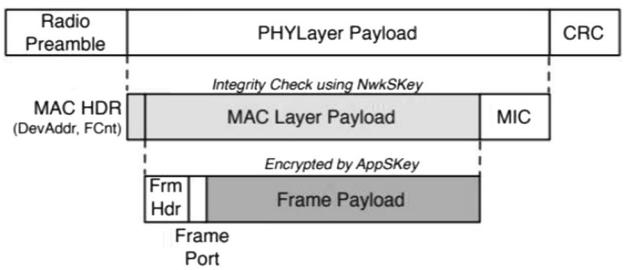

Fig. 10 LoRaWAN packet structure

3.3.4 Authorization

Concerning authorization, LoRaWAN guarantees this requirement by introducing two activation methods (i.e., OTAA and ABP), as described in Sect. 3.3. Thanks to such two mechanisms, a device is allowed to securely communicate with a network server. As described in [56], the OTAA mechanism guarantees the possibility for a device to register with different network servers, while the ABP mechanism will force a device to always communicate with the same network server. The authors of [57] point out that, in recent years, blockchains have also been used to ensure the authorization requirement; in fact, when a device publishes a smart contract, it also defines the set of permissions associated with it. Consequently, when an external device tries to access a resource, its authenticity is first checked and, then, its authorizations are checked against the requested resource. It is possible to consider extending the study carried out in [49], where the authors set up a blockchain-based system, but do not perform any authorization checks. For example, [51], a blockchain-based system is proposed. Authorization is put in the act thanks to the checks carried out by smart contracts against the certificates in order to verify the identities and permissions associated with them. Furthermore, the functions of smart contracts are only accessible from the *Join* server. To conclude, this mechanism, in addition to avoiding unwanted access by a malicious entity, is able to grant access only to legitimate devices.

3.4 AMQP

AMQP[11] (Advanced Message Queuing Protocol) is an open OASIS standard protocol, initially adopted in the banking-financial field; its use was subsequently extended to networks IoT. The AMQP protocol, like MQTT, takes advantage of the TCP protocol and is based on a publish &subscribe asynchronous mechanism. The system consists of: (i) a broker, which receives the information from the devices and distributes them according to specific rules; (ii) the devices, which can operate in producer mode (i.e., they generate a message and send it to the broker) or in consumer mode, by requesting information from a specific queue and processing it. The broker is further composed of two sub-components: (i) an *exchange*, which deals with sorting the information contained in the messages sent by

[11] https://www.amqp.org/

the producers in their respective queues; (ii) the *queues*, which are lists of values generated by producers and uniquely identified by a key, namely binding key; such a key is essential for the *exchange* component to be able to sort the information received into the correct *queue*. Note that a queue can have multiple binding keys and can share a key with other queues. Also, the lists can be volatile, so they are stored in the temporary memory of the server, or persistent, or in file format and stored on a hard disk. A broker node has four operating modes available to forward a received message:

- *Direct Exchange*: the messages received by the broker owns a routing key. If the routing key is compliant with one or more binding keys, the broker will replicate the message for each queue whose identification key matches.
- *Fan-out Exchange*: this operating mode is comparable to the multicast mechanism. The broker completely ignores the routing key and forwards the message to all existing queues.
- *Topic Exchange*: similar to the *Direct Exchange* operating mode, with the only difference that the routing key and the binding key do not have to match completely, but they just need to share wildcards. In this way, a message can be sent to multiple queues.
- *Headers Exchange*: the use of this feature allows to ignore the routing key and base the forwarding mechanisms on the information transmitted within the header of the received packet.

A key feature that has facilitated the development and rapid deployment of AMQP lies in the protocol's flexibility. In fact, AMQP is a cross-platform protocol and, therefore, it can be used in almost any context. Hence, two systems can communicate based on different architectures without making particular changes. For example, a Java-based system can easily communicate with a Python-based system via the AMQP protocol.

Since AMQP is an open standard, there are different types of brokers, which differ mainly in the programming language. The most used are *RabbitMQ*, *Apache Qpid* and *Azure Service Bus*. Figure 11 shows the structure of an AMQP network, where *E* represents the process by which an exchange decides which queues to place your message on. The AMQP protocol provides three QoS mechanisms, as MQTT: at most once, at least once, exactly once.

The authors of [58] state that the AMQP protocol is not suitable for use in IoT systems composed of devices that are particularly limited in terms of energy. To demonstrate this, in [58], different simulations have been carried out comparing different protocols; for each protocol, a test environment has been set up and the number of bytes sent, the time needed to obtain a response and the number of packets sent have been measured. The results showed that AMQP is the protocol that sent the most data over the network with a longer send time than the other protocols. It must also be considered that the AMQP protocol exploits the TLS protocol to guarantee security. In conclusion, as described in [59] and [58], the use of less expensive protocols is recommended in such scenarios.

3.4.1 Authentication

AMQP protocol does not natively meet the authentication requirement, but uses the TLS protocol and/or the *SASL (Simple Authentication and Security Layer)* framework to guarantee the security of transmissions. The first is used to encrypt communications and/or authenticate devices through the use of X.509 certificates, while the second is used to ensure device authentication [60]. The peculiarity of the SASL protocol is to support different authentication mechanisms, such as *OTP, DIGEST-MD5, CRAM-MD5, OAuth10A, OAuthBearer*. Thanks to the use of such mechanisms, supported by the SASL protocol, and the use of the TLS protocol, AMQP is considered one of the most robust and stable protocols acting within a network infrastructure. However, due to its computational requirements, it is not suitable to be adopted in constrained IoT applications.

Hence, one of the most important features of such a protocol is the possibility to choose the authentication mechanism; as just said, SASL provides different authentication mechanisms. Some of them are less expensive than their TLS counterpart. For example, an authentication mechanism based on unique tokens, assigned to individual devices requires sending less information. Furthermore, it is necessary to consider the possibility that X.509 certificates can be chained, increasing the number of bytes that a device is forced to send. In order to take advantage of the SASL framework, it is necessary to carry out the negotiation procedure in order to open a secure connection between sender and recipient. Figure 12 shows the communication scheme during the SASL negotiation.

3.4.2 Confidentiality

As hinted in Sect. 3.4, AMQP exploits the TLS protocol to guarantee data confidentiality, also if such an approach is demonstrated to be expensive for IoT devices. Note that TLS support can be integrated, for example, with certain messaging brokers, which are intermediaries for messaging, such as *RabbitMQ*[12]

As pointed out in [61], protocols that make use of centralized information management expose themselves to

[12] RabbitMQ, https://www.rabbitmq.com/

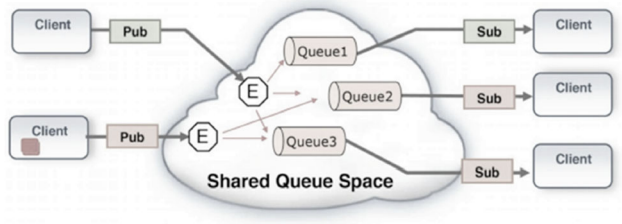

Fig. 11 AMQP network

```
TCP Client                TCP Server
==========================================
AMQP%d3.1.0.0  --------->
               <---------  AMQP%d3.1.0.0
                    :
                    :
          <SASL negotiation>
                    :
                    :
AMQP%d0.1.0.0  --------->                (over SASL secured connection)
               <---------  AMQP%d0.1.0.0
        open   --------->
               <---------  open
```

Fig. 12 AMQP communication scheme during SASL negotiation

the vulnerability of single-point-of-failure. If a malicious entity gains control of the broker node, it would be able to read all the information both stored and transmitted by the device itself. A possible alternative lies in the use of a private blockchain; as proposed in [61], this solution would provide higher information security, eliminating the single-point-of-failure. Since the blockchain is distributed over several devices, even if a malicious entity corrupts a device, the AMQP system would continue to run. Despite this, it is necessary to encrypt the data before they are stored, in order to guarantee privacy even if the storage is violated. The blockchain-based solution would ensure the requirement of confidentiality and data integrity. In citealfandi2020survey, the possibility of extending this system with machine learning techniques is also discussed, which would improve the predictive capabilities of the system against subsequent attacks and the ability to recognize any malicious entities; the final system would thus be equipped with an efficient intrusion detection system.

3.4.3 Integrity

AMQP does not natively ensure data integrity, but uses the TLS protocol. Since this protocol is not applicable in scenarios involving the presence of constrained devices, it is possible to consider the idea of implementing a signature mechanism (e.g., *MD5, SHA1, SHA256, HMAC, the Schnorr algorithm* [20]), capable of computing an integrity value and appending it to the AMQP package payload. This solution would guarantee the desired requirement at the application level. In order to tamper with the data within the package, a malicious entity should initially decrypt the message, then modify the data, and, finally, re-calculate the integrity value. The role of blockchain has been introduced In Sect. 2.1 and, also in the case of AMQP, can be integrated in order to guarantee the requirements of confidentiality and data integrity, as proposed in [61–63].

3.4.4 Authorization

The AMQP protocol does not provide an authorization framework. If a device is able to authenticate itself to the server, it is also able to publish information or register with the provided lists. Hence, an efficient authorization management system must be defined. In [61] and [62], the possibility of adopting a blockchain is analyzed. The blockchain can be configured, so that the transactions also contain the set of permissions for accessing the stored data; or it is possible to structure the blockchain to work through an *ABAC (Attribute-Based Access Control)* mechanism, thus providing a set of permissions based on attribute values. As highlighted in [25], since the AMQP and the MQTT protocol are based on a similar publish &subscribe system, it is possible to consider the implementation of the same solutions with little precautions. The AMQP broker could be configured to define an ACL inside it; this possibility would allow setting permissions to a single device or a group of devices. Currently, there are no studies concerning this aspect, but as highlighted in [25], further research works in this direction are just planned.

3.5 RFID

As described in [64], a RFID system is composed by:

- *Tag*: the element responsible for acquiring information and performing simple computations. This component can be passive (i.e., without power supply, used only for storing data) or active (i.e., powered and able of carrying out simple operations or starting a conversation autonomously, even with other tag devices). Furthermore, *EPCglobal* organization divides the tags into six classes based on the computational- energy capabilities of the device itself: (i) class 0 and class 1: they indicate passive components, programmed during the design phase or by the user; (ii) class 2: indicates the components with encryption and memory read/write capabilities; (iii) class 3: includes the set of devices equipped with a battery, computational logic, and higher coverage range, as well as broadband communication capabilities, with bandwidths higher than 1 MHZ; (iv) class 4: includes active tags able to perform peer-to-peer communications and to acquire data from the environment, acting as sensors; (v) class 5: includes the set of active tags that can be classified as a reader and capable of activating other tags. The tags belonging

to classes 0-2 are passive, and the energy to operate is derived from the wave frequency, received at the input phase.

- *Reader*: a device used to communicate with tags and act as an intermediary between server and tag. It is generally composed of two main components: a radio frequency interface, which is used to communicate with the tags, and a data control unit, which is responsible for communicating with the application server and for validating the data. Moreover, the control unit is able to perform encryption/decryption operations and, in the most complex cases, put in act authentication mechanisms. A reader is able to both read the information contained by a tag, and to write other data within a tag.
- *Server*: the element in charge of storing data and processing them. Figure 13 shows an RFID system.

RFID technology presents a limited transmission capacity, which starts from a few centimeters and can go up to several meters; this is mainly due to the limited energy resources owned by the tags. Tags belonging to classes 0, 1, or 2 are not capable of self-feeding; instead, the amount of energy they receive depends on factors such as: the distance from the reader, the size of the antenna, the frequency of the waves [65]. One of the main issues affecting RFID communications concerns the collisions' management. As described in [64], communications between readers and tags are susceptible to electromagnetic interference. To solve this problem, different protocols have been adopted, such as Query Tree (QT) and Binary Tree (BT), which are able to manage collisions due to the simultaneous communication of multiple components. Note that passive tags (i.e., classes 0, 1, 2) generally have very limited frequencies.

Due to RFID devices without batteries and limited computational capabilities, it is not always possible to adopt complex algorithms into RFID networks. The work in [64] points out that an unauthorized reader can access the information contained in the tags, if adequate access control systems are not integrated. Other attacks scenarios are described in [66]. A first scenario considers a malicious entity that tries to steal information from a specific tag; in this case, it can perform a cloning attack capable of replicating the violated tag. A second scenario concerns the violation of readers; in this situation, a malicious entity could not only steal information by sending it to untrusted servers, but could even send data deletion commands to the tag devices (in case of tags enabled to the write operation). The third scenario regards the scanning of the tag devices, where interference may occur, also introduced by malicious entities, which degrade the quality of the signal leading to data transmission errors.

3.5.1 Authentication

RFID does not provide mechanisms for authenticating the tags. However, some solutions are available in the literature. For example, in [67], the authors propose a system that provides authentication for RFID devices. It is based on the exchange of identification values calculated by means of *cross bit* functions; such values are exchanged among tags, servers and readers, stored within an *Index Data Table (IDT)* and checked every time a session is started, in order to validate the authenticity of both the reader and the tag. Such a solution guarantees: (i) tag's anonymity since the identification data, transmitted by the tags are encrypted; (ii) resistance to replay attacks, thanks to the use of new keys at each new session; (iii) consistent synchronization, since tags use a *mark* value to indicate the synchronization status; (iv) forward secrecy; (v) mutual authentication; (vi) anti-DoS attack.

The authors of [68] propose a similar approach, which consists of a low energy impact authentication system that uses XOR and bit rotation operations. The actors involved in this system are always the server, the reader, and the various tags. However, as demonstrated in [69], the solution proposed in [68], suffers from key disclosure attack, due to the limited values rotated for generating the keys.

In [70], authentication is guaranteed using elliptical encryption. The proposed system includes two steps: (i) *Setup Phase*, when the encryption keys are generated; (ii) *Authentication Phase*, when a tag authenticates itself towards the server; such a phase consists of five different steps, during which the tag and the server exchange temporary information, in order to verify the compliance of the keys.

The mutual authentication approach, proposed in [71], claims to prevent potential attacks such as forward security, eavesdropping, DoS, tracking, spoofing, and replay. It envisions to provide the readers with a cache in order to be more efficient in IoT-5G environments. In this way, recently visited tags can be directly authenticated in the reader, thus significantly reducing computational costs and server's workload, when a large number of tags wants to be authenticated; also, the delay, introduced by reader-server communication, is avoided. To this end, each reader stores the recently visited *tagIds* in its cache memory: if the reader has recently authenticated a tag, it checks the tag's information just available in its cache to validate the tag's request for authentication; otherwise, if the tag is 'unknown', the authentication must be mediated by the server. The only drawback of such a solution lies in the need to have a huge amount of cache memory available in the readers, so as to be able to store a high number of *tagIds*. An ultralight-weight version of the scheme, just described, is proposed in [71], which reduces the amount of memory,

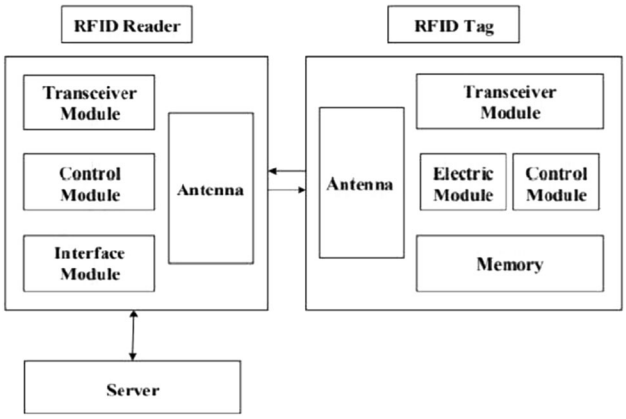

Fig. 13 RFID system

needed by the tag, in case the *tagId* is not already available in the reader's memory. Such an approach is ideal in systems consisting of a large number of tags, which require frequent interactions between reader and server.

The authors of [72] propose the use of the one-way *Cyclic Redundancy Check (CRC)* function, for data encryption, to resist: eavesdropping, tracing attack, replay attack, forward security, spoofing, and de-synchronized attack. The choice of this mechanism is mainly due to its usability for RFID tags belonging to class 1. Furthermore, the CRC function is less expensive regarding energy resources, which further extends its applicability in resource-constrained scenarios. The proposed system consists of two phases: the initialization of secret keys and *tagIds*, and the effective authentication between the tag and the RFID system. In conclusion, the approach envisioned in [72] presents an advantage in terms of computational overhead, derived from the use of CRC functions, compared to the use of hash functions.

3.5.2 Confidentiality

The RFID protocol does not natively satisfy the confidentiality requirement, except for the ISO 14443 standard [73]. In RFID environments, confidentiality management must be divided into two phases. The first one consists in masking the information contained in a tag and sent to the reader. The second one consists in masking the information contained in a reader and sent to the server. Furthermore, also the confidentiality of the information, stored by the server, must be considered; in this case, the server should apply an information hiding mechanism, in order to guarantee data security even if it is violated. Obviously, the tag communications are most exposed to attacks, with respect to the reader and server; this is due to the limited capacity of tag elements, which makes them more vulnerable.

The authors of [66] propose three alternatives: (i) formatting the RFID tag after the reading, thus making it completely unusable; obviously, such a solution is irreversible and is not applicable in IoT scenarios, which are characterized by continuous operations; (ii) setting the RFID tag in a sleeping state; in this way the content and functionality of the tag are not removed, but only temporarily disabled; (iii) setting the tag in the blocking state, simply by changing the value of a flag present in the RFID tag itself; in this state, the tag does not react to external scans on its own initiative; (iv) encrypting the information contained in the device, without altering its capabilities.

In [74], the use of stream-based encryption, through hash functions, is introduced. The proposed system guarantees low energy consumption, and uses a random key generator and a filter for removing the linearity and mathematical correspondences among the generated keys. The same encryption system is also used for the calculation of the MAC value, in order to ensure data integrity. Simulations demonstrated that the system does not introduce computational overhead in RFID tags.

3.5.3 Integrity

As introduced in [74], a hash function to compute the MAC value can guarantee data integrity, since RFID does not natively provide mechanisms to satisfy such a requirement. The approach, presented in [75], verifies that malicious entities have violated no internal or external component to the tags. To this end, a chain of trust is defined, so that each component of the system can verify the integrity of the next one. However, in order to carry out a complete system analysis, it is necessary that all applications within the system are verified. To complete this task, the use of an *Integrity Measurement Architecture (IMA)* is recommended. It is a component natively included in the Linux kernel and capable of performing operations to detect any tampering. Furthermore, the work in [75] points out the need to separate the integrity information manager from the actual validation component, in order to avoid single-point-of-failures.

The main goal is to exclude external or violated devices from the network, since they do not provide reliable data; in addition, [75] highlights the need to encrypt and sign the integrity values, before they are transferred to the validator device, in order to meet also the privacy requirement. S-AES is proposed as the encryption algorithm, which is less expensive in terms of computational resources and required memory than traditional AES. Such a solution can be applied in RFID or NFC (Near Field Communication) based networks. Also, possible integration with blockchain should be considered, in order to completely decentralize the management of integrity certificates.

3.5.4 Authorization

In an RFID scenario, a tag must provide data exclusively to an authenticated and authorized reader; while, an authenticated reader must be able to communicate exclusively with its subset of tags. In addition, access to the resources stored on the server must also be authorized.

The authors of [76] propose a system based on the Hyperledge Fabric (HLF) blockchain to ensure robust access control mechanisms and authorization to the RFID network. A tag sends a request to a reader providing its identity in the envisioned solution. The reader forwards the request to the server that queries the HLF blockchain network through a smart contract. The HLF network, after retrieving the information and the set of permissions, will forward the response to the server that will validate the initial request made by the tag. The access control system is based on ABAC (Attribute-Based Access Control). In this way, a unique key is calculated starting from the device's attributes.

A similar solution, also exploiting the ABAC model, but adopting Ethereum blockchain is presented in [77]. Two systems have been set up for validation purposes: the first uses a local blockchain, while the second uses the Testnet blockchain. The results obtained from the two simulations consider only the time necessary to perform a query and to make an insert into the blockchain. The obtained results are suitable for the local blockchain with respect to the Testnet blockchain. Another interesting comparison is related to the blockchain Casper.

3.6 ZigBee

The ZigBee[13] protocol has been conceived to operate in scenarios that need to preserve the devices' energy; in fact, both the transmission range and the transmission speed are very low. Such limited capabilities allow reducing the energy requirements of the protocol towards the end devices, as described in [78]. A peculiarity of the ZigBee protocol is that it can simultaneously manage a very large number of devices (i.e., more than 65,000) [79]. ZigBee develops three topologies for the cooperation of such a huge number of devices: mesh, tree, and star. In these three cases, neighboring devices can communicate with each other and can operate as forwarders of a signal, in order to extend its range and reach the destination. Figure 14 sketches the three topologies implemented by ZigBee. Such a protocol uses the 802.15.4 transmission standard, which is specifically designed for low energy impact networks and introduces:

- *ZigBee Network Layer (NWK)*: the level that extends the capabilities of the network layer offered by the 802.15.4 standard by introducing the management of the mesh topology, the packets' routing, and the management of ZigBee addresses (note that their format differs from that proposed in 802.15.4)
- *Application Support (APS)*: level responsible for managing the binding table, that is a table containing the information about the devices belonging to the network.
- *Security Service Provider (SSP)*: the layer that deals with managing security both for the network layer and for the application layer, carrying out tasks such as keys' management.
- *ZigBee Device Object (ZDO)*: this layer manages the initialization of the device, the NWK, and APS levels, as well as defining the role that the device will play.
- *Application Framework (APF)*: this layer is responsible for running the user-defined application and is primarily responsible for the information exchange among applications.

Table 4 summarizes the frequencies used by ZigBee and the respective transmission speeds; moreover, in all three cases, the transmission range, if not obstructed, extends up to 100 meters.

As described in [78] and [80], a ZigBee network includes three different kinds of devices:

- *Coordinator*: it is responsible for initializing the network and for allowing other devices to access the system. It can also be configured with additional services.
- *Router*: this component is in charge of performing routing activities. It is always in an active state.
- *End devices*: all the devices that acquire and transmit information. They can only communicate with devices identified as parents. End-devices can enter sleeping mode, in order to save resources.

Figure 15 represents the structure of a ZigBee frame, whose maximum size is equal to 128 Bytes.

As described in [78], the ZigBee protocol natively implements mechanisms to manage encryption keys. More in detail, each ZigBee device stores three types of keys: (i) a *master key*, which is pre-installed inside the device and is used for the key exchange phase; (ii) a *network key*, that is a 128-bit long encryption key shared between all network components and necessary if broadcast communications are carried out; (iii) a *link key*, which is 128 bits long, as the network key, but it is used at the application level for unicast communications; a link key is shared for each pair of communicating ZigBee devices.

[13] https://zigbeealliance.org/

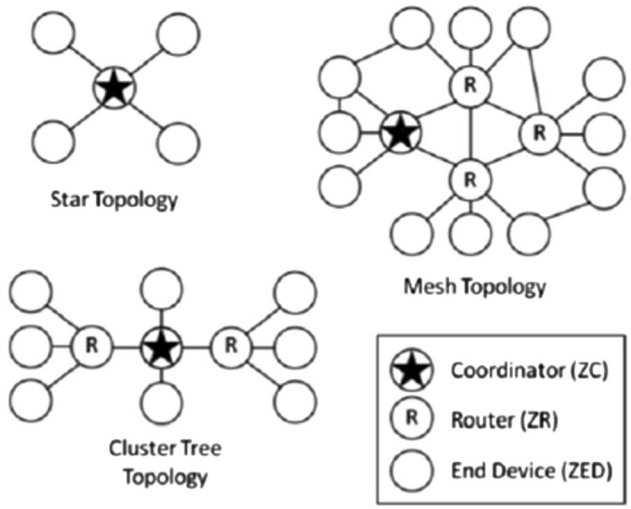

Fig. 14 ZigBee network topologies

Table 4 ZigBee frequency and transmission speed

Frequency	Transmission speed (kb/s)
2.4 GHz	250
915 MHz	40
868 MHz	20

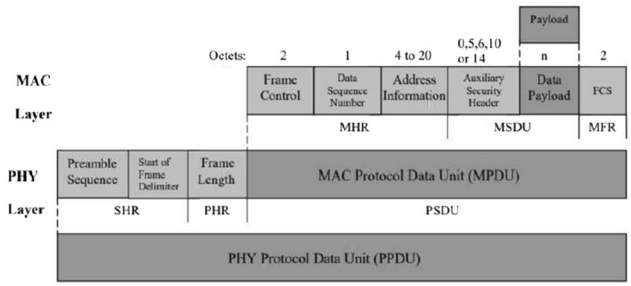

Fig. 15 ZigBee frame

Furthermore, ZigBee natively makes available some mechanisms to secure end-to-end communication [81]. At the application level, it adopts the AES algorithm in Cryptographic Block Ciphers (CBC-MAC) mode to encrypt information and, at the same time, to guarantee authentication during the transmission phase. Regarding data integrity, ZigBee uses the Message Integrity Check (MIC) mechanism, capable of detecting any unauthorized data changes or errors caused by interference. To reduce the computational load, ZigBee allows using the duplicate keys in multiple protocol stack layers.

The authors of [82] point out that the ZigBee end devices do not implement mechanisms capable of guaranteeing security against any physical tampering. While, about attacks against communications, ZigBee, thanks to the use of a counter parameter, is able to resist replay attacks; every time a device receives a duplicate message, it just discards it. Also, [82] highlights that ZigBee offers two security models: centralized and decentralized. The former involves the integration of a *Trust Center (TC)*; this role is played by the coordinator, which is responsible for authenticating end devices and routers; in addition, it generates and shares the network key. In the latter, the TC node is removed and replaced with the routers included in the network. Such routers will be able to perform the tasks previously performed by the CT. In [82], the use of the centralized version is recommended, although it may cause single-point-of-failures.

3.6.1 Authentication

One of the requirements natively guaranteed by the ZigBee protocol is authentication; thanks to the use of the AES-CCM protocol and the implementation of a join phase, the end devices can authenticate to the network. Two separate points must be distinguished: (i) data authentication, intended as the capability of demonstrating that the transmitted data has been generated and communicated by an authentic device, through the use of a key exclusively known by the sender and recipient; (ii) authenticity of the device, which is guaranteed through the phase of joining. Such a phase can operate in two modes: residential mode and commercial mode. The first one foresees that the coordinator sends the network key to a new device; as detailed in the following, such a key is sent over an insecure channel, giving rise to possible vulnerabilities [82]. The second type of join does not expose itself to the vulnerability described above. The TC, impersonated by the coordinator, will send only the master key if needed; through this key, the client and the server will start the phase where the phase is needed the remaining keys will be exchanged. Both commercial and residential modes require devices authentication to occur within a limited period.

The authors of [83] propose the use and control of the radio frequency footprint, which is generated by the analyzed device, in order to improve the authenticity. The basic functionality states that, when a device sends some information, the TC checks the fingerprint produced by the transmission by comparing it with the stored one before proceeding with the standard flow. To make this validation possible, it is necessary to ensure a minimum error in the evaluation, because the generated footprint could be altered by interference. In conclusion, the peculiarity of this system lies in the uniqueness of the imprint and its non-replicability.

3.6.2 Confidentiality

As introduced in Sect. 3.6, ZigBee ensures confidentiality by implementing the AES-CCM mechanism, which is the AES algorithm used in counter mode and *Chiper Block Changing Message Authentication Code (CBC-MAC)*. As presented in [84], this encryption mechanism is one of the most suitable for IoT environments, also in the presence of constrained devices. [84] highlights the possibility of setting the ZigBee network in such a way that only encrypted communications are allowed. Hence, only the devices owning the network key will be able to communicate with the coordinator; moreover, the server will have to periodically start the refresh of the network key, so that the static nature of the key does not turn into a vulnerability.

The work in [82] demonstrated that a malicious entity would be able to listen to the communication during the join phase of the end device to the network, in order to intercept the message containing the password that the TC or the routers send to the new device. The result of the conducted analysis and tests showed that, although the password is sent in an encrypted format, it is not difficult to trace its actual value (e.g., due to the limited length of the password itself). However, it must be considered that a device rarely joins the network. This operation takes place in a very limited time window, effectively reducing the chances of intercepting the correct message. Based on this statement, the authors also demonstrated the ability to induce a device to perform the join phase, thus disclosing the network key. Further tests regarding replay attacks and association flooding have been conducted in [82]; both tests produced a result in favor of the ZigBee protocol proving to be resistant to such attacks.

The authors of [85] propose an improved version of the management of the initial exchange of keys; this solution involves the use of ECDH for key exchange and *subMAC*, which is a reduced version of the MIC, used to fill the gaps in ECDH regarding authentication and man-in-the-middle vulnerabilities. To demonstrate the validity of the approach proposed in [85], a test environment has been set up, which revealed a negligible increase in the amount of energy required by the devices.

In [86] a type of algorithm, different from AES, is considered. The proposed solution is based on the robust, efficient, and unpredictable cryptographic system called *PieceWise Linear Chaotic Map (PWLCM)*; the peculiarities of this algorithm lie in the possibility of parallelizing the encryption processes and in the simplicity of the implemented operations, such as multiplication, division, addition and subtraction. As highlighted in [86], AES-CTR is expensive in terms of computational resources and memory. In order to demonstrate the actual advantage in using the chaotic PWLCM algorithm, different tests have been conducted. The outcome of the experiments revealed that PWLCM requires one-eighth lower times than AES-CTR, and one-third lower time than S-AES. Summarizing, the solution proposed in [86] can resist the most common attacks, making it an excellent candidate for replacing AES.

3.6.3 Integrity

As regards integrity, in the ZigBee protocol, the MAC level is responsible for calculating the MIC of the whole packet; Fig. 16 shows the structure of a MAC packet. The key used to compute the MIC value is set by the upper levels of the protocol stack, so that it is always equivalent to the current network key. Furthermore, AES-CCM provides different MIC calculation methods, which vary their length from 32 bits, to 64 and 128 bits. Combined with this mechanism, it is always possible to calculate an integrity value at the application level. Instead, Fig. 17 shows a sender and a receiver that check the validity of the MIC, through the AES-CCM algorithm.

3.6.4 Authorization

Concerning the authorization requirement, the ZigBee protocol implements an ACL in the 802.15.4 MAC layer, stored in the *PAN Information Base (PIB)*. PIB is a set of attributes owned by each device. The ACL stores the encryption keys, the available security algorithms, the counter that avoids the replay attack, and, finally, the devices enabled to communicate to prevent an unauthorized device from establishing communication. Furthermore, as introduced in Sect. 3.6, ZigBee requires that only parent devices can communicate with child devices. Hence, two communicating devices must share a link key; if an external device tries to communicate with a device inside the network, in addition to not knowing the network key, causing the integrity test to fail, it would not even be able to encrypt the information since it does not have the link key.

In [87], the authors present a system able to recognize the intrusion of an unauthorized device within the ZigBee network; the approach is based on the radio-frequency fingerprint emitted by the device. This solution is based on machine learning and, therefore, requires an initial setup phase. Hence, such an approach is able to authorize or deny access to the requesting devices, guaranteeing a high degree of reliability. Such an identity check phase is

Fig. 16 ZigBee MAC frame

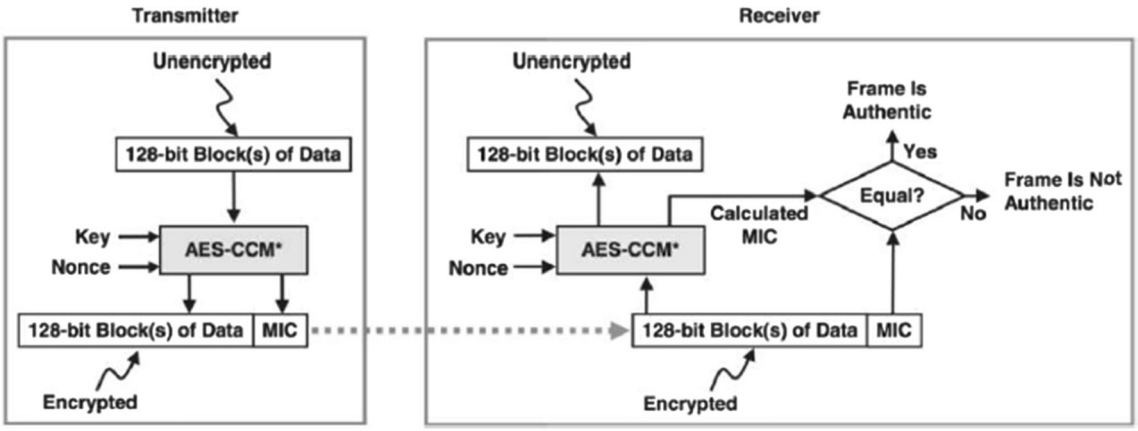

Fig. 17 MIC validation scheme

carried out at the physical level. Therefore, it does not require data decryption and MIC validation, saving resources when the sender node is not authorized.

3.7 Sigfox

The Sigfox[14] protocol has been conceived to have a reduced computational and energy impact on the end devices, but a reduced number of messages that can be transmitted. It mainly finds application in wireless networks connecting low-power devices, which must constantly be on and emit small amounts of data (e.g., electric meters, smartwatches). Besides a maximum transmission speed ranging from 100bps to 600bps, the devices can send a limited number of bytes within the packet payload (i.e., 12 bytes in the uplink and 8 bytes in the downlink), and they can daily perform a maximum of 140 outgoings, and four incoming transmissions [88]. For such a reason, it is necessary to minimize the transmissions of devices; for example, adopting security mechanisms that do not require sending shared parameters or do not require long handshaking phases. It is necessary to consider the possibility of packets loss. The Sigfox protocol is mainly composed of four elements:

- *Object Data*: the end devices that acquire and transmit data to the server. Note that transmissions are carried out in broadcast mode to all the base stations within the transmission's range.
- *Sigfox RAN*: this element has the task of listening to the communications of the end nodes and sending them to the *Sigfox Cloud*.
- *Sigfox Cloud*: it is the Sigfox backend cloud that receives data, validates them, and transmits them to connected applications. In this layer, a graphical interface is available, through which users can interact and manage the network.
- *Customer App*: optional element completely managed by the administrator. It can communicate with the *Sigfox Cloud* via proper API.

Sigfox is based on an LPWAN, which is a low energy consumption WAN network adopting frequencies that vary according to the geographical area: 868 MHz in Europe and 915 MHz in the United States. The transmission range can reach up to 15 kilometers. Sigfox network uses Ultra-Narrow Band (UNB) signals, which help to improve the reliability of transmissions by reducing communication interference. Furthermore, each message is sent three times on three different random frequencies, but without acknowledgment mechanisms performed by the base stations.

The authors of [89] conducted a study regarding the possibility of using Sigfox devices using only the energy accumulated through sunlight. The solution involves the use of a small panel able to load a supercapacitor which allows the device to activate and transmit information. In [89], some tests have been conducted, which, although providing encouraging results for future research, did not fully reflect reality and did not consider the worst cases. This is due to the fact that the packets transmitted by the device contained a payload of only 1 byte, against the maximum 12 accepted by the protocol. However, the approach taken by this study would allow reducing the environmental impact deriving from the production of batteries, decreasing or completely eliminating the maintenance required for their replacement. To conclude, it is necessary to underline the limitations in the use of solar panels, such as the absence of sunlight inside buildings or in shady areas, or the lack of energy caused by unfavorable weather conditions or specific periods of the year.

The security provided by the Sigfox protocol imposes further constraints. It is not enough to consider the

[14] https://www.sigfox.com/en

limitations imposed by the devices, such as energy efficiency or computational complexity. Still, it is also necessary to evaluate the constraints imposed by the structure of the protocol itself, just presented above. For the same reason indicated for CoAP (i.e., the number of messages exchanged in the handshake phase) and the maximum packet size imposed by Sigfox, DTLS cannot be easily adopted in the SigFox protocol. During the device's initialization phase, the following unique parameters are stored: the *deviceID*, which is an identifier associated with the device, and the *Network Authentication Key (NAK)*, that is a symmetric key essential to compute the *MIC* value and guarantee the device authentication. In addition, Sigfox makes it optional to use the $AES-CTR$ algorithm to encrypt data. If this feature is activated, the device will generate an encryption key through a *Key Derivation Function* passing the NAK as a parameter.

In order to guarantee a high level of security, all data transmissions taking place among the base stations and the Sigfox Cloud are protected by a *VPN (Virtual Private Network)*. While the communications that take place between the Sigfox Cloud and the Customer App use the HTTPS protocol, consequently protecting data using the TLS protocol. Sigfox devices use neither TCP nor UDP protocols to carry out communications: it is based on MAC addresses. Therefore, there are no OTAA dynamic activation mechanisms (as described in Sect. 3.3), reducing the possibility of attack [90]. Except in case of a request from a device, the Sigfox protocol does not allow bidirectional communications among end devices [91]. Sigfox inserts a 12-bit Sequence Number (SN) field to protect from replay attacks within each packet. As pointed out by the authors of [92], such a 12-bit SN introduces a vulnerability. Considering that with 12 bits, it is possible to represent only 4096 different values, and the keys used by Sigfox tend never to change, it is clear that a malicious entity could intercept the message, decrypt it, and, consequently, use and modify it without any constraint. Moreover, considering that a device can send a maximum of 140 messages per day, after 30 days, all available SNs would be exhausted, and the device would re-use the same SN, thus making the replay attack possible.

3.7.1 Authentication

Sigfox natively implements a simple mechanism based on checking the *deviceID* and the *NAK* key, in order to guarantee authentication. The server uses the first parameter to check that the correct *deviceID* is associated with the device; while the second is used to compute the MIC value that can also guarantee data integrity. Therefore, Sigfox Cloud has two unique information to validate, and if one of them is incorrect, the package would be discarded. As noted in [91], each base station stores unique parameters that allow authentication to the Sigfox Cloud. In this way, a malicious entity cannot communicate with the Sigfox back-end without owing such parameters.

3.7.2 Confidentiality

Concerning confidentiality, Sigfox natively implements the AES-128 mechanism, albeit optionally. Hence, the administrator should set the use of the security algorithm. AES encrypts the message using the encryption key, calculated using the *Key Derivation Function*. As pointed out in [90], Sigfox still implements a data encryption mechanism at the application level and disable AES.

The authors of [93] propose the adoption of the One Time Pad (OTP) system as a data encryption mechanism. With respect to asymmetric encryption and elliptical encryption, such a solution does not transmit additional packets and does not alter the size of the final packet. Elliptic encryption requires that sender and receiver agree on the parameters used for encryption; while asymmetric encryption uses too long keys. For such reasons, the use of cryptographic systems based on symmetric keys remains the only option. The constraints imposed by the OTP protocol are known, such as: (i) each key used must be random, and it is, therefore, necessary to exploit a module, named *True Random Number Generator (TRNG)*, for generating random keys; (ii) the keys must be exchanged through a secure channel; (iii) each key must be at least as long as the text to be encrypted; (iv) the keys must be used only once. In modern systems, the most complex point to satisfy is the fourth, which imposes the use of different keys. However, this limit does not occur with the Sigfox protocol since, in the worst case (i.e., every day sending the maximum number of messages available in one day), over five years, 255,000 messages are sent; consequently, the device would have no difficulty in generating such a number of keys. In [93], two different simulations are carried out: the first consists in measuring the actual impact in the use of AES-128, while the second simulation compares OTP with AES-128 and ChaCha20[15], which is a lightweight encryption algorithm. In order to emulate different types of devices, different CPU frequencies are set: 4 MHZ, 8 MHZ, and 16 MHZ. As specified in [93], the first simulation showed that the use of an encryption system on a low energy consumption device does not cause a high overhead. The second simulation produced the following results: (i) AES-128-CTR is the least expensive solution in terms of energy and the fastest with the CPU set at 16 MHZ; (ii) OTP is the fastest solution with the CPU set to 4 MHZ and presents a slight increase in consumption

[15] https://tools.ietf.org/html/rfc7539

compared to AES; (iii) ChaCha20 presents higher values in terms of consumption and execution time; (iv) AES-128-CTR is the best solution in terms of execution time and energy overhead, even if ChaCha20, compared to AES, provides resistance to timing attacks and cache-collision attacks. Hence, we can note that OTP is able to offer excellent performance and reduced consumption, as well as being the most robust. As the last feature, the need to integrate an automatic key management mechanism emerges, because, in the proposed system, the keys must be pre-shared, while, in a real system, this is not always possible.

3.7.3 Integrity

As regards data integrity, Sigfox uses the AES algorithm in CMAC mode. Such a mechanism takes the SN's authentication key and produces a MAC value calculated on the whole packet. The result of this operation is subsequently truncated to 2 or 5 bytes. As pointed out in [94], using a 128 bits long MAC value would reduce the battery life of a device by 45%. Furthermore, such a mechanism is applied for messages that are uplinked by an end device; while, regarding the messages received in the downlink, Sigfox does not provide any information [92].

The authors of [94] propose a novel approach for data integrity, which is based on *Cumulative MAC (CuMAC)*. It includes two separate mechanisms: aggregation and accumulation. In the former, the sender generates short *Authentication Tags (AT)* starting from the MAC value; while in the latter, the receiver accumulates the received AT and validates them. Such a mechanism requires that the sender and receiver share confidential content (secret) which, in the case of Sigfox, could be the NAK. As highlighted in [94], a malicious entity cannot generate the same AT value. Figure 18 shows the tag generation scheme. Also, [94] describes a more efficient version of CuMAC, named *CuMAC/S*. This solution has a reduced latency in the validation process. The simulations carried out by the authors, reveal that such a scheme requires a huge amount of memory in the end devices, but, at the same time, it is able to guarantee a higher level of security. In conclusion, it is necessary to assess whether this mechanism is also applicable in Sigfox scenarios, which are particularly limited in energy terms. Note that the simulations carried out in [94] are based on a *CAN (Controller Area Network)* system, that is a serial system for connecting several units in multicast mode.

3.7.4 Authorization

As regards the authorization requirement, Sigfox only implements the device identity control mechanism, as described in Sect. 3.7. If a device does not have the correct *deviceID* and the correct *NAK*, its communications will be ignored by the Sigfox Cloud. This operating mode is motivated by the fact that end devices do not logically and/or physically belong to a specific network without such parameters [94]. Consequently, the Sigfox network administrator will be able to view only the correctly configured devices through the graphical interface made available by Sigfox, and will not be able to interact with other devices.

4 Discussion

The analysis, conducted in Sect. 3 on security requirements related to relevant IoT communication protocols, has emerged many peculiarities and leads to make some important thoughts, examined in the following. It is worth noting that three of the analyzed protocols (i.e., MQTT, CoAP, and AMQP) are inherent to the application layer, and are mainly built on the top of TCP or UDP, while the others (i.e., LoRaWAN, RFID, ZigBee, Sigfox) have their own protocol stack, besides they are more related to network or physical layer, respectively, as described in Sect. 3. For such a reason, a direct comparison in terms of security must consider the underlying network infrastructure for the application layer protocols. Hence, in Table 5, the IoT protocols investigated in Sect. 3 are grouped as stated above, and the topics addressed are split on the basis of the treated security features. What emerges looking at Table 5 (and also confirmed in Sect. 3) is that authentication methods are the most investigated in the literature, along with confidentiality and integrity; while authorization deserves much more attention by the research community, since few solutions are provided on this topic in the investigated field with respect to the other investigated research areas.

AMQP revealed to represent a robust communication protocol, since it is based on TLS and SASL, which guarantee confidentiality, integrity, authentication, and authorization. However, AMQP is not recommended in scenarios involving resource-constrained devices. CoAP or ZigBee protocol can be adopted to reduce the consumption overhead while ensuring a high-security degree. Note that CoAP, being by design suitable for datagram, principally runs on UDP (possible alternatives are NIDD and SMS), and it is the only protocol that provides interoperability with the HTTP protocol. Other protocols do not refer to any possibility to interoperate with other ones. Also, CoAP presents some similarities with MQTT, even in terms of security because they both rely on (D)TLS. MQTT is designed for TCP transport, and it mainly focuses on providing lightweight communication primitives, disregarding

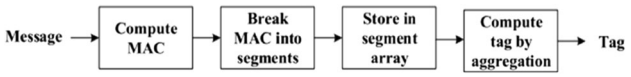

Fig. 18 Tag generation scheme

security features, which are mostly delegated to the underlying TLS protocol. Instead, CoAP, as discussed in Sect. 3.2, is already integrated with higher-layer protocols, such as LwM2M and OSCORE, to provide additional security functionalities to the network. Moreover, CoAP usually relies on DTLS. In case of high latency and Round-Trip Time (RTT) at physical layer, CoAP over datagram transport constitutes a better choice compared to MQTT due to TCP issues in such environments. ZigBee natively ensures the requirements of confidentiality, data integrity, authentication, and authorization. The first two requirements are guaranteed through the use of AES-CCM and a secret key, the third one is achieved through AES, and a device join phase, while the fourth one is ensured by implementing an ACL at the MAC layer of the devices and through the sharing of unique keys for each pair of communicating devices. Suppose the application domain requires to support normal, or at least high. In that case, communication distance, the more suitable protocols to be adopted are LoRaWAN and Sigfox, due to their intrinsic features, as presented in Sects. 3.3 and 3.7, respectively. The latter also offers higher communication speed, not imposing any limit in daily communications, as Sigfox. Table 6 resembles the main features related to the analyzed protocols: the communication speed, the maximum packet size at the application level, and the communication range. Instead, for short-range communications, RFID represents the well-recognized solution. Such protocols do not natively provide security capabilities, but many methods (mainly in the case of RFID) have been proposed in the literature. Table 7 points out, for each IoT protocol, the best application scenario, in terms of communication distance, packets' transmission speed, amount of transmitted information, resource consumption, and robustness (concerning security requirements). Note that two checkmarks ✔✔ are used to indicate that the corresponding feature is highly recommended for the corresponding protocol, only one checkmark ✔ suggests that the requirement is generally met, while the ✗mark indicates that the related requirement is not fulfilled.

It is worth noting that many approaches, independently from the considered protocol, exploit "simple" encryption mechanisms to preserve the resources of IoT devices. Such methods include CRC, hash functions, cross bits functions. On the opposite side, the use of the AES (and its variants) algorithm is still widely recognized, besides it requires more resources on IoT devices to be executed.

A growing number of studies propose the adoption of blockchain technology, as emerged for some of the presented protocols (i.e., MQTT, CoAP, LoRaWAN, AMQP, RFID) to guarantee the requirements of integrity and non-repudiation. Instead, more marginal is the possible integration with machine learning techniques [95], as presented in some solutions tailored to AMQP and ZigBee. Hence,

Table 5 Security requirements in IoT communication protocols

APPLICATION LAYER PROTOCOL	Authentication	Integrity	Confidentiality	Authorization
MQTT	Guaranteed at the application level [15] [17] [18] [19] [20] [22] [23]	Guaranteed by TLS [17] [20] [22] [23]	Guaranteed by TLS [17] [20] [22]	Not guaranteed [20] [21] [24] [25]
CoAP	Guaranteed by TLS/DTLS [11] [33] [34] [35] [36] [37]	Guaranteed by TLS/DTLS [27] [32] [41] [42]	Guaranteed by TLS/DTLS [32] [39] [41] [42]	Guaranteed [27] [34] [43] [44]
AMQP	Guaranteed by TLS and SASL [60]	Guaranteed by TLS [61]	Guaranteed by TLS [63]	Guaranteed [25] [61] [62]
NETWORK SYSTEM	Authentication	Integrity	Confidentiality	Authorization
LoRaWAN	Guaranteed [49] [50] [51] [55]	Guaranteed [53] [54] [55]	Guaranteed [53] [54] [55]	Guaranteed [51] [56] [57]
RFID	Not guaranteed [67] [68] [69] [70] [71] [72]	Not guaranteed [66] [73] [74]	Not guaranteed [74] [75]	Not guaranteed [76] [77]
ZigBee	Guaranteed [78] [81] [83]	Guaranteed [78] [81] [82] [84] [86]	Guaranteed [78] [81] [82] [84] [85] [86]	Guaranteed [78] [81]
Sigfox	Guaranteed [91]	Optional [90] [93]	Guaranteed [92] [94]	Not guaranteed [94]

Table 6 Main features related to IoT communication protocols

IoT PROTOCOL	Communication speed	Maximum packet size	Communication range
MQTT	Delegated to the physical layer	256 MB	Delegated to the physical layer
CoAP	Delegated to the physical layer	64 KB	Delegated to the physical layer
LoRaWAN	From 250 bps to 50 Kbps	255 byte	10 km
AMQP	Delegated to the physical layer	2 GB (RabbitMQ)	Delegated to the physical layer
RFID	Maximum 640 Kbps	2 KB (minimum 5 byte)	From 10 to 200 m
ZigBee	250 Kbps	128 byte	100 m
Sigfox	600 bps (US), 100 bps (EU)	24 Byte	More than 10 km

Table 7 Ideal scenario for each IoT communication protocols

IoT PROTOCOL	High communication Distance	Short range Communication	High transmission Speed	Huge amount of Transmitted data	Low resource Consumption	Security Support
MQTT	✓	✓	✓	✗	✓ ✓	✓
CoAP	✓	✓	✓ ✓	✗	✓ ✓	✓
LoRaWAN	✓ ✓	✗	✓	✗	✗	✓ ✓
AMQP	✓	✓	✗	✓	✗	✓ ✓
RFID	✗	✓ ✓	✓	✗	✓	✗
ZigBee	✗	✓ ✓	✓	✗	✓	✓
Sigfox	✓ ✓	✗	✓ ✓	✗	✗	✓

the following challenges can be pointed out with respect to the conducted analysis:

- Since IoT networks often manage *sensitive* information (e.g., e-health scenarios, smart homes), the communication protocol adopted inside the system should also meet the **privacy** and **data protection** requirements. Ciphering surely represents the first means to protect the information. Still, it implies the definition of a clever key management system, including key distribution and revocation mechanisms, in case of keys' violation. Moreover, the keys' dimension must be taken into account to limit the overhead of data transmitted over the network. End-to-end encryption and decryption mechanisms are preferable in the highly distributed IoT environment, in order to limit intermediate operations, which can hinder the reliability of the information. As cited above, blockchain technology could play a fundamental role in coping with this emerging issue, with the opportune integration with the transmission protocol.
- In an IoT environment, networks' components can be very heterogeneous, and the concept of **trust** of the devices, gateways, and servers that manage the information must be considered, because the end-users must be aware of the reliability and recovery capability of the system in case of violation. This will encourage the adoption of new protocols and technologies.
- More effort must be put in **standardization process** for the security-related features to be associated with the investigated protocols. Uniform and wide-recognized solutions would help to solve **interoperability** issues when different systems need to be integrated.

5 Conclusion

The paper conducted an analysis of the state-of-the-art about how the most widespread and important IoT communication protocols satisfy security requirements (i.e., authentication, confidentiality, integrity, authorization) For each protocol (i.e., MQTT, CoAP, LoRaWAN, AMQP, RFID, ZigBee, and Sigfox), a general overview has been provided and, subsequently, the authentication, confidentiality, integrity, and authorization requirements have been investigated in depth. Finally, the strengths and vulnerabilities are described, pointing out solutions or alternatives. The conducted analysis allowed to reason about which protocol is more suitable to adopt in a specific scenario on the basis of application requirements, such as the communication distance, the packets' transmission speed, the amount of transmitted information, the resource

consumption, and the level of security required. Privacy, trust, standardization process, interoperability emerged as open issues, as well as the integration of blockchain technology. Moreover, other protocols could be analyzed, such as *Data Distribution Service (DDS)*[16], which is conceived for real-time systems and it is based on a publish &subscribe pattern, and *Extensible Messaging and Presence Protocol (XMPP)*[17], which is designed for instant messaging. The worldwide scientific community should spend further effort to address the topics above, since still many challenges must be faced in the field of communication inside IoT networks.

Funding Open access funding provided by Università degli Studi dell'Insubria within the CRUI-CARE Agreement.

References

1. Marinakis, Y. D., Walsh, S. T., & Harms, R. (2017). Internet of things technology diffusion forecasts. In 2017 Portland international conference on management of engineering and technology (PICMET) (pp. 1–5). IEEE.
2. Alaba, F. A., Othman, M., Hashem, I. A. T., & Alotaibi, F. (2017). Internet of things security: A survey. *Journal of Network and Computer Applications, 88*, 10–28.
3. Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2015). Security, privacy and trust in internet of things: The road ahead. *Computer Networks, 76*, 146–164.
4. Yaqoob, I., Ahmed, E., Hashem, I. A. T., Ahmed, A. I. A., Gani, A., Imran, M., & Guizani, M. (2017). Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications, 24*(3), 10–16.
5. Alqahtani. B., AlNajrani, B. (2020). A study of internet of things protocols and communication. In 2020 2nd International conference on computer and information sciences (ICCIS) (pp. 1–6). IEEE.
6. Al-Sarawi, S., Anbar, M., Alieyan, K. & Alzubaidi, M. (2017). Internet of things (iot) communication protocols. In 8th International conference on information technology (ICIT) pp. 685–690. IEEE.
7. Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing, 3*(1), 11–17.
8. Adat, V., & Gupta, B. (2018). Security in internet of things: Issues, challenges, taxonomy, and architecture. *Telecommunication Systems, 67*(3), 423–441.
9. Das, A. K., Zeadally, S., & He, D. (2018). Taxonomy and analysis of security protocols for internet of things. *Future Generation Computer Systems, 89*, 110–125.
10. Nguyen, K. T., Laurent, M., & Oualha, N. (2015). Survey on secure communication protocols for the internet of things. *Ad Hoc Networks, 32*, 17–31.
11. Granjal, J., Monteiro, E., & Silva, J. S. (2015). Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials, 17*(3), 1294–1312.
12. Dai, H.-N., Zheng, Z., & Zhang, Y. (2019). Blockchain for internet of things: A survey. *IEEE Internet of Things Journal, 6*(5), 8076–8094.
13. Zhang, Y., Kasahara, S., Shen, Y., Jiang, X., & Wan, J. (2018). Smart contract-based access control for the internet of things. *IEEE Internet of Things Journal, 6*(2), 1594–1605.
14. Beckwith, E., & Thamilarasu, G. (2020). Ba-tls: Blockchain authentication for transport layer security in internet of things. In 2020 7th International conference on internet of things: systems, management and security (IOTSMS) (pp. 1–8). IEEE.
15. Sicari, S., Rizzardi, A., & Coen-Porisini, A. (2020). Increasing the pervasiveness of the iot: fog computing coupled with pub &sub and security. In 2020 IEEE international conference on smart internet of things (SmartIoT) (pp. 64–71). IEEE.
16. Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2021). Testing and evaluating a secure-aware pub &sub protocol in a fog-driven iot environment. In 19th International conference on ad hoc networks and wireless (AdHoc-Now 2020).
17. Andy, S., Rahardjo, B., & Hanindhito, B. (2017). Attack scenarios and security analysis of mqtt communication protocol in iot system. In 2017 4th International conference on electrical engineering, computer science and informatics (EECSI) (pp. 1–6). IEEE.
18. Buccafurri, F., De Angelis, V., & Nardone, R. (2020). Securing mqtt by blockchain-based otp authentication. *Sensors, 20*(7), 2002.
19. Diro, A., Reda, H., Chilamkurti, N., Mahmood, A., Zaman, N., & Nam, Y. (2020). Lightweight authenticated-encryption scheme for internet of things based on publish-subscribe communication. *IEEE Access, 8*, 605 39-605 51.
20. Malina, L., Srivastava, G., Dzurenda, P., Hajny, J., & Fujdiak, R. (2019). A secure publish/subscribe protocol for internet of things. In Proceedings of the 14th international conference on availability, reliability and security, pp. 1–10.
21. Dinculeană, D., & Cheng, X. (2019). Vulnerabilities and limitations of mqtt protocol used between iot devices. *Applied Sciences, 9*(5), 848.
22. Sahmi, I., Abdellaoui, A., Mazri, T., & Hmina, N. (2021). Mqtt-present: Approach to secure internet of things applications using mqtt protocol. *International Journal of Electrical & Computer Engineering (2088–8708), 11*(5), 4577–4586.
23. Iyer, S., Bansod, G., Naidu, P., & Garg, S. (2018). Implementation and evaluation of lightweight ciphers in mqtt environment. In 2018 International conference on electrical, electronics, communication, computer, and optimization techniques (ICEECCOT), pp. 276–281. IEEE.
24. Katsikeas, S., Fysarakis, K., Miaoudakis, A., Van Bemten, A., Askoxylakis, I., Papaefstathiou, I., & Plemenos, A. (2017). Lightweight & secure industrial iot communications via the mq

[16] Data Distribution Service, https://www.dds-foundation.org/

[17] Extensible Messaging and Presence Protocol, https://xmpp.org/

telemetry transport protocol. In IEEE symposium on computers and communications (ISCC), pp. 1193–1200. IEEE.
25. Cruz-Piris, L., Rivera, D., Marsa-Maestre, I., De La Hoz, E., & Velasco, J. R. (2018). Access control mechanism for iot environments based on modelling communication procedures as resources. *Sensors, 18*(3), 917.
26. Rizzardi, A., Sicari, S., Miorandi, D., & Coen-Porisini, A. (2016). Aups: An open source authenticated publish/subscribe system for the internet of things. *Information Systems, 62*, 29–41.
27. Albalas, F., Al-Soud, M., Almomani, O., & Almomani, A. (2018). Security-aware coap application layer protocol for the internet of things using elliptic-curve cryptography. *Power (mw), 1333*, 151.
28. Colitti, W., Steenhaut, K., & De Caro, N. (2011). Integrating wireless sensor networks with the web. Extending the Internet to Low power and Lossy Networks (IP+ SN 2011).
29. Kumar, V., Jha, R. K., & Jain, S. (2020). Nb-iot security: A survey. *Wireless Personal Communications, 113*(4), 2661–2708.
30. Dawaliby, S., Bradai, A., & Pousset, Y. (2016). In depth performance evaluation of lte-m for m2m communications. In 2016 IEEE 12th international conference on wireless and mobile computing, networking and communications (WiMob), pp. 1–8.
31. Basu, S. S., Haxhibeqiri, J., Baert, M., Moons, B., Karaagac, A., Crombez, P., Camerlynck, P., & Hoebeke, J. (2020). An end-to-end lwm2m-based communication architecture for multimodal nb-iot/ble devices. *Sensors, 20*(8), 2239.
32. Alliance, O. M. (2018). Lightweight machine to machine technical specification: Core. *OMA, 6*, 21–28.
33. Tamboli, M. B., & Dambawade, D. (2016). Secure and efficient coap based authentication and access control for internet of things (iot). In 2016 IEEE international conference on recent trends in electronics, information & communication technology (RTEICT), pp. 1245–1250. IEEE.
34. Pereira, P. P., Eliasson, J., & Delsing, J. (2014). An authentication and access control framework for coap-based internet of things. In IECON 2014-40th annual conference of the IEEE industrial electronics society, pp. 5293–5299. IEEE.
35. Esfahani, A., Mantas, G., Matischek, R., Saghezchi, F. B., Rodriguez, J., Bicaku, A., Maksuti, S., Tauber, M. G., Schmittner, C., & Bastos, J. (2017). A lightweight authentication mechanism for m2m communications in industrial iot environment. *IEEE Internet of Things Journal, 6*(1), 288–296.
36. Ukil, A., Bandyopadhyay, S., Bhattacharyya, A., Pal, A., & Bose, T. (2014). Auth-lite: lightweight m2mauthentication reinforcing dtls for coap. In 2014 IEEE international conference on pervasive computing and communication workshops (PERCOM WORKSHOPS), pp. 215–219. IEEE.
37. Park, J., & Kang, N. (2014). Lightweight secure communication for coap-enabled internet of things using delegated dtls handshake. In 2014 International conference on information and communication technology convergence (ICTC), pp. 28–33. IEEE.
38. Bormann, C., Ersue, M., & Keranen, A. (2014). Terminology for constrained-node networks. Internet Engineering Task Force (IETF): Fremont (pp. 2070–1721). USA: CA.
39. Halabi, D., Hamdan, S., & Almajali, S. (2018). Enhance the security in smart home applications based on iot-coap protocol. In 2018 Sixth international conference on digital information, networking, and wireless communications (DINWC), pp. 81–85. IEEE.
40. Tamayo, G. (2017). Lempel-ziv-welch (lzw) compression. In https://sites.google.com/site/datacompressionguide/lzw
41. Bhattacharjya, A., Zhong, X., Wang, J., & Li, X. (2020). Coap–application layer connection-less lightweight protocol for the internet of things (iot) and coap-ipsec security with dtls supporting coap. In M. Farsi (Ed.), *Digital Twin Technologies and Smart Cities* (pp. 151–175). Springer.
42. Granjal, J., Monteiro, E., & Silva, J. S. (2013). Application-layer security for the wot: Extending coap to support end-to-end message security for internet-integrated sensing applications. In International conference on wired/wireless internet communication. Springer, pp. 140–153.
43. Beltran, V., & Skarmeta, A. F. (2016). An overview on delegated authorization for coap authentication and authorization for constrained environments (ace). In IEEE 3rd world forum on internet of things (WF-IoT). pp. 706–710. IEEE.
44. Alphand, O., Amoretti, M., Claeys, T., Dall'Asta, S., Duda, A., Ferrari, G., Rousseau, F., Tourancheau, B., Veltri, L., Zanichelli, F. (2018). Iotchain: A blockchain security architecture for the internet of things. In IEEE wireless communications and networking conference (WCNC), 1–6. IEEE.
45. Kane, A., Daniel, F., Zamfir, V. (2020). casperlabs.io. In https://docs.casperlabs.io/en/latest/theory/index.html
46. Kim, J., & Song, J. (2017). A dual key-based activation scheme for secure lorawan. *Wireless Communications and Mobile Computing*. https://doi.org/10.1155/2017/6590713
47. Virk, H. (2019). Introducing lorawan 1.1 support. In https://os.mbed.com/blog/entry/Introducing-LoRaWAN-11-support/
48. Dönmez, T. C., & Nigussie, E. (2018). Security of join procedure and its delegation in lorawan v1. 1. *Procedia Computer Science, 134*, 204–211.
49. Danish, S. M., Lestas, M., Asif, W., Qureshi, H. K., & Rajarajan, M.(2019)"A lightweight blockchain based two factor authentication mechanism for lorawan join procedure," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, pp. 1–6.
50. Navarro-Ortiz, J., Chinchilla-Romero, N., Ramos-Munoz, J. J., & Munoz-Luengo, P. (2019). Improving hardware security for lorawan. In 2019 IEEE conference on standards for communications and networking (CSCN), pp. 1–6. IEEE.
51. Ribeiro, V., Holanda, R., Ramos, A., & Rodrigues, J. J. (2020). Enhancing key management in lorawan with permissioned blockchain. *Sensors, 20*(11), 3068.
52. Sanchez-Iborra, R., Sánchez-Gómez, J., Pérez, S., Fernández, P. J., Santa, J., Hernández-Ramos, J. L., & Skarmeta, A. F. (2018). Enhancing lorawan security through a lightweight and authenticated key management approach. *Sensors, 18*(6), 1833.
53. Yang, X., Karampatzakis, E., Doerr, C., & Kuipers, F. (2018). Security vulnerabilities in lorawan. In 2018 IEEE/ACM third international conference on internet-of-things design and implementation (IoTDI), pp. 129–140. IEEE.
54. Raad, N., Hasan, T., Chalak, A., & Waleed, J. (2019). Secure data in lorawan network by adaptive method of elliptic-curve cryptography. In 2019 International conference on computing and information science and technology and their applications (ICCISTA), pp. 1–6. IEEE.
55. Kim, J., & Song, J. (2018). A secure device-to-device link establishment scheme for lorawan. *IEEE Sensors Journal, 18*(5), 2153–2160.
56. Ayoub, W., Samhat, A. E., Nouvel, F., Mroue, M., & Prévotet, J.-C. (2018). Internet of mobile things: Overview of lorawan, dash7, and nb-iot in lpwans standards and supported mobility. *IEEE Communications Surveys & Tutorials, 21*(2), 1561–1581.
57. Riabi, I., Ayed, H. K. B., & Saidane, L. A. (2019). A survey on blockchain based access control for internet of things. In: 15th International wireless communications & mobile computing conference (IWCMC), pp. 502–507. IEEE.
58. Năstase, L., Sandu, I. E., & Popescu, N. (2017). An experimental evaluation of application layer protocols for the internet of things. *Studies in Informatics and Control, 26*(4), 403–412.

59. Verma, S., & Rastogi, M. A. (2020). Iot application layer protocols: A survey. *Journal of Xi'an University of Architecture & Technology, VII,* 57.
60. McAteer, I. N., Malik, M. I., Baig, Z., & Hannay, P. (2017). Security vulnerabilities and cyber threat analysis of the amqp protocol for the internet of things.
61. Alfandi, O., Khanji, S., Ahmad, L., & Khattak, A. (2020). A survey on boosting iot security and privacy through blockchain. *Cluster Computing, 24*(1), 37–55.
62. Kuzmin, A. (2017). Blockchain-based structures for a secure and operate iot. In Internet of Things Business Models, Users, and Networks, pp. 1–7. IEEE.
63. Hamad, S. A., Sheng, Q. Z., Zhang, W. E., & Nepal, S. (2020). Realizing an internet of secure things: A survey on issues and enabling technologies. *IEEE Communications Surveys & Tutorials, 22*(2), 1372–1391.
64. Jia, X., Feng, Q., Fan, T., & Lei, Q. (2012). Rfid technology and its applications in internet of things (iot). In 2nd international conference on consumer electronics, communications and networks (CECNet), pp. 1282–1285. IEEE.
65. Pessl, P., & Hutter, M. (2013). Pushing the limits of sha-3 hardware implementations to fit on rfid. In International workshop on cryptographic hardware and embedded systems, pp. 126–141. Springer.
66. Nie, X., & Zhong, X. (2013). Security in the internet of things based on rfid: issues and current countermeasures. In Proceedings of the 2nd international conference on computer science and electronics engineering. Atlantis Press.
67. Fan, K., Jiang, W., Li, H., & Yang, Y. (2018). Lightweight rfid protocol for medical privacy protection in iot. *IEEE Transactions on Industrial Informatics, 14*(4), 1656–1665.
68. Tewari, A., & Gupta, B. (2017). Cryptanalysis of a novel ultra-lightweight mutual authentication protocol for iot devices using rfid tags. *The Journal of Supercomputing, 73*(3), 1085–1102.
69. Wang, K.-H., Chen, C.-M., Fang, W., & Wu, T.-Y. (2018). On the security of a new ultra-lightweight authentication protocol in iot environment for rfid tags. *The Journal of Supercomputing, 74*(1), 65–70.
70. Liao, Y.-P., & Hsiao, C.-M. (2014). A secure ecc-based rfid authentication scheme integrated with id-verifier transfer protocol. *Ad Hoc Networks, 18,* 133–146.
71. Fan, K., Gong, Y., Liang, C., Li, H., & Yang, Y. (2016). Lightweight and ultralightweight rfid mutual authentication protocol with cache in the reader for iot in 5g. *Security and Communication Networks, 9*(16), 3095–3104.
72. Shi, Z., Chen, J., Chen, S., & Ren, S. (2017). A lightweight rfid authentication protocol with confidentiality and anonymity. InIEEE 2nd advanced information technology, electronic and automation control conference (IAEAC), pp. 1631–1634. IEEE.
73. Knospe, H., & Pohl, H. (2004). Rfid security. *Information Security Technical Report, 9*(4), 39–50.
74. Shen, S.-S., Liao, H.-R., Lin, S.-H., & Chiu, J.-H. (2011). A novel stream cipher with hash function for the rfid device. In 2011, Fifth international conference on innovative mobile and internet services in ubiquitous computing, pp. 532–536. IEEE
75. Mubarak, M. F., Yahya, S., et al. (2011). Trusted anonymizer-based rfid system with integrity verification. In 2011 7th International conference on information assurance and security (IAS), pp. 98–103. IEEE.
76. Figueroa, S., Añorga, J., Arrizabalaga, S., Irigoyen, I., & Monterde, M. (2019). An attribute-based access control using chain-code in rfid systems. In 2019 10th IFIP international conference on new technologies, mobility and security (NTMS), pp. 1–5. IEEE.
77. Figueroa, S., Añorga, J., & Arrizabalaga, S. (2019). An attribute-based access control model in rfid systems based on blockchain decentralized applications for healthcare environments. *Computers, 8*(3), 57.
78. Sethi, P., & Sarangi, S. R. (2017). Internet of things: architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*. https://doi.org/10.1155/2017/9324035.
79. Zillner,T., & Strobl, S. (2015). Zigbee exploited: The good, the bad and the ugly. Black Hat
80. Aju, O. G. (2015). A survey of zigbee wireless sensor network technology: Topology, applications and challenges. *International Journal of Computer Applications, 130*(9), 47–55.
81. Khanji, S., Iqbal, F., & Hung, P. (2019). Zigbee security vulnerabilities: Exploration and evaluating. In 2019, 10th International conference on information and communication systems (ICICS), pp. 52–57. IEEE.
82. Fan, X., Susan, F., Long, W., & Li, S. (2017). Security analysis of zigbee. Comput. Netw. Secur. Class, Massachusetts Inst. Technol., Cambridge, MA, USA. Rep.
83. Ramsey, B. W., Temple, M. A., & Mullins, B. E. (2012). Phy foundation for multi-factor zigbee node authentication. In 2012, IEEE global communications conference (GLOBECOM), pp. 795–800. IEEE.
84. Ferreira, H. G. C., de Sousa, R. T., de Deus, F. E. G., & Canedo, E. D. (2014). Proposal of a secure, deployable and transparent middleware for internet of things. In 2014, 9th Iberian conference on information systems and technologies (CISTI), pp. 1–4. IEEE.
85. Choi, K., Yun, M., Chae, K., & Kim, M. (2012). An enhanced key management using zigbee pro for wireless sensor networks. In The international conference on information network 2012, pp. 399–403. IEEE.
86. Bakhache, B., Ghazal, J. M., & El Assad, S. (2013). Improvement of the security of zigbee by a new chaotic algorithm. *IEEE Systems Journal, 8*(4), 1024–1033.
87. Sadikin, F., van Deursen, T., & Kumar, S. (2020). A zigbee intrusion detection system for iot using secure and efficient data collection. *Internet of Things, 12*, 100306.
88. Fard, M. A. H. B., Chouinard, J.-Y., & Lebel, B. (2020). Rogue device discrimination in zigbee networks using wavelet transform and autoencoders. *Annals of Telecommunications, 76*(1), 27–42.
89. Bate, K. O., Kumar, N., & Khatri, S. K. (2017). Framework for authentication and access control in iot. In 2017, 2nd International conference on telecommunication and networks (TELNET), pp. 1–6. IEEE.
90. Chacko, S., Job, M. D. et al. (2018). Security mechanisms and vulnerabilities in lpwan. In IOP Conf. Ser. Mater. Sci. Eng, vol. 396, no. 1.
91. Paper, W. (2017). Introducing sigfox features. In https://www.sigfox.com/en/whatsigfox/technology/id_security.
92. Coman, F. L., Malarski, K. M., Petersen, M. N., & Ruepp, S. (2019). Security issues in internet of things: Vulnerability analysis of lorawan, sigfox and nb-iot. In 2019, Global IoT Summit (GIoTS), pp. 1–6. IEEE.
93. Fujdiak, R., Blazek, P., Mikhaylov, K., Malina, L., Mlynek, P., Misurec, J., & Blazek, V. (2018). On track of sigfox confidentiality with end-to-end encryption. In Proceedings of the 13th international conference on availability, reliability and security, pp. 1–6.
94. Li, H., Kumar, V., Park, J.-M., & Yang, Y. (2020). Cumulative message authentication codes for resource-constrained networks. arXiv preprint arXiv:2001.05211.
95. Tahsien, S. M., Karimipour, H., & Spachos, P. (2020). Machine learning based solutions for security of internet of things (iot): A survey. *Journal of Network and Computer Applications, 161*, 102630.

Alessandra Rizzardi is Assistant Professor at University of Insubria (Varese),where she received BS/MS degree in Computer Science 110/110 cum laude in 2011/2013. In2016 she got Ph.D. in Computer Science and Computational Mathematics at the sameuniversity, under the guidance of Prof. Sabrina Sicari. Her research activity is on WSN andIoT security issues.

Sabrina Sicari is Associate Professor at University of Insubria (Varese). Shereceived degree in Electronical Engineering, 110/110 cum laude, from University of Catania,in 2002, where in 2006 she got Ph.D. in Computer and Telecommunications Engineering,followed by Prof. Aurelio La Corte. She is member of COMNET, IEEE IoT, ETT, ITLeditorial board. Her research activity security, privacy and trust in WSN, WMSN, IoT, anddistributed systems.

Alberto Coen-Porisini received Dr. Eng. degree and Ph.D. in ComputerEngineering from Politecnico di Milano in 1987 and 1992. He is Professor of SoftwareEngineering at Università degli Studi dell'Insubria since 2001, Dean of the School of Sciencefrom 2006 and Dean since 2012. His research regards specification/design of real-timesystems, privacy models and WSN.