IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 27, 2020, accepted June 17, 2020, date of publication June 22, 2020, date of current version July 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3003998

Message Queuing Telemetry Transport (MQTT) Security: A Cryptographic Smart Card Approach

EDUARDO BUETAS SANJUAN, ISMAEL ABAD CARDIEL, JOSE A. CERRADA, AND CARLOS CERRADA

Department of Software and Systems Engineering, Universidad Nacional de Educación a Distancia (UNED), 28040 Madrid, Spain

Corresponding author: Eduardo Buetas Sanjuan (eduardo@buetassanjuan.name)

This work was supported in part by the Spanish Ministry of Science, Innovation, and Universities, under Project DPI2016-77677-P and Project DPI2017-84259-C2-2-R, and in part by the Community of Madrid under Grant RoboCity2030-DIH-CM P2018/NMT.

ABSTRACT The Message Queuing Telemetry Transport (MQTT) protocol is one of the most extended protocols on the Internet of Things (IoT). However, this protocol does not implement a strong security scheme by default, which does not allow a secure authentication mechanism between participants in the communication. Furthermore, we cannot trust the confidentiality and integrity of data. Lightweight IoT devices send more and more sensible data in areas of Smart Building, Smart City, Smart House, Smart Car, Connected Car, Health Care, Smart Retail, Industrial IoT (IIoT), etc. This makes the security challenges in the protocols used in the IoT particularly important. The standard of MQTT protocol strongly recommends implement it over Transport Layer Security (TLS) instead of plain TCP. Nonetheless, this option is not possible in most lightweight devices that make up the IoT ecosystem. Quite often, the constrained resources of IoT devices prevent the use of secure asymmetric cryptography algorithms implemented by themselves. In this article, we propose making a security schema in MQTT protocol using Cryptographic Smart Cards, for both challenges, the authentication schema and the trusted data confidentiality and data integrity. We carry out this security schema without modifying the standard protocol messages. And finally, we present a time results experiment using an example implementation model with JavaCard library.

INDEX TERMS Internet of Things (IoT), javacard, message queuing telemetry transport (MQTT), mutual authentication, smart card.

I. INTRODUCTION

The Internet of Things (IoT) is an ecosystem that provides the possibility of communications on the Internet to countless devices of very different types: environment sensors [1], vehicles [2], remotely controlled actuators [3], home appliances [4], health care sensors [5], industrial devices (IIoT) [6], etc. It is expected that by the end of 2022 will be 20.4 billions of IoT devices connected [7]. This new ecosystem raises new challenges in the security of its communications [8].

One of the most appropriate communications protocols for the IoT is the MQTT protocol, due to its capacity for easy implementation on lightweight, cheap, low-power, and low memory devices [9].

MQTT protocol was designed by IBM and in 2013 was standardized by OASIS (Open Architecture System). It has been approved as ISO standard, called ISO/IEC 20922 from June 2016. The protocol continues the evolution including new functions and formalizing common capability options. The last published version is MQTT v5.0 from 2018 [10].

This protocol has a Publisher/Subscriber structure with a star topology, as we can see in Fig. 1. It is possible to create a tree topology including more than one broker in the system [11].

MQTT has three types of participants:

1) BROKER, is the centre of the star in MQTT protocol and it is in charge of the exchange of messages between the other participants. All other participants connect with it and only with it, so it is in charge too of the authentication of all participants in the network.
2) PUBLISHERs, are the elements that send data to the broker so that it sends this data to one or more subscribers that require it.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Maaz Rehan.

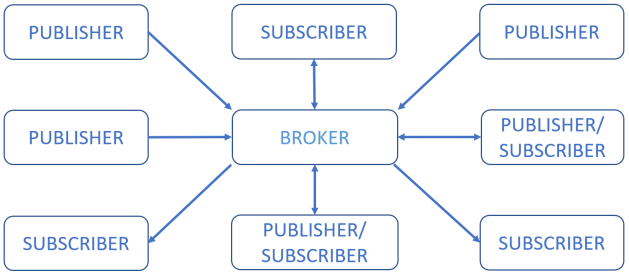

FIGURE 1. Publish/subscribe model of MQTT protocol.

3) SUBSCRIBERs, are the elements that receive data to the broker. The data they receive is the data sent by publishers.

The authentication by default in this protocol is based on a user-password scheme and the password is sent without encryption. In the connect message you can indicate an external alternative authentication method for this authentication (SCRAM [12], Kerberos [13], etc.). In this article, we create our own authenticate method based on Cryptographic Smart Card.

The MQTT has no encryption method for sending data, leaving this as the responsibility of the implementer. Also, the MQTT standard specification strongly recommends that the server implements the protocol over TLS [14].

The TLS protocol is a security standard *(RFC5246)* for the communications between two applications widely extended on the Internet. The TLS protocol consumes more than 100 KB of memory [15] and it requires a lot of resource consumption.

There are, in the current bibliography, many comprehensive studies about applying security frameworks to secure communications in the IoT, i.e. [16], [17], or [18]. And in particular, about MQTT communications, applying cryptographic schemes, [19], [20] or [21]. These papers use different schemes and cryptography primitives to secure MQTT communications, both to authenticate and to encrypt the payload. In these proposals, the IoT microcontrollers implement the security solution.

In this paper, we propose including a Cryptographic Smart Card: hardware secure, trustworthy, well tested and with low economic cost in the IoT devices to execute all necessary cryptographic functions, and a public key repository accessible for the broker (Fig. 2). Using these new elements, we present a new method for mutual authentication [22] in the MQTT protocol. Also, we define an encryption schema for encoding the data exchange between then clients and the broker, in both directions. And of course, without including modifications in the specification of the protocol messages.

The article is presented in four major sections before it is concluded. After this introductory section, our proposed security schema is presented, for the mutual authentication and encrypt data in exchanges. In the third section, we propose how to integrate our security schema in the protocol with

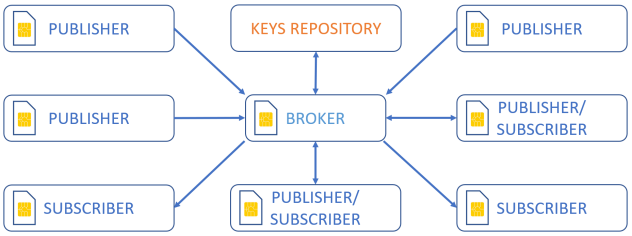

FIGURE 2. Publish/subscribe model of MQTT protocol with security schema based on cryptographic smart card.

no modifications in any message format and following the standardization of MQTT. The next section proposes how to implement the cryptographic requirements for our security schema with Cryptographic Smart Card, specifically with a JavaCard, with a time study of the process. And the last section includes conclusions details and describes the opened research trends about security in IoT communications.

II. GENERAL SCHEME OF SECURITY

In order to achieve the objectives presented in this article, it has been necessary to introduce three new elements in the system. These elements are enumerated below:

A. PUBLISHER/SUBSCRIBER CRYPTOGRAPHIC SMART CARD

The publishers and the subscribers MQTT must have a Cryptographic Smart Card that must be communicated with a microcontroller that manages the communication with the broker.

B. BROKER CRYPTOGRAPHIC SYSTEM

The broker must have a cryptographic system, either a Cryptographic Smart Card or an HSM (Hardware Security Manager) system [23].

The broker system must complete several cryptographic functions for all messages sent between publishers and subscribers. If the number of clients and the frequency of messages produces messages overlaps, the system should execute several cryptographic operations in parallel with high speed. So, for these situations, we suggest using an HSM and if it was necessary, using a balanced system for the broker processor.

C. PUBLIC KEY REPOSITORY

The system must have a public key repository accessible, through a secure protocol, by the broker.

The implementer of this system can select the asymmetric cryptography algorithm (RSA_NOPAD, RSA_PKCS1, ECC, etc.) [24], that he prefers for authentication proposes and the block cipher algorithm (AES, DES, TEA, NOEKEON,etc.) [25] for payload encryption.

The key pair (public and private) to make the authentication and the random numbers used by the devices in the process, must be generated directly inside of the cryptographic devices. The private key never leaves the device [26], the random numbers only leave the device encrypted, and a secure

random number generator produces its. In this way, we avoid that this data will be disclosed by an attacker using Man-in-the-Middle techniques, defined by NIST [27].

All public keys are published in the public key repository of the system, each public key is associated with a unique identifier (*UID*). That identifies a unique element in the system. Also, the Cryptographic Smart Card of the publishers and subscribers when they generate its owner pair, integrates the public key of the brokers, before its inclusion in the system. Thusly, the Cryptographic Smart Card of each publisher or subscriber includes its private key and public key, and the broker public key to which it should connect.

It is necessary that the messages exchange between participants ensure that each message is received once, and only once, by the intended recipient, in order to apply the security scheme proposed in this paper and to guaranty the safety in the communications. So it is required to use MQTT protocol with level QoS (Quality of Service) equal two [28].

In this section and beyond we use the following notation (Table. 1) to refer to the processes of data exchange between the participants.

TABLE 1. Notation.

Notation	Description
UID	The unique identifier of a participant
$Cpr_b(a)$	Encryption of data "a" with private key of the broker
$Cpu_b(a)$	Encryption of data "a" with public key of the broker
$Cpr_{cx}(a)$	Encryption of data "a" with private key of the client x
$Cpu_{cx}(a)$	Encryption of data "a" with public key of the client x
$BCE_x(a)$	Encryption of data "a" with block cipher algorithm with key x
RN_x	Random number x
$A; B$	Data A concatenated with data B
$HexStr(a)$	Function that encodes a binary array (a) to its representation in hexadecimal numbers encoded in UTF-8 format

D. AUTHENTICATION

In this article, we propose a mutual authentication will be carried out in three steps (Fig. 3).

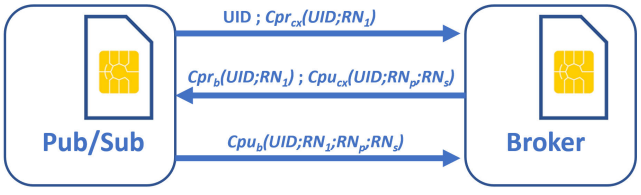

FIGURE 3. Mutual authentication process.

The first step of authentication is initiated by the client, it sends a connection message to the broker. In this message, the client sends its unique client identifier (*UID*) and concatenates the result of encryption with its private key of its *UID* concatenated with a random number generated by its Cryptographic Smart Card (RN_1).

$$UID; Cpr_{cx}(UID; RN_1)$$

The broker receives this message and with *UID* obtains, from the public key repository, the client public key. With this public key, the broker decrypts the second part of the message. With data decrypt the broker compares *UID* that has received in the first part of the message with the *UID* receives encrypted in the second part of the message. With this comparison, the broker makes sure the client has encrypted the message with its private key.

The next step in the authentication scheme involves that the broker sends to the client a message composed of two components concatenated, in the first component the broker sends the client *UID* plus the RN_1, all encrypted with the broker private key. And in the second component send the client *UID* plus two random numbers (RN_p and RN_s) generated by the cryptographic broker device, all encrypted with the client public key that initiated the authentication.

$$Cpr_b(UID; RN_1); Cpu_{cx}(UID; RN_p; RN_s)$$

In this step, the client has already authenticated the broker because it receives his random initial number encrypted with the broker private key. This encryption can only be done by the broker since it is unique that knows its private key.

The last step in the authentication scheme is the transmission from the client to the broker with the public broker key, the concatenation of client *UID*, plus RN_1, RN_p and RN_s (these random numbers are only known by the client and the broker because they are generated inside the cryptographic system of the broker and has exited from the cryptographic system encrypted with the client public key).

$$Cpu_b(UID; RN_1; RN_p; RN_s)$$

Once the broker receives and decrypts the message, it must check the *UID*, RN_1, RN_p, and RN_s. If verification is correct, the broker finishes the mutual authentication correctly.

If for any reason the connection is reset, either by a problem with the authentication operations or for an out planned network shutdown, the system applies a timeout for accepting another connection for the client, to prevent DoS attack as proposed by [29].

E. PAYLOAD ENCRYPT

To perform the encryption of payload we use a symmetric block cipher algorithm because the encryption time is much lower than with an asymmetric algorithm [30].

In the process of sending a message with data from a publisher to the broker, the publisher sends the data (payload) together with its *UID* and encrypted with the Block Cipher selected using a random key, only used in one exchange of data, and only known by the broker and the publisher (RN_{pn}).

This solution avoids attacks for sending of repeat messages (replay-attack), already treated by M. Shuai *et al.* in [31], also to avoid the statistical attacks, like statistical disclosure attack (SDA) described for N. Emandoost *et al.* in [32], and to protect avoid attacks of Man-in-the-Middle.

In the first publish message we use the random number RN_p, as the encryption key, created in the authentication process. (Fig. 4).

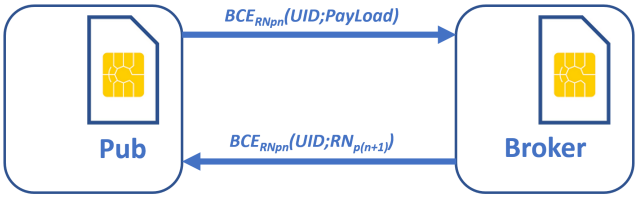

FIGURE 4. Publish from publisher to broker process.

The random number RN_p, in the authentication process, is sent in the second step, encrypted with the client public key (only the client can decrypt) and in the third step, is sent (concatenated with RN_1 and RN_s) encrypted with the broker public key (only the broker can decrypt), therefore is a random number known only by the client and the broker, so the broker can trust in this message.

In the acknowledgement of this message, the broker sends, to the publisher, the *UID* of the publisher concatenated with the other random number generated by the cryptographic system of the broker ($RN_{p(n+1)}$), all of this encrypted with block cipher selected with RN_{pn} as the key. This new random number, $RN_{p(n+1)}$, is the one that the publisher must use in the next payload sent as the encryption key. This new random number ($RN_{p(n+1)}$) is only known by the broker, who generates it, and the publisher, because it is sent encrypted with the block cipher. The encryption key is known for both: broker and publisher.

When the broker sends a message with payload to a subscriber, a similar cryptographic scheme is used (Fig. 5). The broker sends the message with subscriber *UID* concatenated with the payload, all encrypted with the block cipher selected and with a random key that only the broker and the subscriber known, so only the subscriber can decrypt the message, and it can trust in the message.

In the first message, this random key is RN_s, generated by the broker in the second step of the authentication process and sent to the subscriber encrypt with its public key, so only the broker, who generated it, and only the subscriber can decrypt the message with its private key. So the subscriber can trust in this message.

In the acknowledgement for this message, the subscriber sends, to the broker, its *UID* concatenated with a new random key, generated by its cryptographic smart card, $RN_{s(n+1)}$, all encrypted with the actual random key RN_{sn}. This new random key $RN_{s(n+1)}$, is only known by the subscriber, who generated it, and by the broker. The broker can decrypt the message

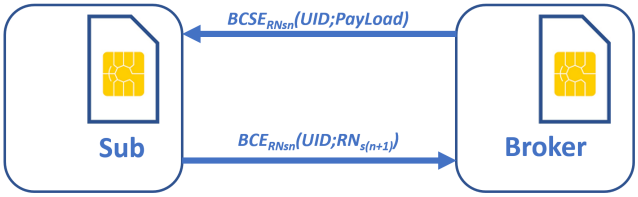

FIGURE 5. Publish from broker to subscriber process.

with RN_{sn}, and $RN_{s(n+1)}$ is used in the next exchange as the encrypted key.

In both messages exchanges, publisher to broker and broker to the subscriber, if any *UID* verification with the payload fails, the participant that has checked the inconsistency will disconnect immediately…They should make a new connection and a new authentication process for a new exchange of messages between these two participants.

In all communication process, authentication, publish or subscriber, if the connection is reset when the communication is restarted a new authentication process is necessary for restart the exchange of messages.

III. PROTOCOL IMPLEMENTATION

This section proposes, following the standard MQTT v5.0 [10], to include in the exchange of messages, the components necessary to comply with security processes described in section two of this paper without modifying any standard message.

In this way, some broker implementation that meets this specification, like for example [33], could be able to work with the proposed mutual authentication process and encrypting payload messages, without protocol modification and only using the possibilities already the protocol incorporates.

We define random numbers (RN_x), that use in the exchange of messages between client and server and vice versa, with the same length that the length key use in block cipher encryption and unique client identification (*UID*) like a set of 8 characters encoded in UTF-8 format.

A. AUTHENTICATION

In the mutual authentication process, we follow the Enhanced Authentication method included in section 4.12 of the protocol specification.

Once established the network link between client and broker, the next action to take should be to send the message CONNECT (Fig. 6, standard filling components are shown in blue and specific filling components are shown in orange) from client to broker according to the conformance statement [MQTT-3.1.0-1] of the specification.

In the CONNECT message, we must specify the *Authentication Method* defined in CONNECT message like a string of characters.

We define this string like *"SCACAuth-"+Auth Algorithm+"−"+Auth Key Length+"−" + Block Cipher Algorithm + "−" + Block Cipher Algorithm Key Lenght*, in this way,

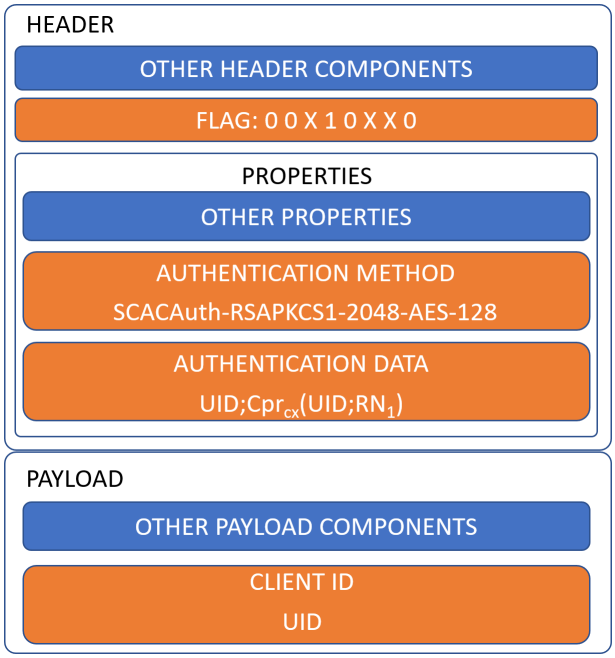

FIGURE 6. CONNECT message.

we can identify that the *Authentication Method* is the *Authentication Method* propose in this paper (*"SCACAuth"* = Smart Card Asymmetric Cryptography Authentication), the Auth Algorithm (*"RSANOPAD"*,*"RSAPKCS1"*,*"ECC"*, etc.), Auth Key Length used (*"512"*,*"1024"*, etc.), Block Cipher Algorithm (*"AES"*,*"DES"*,*"3DES"*, etc.) and Block Cipher Algorithm Key Length (*"64"*,*"128"*,*"256"*, etc.).

For example, if we want to realize authentication with RSA [34] algorithm with PKCS1 [35] padding schema with a key length of 2048 bits and AES [36] with 128 bits key length, we must send like an authentication method *"SCACAuth-RSAPKCS1-2048-AES-128"*.

In order to send characters strings in data exchange over MQTT, we must encode these characters strings following the encoded format presents in section 1.5.4 of the protocol specification (Fig. 7), encoding first in two bytes the length of the string and second, in the third byte, the string characters in UTF-8 format.

Bit	7	6	5	4	3	2	1	0
byte 1	String length MSB							
byte 2	String length LSB							
byte 3	UTF-8 encoded character data, if length > 0.							

FIGURE 7. String UTF-8 definition in MQTT specification.

In the *Authentication Data* field, we send, in binary format, the unique client identifier (*UID*) without encrypting plus the *UID* and a random number generated by the cryptographic smart card of the client (RN_1), both encrypted with its private key.

In *Connect Flags* field (Fig. 8), we must write 0 in *User Name* and *Password* flags, because in the *SCACAuth* method

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Start	Reserved
byte 8	0	0	X	1	0	X	X	0

FIGURE 8. Flag connect message.

send neither, furthermore, in the flag *Will QoS* must be used 0×10 (QoS = 2).

Also in the first field of the payload of CONNECT message, we must send the Client-ID [MQTT-3.1.3-3], in our case we use *UID* like Client-ID.

For the rest of the data in CONNECT message, we should send data as if it was a standard MQTT message.

When this CONNECT message is received by the broker, it must decrypt the second part of the *Authentication Data* and must compare *UID* in payload, *UID* not encrypted in *Authentication Data* and *UID* encrypted in *Authentication Data*.

If this comparison or the decryption process is not successfully or some of the flags are not compatible with the *SCACAuth* authentication method, the broker must send a CONNACK message, with *Reason Code* 0×87, Not Authorized (Fig. 9), and disconnects the link with the client after this sending it.

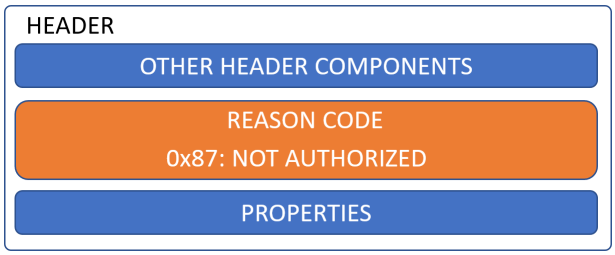

FIGURE 9. CONNACK message reason code 0×87.

If instead, all verification is successful, the broker must continue with mutual authentication process sending an *AUTH* message, with *Reason Code* 0×18, Continue Authentication (Fig. 10).

In the message properties, we must include the *Authentication Method*, this method must be equal that the method received in CONNECT message.

Also, we must include in the property *Authentication Data*: first, the client *UID* plus the RN_1, both encrypted with the broker private key, and second the client *UID* plus two random numbers generated by the cryptographic system of the broker (RN_p and RN_s) encrypted with the client public key.

Once the client has received this message, it must realize four checks:

1) The *Authentication Method* is the same that it sent in CONNECT message.
2) It is able to decrypt the first part of *Authentication Data* with the broker public key, and the second part with its private key.

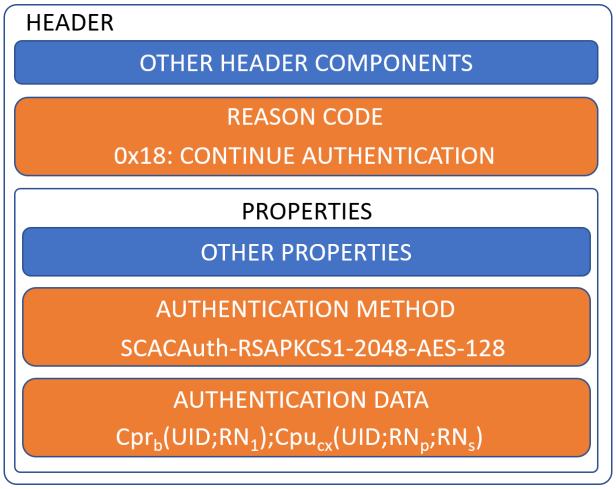

FIGURE 10. AUTH message reason code 0 × 18 authentication process step 2.

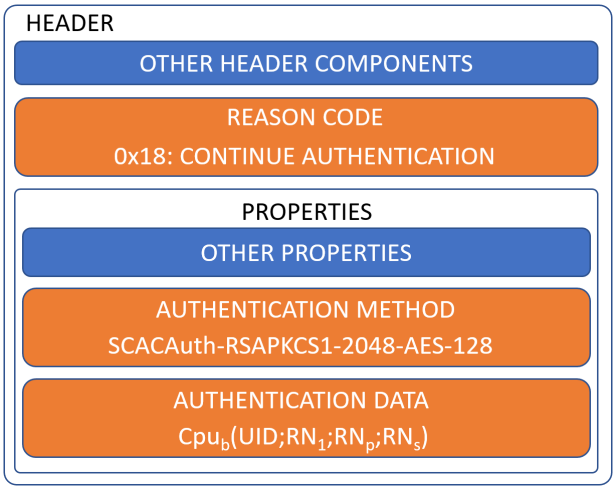

FIGURE 12. AUTH message reason code 0 × 18 authentication process step 3.

3) The *UID* sent in both parts of *Authentication Data* is its *UID*.
4) The RN_1 sent in the first part of *Authentication Data* is the same that it sent in the CONNECT message.

If some of these checks are not successfully, the client must send a DISCONNECT message, with *Reason Code* 0 × 80, Unspecified Error (Fig. 11). We use this *Reason Code* because the *Reason Code* 0 × 87, Not Authorized is reserved for disconnection by the broker according to the protocol specification. And in the *Reason String* property of DISCONNECT message, we must write the data ''Auth Error'', encoded as a characters string according to section 1.5.4 of the MQTT specification.

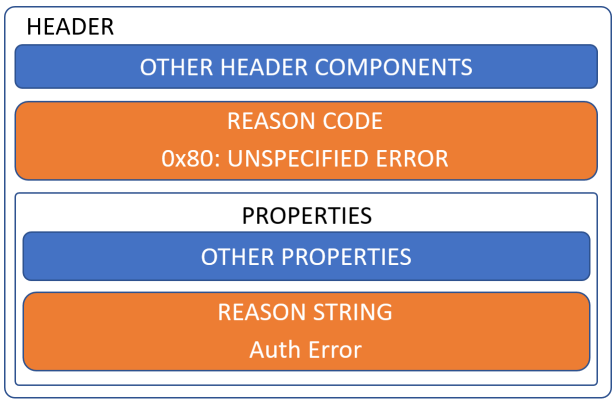

FIGURE 11. DISCONNECT message Auth Error.

On the other hand, if all of the checks are successful, the client must send an *AUTH* message with its *Reason Code* 0 × 18, Continue Authentication (Fig. 12). In this message, the client must send the same *Authentication Method* that in CONNECT message. And the Authentication Data must include its *UID* plus all the random numbers created in the authentication process (RN_1, RN_p and RN_s), all of these encrypted with the broker public key.

When the broker has received this message, it must decrypt the data with its private key, and to confirm, that received data are *UID*, RN_1, RN_p and RN_s.

If everything is correct the broker must send a CONNACK message, with *Reason Code* 0 × 00, Success (Fig. 13), and the mutual authentication process will be finished successfully. And it can thus start the exchange of messages with payload.

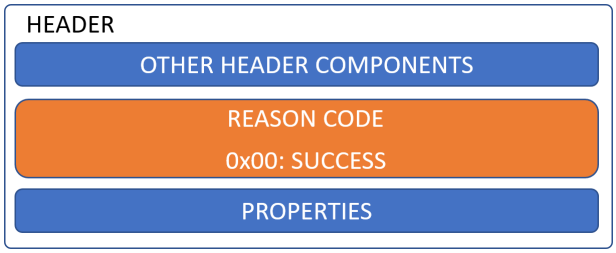

FIGURE 13. CONNACK reason code 0 × 00.

But, if the broker cannot decrypt the *Authentication Data*, or the contained data are not correct, the broker must send a CONNACK message with *Reason Code* 0 × 87, Not Authorized (Fig. 8), and after it must close the network link with the client.

B. PUBLICATION

The message exchange, for publishing with QoS = 2, either by from the publisher to the broker or, from the broker to the subscriber, is formed by the messages, PUBREC, PUBREL, and PUBCOMP, as we can see in the messages flow in Fig. 14.

The first message sent by the sender, the PUBLISH message is generated in the same form that a standard MQTT message except for its payload, QoS flags should be equal 2 (Fig. 15).

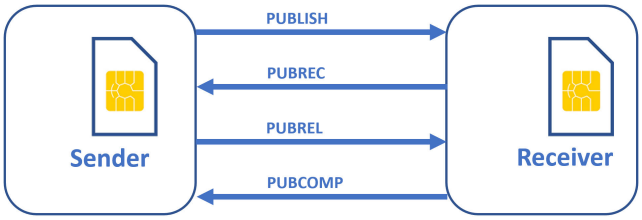

FIGURE 14. Publish exchange with QoS = 2.

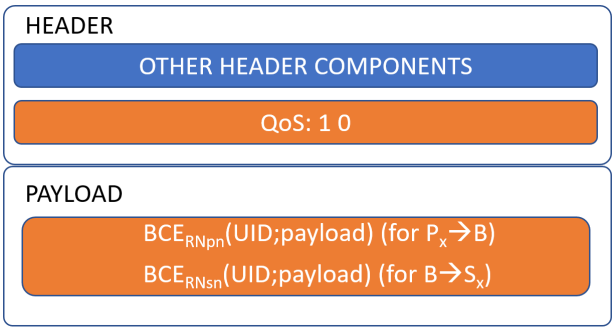

FIGURE 15. PUBLISH message.

The payload is composed with the *UID* of the publisher concatenated with the useful data that the publisher sends to the broker, and this is encrypted with the block cipher algorithm using a random number as the key.

In the first message we have two options. First, the random number generated in the mutual authentication process is used as the key, RN_p, for the first message from the publisher to the broker. Or second, the RN_s for the first message from the broker to the subscriber.

When the receiver receives this message, it must decrypt the payload with the current random number as the key and verify that the *UID* is correct. If it is correct, it trusts in the data sends in the payload.

If the receiver cannot decrypt the message or the *UID* is not correct, the receiver must send a DISCONNECT message with its *Reason Code* 0×80, Unspecified Error (Fig. 16) and including ''Crypt Error'' in the *Reason String*. It disconnects the link after sending the message.

If all is correct, the receiver must send a PUBREC message, in the same way as with a standard MQTT exchange of messages, with *Reason Code* 0×00, Success (Fig. 17). Any other error not related to the payload encryption must be answered as if it was a standard MQTT message.

Once the sender receives the PUBREC message sent from the receiver, following the publishing process with QoS = 2, it answers the PUBREL message.

In *User Property* of PUBREL message, the sender includes the *UID* sent in the PUBLISH message concatenated with a new random number, this new random number will be utilized in the next message exchange, $RN_{p(n+1)}$ if the sender is a publisher or $RN_{s(n+1)}$ if the sender is the broker. All of these encrypts with the block cipher selected and the current random number as the key.

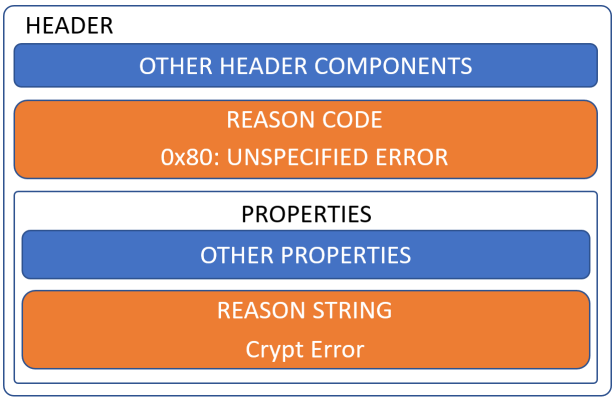

FIGURE 16. DISCONNECT message crypt error.

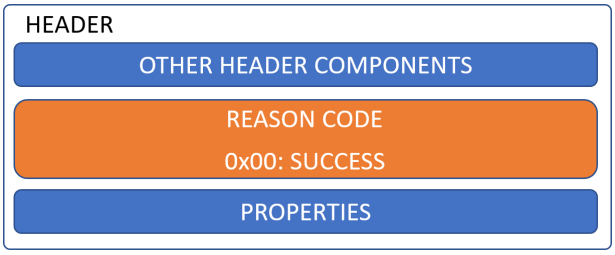

FIGURE 17. PUPREC message reason code 0×00.

Since the protocol definition establishes that the *User Property* of PUBREL message must contain a pair of character strings, in the first string of the pair is transmitted the string ''Ns'' in the case of the sender is the broker or ''Np'' if the sender is a publisher. The second string is composed by the set of hexadecimal numbers in UTF-8 encoding that represents the result of the encryption of the concatenating of *UID* with the new random number (Fig. 18).

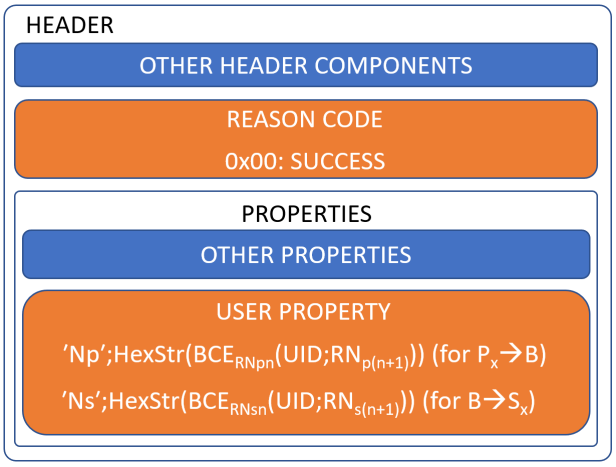

FIGURE 18. PUPREL message.

All other PUBREL message behaviours will be performed, as if it was a standard PUBREL message.

The receiver, once it receives the PUBREL message, checks if in the first string of the *User Property* is ''Np'' in the case of an exchange message from publisher to broker or ''Ns'' if the receiver is a subscriber. After this, convert the second string to an array of bytes and decrypts it with the current random number as the key.

Once the decryption has been carried out, the receiver must verify if the *UID* is correct, if it is, it must save the new random number for the next exchange and the process will continue like in a MQTT standard exchange, sending, for finish the process, a PUBCOMP message with *Reason Code* 0×00, Success (Fig. 19).

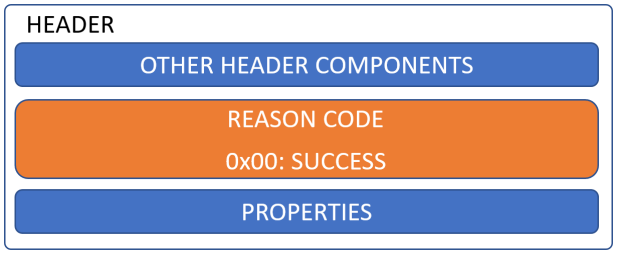

FIGURE 19. PUBCOMP message reason code 0×00.

If it cannot decrypt data, or the *UID* is not the expected, the receiver must send a DISCONNECT message with *Reason Code* 0×80, Unspecified Error and in the *Reason String* will send ''Crypt Error'' (Fig. 16), disconnecting the link of the network after sending the message.

All of the other behaviours of this messages exchange must be according to the MQTT standard protocol.

C. SUMMARY OF ATTACK MITIGATION

In this subsection, we expose the main attack mitigation techniques used with this security schema:

1) Spoofing Attack mitigation:
   a) In the Authentication Process: In step 1, the broker receives *UID* and RN_1 encrypted with the client private key. The client can encrypt with the correct *UID* because he is the only one who knows its private key. After this, in step 2, the client receives encrypted *UID* and RN_1 with the broker private key. The broker can encrypt because he is the only one who knows its private key.
   b) In the Exchange data process: Client *UID* and data are encrypted with a key that only the sender and the receiver know. The receiver checks the *UID* after decrypt. This key is different for each client and change for each message exchange.
2) Man-in-the-Middle Attack mitigation:
   a) In the Authentication Process: In Step 2, *UID*, RN_p and RN_s are encrypted with the client public key, and only the client that knows its private key can decrypt data. And in Step 3, *UID*, RN_1, RN_s and RN_p are encrypted with broker public key, so the broker is the only one that can decrypt this message.
   b) In the Exchange data process: Client *UID* and data are encrypted with a key that only the sender and the receiver can know. The receiver checks the *UID* after decrypting it. This key is different for each client and change for each message exchange.
3) Reply-Attack mitigation:
   a) In the Authentication Process: In this process, all *RN* values are different and randoms for each connection. In step 2 and 3, with the client or the broker check, the validator downs the connection if an attacker tries to send a reply message with RN values for other trying to connect.
   b) In the Exchange data process: The cryptographic key changes each data transaction. So, in the message check, if an attacker sends a reply attack, the receiver discards the data and downs the connection.
4) Statical Disclosure mitigation:
   a) In the Authentication Process: Random number used in the authentication process is different from each attempt.
   b) In the Exchange data process: All messages exchanges use a new random cryptographic key, so all the messages are different.
5) Denial of Service mitigation:
   a) Against Broker: It is necessary to set a timeout to avoid connections after an authentication failure to prevent a DoS with CONNECT messages.
   b) Against Publisher/Subscribers: These elements never accept new connections from other devices, they always start by itself the connection.

IV. SMART CARD CRYPTOGRAPHY IMPLEMENTATION

Smart cards are present in multiple processes in our daily lives, in order to identify ourselves digitally, banking processes, mobile communications, digital signatures, etc. In this article we propose to use these devices, and their cryptographic capabilities, to carry out encryption necessary for the security of communications in the IoT.

We have used a Java Card to solve the Smart Card Cryptography. This kind of Smart Card executes a Java environment, Java Card Virtual Machine (JCVM), that is defined by Oracle in [37], this technology allows making cross-manufacturer applications, denominated applets, in order to abstract the software development and the final device where the applet is executed.

For this implementation, we have created a new applet (*SCACAuth_Applets*) and make the test with this applet in a NXP J3H145 card (with secure smart card controller P60D144 [38]), that supports version 3.0.4 of the java card specification.

In order to calculate the time to perform the operations (Table. 2), we measure the time from sending the command to Java Card until it receives its response.

TABLE 2. Time spend for cryptographic executions.

Process	Avg. (ms)	Max. (ms)	Min. (ms)	sd(σ) (ms)
Create Key Pair and initialize UID and Broker Public Key	8906.0	23068.0	3507.0	4225.30
Cipher Init	349.19	410.0	382.0	7.70
Authentication	1265.31	1279.0	1250.0	5.85
Send Message Publish to Broker	147.58	151.0	146.0	1.09
Send Message Broker to Subscriber	190.66	193.0	189.0	5.38

We carry out the process implementation, to check the performance of the SCACAuth scheme with RSA with PKCS1 as an asymmetric cryptography algorithm and padding scheme with key length 2048 and AES with key length 128 as block cipher algorithm. In order to create random numbers, we use the ALG_SECURE_RANDOM algorithm implemented in the javacard.security.RandomData library.

All code, both the Java Card Applet code and the test application, are accessible in the repository https://github.com/EBuetas78/MQTT-SCACAuth.

A. CREATE KEY PAIR AND INITIALIZE UID AND BROKER PUBLIC KEY

This process is executed once before installing the device in the system. For executing these actions we create tree functions in the Java Card applet.

1) `Put_UID`

Receives as a parameter the *UID* of the device and save it in EEPROM memory.

2) `Put_Broker_Public_Key`

Receives as parameters the *exponent* and *module* of the broker public key, and with this data generate a public key and store this in EEPROM memory.

3) `Create_Pair`

Creates the Public-Private Pair of the device and store this pair in EEPROM memory and return in the response the *module* and the *exponent* of the public key of the pair to allow store it in the Public Key Repository of the system.

This process takes an irregular total time because the generated RSA pair in the Smart Card spend an irregular value of time, while the other functions quite less time consumers (Fig. 20).

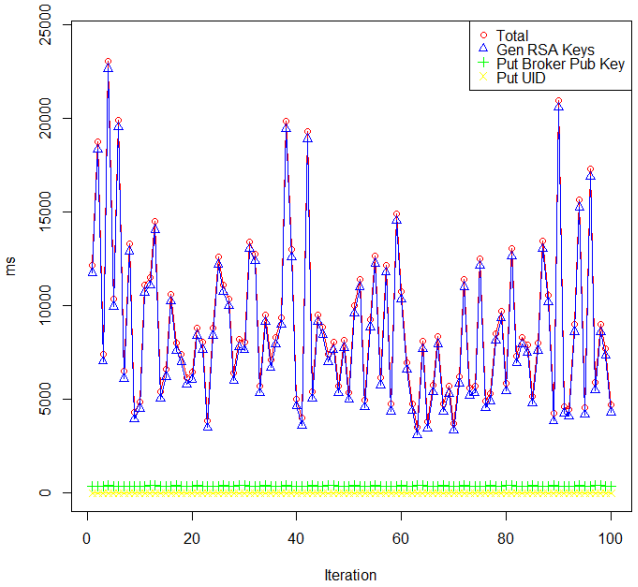

FIGURE 20. Times of create key pair and initialize UID and Broker public key process.

B. CIPHER INIT

After we have finished the creation pair process we need to initiate the ciphers functions in Smart Card, this process is needed each time that we power on the smart card and implement only with one function CipherInit (Fig. 21), that initializes all function ciphers used in the applet and reserves RAM memory for the next process.

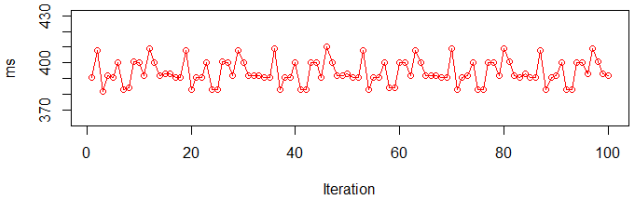

FIGURE 21. Times cipher inits process.

C. AUTHENTICATION

This process is executed each time that is initialized the communication between the broker and the client. We create three functions for this process, one for each step of the authentication.

1) `Create_Step1`

This function generates the random number RN_1. It concatenates the *UID* stores in EEPROM with the generated random number, and encrypt all of them with RSA-PKCS1 algorithm using the private key of the device. It returns the encrypted message.

2) `Check_Step2`

It receives as a parameter the data sending by the broker like an *Authentication Data* in the *Auth* message, decrypts both

part, the first with public key of the broker, and the second with its private key, and checks *UID* in both part and RN_1 in the first part. If everything is ok, it returns ok and stores in RAM memory, the RN_p and RN_s random numbers. if there is some error it returns fail.

3) `Create_Step3`

It creates an encryption with public key of the broker of the concatenated *UID*, RN_1, RN_p and RN_s and returns this data.

We can see the time process over the iteration in Fig. 22

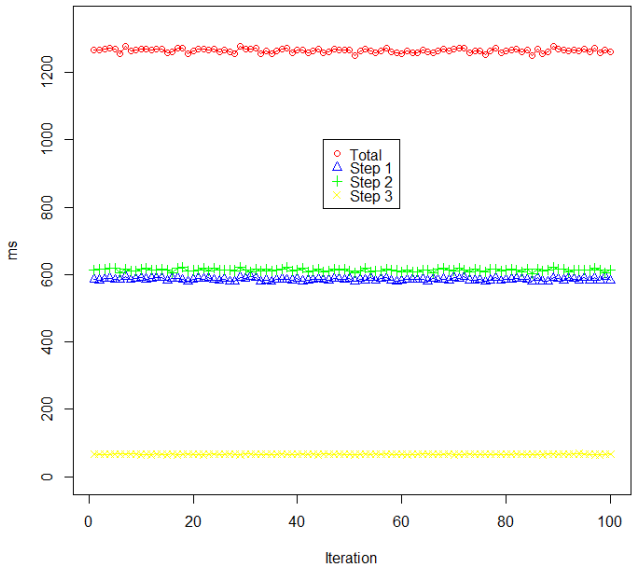

FIGURE 22. Times of authentication process.

D. SEND MESSAGE PUBLISH TO BROKER

This process is executed every time a publisher sends a message to the broker, in this test we use a useful data length of 50 bytes to communicate (Fig.23).

We use two functions for this process.

1) `Create_Publish_Message`

It receives the useful data to transmit. First, it adds to the useful data two bytes with its length to prepare the encryption process with a block cipher that needs to add padding bytes to calculate the length of the data to encrypt divisible for key length. So, it is necessary to add this length to the useful data in order to provide information to the later decrypting message process in the broker

Concatenates *UID* with this length and data and encrypts it with the RN_{pn} stored in the RAM memory as the key. Once this encryption is done, it sends encrypted data as function response.

2) `Check_PubRel`

It receives as a parameter the data sending by the broker in the PUBREL message, it decrypts the message with RN_{pn} stored in RAM as the key and checks that eight first characters of the decrypted message are the same that *UID* stored in EEPROM,

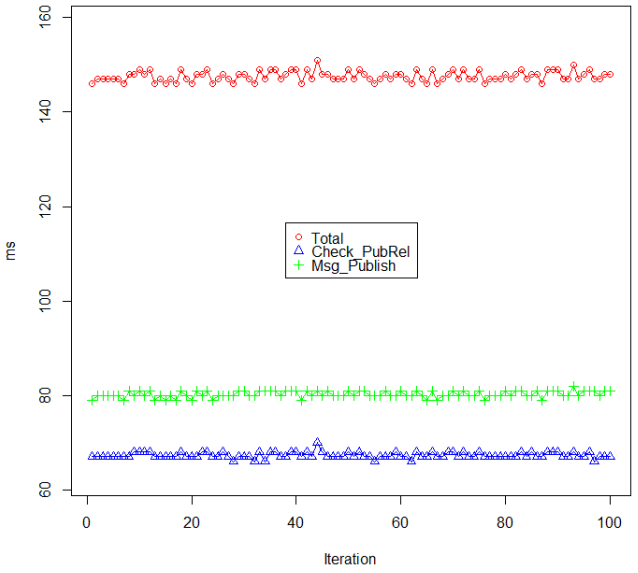

FIGURE 23. Times of publishing a message from publisher to broker process.

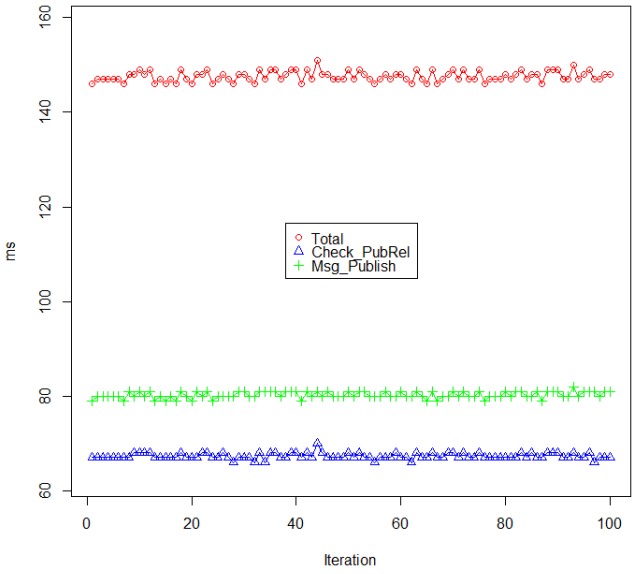

FIGURE 24. Times of publishing a message from broker to subscriber process.

if it is true, it takes the sixteen next characters, stores in RAM this content as the new RN_{pn} and response successful, if the check of *UID* is not ok, response with a fail code.

E. SEND MESSAGE BROKER TO SUBSCRIBER

This process is executed every time a broker sends a message to a subscriber, in this test we use a useful data length of 50 bytes to communicate (Fig.24).

We use two functions for this process.

1) `Read_Payload`

It receives the data included in the payload of PUBLISH message sends by the Broker. It decrypts the message with

RN_{sn} stored in RAM as the key and checks that eight first characters of the decrypted message are the same that *UID* stored in EEPROM. If the check is not correct returns a fail as the response.

If the *UID* check is correct, it reads the next two bytes. These two bytes are the length of the payload. It reads this length and returns as the response the length and the data of the payload.

2) `Create_PubRel`

This function creates a new random number $RN_{s(n+1)}$, and concatenates *UID*, it stores in EEPROM. Also, it encrypts these data with RN_{sn} as the key, and after it replaces RN_{sn} for $RN_{s(n+1)}$ in RAM memory. So this new random number is the new key for the decrypt process in the next `Read_Payload` execution.

V. CONCLUSIONS AND FUTURE WORKS

A. CONCLUSIONS

In this article, we have exposed an authentication and encryption schema to secure MQTT communications, and in what manner we can develop this schema in the MQTT protocol without modifying the MQTT standard specification. This new schema is carried out appending a Cryptographic Smart Card for each publisher and for every subscriber, and another cryptographic device, or a Cryptographic Smart Card or an HSM device, to the broker. With these devices, we make all cryptographic process to complete the process of the security scheme. Additionally, we proposed a JavaCard implementation of this schema and we have included an execution time study for this implementation.

The time study results shows that this schema is convenient for network configuration with devices maintaining the connection along the time as industrial networks: production lines, manufacturing automation,... and the schema works worst if the devices need to authenticate in short intervals.

B. FUTURE TRENDS

This security schema can execute over other cryptographic devices, current or future. In our implementation, we have used cryptographic smart cards, but future research can work to find other devices in order to improve the authentication time executing asymmetric cryptography faster.

In this article, we have used the standard JavaCard which restrains to work directly with low-level cryptographic primitives of the Smart Card. There are some researches using libraries which allows to work directly with cryptographic microcontrollers, like *JCMathLib* [39], this is another possibility of future research.

Another research option to improve the system performance is to change the used algorithms, especially the asymmetric algorithm to another option as ECC that is faster than RSA [40], [41].

REFERENCES

[1] D. Sehrawat and N. S. Gill, "Smart sensors: Analysis of different types of IoT sensors," in *Proc. 3rd Int. Conf. Trends Electron. Informat. (ICOEI)*, Tirunelveli, India, Apr. 2019, pp. 523–528.

[2] B. V. Philip, T. Alpcan, J. Jin, and M. Palaniswami, "Distributed real-time IoT for autonomous vehicles," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 1131–1140, Feb. 2019.

[3] S. Nuratch, "Applying the MQTT protocol on embedded system for smart sensors/actuators and IoT applications," in *Proc. 15th Int. Conf. Electr. Eng./Electron., Comput., Telecommun. Inf. Technol. (ECTI-CON)*, Chiang Rai, Thailand, Jul. 2018, pp. 628–631.

[4] A. Cornel- Cristian, T. Gabriel, M. Arhip-Calin, and A. Zamfirescu, "Smart home automation with MQTT," in *Proc. 54th Int. Univ. Power Eng. Conf. (UPEC)*, Bucharest, Romania, Sep. 2019, pp. 1–5.

[5] D. Yi, F. Binwen, K. Xiaoming, and M. Qianqian, "Design and implementation of mobile health monitoring system based on MQTT protocol," in *Proc. IEEE Adv. Inf. Manage., Communicates, Electron. Autom. Control Conf. (IMCEC)*, Xi'an, China, Oct. 2016, pp. 1679–1682.

[6] Z. Bi, L. Da Xu, and C. Wang, "Internet of Things for enterprise systems of modern manufacturing," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1537–1546, May 2014.

[7] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on IoT security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[8] C. Shouqi, L. Wanrong, C. Liling, H. Xin, and J. Zhiyong, "An improved authentication protocol using smart cards for the Internet of Things," *IEEE Access*, vol. 7, pp. 157284–157292, 2019.

[9] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 2015.

[10] (Jan. 2018). *MQTT 5.0. Specification, OASIS 2018*. [Online]. Available: http://doc.oasis-open.org/mqtt/v5.0/mqtt-v5.0.html

[11] E. Longo, A. Enrico Cesare Redondi, M. Cesana, A. Arcia-Moret, and P. Manzoni, "MQTT-ST: A spanning tree protocol for distributed MQTT brokers," 2019, *arXiv:1911.07622*. [Online]. Available: http://arxiv.org/abs/1911.07622

[12] S. Nafie, J. Robert, and A. Heuberger, "SCRAM: A novel approach for reliable ultra-low latency M2M applications," in *Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall)*, Chicago, IL, USA, Aug. 2018, pp. 1–5.

[13] Z. Tbatou, A. Asimi, Y. Asimi, and Y. Sadqi, "Kerberos v5: Vulnerabilities and perspectives," in *Proc. 3rd World Conf. Complex Syst. (WCCS)*, Marrakesh, Morocco, Nov. 2015, pp. 1–5.

[14] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, document RFC 5246, Aug. 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246

[15] R. R. Pahlevi, P. Sukarno, and B. Erfianto, "Implementation of event-based dynamic authentication on MQTT protocol," *J. Rekayasa Elektrika*, vol. 15, no. 2, pp. 125–133, Aug. 2019.

[16] L. Malina, G. Srivastava, P. Dzurenda, J. Hajny, and S. Ricci, "A privacy-enhancing framework for Internet of Things services," in *Network and System Security*. Cham, Switzerland: Springer, 2019, pp. 77–97.

[17] A. Roukounaki, S. Efremidis, J. Soldatos, J. Neises, T. Walloschke, and N. Kefalakis, "Scalable and configurable end-to-end collection and analysis of IoT security data : Towards end-to-end security in IoT systems," in *Proc. Global IoT Summit (GIoTS)*, Aarhus, Denmark, Jun. 2019, pp. 1–6, doi: 10.1109/GIOTS.2019.8766407.

[18] S. Sezer, "T1C: IoT security:—Threats, security challenges and IoT security research and technology trends," in *Proc. 31st IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Arlington, VA, USA, Sep. 2018, pp. 1–2, doi: 10.1109/SOCC.2018.8618571.

[19] L. Malina, G. Srivastava, P. Dzurenda, J. Hajny, and R. Fujdiak, "A secure publish/subscribe protocol for Internet of Things," in *Proc. 14th Int. Conf. Availability, Rel. Secur. (ARES)*. New York, NY, USA: ACM, 2019, pp. 1–17.

[20] S. P. Mathews and R. R. Gondkar, "Protocol recommendation for message encryption in MQTT," in *Proc. Int. Conf. Data Sci. Commun. (IconDSC)*, Bengaluru, India, Mar. 2019, pp. 1–5, doi: 10.1109/IconDSC.2019.8817043.

[21] E. Elemam, A. M. Bahaa-Eldin, N. H. Shaker, and M. A. Sobh, "A secure MQTT protocol, telemedicine IoT case study," in *Proc. 14th Int. Conf. Comput. Eng. Syst. (ICCES)*, Cairo, Egypt, Dec. 2019, pp. 99–105, doi: 10.1109/ICCES48960.2019.9068129.

[22] Sudhakar K, Srikanth S, and Sethuraman M, "Secured mutual authentication between two entities," in *Proc. IEEE 9th Int. Conf. Intell. Syst. Control (ISCO)*, Coimbatore, India, Jan. 2015, pp. 1–5.

[23] R. De Prisco, A. De Santis, and M. Mannetta, "Reducing costs in HSM-based data centers," *J. High Speed Netw.*, vol. 24, no. 4, pp. 363–373, 2018, doi: 10.3233/JHS-180600.
[24] M. N. B. Anwar, M. Hasan, M. Hasan, J. Loren, S. T. Hossain, "Comparative study of cryptography algorithms and its' applications," *Int. J. Comput. Netw. Commun. Secur.*, vol. 7, no. 5, pp. 141–184, 2018.
[25] G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, and C. Manifavas, "A review of lightweight block ciphers," *J. Cryptograph. Eng.*, vol. 8, no. 2, pp. 96–103, 2019, doi: 10.1007/s13389-017-0160-y.
[26] M. Bahadori, M. R. Mali, O. Sarbishei, M. Atarodi, and M. Sharifkhani, "A novel approach for secure and fast generation of RSA public and private keys on SmartCard," in *Proc. 8th IEEE Int. NEWCAS Conf.*, Montreal, QC, Canada, Jun. 2010, pp. 265–268.
[27] A. P. Grassi, E. M. Garcia, and L. J. Fenton, "Digital Identity Guidelines," NIST, Nat. Inst. Standards Technol. U.S. Dept. Commerce, Gaithersburg, MD, USA, NIST Special Publication 800-63-3, Jun. 2017, doi: 10.6028/NIST.SP.800-63-3.
[28] A. S. Sadeq, R. Hassan, S. S. Al-rawi, A. M. Jubair, and A. H. M. Aman, "A QoS approach for Internet of Things (Iot) environment using MQTT protocol," in *Proc. Int. Conf. Cybersecur. (ICoCSec)*, Negeri Sembilan, Malaysia, Sep. 2019, pp. 59–63.
[29] G. Potrino, F. de Rango, and A. F. Santamaria, "Modeling and evaluation of a new IoT security system for mitigating DoS attacks to the MQTT broker," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Marrakesh, Morocco, Apr. 2019, pp. 1–6, doi: 10.1109/WCNC.2019.8885553.
[30] S. K. Rao, D. Mahto, and D. A. Khan, "A survey on advanced encryption standard," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 3, pp. 711–724, 2017, doi: 10.21275/art20164149.
[31] M. Shuai, N. Yu, H. Wang, and L. Xiong, "Anonymous authentication scheme for smart home environment with provable security," *Comput. Secur.*, vol. 86, pp. 132–146, Sep. 2019.
[32] N. Emamdoost, M. S. Dousti, and R. Jalili, "Statistical disclosure: Improved, extended, and resisted," in *Proc. 6th Int. Conf. Emerg. Secur. Inf., Syst. Technol.*, 2012, pp. 119–125.
[33] D. L. de Oliveira, A. F. da S. Veloso, J. V. V. Sobral, R. A. L. Rabelo, J. J. P. C. Rodrigues, and P. Solic, "Performance evaluation of MQTT brokers in the Internet of Things for smart cities," in *Proc. 4th Int. Conf. Smart Sustain. Technol. (SpliTech)*, Split, Croatia, Jun. 2019, pp. 1–6.
[34] R. Rivest, A. Shamir, and L. Aldeman, "A method for obtaining digital signatures and public-key cryptosystems," *J. Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.
[35] B. Kaliski, J. Jonsson, and A. Rusch, *PKCS #1: RSA Cryptography Specifications Version 2.2*, document RFC 8017, Nov. 2016. [Online]. Available: https://tools.ietf.org/html/rfc8017
[36] T. Jamil, "The rijndael algorithm," *IEEE Potentials*, vol. 23, no. 2, pp. 36–38, Apr./May 2004.
[37] Oracle.com. (2020). *Java Card Overview*. Accessed: Mar. 2, 2020. [Online]. Available: https://www.oracle.com/java/technologies/java-card-tech.html
[38] NXP Semiconductors, SmartMX2 P60 Family, P60D080 and P60D144, Datasheet, Sep. 2010.
[39] V. Mavroudis and P. Svenda, "Towards low-level cryptographic primitives for JavaCards," 2018, *arXiv:1810.01662*. [Online]. Available: http://arxiv.org/abs/1810.01662
[40] F. Mallouli, A. Hellal, N. Sharief Saeed, and F. Abdulraheem Alzahrani, "A survey on cryptography: Comparative study between RSA vs ECC algorithms, and RSA vs el-gamal algorithms," in *Proc. 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)/5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Jun. 2019, pp. 173–176.
[41] D. Pharkkavi, "Time complexity analysis of RSA and ECC based security algorithms in cloud data," *Int. J. Adv. Res. Comput. Sci.*, vol. 9, no. 3, pp. 201–208, Jun. 2018. [Online]. Available: https://search.proquest.com/docview/2101252615?accountid=14609

EDUARDO BUETAS SANJUAN received the B.S. degree in telecommunications from the Escuela Universitaria Politécnica de Teruel, University of Zaragoza, Spain, in 2000, and the master's degree in software engineering and computer system engineering from the Universidad Nacional de Educación a Distancia (UNED), in 2018, where he is currently pursuing the Ph.D. degree in systems and control engineering. Since 2000, he has been working on different automation and industrial communications projects, mostly for automation manufacturers, such as General Motors, Volkswagen, and Seat, and Tier 1 suppliers such as CEFA, Faurecia, and FFT. He is interested in RFID systems, industrial communications, and automation systems.

ISMAEL ABAD CARDIEL received the Ph.D. degree in software engineering and computer systems from the Universidad Nacional de Educación a Distancia (UNED), in 2016. He belongs to the Software Engineering and Computer Systems Research Group. This research group has been involved in software engineering, robotics, and RFID research projects, since 2004. He is currently an Associate Professor with the UNED. His current research interests include ubiquitous computing with hybrid systems (vision and RFID) and new software architectures for the industrial IoT.

JOSE A. CERRADA received the M.S. degree in industrial engineering and the Ph.D. degree from the Polytechnic University of Madrid, Spain, in 1979 and 1983, respectively. He is currently a Full Professor and has been the Head of the Department of Systems and Software Engineering, Universidad Nacional de Educación a Distancia (UNED), since 2015. He is teaching in the area of software engineering, specifically in the domains of software process management and improvement. He has participated in more than 30 research projects (European and Spanish public administration). His current research interests include RFID technologies and software engineering.

CARLOS CERRADA received the M.S. and Ph.D. degrees in industrial engineering from the Polytechnic University of Madrid, Spain, in 1983 and 1987, respectively. From 1989 to 1990, he was a Fulbright Scholar with the Robotics Institute, Carnegie Mellon University, PA, USA. Since 2006, he has been a Full Professor with the Department of Software Engineering and Computer Systems, Universidad Nacional de Educación a Distancia (UNED), Spain. His research interests include software engineering and robotics.