SN Computer Science (2020) 1:24
https://doi.org/10.1007/s42979-019-0022-z

ORIGINAL RESEARCH

Access Controls for IoT Networks

Alban Gabillon[1] · **Romane Gallier**[1] · **Emmanuel Bruno**[2]

Received: 16 April 2019 / Accepted: 6 September 2019 / Published online: 20 September 2019

Abstract
The message queuing telemetry transport (MQTT) protocol is becoming the main protocol for the internet of things (IoT). In this paper, we define a highly expressive attribute-based access control (ABAC) security model for the MQTT protocol. Our model allows us to regulate not only publications and subscriptions, but also distribution of messages to subscribers. We can express various types of contextual security rules (temporal security rules, content-based security rules, rules based on the frequency of events, etc.).

Keywords Security policy · MQTT · ABAC · IoT · First-order logic

Introduction

The Message Queuing Telemetry Transport (MQTT) protocol is becoming the main protocol behind pub-sub networks for the Internet of Things, that is, in networks implementing the publication–subscription paradigm. The MQTT protocol is an ISO standard (ISO/IEC PRF 20922) [1] and the 3.1 version became an OASIS specification in 2013 [2]. Basically, the MQTT protocol works as follows: publishers post messages to logical channels called topics; subscribers receive messages published to the topics to which they subscribed; and the MQTT broker routes' messages from publishers to subscribers.

The MQTT protocol supports very few security features. It includes a MQTT client identification mechanism and supports the basic login/password authentication scheme. Consequently, there have been several papers aiming at defining security solutions for the MQTT protocol or more generally for the pub-sub pattern. These papers address various issues like how to implement a security policy regulating publications and subscriptions [3–5], how to distribute the evaluation and the enforcement of the security policy at the edge of the IoT network [6, 7], how to distribute and synchronize the security policy between different pub-sub architectures [8], or how to protect the confidentiality of the messages from the broker or the pub-sub architecture itself [9, 10]. Although these issues are all very important, we noticed that none of these papers fully addressed the definition of a security model allowing to express security policies for regulating IoT messages. Some of the papers [4, 5] mention that they are using the ABAC (Attribute-Based Access Control) model [11] for expressing the security policy controlling publications and subscriptions. However, they do not go much into details and do not elaborate on the expressive power of the security policy. In this paper, we define a highly expressive ABAC model for regulating IoT messages in an MQTT network. We believe that the definition of such a security model (which does not contradict the solutions proposed by the aforementioned papers) has been missing in the literature related to security solutions for pub-sub architectures. Our model allows us to regulate not only publications or subscriptions, but also distribution of messages by the broker to subscribers. Our model supports positive and negative authorizations and allows us to express various types of context-based policies, including policies based on the frequency of events. This paper is an extension of two papers we previously published [12, 13]. In this paper, we give a

This article is part of the topical collection "Future Data and Security Engineering" guest edited by Tran Khanh Dang.

✉ Alban Gabillon
alban.gabillon@upf.pf

Romane Gallier
romane.gallier@doctorant.upf.pf

Emmanuel Bruno
emmanuel.bruno@univ-tln.fr

[1] Université de la Polynésie Française, Punaauia, BP 6570, 98702 Fa'a'ā, French Polynesia

[2] Université de Toulon, CNRS, LIS, UMR 7020, 83957 La Garde, France

more complete definition of our security model, including the security policy administration model. We also present a new version of our prototype of access control enforcement system. The prototype we sketched in [12, 13] was based on SWRL and OWL2. Although it showed the feasibility of our approach, it proved to be inefficient in term of performance. The prototype we present in this paper is based on RDF [14] and SHACL [15] and the experiments we conduct show that it is efficient and scalable. Finally, in this paper, we also update our review of the literature on security models for IoT networks.

The remainder of this paper is organized as follows: in "ABAC Model", we define our model. In "Security Administration Model", we present our security administration model. In "Prototype", we present our secure MQTT broker prototype based on our model and we study its performance. In "Related Works", we review related works before the conclusion in "Conclusion".

ABAC Model

Some papers [4, 5] mention that they are using the attribute-based access control (ABAC) model [11] for expressing the security policy controlling publications and subscriptions in a pub-sub network. However, these papers do not go much into details and do not elaborate on the expressive power of the security policy. Moreover, none of these papers address the security administration issue. Our aim in this paper is to define a security model which can be seen as a profile of the ABAC model for pub-sub networks based on MQTT. We first identify some requirements specific to IoT security policies. Then, we make some assumptions on the IoT network and on some security aspects that we shall not cover. Finally, we devise our model starting from the requirements we identified.

Requirements

- Our model should offer the possibility to regulate not only publications and subscriptions to topics but also distribution[1] of messages by the broker to subscribers. Controlling distribution of messages is essential to regulate the various flows of messages coming from the broker. Solely controlling subscriptions is too coarse grained to achieve that task.
- Our model should allow for various types of dynamic and contextual authorization rules, i.e., authorization rules whose outcome (permit ort deny) depend on some contextual conditions applying to the nodes, the messages (including the content of the messages) or the environment. In particular, authorization rules based on the frequency of events should be supported since controlling the rate at which a node may send or receive messages is important in many IoT applications.

Assumptions

- For the sake of simplicity, we assume a pub-sub architecture with only one MQTT broker. Since we focus on the expressive power of the security policy, we do not investigate issues like distributing and synchronizing the security policy between different bridged brokers or evaluating the security policy at the edge of the network [6–8].
- We assume the broker to be trusted, i.e., we do not investigate solutions to protect the confidentiality of the messages from the broker [9, 10].
- We do not investigate authentication techniques. We believe that standard authentication techniques can be used to authenticate both nodes and attributes.
- Finally, we assume that TLS/SSL is used at the transport layer between all nodes of the IoT network. Most existing MQTT servers support the use of TLS/SSL.

Language

We use first-order logic with equality to define our model, i.e., we define a logical language allowing us to represent nodes, attributes, events (such as publications, subscriptions, and message distribution), and authorization rules. Note, however, that the reader who is not familiar with logic should be able to understand the main principles of our model, since we translate in plain English each logical formula.

Although we define our own logical language, we wish to make it clear that this paper is not about a new logic-based policy language. To specify our model, we could use XACML [16] (but it would be unreadable by a human), or an existing logical language like SecPAL [17]. However, we prefer defining our own language, so that we can restrict ourselves to Horn clauses which can easily be read by a human and for which there exists efficient resolution methods.

Constants

Constants of our language are string expressions. They are node identifiers such as *sensor1*, *user1*, etc. or the special string broker referring to the MQTT broker.

[1] Note that in the MQTT protocol the distribution of messages by the broker is implemented by means of *publish* messages. From a security point of view, we prefer to make a clear distinction between the privilege to publish in a given topic (this privilege can be held by any node) and the privilege to deliver messages to subscribers (this privilege can be held only by the broker).

Table 1 Node predicates

Predicate	Meaning
node(*N*)	*N* is an IoT node
broker(*N*)	*N* is the broker
sensor(*N*)	*N* is a sensor
client(*N*)	*N* is a client

Topics are defined by path expressions (written as strings) such as *temperatures/sensor1*. Several topics can be referenced using wildcards # and +. For examples, *temperatures/#* addresses any topic having temperature as path root and *home/+/temperatures* addresses topics such as *home/room1/temperature*, *home/room2/temperature,* etc. See [2] for more details about the use of wildcards in MQTT topics.

Note that, to lighten the notations, we omit the quotation marks for the strings.

Variables

Variables are written in capitalized letters like in Prolog. Our language includes the anonymous variable _ which means anything. If variable *S* contains a string value, then we assume that this value can be referred to in a path expression. For example, if *S* contains the string *sensor1,* then *temperatures/S* represents the topic *temperatures/sensor1*.

In this paper, to distinguish variables from constants, we constrain ourselves to consider only constants written as strings of lowercase characters.

Predicates

Authorizations can be derived from a set of facts $\mathscr{F}$ and from a set of logical rules $\mathscr{R}$. Set $\mathscr{F}$ keeps track of registered nodes and events (publications, subscriptions and distributions), whereas set $\mathscr{R}$ records the nodes hierarchy (Table 1).

Set $\mathscr{F}$ includes instances from the following node predicates:

Registering a node creates an instance of one of these node predicates.

Set $\mathscr{R}$ includes the following rules:

$$node(N) \leftarrow broker(N) \tag{1}$$

Table 3 Matching predicate

Predicate	Meaning
$addresses(T, T')$	Topic *T* addresses topic *T'*

$$node(N) \leftarrow sensor(N) \tag{2}$$

$$node(N) \leftarrow client(N). \tag{3}$$

These three rules can be used to derivate that the broker or a sensor or a client is also a node. These rules define a roles hierarchy that could be expanded according to the needs of the application.

Set $\mathscr{F}$ also includes instances from the following event predicates (Table 2):

Publishing a message creates an instance of the *hasPublished/3* predicate. Subscribing to a topic creates an instance of the *hasSubscribed/3* predicate. Delivering a message creates an instance of the *hasDelivered/3* predicate. As we shall see in "Language", recording these events allows us in particular to express security rules controlling the frequency of publishing/delivering messages.

As we said previously, topics are path expressions possibly written with wildcards. Therefore, set $\mathscr{F}$ also includes instances from the following topic predicate (Table 3):

For example, fact *addresses*(*temperature*/∗, *temperature*/*sensor*1) belongs to $\mathscr{F}$. For the sake of simplicity, we do not give the logical rules allowing us to derive instances of the *addresses/2* predicate.

Functions

Functions of our language represent attributes. They are either,

- Functions applying to messages or
- Functions for evaluating temporal conditions or any other contextual conditions.

Lists of functions in Tables 4 and 5 are not exhaustive and can be extended depending on the needs.

Table 2 Event predicates

Predicate	Meaning
hasPublished(*N*, *T*, *D*)	At time *D*, node *N* has published a message in topic *T*
hasSubscribed(*N*, *T*, *D*)	At time *D*, node *N* has subscribed to topic *T*
hasDelivered(*T*, *N*, *D*)	At time *D*, the broker has delivered a message from topic *T* to node *N*

Table 4 Message attribute functions

Function	Purpose
length(M)	Returns the length of the message M
retained(M)	Returns true if the message M is retained, false else
value(M)	Returns the content of the message M
encoding(M)	Returns the character encoding of the message M
ciphered(M)	Returns true if the message M is encrypted[a], false else

[a]Encrypting a message means encrypting the payload of the MQTT packet transporting the message. This should not be confused with encrypting the whole communication between nodes at the transport layer by means of TLS/SSL

Table 5 Contextual functions

Function	Purpose
time()	Returns the current time
date()	Returns the current date
latency()	Returns the network's latency
bandwidth()	Returns the network's bandwidth

Table 6 Actions

Term	Action
publish(M, T, Q)	Publishing message M in topic T at QoS Q
subscribe(T, Q)	Subscribing to topic T at QoS Q
deliver(M, T, N)	Delivering message M from topic T to node N

Security Policy

Actions

We define the three compound terms to represent the following three actions:

Variables represent action parameters. Note that there is no QoS parameter for the deliver operation. This is because the QoS used by the broker to deliver a message to node N is the QoS chosen by node N when it subscribed to topic T. This means that if, in our security policy, we need to restrict the QoS used by the broker to deliver messages, then it should be done during the subscription step.

Contextual Authorization Rules

We consider positive authorizations and negative authorizations represented by the two following predicates:

Variable A contains any of the three compound terms of Table 6. Note that if A is a deliver action, then we assume that N cannot be different from broker (Table 7).

Table 7 Authorizations

Predicate	Meaning
allow(N, A)	Node N is allowed to perform action A
deny(N, A)	Node N is denied performing action A

The security policy $\mathscr{P}$ regulates publish, subscribe, and deliver operations. It consists of a set of authorization rules. Any authorization rule is an instance of one of the following rule templates:

$$allow(N, A) \leftarrow conditions \tag{4}$$

$$deny(N, A) \leftarrow conditions. \tag{5}$$

Symbol *conditions* stands for a possibly empty conjunction of *contextual conditions* on nodes, topics, QoS, messages, and the environment. Here are a few examples of authorization rules:

$$deny(sensor1, publish(\_, alarms/sensor1, \_)) \leftarrow time() > 8 \wedge time() < 20 \; . \tag{6}$$

Rule 6 denies *sensor1* to publish messages (whichever the QoS is), in topic *alarms/sensor1* during daytime:

$$allow(N, subscribe(alarms/\#, \_)) \leftarrow guest(N) \; . \tag{7}$$

Rule 7 allows guest nodes to subscribe to the alarm hierarchy of topics. Here we assume *guest/1* is a role predicate expanding the hierarchy defined in "Assumptions".

Regarding the delivering operation, we should first note that the normal MQTT behavior is to deliver messages from topic T to the nodes which subscribed to topic T.

This can be expressed by the following default policy rule:

$$allow(broker, deliver(\_, T, N)) \leftarrow hasSubscribed(N, T, \_) \; . \tag{8}$$

Rule 8 allows the broker to deliver any messages from topic T to the nodes which subscribed to topic T. However, this default policy can be overridden in some specific cases (see "Security Policy" for conflicts resolution between rules):

$$deny(broker, deliver(M, alarms/\#, N)) \leftarrow guest(N) \wedge value(M) =' failure' \; . \tag{9}$$

Rule 9 overrides rule 8 and denies the broker to deliver failure messages from the alarm hierarchy of topics to guest nodes. Rule 9 is an example of a content-based authorization rule.

In rules 7 and 9, there is a path expression referring to the set of topics alarms/#. Therefore, we need to include in set $\mathscr{P}$ some rules to derive instances of predicates *allow/2* and *deny/2* addressing any subset of a set of topics expressed by means of wildcards:

Table 8 Frequency predicate

Predicate	Meaning
$freq(E, F, I)$	F is the instant frequency of repeating event E per unit of time I

$$allow/deny\big(N, publish\big(M, T', Q\big)\big) \leftarrow allow/deny(N, publish(M, T, Q)) \wedge addresses\big(T, T'\big) . \quad (10)$$

Rule 10 says that if publication is allowed/denied for a set of topics *T,* then publication is also allowed/denied for each subset *T'* of *T*. We could write similar rules for the *subscribe/3* and *deliver/3* predicates.

For example, since *addresses(alarms/#,alarms/sensor1)* is true, then *allow(subscribe(user1,alarms/sensor1,1))* can be derived from *allow(subscribe(user1, alarms/#,1))*.

Controlling the Frequency of Events

Our experience has shown us that in some applications being able to control the frequency of publications, subscriptions and messages distribution is important. Consider, for example, an online-trading broker. An online trading broker is a pub-sub service, where clients may send trade orders and receive various tips and hints related to the stock market. Assume that the online broker sells standard accounts and premium accounts. Premium account holders receive more hints and tips per day than standard account holders. Moreover, premium account holders can send more trading orders per day than standard account holders. In such a scenario, we would need to express authorization rules controlling the frequency of publications (e.g., trade orders) and the frequency of messages (e.g., hints and tips) delivered by the broker. Another obvious use of having authorization rules based on the frequency of publications would be to mitigate the effects of compromised sensors involved in DDOS attacks against the pub-sub architecture.

To define authorization rules allowing us to express conditions on the frequency of events, we define the following high-order predicate (Table 8):

Frequencies are always evaluated at the time that the policy is evaluated. This explains why instances of the *freq/3* predicates represent instant frequencies.

Variable *E* refers to any formula instance of the three event predicates *hasPublished/3*, *hasSubscribed/3,* and *hasDelivered/3* defined in "Assumptions", with the last variable referring to the timestamp of the event always equal to the anonymous variable _.

Here are two examples of frequencies:

$$freq(hasPublished(sensor1, alarms/sensor1, \_), 5, 24). \quad (11)$$

Formula 11 says that the instant frequency of publications made by *sensor1* in topic *alarms/sensor1* is 5 in the last 24 h:

$$freq(hasPublished(\_, alarms/\#, \_), 152, 24). \quad (12)$$

Formula 12 says that the instant frequency of publications (made by all sensors) in topics hierarchy *alarms/#* is 152 in the last 24 h.

Note that, by defining the high-order predicate *freq/3*, we are no longer in strict first-order logic. However, computing instances of the *freq/3* predicate can easily be done using some aggregate predicate which would be implemented in many inference engines. For example, the rule below is the SWI Prolog [18] definition of the *freq/3* predicate for the *hasPublished/3* predicate. It uses the Prolog built-in *aggregate_all/3* predicate:

$$freq(hasPublished(N, T, \_), F, I)) \leftarrow aggregate\_all(count, (hasPublished(N, T, D) \wedge (time() - D) < I)), F) . \quad (13)$$

Basically, Prolog rule 13 counts the number of instances of the *hasPublished/3* predicate referring to node *N* and topic *T* with a timestamp not older than *I* hours.

The following rules are examples of authorization rules regulating the frequency of publications and messages distribution:

$$allow(sensor1, publish(\_, alarms/sensor1, \_)) \leftarrow freq(hasPublished(sensor1, alarms/sensor1, \_), F, 24) \wedge F < 5 . \quad (14)$$

Rule 14 allows *sensor1* to publish messages in topic *alarms/sensor1* as long as it does not post more than 5 alert messages per 24 h:

$$deny(broker, deliver(\_, alarms/sensor1, N)) \leftarrow guest(N) \wedge freq(hasDelivered(alarms/sensor1, N, \_), F, 24) \wedge F > 1 . \quad (15)$$

Rule 15 denies the broker to deliver to guest nodes more than one alert message per 24 h from topic *alarms/sensor1*.

Conflict Resolution Policy

Since our authorization model allows for positive and negative authorizations, conflicts between rules may arise. For example, consider the following two rules:

$$deny(N, subscribe(\_, \_)) \leftarrow sensor(N) \quad (16)$$

$$allow(N, subscribe(N/\#, \_)) \leftarrow sensor(N). \quad (17)$$

Rule 16 says that subscriptions are forbidden for sensors, while rule 17 says that sensors can subscribe (at any

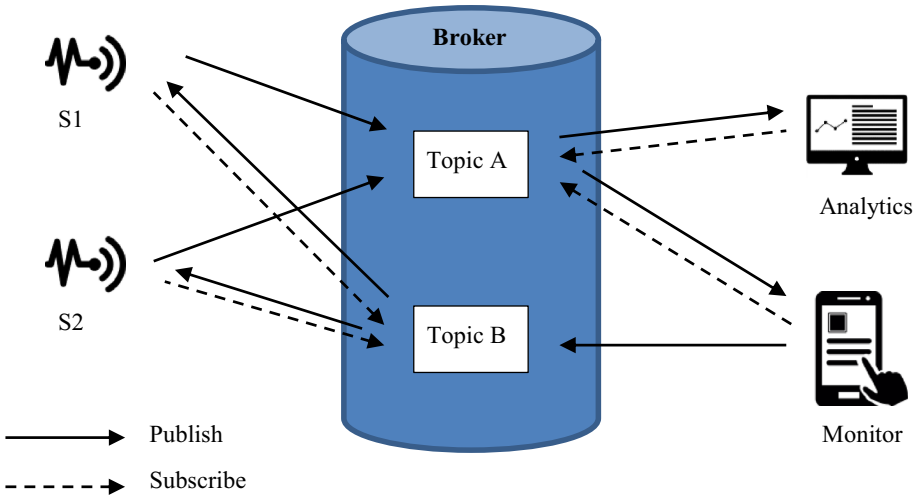

Fig. 1 IoT network

QoS) to topic for which the path root corresponds to their identifier. Clearly, these two rules conflict whenever a sensor subscribes to a topic for which the path root corresponds to the sensor identifier.

There are many possible solutions to solve conflicts between authorization rules. The XACML standard [16] enumerates several combining algorithms to solve conflicts between rules (deny overrides, permit overrides, first applicable overrides, permit unless deny, deny unless permit, etc.). We can use any of these algorithms depending on our needs. Regarding the small example above, the permit overrides algorithm would allow a node subscribing to a topic for which the path root corresponds to the node identifier.

Security Administration Model

Principles

Definition of a security model must include the definition of a model for administering the security policy. To introduce our model, let us first consider the scheme, as depicted in Fig. 1.

Sensors (*S1* and *S2*) sends messages to Analytics through topic *A*. Monitor sends commands to sensors through topic *B*. Monitor owns sensors *S1* and *S2* and created topics *A* and *B*. This scenario suggests us that Monitor could be the administrator defining the security policy regulating messages going through channels *A* and *B*. Of course, this is not the only possible scenario. The IoT network could be more centralized; topics *A* and *B* could also be shared by other applications and sensors. Nevertheless, decentralizing the security administration should be possible even if the network contains only one broker. Moreover, to give flexibility, delegation of rights should also be supported.

In our model, security administration is topic-based. We state that there is at least one security administrator for each topic. A security administrator for a given topic *T* is responsible for defining the security policy regulating publications/subscriptions to topic *T* and distribution of messages from topic *T*. There is also one *Root Administrator* (*RA*) who can administrate the security policy for all topics.

More precisely, the *RA* can perform the following tasks:

- Administrate (i.e., define the security policy for) all or some topics.
- Grant to another user the admin privilege on a given topic (with possibly the right to transfer this right).
- Revoke from a user the admin privilege on a topic.

Each admin for a given topic *T* can define the security policy for that topic *T*. If s/he has also been granted the right to transfer this right, then s/he may also grant to another node the right to administrate topic *T*. We believe that this administration scheme is flexible enough to support various cases of application.

In the following sections, we show how we extend our logical language to define our administration policy.

Constants

We define the following constant to represent the *RA*: *root*

Function

We define the following function which returns the topic addressed by an authorization rule *R* (instance of either template 4 or template 5) (Table 9).

Table 9 Topic function

Function	Purpose
topic(*R*)	Returns the topic appearing in the action parameter *A* of rule *R*

Table 10 Rights' delegation predicate

Predicate	Meaning
$hasGranted(N, T, N', G)$	Node *N* has granted the *admin privilege* on topic *T* to node *N'* with or without the *grant option* (depending on the value of the Boolean variable *G*)

Recall that action *A* is represented by one of the compound terms *publish/3*, *subscribe/2* or *deliver/3* defined in Table 6.

Predicate

We extend set F with instances of the following event rights delegation predicate (Table 10):

Granting an admin right creates a new instance of this predicate. Revoking the right deletes the corresponding instance of this predicate. The grant option is similar to the grant option of the SQL grant statement [19].

Security Administration Policy

Let *TA* be an administrator node of topic *T*. Admin *TA* can add and delete authorization rules addressing topic *T* in the security policy $\mathscr{P}$. If admin *TA* holds the grant option on topic *T*, then it can also grant and revoke the admin rights on topic *T* to some other nodes.

Actions

We define four compound terms to represent the four following actions (Table 11):

Note that the grant option cannot be granted nor revoked separately. The same principle applies in the SQL delegation scheme.

Security Administration Rules

The security administration policy $\mathscr{a}$ is mandatory and consists of the five following administration rules:

$$hasGranted(root, \#, root, true). \tag{18}$$

Rule 18 says that the *RA* has granted to himself the admin option on the whole topic hierarchy with the admin option:

$$allow(N, ruleAdd(R)) \leftarrow hasGranted(\_, T, N, \_) \wedge addresses(T, T') \wedge topic(R) = T'. \tag{19}$$

Rule 19 says that if node *N* was granted the admin privilege on topic *T*, then it can add authorization rules referring to topic *T* (or to a subset of topics *T* if *T* represents a set of topics):

$$allow(N, ruleDel(R)) \leftarrow hasGranted(\_, T, N, \_) \wedge addresses(T, T') \wedge topic(R) = T'. \tag{20}$$

Rule 20 says that if node *N* was granted the admin privilege on topic *T*, then it can delete authorization rules referring to topic *T* (or to a subset of topics *T* if *T* represents a set of topics):

$$allow(N, grant(T', \_, \_)) \leftarrow hasGranted(\_, T, N, true) \wedge addresses(T, T'). \tag{21}$$

Rule 21 says that if node *N* was granted the admin privilege on topic *T* with the grant option, then it can grant the admin option on topic *T* (or on a subset of topics *T* if *T* represents a set of topics):

$$allow(N, revoke(T', N')) \leftarrow hasGranted(N, T, N', \_). \tag{22}$$

Rule 22 says that if node *N* has granted to node *N'* the admin privilege on topic *T,* then it can revoke this privilege from node *N'*. In other words, only the node which transferred a privilege can revoke it. Moreover, as we said previously revoking an admin privilege deletes the corresponding instance of the *hasGranted/4* predicate. Since the grantee *N'* might also have transferred this right to some other nodes, revocation would also delete all the instances of *hasGranted/4* corresponding to the delegation chain originating from *N'*. This mechanism is usually referred to as cascade revocation.

Finally, let us mention the following two points:

Table 11 Security administration actions

Term	Action
ruleAdd(*R*)	Adding rule *R* in policy $\mathscr{P}$
ruleDel(*R*)	Deleting rule *R* from policy $\mathscr{P}$
grant(*T*, *N*, *G*)	Granting the admin privilege on topic *T* to node *N* with or without (depending on the value of *G*) the grant option
revoke(*T*, *N*)	Revoking the admin privilege on topic *T* from node *N*

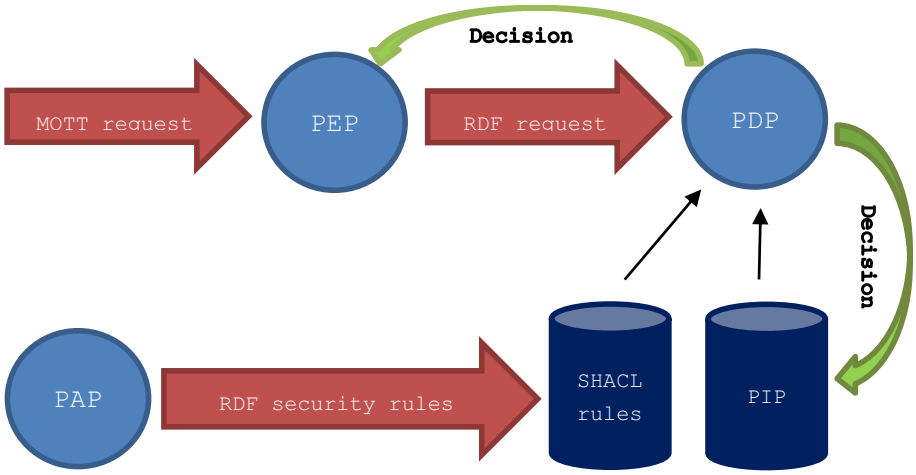

Fig. 2 Access control engine workflow

- The combining algorithm enforced in $\mathscr{a}$ is obviously deny unless permit [16], i.e., the default policy is deny. This default policy is overridden by rules 18–22.
- If T represents a set of topics, then it is not possible to revoke the admin privilege for a subset of T. One would have to revoke the admin privilege for the set T and then grant again the admin privilege on a subset of T.

Prototype

In [12, 13], we describe a first proof-of-concept prototype. This first prototype is based on (i) OWL2 ontologies for representing nodes, topics and events, (ii) SWRL [20] rules for representing the security policy, and (iii) an OWL2 [21] inference engine for computing a security decision (allow/deny). However, this approach has proved to be inefficient in terms of performance. In this paper, we present a new prototype based on RDF [14] and SHACL [15] ("Architecture") and we evaluate its performance ("Performance Analysis").

Architecture

The implantation of our security model relies on the W3C Resource Description Framework (RDF 1.1) Model [22]. We use RDF as a logical model to formally represent the security policy and its processing. We also use RDF as a physical model, since each identifiable object (node, topic, event, security rule, etc.) is represented as a resource. In other words, RDF statements describe resources, contextual information, events occurring in the system, and the security policy. We use an inference engine to compute security decisions. Each security decision and the proof graph which has led to the decision are automatically added to our contextual database for further processing. As a general principle, any event occurring within our access control enforcement system is monitored and saved into the contextual database. Recording events such as publication/subscription requests, security decisions, processing errors, etc. is not only important for traceability purpose, but also allows us to refer to these events in dynamic contextual access control rules.

Recently, the W3C has proposed the Shapes Constraint Language (SHACL) recommendation [15] as a solution to define and validate constraints on RDF graphs by means of SHACL shapes. SHACL has also been extended with a rule mechanism [23], where conditions can even include SPARQL queries [24]. Therefore, we can translate any security rule defined with our model into an SHACL shape/rule. Instead of processing security rules with a generic rule reasoner as we did in our first prototype [12, 13], we now translate our security rules into SHACL shapes/rules. We apply that these SHACL shapes/rules to RDF publication/subscription requests to produce security decisions. Processing SHACL shapes/rules has proved to be much more efficient than processing SWRL rules and ontologies with a generic rule reasoner.

Our access control enforcement system is built according to the XACML architecture [16], that is, it has a Policy Enforcement Point (PEP), a Policy Decision Point, a Policy Information Point (PIP—contextual database), and a Policy Administration Point (PAP). Figure 2 depicts the general workflow of our prototype. RDF security rules are first translated into SHACL shapes/rules. Any MQTT request is intercepted by the PEP, which has been implemented as a proxy between end points and the MQTT broker. The PEP submits to the PDP the RDF request. The PDP uses knowledge from the PIP and applies the SHACL rules on the request to compute a decision together with the RDF proof graph which has led to the decision. Decision and proof graph are returned to the PEP and saved into the PIP.

Our prototype is written in Java 9. It is based on the Apache Jena Framework [25] to store and manipulate RDF data. SHACL validation is done using TopBraid [26] which

Table 12 Configuration scenarios

	C1	C2	C3	C4	C5	C6
Number of subscribers	50	100	250	500	750	1000
Number of publishers	1000	1000	1000	1000	1000	1000

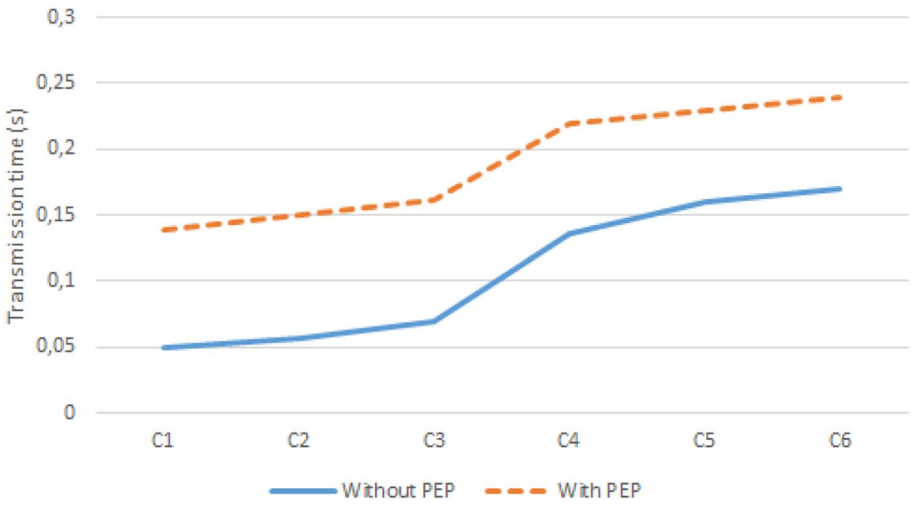

Fig. 3 Average transmission time per scenario

is an open source plugin for Jena. Our prototype provides the user with a REST API to manage the PIP, the policies, and the request/response workflow

Performance Analysis

We analyze the transmission time of messages, i.e., the time taken by a published message to reach a subscriber. We compare the transmission time in a configuration, where the security policy is enabled and the requests intercepted by the PEP with a configuration, where the clients are directly connected to the server with no security enforcement. We conduct our tests on a desktop PC equipped with an i764bit Quad Core CPU and 16Gbit of RAM. The broker is Mosquitto 1.6.0 which is an MQTT v3.1.1 broker. The clients (publishers and subscribers) are implemented in python and rely on the Paho-mqtt1.4.0 libraries. Paho-mqtt implements versions 3.1 and 3.1.1 of the MQTT protocol.

Our experiment configuration is inspired from a similar work presented in [27]. It is the following:

- The subject tree is generated randomly, so that each node has between 0 and 5 children. The maximum height of the tree is 5. From the topic tree, a list of possible topics for subscription and publication is generated. Similarly, a list of possible topic filters is created.
- Each subscriber issues a single subscription on a topic generated randomly in the subject tree, with a random quality of service between 0 and 2.
- Each publisher posts one message on a randomly generated topic. The payload of the message contains the identifier of the publisher, the topic, a timestamp corresponding to the time of emission, and a random quality of service.
- The security policy is randomly generated. It contains 100 rules. Each rule contains:
  - A priority between 0 and 5
  - An action: (publish or subscribe)
  - A target: (a random topic or a set of topics)
  - A decision: (allow or deny)
- Clients are all generated at the same time. Upon receipt of a message by a subscriber, the timestamp corresponding to the time of emission is retrieved from the payload and subtracted to the reception time to compute the transmission time.
- We consider a set of experiments for scenarios composed of 50, 100, 250, 500, 750, and 1000 subscribers each including 1000 publishers (see Table 12)?

Results of our measurements are shown in Fig. 3. Figure 3 shows that the difference between the average transmission time of one request with the PEP and the average transmission time of one request without the PEP is constant and approximately equal to 100 ms, regardless of the scenario. This shows that our prototype is scalable and that our approach is very promising. In a similar work presented in [27], the authors developed a prototype, where the average

transmission time increased linearly with the number of subscribers.

The difference of 100 ms mainly consists of,

- the time required to process the event by the PEP,
- the time required to process the request by the PDP,
- the transmission times between the PDP and the PEP.

We separately measured the average time to process the request by the PDP and we found out that it is constant and approximately equal to 55 ms.

Our prototype can still be optimized. For instance, we are planning to use web sockets instead of the REST API. This should improve transmission times between the PEP and the PDP.

Related Works

As we already said, to our knowledge, there is no paper directly addressing the definition of a security policy model for IoT messages. Nevertheless, in this section, we review security works related to pub-sub architectures and MQTT protocol.

This paper which resembles the most to our approach is [27]. Like us, the authors consider that the protection object is the message. Therefore, they also regulate the distribution of messages by the broker. They use the ABAC model, although they do not define, as we do, an ABAC profile for IoT applications. They consider two types of security policies: the security policy expressed by administrators regulating the right to publish/receive messages and the security policy expressed by users in terms of preferences. They do not mention rights delegation. Their Policy Enforcement Point (PEP) is implemented as a proxy between the MQTT broker and the nodes. Having in mind the performances, they manage the security policy within a key value datastore (Redis[2]). However, they do not say much on the Policy Decision Point (PDP) and on analyzing the security policy using a reasoning engine. Moreover, Redis does not allow for complex queries and does not seem to be the right choice for storing highly expressive contextual policies. Finally, our prototype has shown better performance than their prototype (see "Performance Analysis").

In [3], the authors define the *secKit* tool integrated in the IOT network simulator developed as part of the FP7 iCore project [28]. This tool is used to define security policies protecting the data exchanged between the different components (virtual objects, composite virtual objects) which abstract the IoT network. *secKit* is based on a collection of models for modeling objects, data, time, roles, activities, interactions, risk, contexts, trust management, and so on. It was implemented as a Mosquitto plugin [29]. Authorizations rules can be positive or negative and include obligations. Authorization rules are Event Condition Action (ECA) rules [30]. This formalism makes it possible to express contextual and dynamic authorizations, but requires the implementation of an event manager capable of intercepting all events. Many aspects related to this tool are not clearly defined. Authors claim that the tool supports many features, but they do not elaborate on the features (expressive power, risks, trust management, obligations, conflicts resolution, etc.). Therefore, it is very difficult to have a clear view of the model supported by *secKit*. Note also that the tool *secKit* seems to be abandoned. The source code of an alpha version can be downloaded from gitHub (https://github.com/iot-icore/iCore-security-toolkit), but has not been modified for 4 years.

In [4], the authors describe a NetwOrked Smart object(NOS) middleware, located between the objects and the MQTT client. NOS intercepts the messages intended to be published, normalizes them (i.e., extracts metadata), and according to a security policy, implementing the ABAC model decides whether to grant the publication of the message. If the message is authorized, it is encrypted by means of a temporary key corresponding to the subject (topic), where the message is to be published. Once encrypted, the message is published by the MQTT client. Customers wishing to subscribe to a topic, contact the NOS which according to the security policy will issue them or not the key to decipher the messages of the subject. This approach frees the MQTT broker from the evaluation and enforcement of the authorization policy. It requires, however, to set up a key management mechanism and offers a rather coarse level of granularity, since the object of protection is not the message but the subject (thus including all the messages published in the subject). NOS has been implemented using the node.js platform and the objects transmit their messages via the http protocol. Currently, NOS is not available for download. In [8], the same authors improve their architecture by proposing a solution to distribute and synchronize the security policies hosted by the NOS of several IoT networks. Their synchronization protocol uses the MQTT protocol.

In [5], the authors propose a solution to implement the ABAC model in a federation of IoT platforms. Their solution decouples the authorization process of the authentication process. An application first connects to an authorization manager to obtain a set of tokens. Each token represents an attribute of the application. Once in possession of its tokens, the application that needs to access a resource turns to an authorization manager who will accept or reject access to the requested resource based on the tokens presented. If the application wants to access a resource belonging to a foreign IoT platform, then it must present its tokens to the authentication manager of the foreign platform to obtain foreign

[2] https://redis.io/.

tokens. These foreign tokens are computed by means of a token conversion function (little information is given on this function in the article). Once in possession of the foreign tokens, the application can then turn to the foreign authorization manager who will accept or reject access. The authors suggest to implement the tokens either in the form of Google macaroons [31] or in the form of JWT [32] tokens (Json Web Tokens).

In [33], the authors implement the RBAC model in a pub-sub network. They consider the privilege of logging in, the privileges of adding a topic and deleting it, and the privileges of publishing and subscribing. They define a solution to disseminate security policy [34] in a network of partially trusted brokers so that access control decisions are taken and enforced at the earliest. Their solution is implemented in the Hermès broker [35] using the http protocol.

In [36], the authors propose to adapt the OAuth protocol [37] to the case of an IoT network, where resources are discovered and exposed according to the IETF standard [38]. The data (resources) produced by the sensors of the low power IoT network are transmitted to the gateway between the low power network and the IP network. This schema ignores the MQTT protocol, resources being logged and exposed at the gateway level. Within this gateway, an authorization server is installed. According to the OAuth protocol, a third-party application that wants to access a resource exposed by the gateway, (1) requests access delegation from the owner of the resource by asking him to authenticate, (2) obtains an access token to the resource provided by the authorization server and (3) presents the access token to the authorization server to access the resource. The authors do not mention any case study that could benefit from such a security architecture.

In [39], the authors present above all a method of authorization using Elliptical Curve Cryptography (ECC). However, they also implement an authorization scheme using capabilities. Capabilities are tokens distributed to users holding the access rights of the users to the data produced by the network. Very little is said about the expressive power of the security policy and interaction of their scheme with the MQTT protocol is unclear.

There is a general trend in the world of IoT which consists, for scaling purposes, of moving the processing and controls at the edge of the IoT network towards the objects themselves [40]. Thus, there are several approaches [6, 7] for moving security-related services and processes to end-of-network gateways or servers. In [6], the authors even use the concept of sticky policy [41] which implies that data owners encapsulate the policy protecting the data with the data itself.

There are several articles that deal with security issues in pub-sub networks, without specifically considering the MQTT protocol. In [9], the authors attempt to identify the security problems specific to this type of network. They are particularly interested in protecting the confidentiality of messages and subscriptions in case the pub-sub infrastructure is not trusted. They suggest some lines of research such as the use of numerical calculation on encrypted data [42, 43]. To protect the content of the messages and subscription schemes of the brokers, the authors in [10] propose a technique based on the encryption preserving the asymmetric scalar product [44].

In [45], the authors propose a model that implements the reliable distribution service, that is, the messages received by a client do not depend on the connection location, the network latency, or the possible points of network failure. These messages depend only on the customer's subscription filter and her access rights, which are uniformly enforced throughout the pub-sub network.

Conclusion

In this paper, we have defined a model to express security policies for a pub-sub architecture consisting of a single MQTT broker. The most important contributions of our paper are the followings:

- Our model allows us to regulate not only publications and subscriptions but also distribution of messages. To our knowledge, this feature has not been addressed in any other paper related to IoT security except [27].
- Our model is an interpretation of the ABAC model for the pub-sub architecture with some unique features like the possibility to control the frequency of events.
- We have developed a scalable prototype based on RDF and SHACL.

Regarding future works, we are planning to investigate the following issues:

- We will extend our model to the case of a pub-sub architecture consisting of several bridged brokers. In such a scenario, we might need to apply the solution presented in [8] to synchronize the security policy at every node of the pub-sub architecture.
- We will also consider an IoT network consisting of a TCP/IP network hosting the pub-sub architecture coupled with a Low Power Wide Area Network (LPWAN) hosting the sensors. In such a scenario, we might also need to implement solutions proposed by others [6, 7] to move, for scaling purposes, the security controls at the various gateways between the TCP/IP network and the LPWAN network.

- We are also planning to include the possibility to declare obligations in the security policy.

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. ISO/IEC 20922:2016—information technology—message queuing telemetry transport (MQTT) v3.1.1. 2016. https://www.iso.org/standard/69466.html. Accessed 11 Jan 2018.
2. Banks A, Gupta R. MQTT version 3.1. 1. OASIS Stand 2014;29.
3. Neisse R, Steri G, Fovino IN, Baldini G. SecKit: a model-based security toolkit for the internet of things. Comput Secur. 2015;54:60–76.
4. Rizzardi A, Sicari S, Miorandi D, Coen-Porisini A. AUPS: an open source AUthenticated publish/subscribe system for the internet of things. Inf Syst. 2016;62:29–41.
5. Sciancalepore S, et al. Attribute-based access control scheme in federated IoT platforms. In: International Workshop on Interoperability and Open-Source Solutions. 2016, pp. 123–138.
6. Sicari S, Rizzardi A, Miorandi D, Coen-Porisini A. Security towards the edge: sticky policy enforcement for networked smart objects. Inf Syst. 2017;71:78–89.
7. Phung PH, Truong HL, Yasoju DT. P4SINC-an execution policy framework for IoT services in the edge. In: Internet of Things (ICIOT), 2017 International Congress on IEEE. 2017, pp. 137–142
8. Sicari S, Rizzardi A, Miorandi D, Coen-Porisini A. Dynamic policies in internet of things: enforcement and synchronization. IEEE Internet Things J. 2017;4(6):2228–38.
9. Wang C, Carzaniga A, Evans D, Wolf AL. Security issues and requirements for internet-scale publish-subscribe systems. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences. Big Island, HI, USA: IEEE; 2002.
10. Choi S, Ghinita G, Bertino E. A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations. In: DEXA'10 Proceedings of the 21st international conference on Database and expert systems applications: Part I. Berlin, Heidelberg: Springer; 2010. pp. 368–384.
11. Yuan E, Tong J. Attributed based access control (ABAC) for web services. In: IEEE International Conference on Web Services (ICWS'05). Orlando, FL, USA: IEEE; 2005.
12. Gabillon A, Bruno E. Regulating IoT messages. In: Presented at the 14th international conference on information security practice and experience (ISPEC 2018)—short paper, Tokyo. 2018.
13. Gabillon A, Bruno E. A security model for IoT networks. In: International conference on future data and security engineering. Ho Chi Minh Ville, Vietnam; 2018, pp. 39–56.
14. McBride B. The resource description framework (RDF) and its vocabulary description language RDFS. In: Handbook on ontologies. New York: Springer; 2004, pp. 51–65.
15. Knublauch H, Kontokostas D. Shapes constraint language (SHACL). W3C Candidate Recomm. 2017;11(8).
16. Moses T, et al. Extensible access control markup language (xacml) version 2.0. Oasis Stand. 2005;02.
17. Becker MY, Fournet C, Gordon AD. SecPAL: design and semantics of a decentralized authorization language. J Comput Secur. 2010;18(4):619–65.
18. Wielemaker J, Ss S, Ii I. SWI-Prolog 2.7-reference manual. 1996.
19. Date CJ, Darwen H. A guide to the SQL standard, vol. 3. New York: Addison-Wesley; 1987.
20. Horrocks I, et al. SWRL: a semantic web rule language combining OWL and RuleML. W3C Memb Submiss. 2004;21:79.
21. Group WOW, et al. OWL 2 web ontology language document overview. 2009.
22. Status for resource description framework (RDF) model and syntax specification. https://www.w3.org/1999/.status/PR-rdf-syntax-19990105/status. Accessed 25 May 2019.
23. SHACL advanced features. https://w3c.github.io/data-shapes/shacl-af/#rules. Accessed 23 Jun 2019.
24. Pérez J, Arenas M, Gutierrez C. Semantics and complexity of SPARQL. ACM Trans Database Syst TODS. 2009;34(3):16.
25. Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K. Jena: implementing the semantic web recommendations. In: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. New York, NY, USA: ACM; 2004, pp. 74–83.
26. SHACL API in Java based on Apache Jena. Contribute to TopQuadrant/shacl development by creating an account on GitHub. TopQuadrant, Inc, 2019.
27. Colombo P, Ferrari E. Access Control Enforcement within MQTT-based Internet of Things Ecosystems. In: Proceedings of the 23nd ACM on symposium on access control models and technologies. New York, NY, USA: ACm; 2018. pp. 223–234.
28. Giaffreda R. iCore: a cognitive management framework for the internet of things. In: The future internet assembly. 2013, pp. 350–352.
29. Light R. Mosquitto-an open source mqtt v3. 1 broker. URL Http-mosquitto Org. 2013.
30. Han W, Lei C. A survey on policy languages in network and security management. Comput Netw. 2012;56(1):477–89.
31. Birgisson A, Politz JG, Erlingsson U, Taly A, Vrable M, Lentczner M. Macaroons: cookies with contextual caveats for decentralized authorization in the cloud. In: NDSS. 2014.
32. Jones M, Bradley J, Sakimura N. Json web token (jwt). 2015.
33. Belokosztolszki A, Eyers DM, Pietzuch PR, Bacon J, Moody K. Role-based access control for publish/subscribe middleware architectures. In: Proceedings of the 2nd international workshop on Distributed event-based systems. 2003, pp. 1–8.
34. Singh J, Vargas L, Bacon J, Moody K. Policy-based information sharing in publish/subscribe middleware. In: 2008 IEEE workshop on policies for distributed systems and networks. 2008, pp. 137–144.
35. Hermes. http://hermes-pubsub.readthedocs.io/en/latest/. Accessed 04 Nov 2017.
36. Sciancalepore S, Piro G, Caldarola D, Boggia G, Bianchi G. OAuth-IoT: An access control framework for the Internet of Things based on open standards. In: Computers and communications (ISCC), 2017 IEEE symposium on IEEE. 2017, pp. 676–681.
37. Hardt D. The OAuth 2.0 authorization framework. 2012.
38. Shelby Z. Constrained RESTful environments (CoRE) link format. Internet Eng. Task Force IETF. 2012;RFC6690.
39. Lohachab A, et al. ECC based inter-device authentication and authorization scheme using MQTT for IoT networks. J Inf Secur Appl. 2019;46:1–12.
40. Hu YC, Patel M, Sabella D, Sprecher N, Young V. Mobile edge computing—A key technology towards 5G. ETSI White Pap. 2015;11(11):1–16.

41. Pearson S, Casassa-Mont M. Sticky policies: an approach for managing privacy across multiple parties. Computer. 2011;44(9):60–8.
42. Abadi M, Feigenbaum J, Kilian J. On hiding information from an oracle. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing. 1987, pp. 195–203.
43. Feigenbaum J. Encrypting problem instances. In: Williams HC, editor. Advances in cryptology—CRYPTO'85 proceedings. Berlin: Springer; 1986. p. 477–88.
44. Wong WK, Cheung DW, Kao B, Mamoulis N. Secure kNN computation on encrypted databases. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 2009, pp. 139–152.
45. Zhao Y, Sturman DC. Dynamic access control in a content-based publish/subscribe system with delivery guarantees. In: 26th IEEE international conference on distributed computing systems (ICDCS'06). 2006, pp. 60–60.