Improving MQTT by Inclusion of Usage Control

Antonio La Marra[1], Fabio Martinelli[1], Paolo Mori[1], Athanasios Rizos[1,2(✉)], and Andrea Saracino[1]

[1] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy
{antonio.lamarra,fabio.martinelli,paolo.mori,athanasios.rizos,andrea.saracino}@iit.cnr.it
[2] Department of Computer Science, University of Pisa, Pisa, Italy

Abstract. Due to the increasing pervasiveness of Internet of Things (IoT) and Internet of Everything (IoE) devices, securing both their communications and operations has become of capital importance. Among the several existing IoT protocols, Message Queue Telemetry Transport (MQTT) is a widely-used general purpose one, usable in both constrained and powerful devices, which coordinates data exchanges through a publish/subscribe approach. In this paper, we propose a methodology to increase the security of the MQTT protocol, by including Usage Control in its operative workflow. The inclusion of usage control enables a fine-grained dynamic control of the rights of subscribers to access data and data-streams over time, by monitoring mutable attributes related to the subscriber, the environment or data itself. We will present the architecture and workflow of MQTT enhanced through Usage Control, also presenting a real implementation on Raspberry Pi 3 for performance evaluation.

1 Introduction

Over the last years, Internet of Things (IoT) devices have become more and more pervasive to our daily life. As a matter of fact, we are currently using many connected objects, such as smart house appliances, connected cars, remote surveillance cameras, smart meters etc. According to Ericsson [5], in 2020 we should expect the total number of IoT devices to reach 50 billions, and this number becomes even more dramatic if we consider the Internet of Everything (IoE) paradigm, which also includes user devices such as smartphones, smartwatches, tablets, etc.

IoT devices could be very different, because they typically have different types of hardware, depending on the provided functionalities, and software applications to manage them. Hence, in order to have a unique application which eases the control of all the smart devices owned by the same user, a necessity has arisen to be able to easily communicate with a set of distinct IoT devices. To this aim, several application layer protocols have been proposed in the scientific literature, and among them, MQTT is one of the most widely used [1].

G. Wang et al. (Eds.): SpaCCS 2017, LNCS 10656, pp. 545–560, 2017.
https://doi.org/10.1007/978-3-319-72389-1_43

MQTT[1] is also recently standardized by OASIS[2] and works according to the Publish/Subscribe protocol pattern, where a central *Broker* handles the communications and data sharing, collecting data from a set of *Publishers* and redistributing them to a set of *Subscribers*, according to their specified interests.

According to [20], the MQTT standard and the existing implementations, provide support only for basic authentication and simple authorization policies, applied to Subscribers at subscription time. Since MQTT is based on HTTP functionalities, most of the MQTT security solutions seem to be either application specific, or just leveraging TLS/SSL protocols [1]. Currently, OASIS MQTT security subcommittee is working on a standard to secure MQTT messaging using MQTT Cybersecurity Framework [17]. Although the effort concerning security of MQTT protocol is rising, two main obstacles occur. The first one is that, although the protocol has the ability to deal with various components that become Publishers or Subscribers, the fact that they use different platforms makes it difficult to create and enforce a generic security policy. The second problem is that the current efforts are mainly directed to message communication security, to avoid eavesdropping, integrity violation and MITM attacks. Still no efforts have been done in the directions of supporting policy enforcement at Broker level, nor it has been considered the possibility of dynamically revoking subscriptions.

In this paper, we propose to enhance the security of the MQTT protocol by adding Usage Control (UCON) in the MQTT architecture and workflow. UCON is an extension of traditional access control which enforces continuity of access decision, by evaluating policies based on mutable attributes, i.e. attributes changing over time [10]. Adding Usage Control in MQTT we aim at enforcing dynamically fine grained policies, which do not only consider the identity of the Subscriber as a parameter for granting access to data, but also dynamic attributes such as Subscriber reputation, data reliability, or environmental conditions of a specific application. After surveying the main IoT application protocols, and motivating the choice of focusing on MQTT, this work will discuss the architecture and the workflow of the MQTT - UCON integration. The proposed framework is designed to be general, easy to integrate in the Broker component, remaining oblivious to both Publishers and Subscribers. The addition of UCON in fact, does not modify the MQTT protocol, enforcing the policies independently from the implementation of Publisher and Subscriber, which allows the proposed solution to be compatible with any Off-the-Shelf MQTT Publisher/Subscriber application. Furthermore, we will demonstrate the viability of the approach by presenting a real implementation of the framework on both general purpose and performance constrained devices, discussing also the performance measured on two Rasperry Pi 3 model b[3], used respectively as Broker and Subscriber.

The rest of the paper is organized as follows: In Sect. 2 a comparison between the main IoT application protocols is reported, detailing afterward the MQTT

[1] http://mqtt.org.
[2] https://www.oasis-open.org/.
[3] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

protocol and motivating our choice to focus on it. Furthermore, some background information about usage control are reported. Section 3 describes the integration of UCON and MQTT detailing the architecture and the operative workflow. Section 4 details the results of the performance analysis. In Sect. 5 are reported a set of related works about security in MQTT and application of UCON in IoT. Finally, Sect. 6 concludes by proposing future directions which stem from this preliminary work.

2 Background

In this section we will survey the main protocols for IoT, motivate the choice to focus on MQTT, briefly describing the protocol and we will recall some background notions on the concept of usage control.

2.1 IoT Application Protocols

The most known application layer protocols in IoT are CoAP, MQTT, XMPP, HTTP, AMQP and WebSocket. In [7], the authors claim that CoAP is more Resource-friendly than MQTT but in terms of Message Oriented Approach (MOA), MQTT stands out. They report also that MQTT needs less RAM but more CPU load than CoAP. All the protocols mentioned above use TCP as transport layer. Only CoAP uses UDP. The same happens as for the security layer. All protocols use TLS/SSL except from CoAP that uses DTLS. In fact, CoAP targets to very constrain environments.

Furthermore, according to [8] MQTT provides the smallest header size of two bytes, although it is based on TCP. Moreover, it provides three levels of QoS which puts this protocol in the first place in terms of QoS, even though it needs extra load in the network for message retransmission. On the other hand, XMPP requires processing and storing XML data, which necessitates memory space too large for most IoT devices. In addition, HTTP performs better in non constrained environments when PC, Laptop and Servers are used. It is generally not applicable in IoT devices due to its high overhead. AMQP [14], is more suitable for server-to-server communication than device-to-device communication. Websocket is neither a request/response nor a Publish/Subscribe protocol. In Websocket, a client initializes a handshake with a server to establish a Websocket session. The handshake process is intended to be compatible with HTTP-based server-side software so that a single port can be used by both HTTP and Websocket clients [4]. According to [21], MQTT messages experience lower delays than CoAP for lower packet loss and vice versa. When the message size is small the loss rate is equal. AllJoyn [22], is a full stack of protocols intended for IoT. Though quite popular, the main disadvantage of AllJoyn is that the application protocol cannot be separated from the rest of the protocol stack. Due to this fact, Alljoyn is a complete framework and not only an application layer protocol. Thus, it is not taken into consideration in this study. As a synopsis to the basis, reader can also consult Table 1. This comparison gives the details about

the existence of Quality of Service (QoS). Also it refers to the communication pattern which in the case of MQTT is the Publish/Subscribe. The most significant column is the third. In this column, we identified that MQTT is more general purpose.

Table 1. Application layer protocol comparison

Protocol	QoS	Communication pattern	Target devices
CoAP	YES	Req/Resp	Very constrained
MQTT	YES	Pub/Sub	Generic
XMPP	NO	Req/Resp Pub/Sub	High memory consumption
HTTP	NO	Req/Resp	High performance
AMQP	YES	Pub/Sub	Ser-2-Ser communication
Web socket	NO	Client/Server Pub/Sub	Needs less power than HTTP still need high power
AllJoyn	NO	Client/Server Pub/Sub	High computational power

2.2 The MQTT Protocol

MQTT is an open pub/sub protocol designed for constrained devices used in telemetry applications. MQTT is designed to be very simple on the client side either this is the Subscriber or the Publisher. Hence, all of the system complexities reside on the Broker which performs all the necessary actions for the MQTT functionality. MQTT is independent from the routing or networking specific algorithms and techniques. However, it assumes that the underlying network provides a point-to-point, session-oriented, auto-segmenting data transport service with in-order delivery (such as TCP/IP) and employs this service for the exchange of messages.

MQTT is a topic-based Publish/Subscribe protocol that uses character strings to provide support of hierarchical topics. This means that there is the ability to create and control the hierarchy of the topics. There is also the opportunity to the subscription to multiple topics. In Fig. 1, we can see the topology of the protocol. It consists of the Publisher(s) that send the data to the Broker for publishing. A Subscriber authenticates and subscribes to the Broker for certain topics. Moreover, the Broker sends the data to the specific Subscriber(s) that are subscribed to the specific topic of the message. The Broker is responsible to distribute them to the related Subscribers correctly. The Publishers and the Subscribers can be very constrained devices, especially in the case of Publishers that can be even a sensor. On the other hand, though, the Broker has to have enough computational power so as to be able to handle the amount of data being distributed. MQTT supports basic end-to-end Quality of Service (QoS) [3]. Depending on how reliably messages should be delivered to their receivers, MQTT provides three QoS levels. QoS level 0 only offers a best-effort delivery

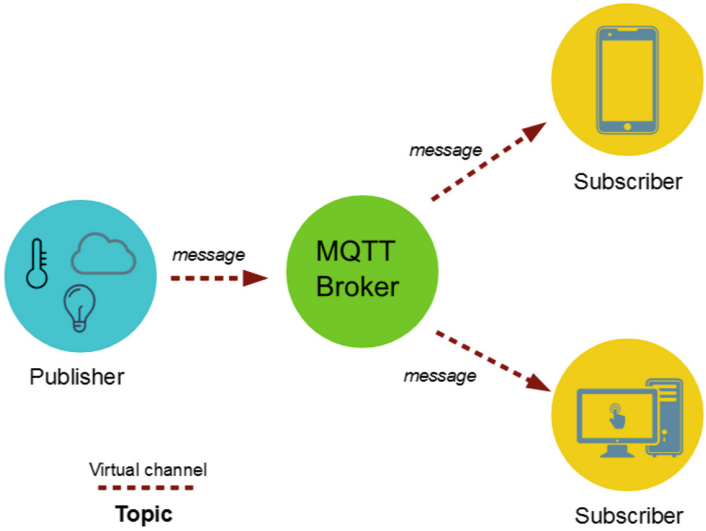

Fig. 1. MQTT topology diagram.

service, in which messages are delivered either once or not at all to their destination. No retransmission or acknowledgment is defined. QoS level 1 retransmits messages until they are acknowledged by the receivers. Hence, QoS level 1 messages may arrive multiple times at the destination because of the retransmissions, still multiple copies are not natively handled. QoS level 2 ensures not only the reception of the messages, but also that they are delivered only once.

We focus on the MQTT protocol since it is the most generic among the IoT protocols described, and libraries are available for all major IoT development platforms, like Arduino, for several programming languages (C, Java, PHP, Python, Ruby, Javascript) and for the two major mobile platforms (iOS and Android) [5]. The authentication to the Broker can be done by providing the following credentials [13]: Topic to be Subscribed on, Username and Password. The most known effort to add more security features in MQTT is SMQTT [18], but no solution is given to the policies that are followed by the information after it is delivered to the Subscribers. Our proposal addresses this problem, as long as the continuous control of Publishers/Subscribers on both authentication and access.

2.3 Usage Control

The UCON model extends traditional access control models. It introduces mutable attributes and new decision factors besides authorizations; these are obligations and conditions. Mutable attributes represent features of subjects, object, and environment that can change their values as a consequence of the operation of the system [6].

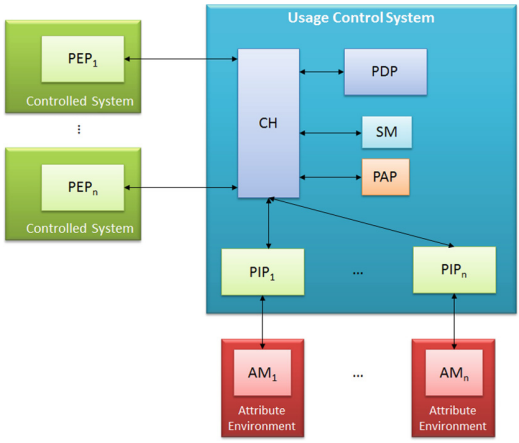

Fig. 2. Usage control framework diagram.

Since mutable attributes change their values during the usage of an object, UCON model allows to define policies which are evaluated before and continuously during the access. In particular, a usage control policy consists of three components: authorizations, conditions and obligations. Authorizations are predicates which evaluate subject and object attributes, and also the actions that the subject requested to perform on the object. Pre-Authorizations are evaluated when the subject requests to access the object, while Ongoing-Authorizations are predicates which are continuously evaluated while the access is in progress. Obligations are predicates which define requirements that must be fulfilled before the access (Pre-Obligations), or that must be continuously fulfilled while the access is in progress (Ongoing-Obligations). Conditions are requirements that evaluate the attributes of the environment. In this case too, Pre-Conditions are evaluated when the subject requests to access the object, while Ongoing-conditions are continuously evaluated while the access is in progress.

The continuous evaluation of the policy when the access is in progress is aimed at interrupting the access when the execution right is no more valid, in order to reduce the risk of misuse of resources. Hence, in the Usage Control model it is crucial to be able to continuously retrieve the updated values of the mutable attributes, in order to perform the continuous evaluation of the policy and to promptly react to the attributes change by interrupting those ongoing accesses which are no longer authorized.

The main blocks of UCON is the *Usage Control System (UCS)* surrounded by the *Controlled Systems* and the *Attribute Environment* are shown in the Fig. 2. The Controlled Systems are those components on which the UCON policy can be enforced. Each Controlled System communicates with the UCS issuing the request to access a resource by performing a specific operation on it. These components are the *Policy Enforcement Points (PEPs)*. For more information about UCON, readers can refer to [10].

UCS has its own components which are the following [16]:

Policy Decision Point (PDP): This component takes as an input an access (usage) request and an access (usage) policy returning one of the following decisions: *Permit, Deny, Undetermined.*

PIPs communicate with the Attribute Environment through Attribute Managers (AMs) which are not part of the UCS [2].

Policy Information Points (PIPs): These components retrieve attributes related to subject, object and environment of received access requests. Each PIP acts as the interface between the UCS and a specific Attribute Manager. Each PIP has custom implementation for each specific application, AM and the kind of attribute that should be retrieved.

Session Manager (SM): This component is a database which stores all the active sessions, with the necessary information to perform policy reevaluations.

Context Handler (CH): This component is the main core of the UCS, where it is responsible of routing messages among the various components. Firstly, it has to forward the access request to the various PIPs for attribute retrieval, then the complete access to the PDP and as a result to return the decision to the PEP. Finally, it receives notification from PIPs when the value of an attribute changes, forwarding to the PDP the new value for policy reevaluation. UCON framework consists of the following actions [9]:

`TryAccess:` This function is invoked by the PEP to send to the UCS the request to perform an action or access a resource, to be evaluated against a policy. The UCS will respond with a Permit or Deny decision, eventually collecting the needed attributes from the PIPs. If the answer is Permit, this response is also containing the Session ID for the session that is about to start.

`StartAccess:` This function is invoked by the PEP having the SessionID as a parameter. This is the actual start of using the service requested. There is again evaluation from the PDP and after an affirmative response the CH confirms the session to the SM as active.

`RevokeAccess:` If a mutable attribute changes its value, the PIP sends it to the CH for reevaluation because it might change the policy decision. If this event occurs, the usage has to be revoked. The CH informs both PEP and SM that this session is revoked. On one hand, the SM keeps the session recorded but in an inactive state, whereas on the other hand the PEP blocks the usage to the resource.

`EndAccess:` This function is invoked when the usage of the resource terminates. When received by the UCS, it deletes the session details from the SM and communicates to the PIPs that the attributes related to that policies are not needed anymore, unless other sessions are using it.

3 Introducing UCON in MQTT

In this section we present the proposed architecture, presenting first the model, then the operative workflow and the performed implementation.

3.1 System Model

As previously mentioned, MQTT protocol is based on the Publish/Subscribe model, thus the entities participating to the protocol can act either as *Publishers* or *Subscribers*. Publishers could be sensors or other devices which collect and provide specific data, when available, periodically or even as a stream. Subscribers are instead entities that register to the broker to receive, when available, specific data or set of information grouped under a *Topic*. The *Broker* acts as middleware and coordinator, managing the subscription requests and dispatching data to Subscribers, when made available by prosumers.

The MQTT protocol supports ID and password-based authentication for both Publishers and Subscribers. The enforcement is performed on Broker's side, which keeps track of the ID and authentication password of authorized Publishers and Subscribers. However, we argue that this authentication model is too simplistic and coarse grained, making impossible to check the right to access information over time. In fact, once a Subscriber has been authorized, the subscription remains valid until the Subscriber explicitly invokes an `unsubscribe` for the topic(s) it was registered for. The same goes for Publishers which keeps the right to publish continuously or on demand, till they have valid credentials. In real applications, several features might imply a condition for a subscription to decay, or for a publication to be denied. Detected Publisher malfunction or corruption, conditions on time spans in which a subscription should be allowed, and Subscriber reputation, are just few examples of aspects on which a more complex policy should be enforced. To be able to enforce policies with similar conditions to the aforementioned ones, and to have the possibility of revoking a subscription, usage control has been added to the MQTT logical architecture.

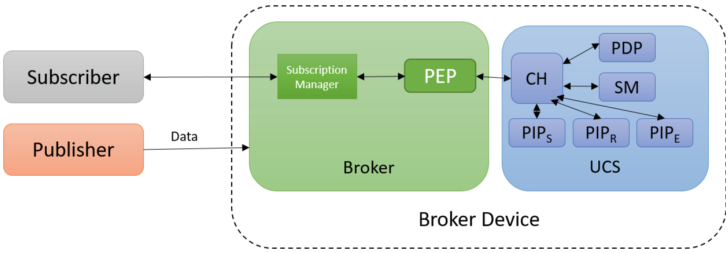

Fig. 3. UCON implementation in MQTT.

In Fig. 3 we depict the logical architecture of the proposed framework. As shown, the UCS is physically integrated in the Broker Device, i.e. the physical

machine that is hosting the Broker software, which enables the MQTT protocol. It is worth noting that we consider in this example three abstract PIPs, which are conceptually grouping the PIPs reading attributes related to the subject (PIP_S), to the resource (PIP_R) and to the environment (PIP_E). The PEP is (partially) embedded in the broker, to dynamically control the subscription events. In particular, the PEP will intercept the subscription events and interact directly with the Broker subscription manager, deleting and inserting the entries for Subscribers from the list of authorized ones, according on the UCS decision. In such a way, the PEP ensures that no Subscribers can register by avoiding the enforcement of the usage control policy. Since the PEP is embedded in the Broker, the proposed architecture remains compatible with any implementation of MQTT Subscribers. The only requirement is that the Subscriber is configured to access with username and password, otherwise the connection will be refused by the Broker.

3.2 Operative Workflow

In Fig. 4, we report the envisioned workflow. For the sake of simplicity, we will consider a simple system made out of a Broker and a single Publisher and Subscriber.

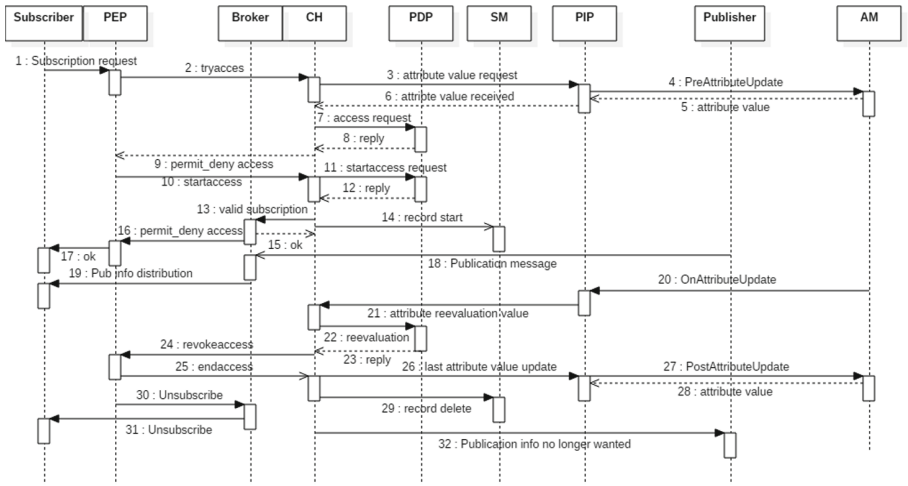

Fig. 4. Full workflow sequence diagram.

The workflow is initiated by a subscription request from the Subscriber to the Broker. This request is intercepted by the PEP, which interprets it, so as to take the credentials of the Subscriber that are needed in order to create and send the request to the UCS for evaluation. Hence the PEP invokes the `TryAccess` sending to the CH request and policy. The request is eventually filled

by attributes retrieved through the PIP, then is sent, together with the policy, to the PDP for evaluation, which should return a Permit or Deny decision. In case of Deny, the subscription request is dropped and the Subscriber will be notified, as if a wrong username/password has been inserted. In case of Permit, the Session Manager (SM) creates the session and sends its ID to the PEP (via the CH) which is informed about this decision and performs the `StartAccess`. Supposing a permit decision has been received, the Broker informs the Subscriber about the successful subscription and starts to send data related to the topic when available, eventually stimulating Publishers in an idle state.

To illustrate the `revoke` workflow, we suppose that one of the attributes relevant for the Subscriber policy changes its value `(OnAttributeUpdate)`. This causes the PIP to send this new attribute to the CH that forwards it to the PDP for reevaluation. Supposing that the value of this attribute leads to a conclusion that this session must be revoked (Deny decision), the CH invokes the `RevokeAccess` on the PEP, also informing the Subscriber that the access is no longer granted `(RevokeAccess)`. The termination of the access could happen also if the Subscriber is no longer interested to the data, invoking the `Unsubscribe`. The unsubscribe triggers the PEP to send an `EndAccess` to the CH. The latter informs the PIP to take the last value of the attribute `(PostAttributeUpdate)`. Also the UCS informs the Broker that the Subscriber is no longer subscribed and forces the unsubscription. Moreover, the SM is also informed that this session is over so that the record should be archived or deleted. Finally, if this Subscriber is assumed to be the only one that was interested to the Publisher, the Broker informs him to stop data publication due to fact that there is no more any interest from any Subscriber.

We point out that the simplification of considering a single Publisher/Subscriber does not harm generality. In fact, the protocol is not modified and multiple Subscribers/Publishers do not introduce any additional criticalities, since concurrency is natively supported in both the UCS and MQTT.

3.3 Implementation

As previously mentioned the UCS is a Java-based configurable framework, easy to integrate in any system with a Java runtime environment. The software used to implement the Broker is the open source MQTT Broker Moquette[4]. Though not largely used as the Mosquitto[5] Broker, Moquette is easier to integrate with the UCS framework, since they are based on the same programming language. The Broker has been partially modified to include in it the PEP functionalities. In particular, the subscription request is intercepted by hooking the subscription handling method, as to invoke `TryAccess` and `StartAccess` and waiting results before allowing or denying the subscription. If a Deny decision is received, the Broker will return a *wrong user/password* message to the subscriber.

[4] https://github.com/andsel/moquette.
[5] https://mosquitto.org.

If there is a policy violation, the `RevokeAccess` is invoked. Hence, the PEP calls the Unsubscribe function so as to prevent the Subscriber from receiving messages, while the `EndAccess` is invoked to remove the session details on the UCS side.

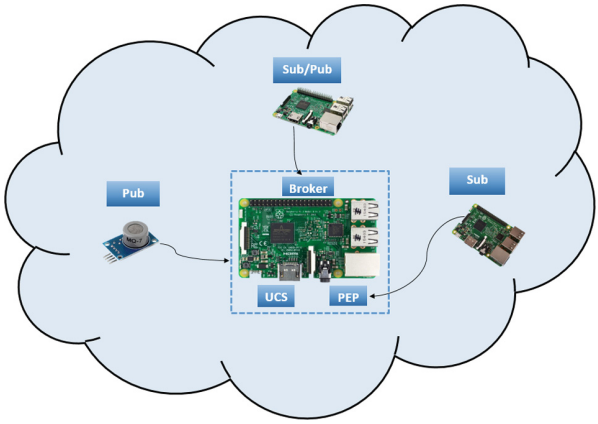

Fig. 5. Testbed logical representation.

In Fig. 5, is depicted the architecture of our testbed. In one Raspberry (central in Fig. 5) we run the Broker which includes the PEP, and the UCS as JARs. The code of the Subscriber[6] and the Publisher[7] were running unmodified in different Raspberries. Furthermore, additional tests have been performed by having the Subscriber host in an Android application called *MyMQTT*, which can be accessed through Google Play. Hence, the Subscriber code can be almost completely executed in the same device of the Publisher. Moreover, it is or the latter can be a small sensor that gives the data to the Broker as shown in Fig. 5. Since the framework is general, none of these configurations affects the functionality or requires any modifications to the framework.

4 Results

To demonstrate the viability of the proposed approach, the overhead introduced by usage control has been measured in a simulated and in a real environment. The framework has been tested in two different environments: the first one is a virtual machine installing Ubuntu 16.04 64-bit, equipped with an Intel i7-6700HQ with 8 cores enabled, 8 GB-DDR4 RAM running in 2133 MHz, the second one is a

[6] https://github.com/pradeesi/MQTT_Broker_On_Raspberry_Pi/blob/master/subscriber.py.

[7] https://github.com/pradeesi/MQTT_Broke_On_Raspberry_Pi/blob/master/publisher.py.

Raspberry Pi 3 with a Broadcom ARRMv7 Quad Core Processor running on 1.2 GHz and 1 GB of LPDDR2 RAM on 900 MHz, running official Raspbian as operative system. The Publisher and the Subscriber were installed in two other Raspberries.

The complete set of results is reported in Table 2. All values have been extracted as the average times computed on 10 runs of the framework in every setting. The first column describes the title of the timings which are all described in milliseconds. The second column describes the timings when the Raspberries are used, and the third one the scenario where we used the Desktop-PC.

Table 2. Result comparison

Event timings (ms)	Raspberry	Desktop
Total tryaccess time	770	91
Total startaccess time	169	26
Total subscription time	969	121
UCON subscription part	939	118
MQTT subscription part	30	3
Total endaccess time	211	27
Unsubscribe time with UCON	213	27
Unsubscribe time without UCON	2	0
Revoke duration in broker	216	27
Revoke duration on UCON	455	41

In Table 2, there are reported the detailed timings, considering a policy with a single attribute. In Figs. 6 and 7 are reported the performance variation at the increase of the number of attributes used in the policy. As shown, the timing behavior is almost linear to the amount of attributes, which is expected, due to the longer time needed to collect a larger number of attributes and for the evaluation performed by the PDP. However, in the real case, even considering 40 attributes, the timings are still acceptable for most of applications. Moreover, it is worth noting that policies with a large number of attributes such as 40 are quite unusual [15].

As expected, the low computational power of the Raspberry alongside the existence of a real network among the MQTT components, explains the longer timings than in the simulated environment. However, also considering a limited amount of attributes which is usual as mentioned above the overhead is slightly bigger than 1 s.

Considering the subscription time, we see that there is some overhead caused be UCS. This is not considered as a constraint because, since the Broker provides a buffer, we can still send all the published messages between the time of the request and the actual acceptance of the Subscriber. This causes no packet loss

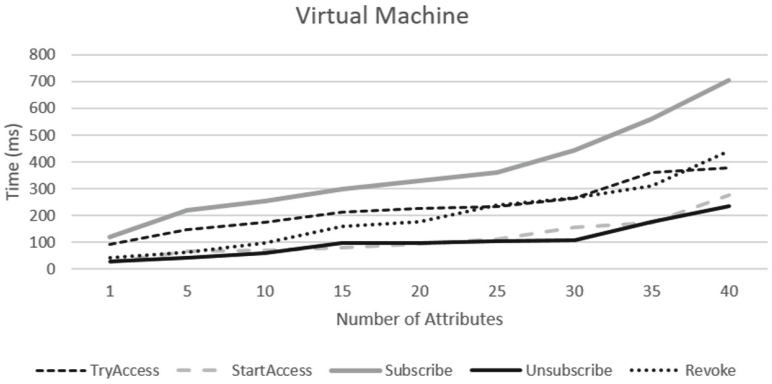

Fig. 6. Timings on the simulated testbed.

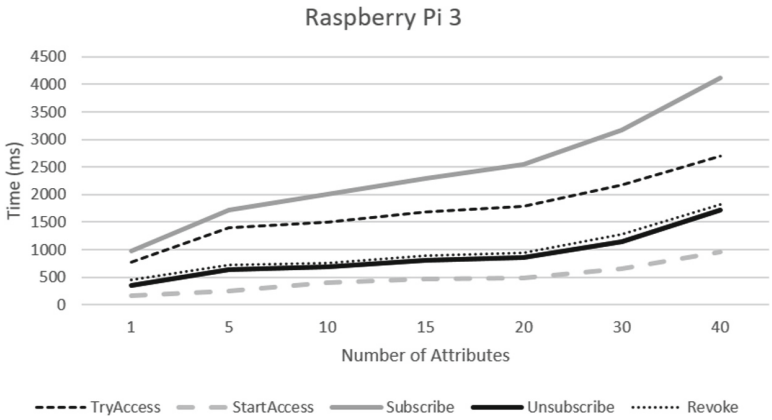

Fig. 7. Performance on the real testbed.

to the Subscriber and high QoS. Furthermore, the most significant time is the one of the revocation. This time is in fact the actual time in which the policy is violated and should be minimized. As shown, this time is equivalent to 216 ms in the real use case and 27 ms in the Virtual Machine, considering a policy with a single attribute. For several applications, this time can be considered as negligible. As shown, the time between a non-valid value is taken and revocation of the access is very small.

Finally, it is worth mentioning that in the ongoing phase, i.e. after a successful `StartAccess`, no delay is introduced by UCON while delivering messages to the Subscribers independently also of the number of attributes.

5 Related Work

IoT is a paradigm which includes applications spanning from e-health to industrial controls. IoT architectures are distributed targeting on constrained devices. The different nature of these devices makes introduction of security mechanisms very difficult, especially when there exists the requirement of dynamic policy (re)evaluation. Although there exist applications of UCON in GRID [16] and Cloud [2] systems, alongside another one Android mobile devices [11], there is only one targeting on IoT [15], where the authors present a modified version of the standard usage control framework, called *UCIoT* that aims to bring UCON on IoT architectures. The architecture is designed to be seamless, configurable and dynamic. However, the authors did not consider any specific IoT protocol. The integration with the MQTT extension we are proposing and further evaluation, could be considered a valuable extension.

In [19], the authors present EventGuard in order to secure generally Publish/Subscribe overlay services. EventGuard is a dependable framework and a set of defense mechanisms for securing such a service. It comprises of a suite of guards to enhance security. But their solution does not target on MQTT but general in these type of protocols which means it is not targeting on constrained devices and protocols for IoT.

The authors of [12], propose a solution to securing Smart Maintenance Services. Their goal is to proactively predict and optimize the Maintenance, Repair and Operations (MRO) processes carried out by a device maintainer for industrial devices deployed at the customer side. They focus on the MQTT routing information asset and they define two elementary security goals regarding the client authentication. Their solution is based on Transport Layer Security (TLS) which is already a basic feature of the protocol. They proposed on how to use it more efficiently as a hardware element. Although they claim that the performance impact is not significant, the adoption of an extra hardware component might be critical in the constrained environment of IoT.

The most significant effort that targets in securing MQTT is a variation of it, called SMQTT [18]. It adds a security feature that is augmented to the existing MQTT protocol based on Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE) using lightweight Elliptic Curve Cryptography. This type of lightweight Attribute Based Encryption, produces extra overhead caused by the time and the computational power and is significant in the constrained environment of IoT. Moreover, our solution, has the advantage of using computational power only in the Broker which by default has sufficient computational power compared to Publishers and Subscribers. However, their implementation needs specific Publishers and Subscribers in order to decrypt the data whereas our solution works silently without being noticed by them and can work with any type of Publishers or Subscribers.

6 Conclusion

Current security mechanisms for IoT protocol are mainly focused on ensuring standard security properties such as message confidentiality and integrity, together with authentication. To the best of our knowledge, up to now the efforts for policies enforcement, which would ensure much more flexible, expressive and effective properties, are still quite limited. In this paper we have presented a first preliminary effort to increase the security of the MQTT protocol, by enabling the dynamic enforcement of usage control policies. We have presented a general methodology which allows to integrate UCON in a seamless way, without requiring protocol modifications. A real implementation has been presented, with performance evaluation to demonstrate the viability of the approach.

As future work, we plan to test the presented framework on a larger testbed with a larger number of attributes for the definition and enforcement of more complex policies, with a possible evaluation in a real applicative setting. Furthermore, we point out that the applied methodology can be easily extended to other IoT application protocols, where the benefits of integration are worth to be investigated in future works.

Acknowledgments. This work has been partially funded by EU Funded projects H2020 C3ISP, GA #700294, H2020 NeCS, GA #675320 and EIT Digital HII on Trusted Cloud Management.

References

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutorials **17**(4), 2347–2376 (2015, fourthquarter)
2. Carniani, E., D'Arenzo, D., Lazouski, A., Martinelli, F., Mori, P.: Usage control on cloud systems. Future Gener. Comput. Syst. **63**(C), 37–55 (2016)
3. Chen, D., Varshney, P.K.: QoS support in wireless sensor networks: a survey (2004)
4. Colitti, W., Steenhaut, K., De Caro, N., Buta, B., Dobrota, V.: Evaluation of constrained application protocol for wireless sensor networks. In: 2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN), pp. 1–6, October 2011
5. Collina, M., Corazza, G.E., Vanelli-Coralli, A.: Introducing the QEST broker: scaling the IoT by bridging MQTT and REST. In: 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 36–41, September 2012
6. Faiella, M., Martinelli, F., Mori, P., Saracino, A., Sheikhalishahi, M.: Collaborative attribute retrieval in environment with faulty attribute managers. In: 2016 11th International Conference on Availability, Reliability and Security (ARES), pp. 296–303, August 2016
7. Fysarakis, K., Askoxylakis, I., Soultatos, O., Papaefstathiou, I., Manifavas, C., Katos, V.: Which IoT protocol? Comparing standardized approaches over a common M2M application. In: 2016 IEEE Global Communications Conference (GLOBECOM), pp. 1–7. IEEE (2016)

8. Karagiannis, V., Chatzimisios, P., Vzquez-Gallego, F., Alonso-Zrate, J.: A survey on application layer protocols for the internet of things. Trans. IoT Cloud Comput. **1**(1), 11–17 (2015)
9. Karopoulos, G., Mori, P., Martinelli, F.: Usage control in SIP-based multimedia delivery. Comput. Secur. **39**, 406–418 (2013)
10. Lazouski, A., Martinelli, F., Mori, P.: Survey: usage control in computer security: a survey. Comput. Sci. Rev. **4**(2), 81–99 (2010)
11. Lazouski, A., Martinelli, F., Mori, P., Saracino, A.: Stateful data usage control for android mobile devices. Int. J. Inf. Secur. **16**(4), 345–369 (2017)
12. Lesjak, C., Hein, D., Hofmann, M., Maritsch, M., Aldrian, A., Priller, P., Ebner, T., Ruprechter, T., Pregartner, G.: Securing smart maintenance services: hardware-security and TLS for MQTT. In: 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), pp. 1243–1250, July 2015
13. Locke, D.: MQ telemetry transport (MQTT) v3. 1 protocol specification. IBM developerWorks Technical Library (2010)
14. Luzuriaga, J.E., Perez, M., Boronat, P., Cano, J.C., Calafate, C., Manzoni, P.: A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. In: 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), pp. 931–936, January 2015
15. La Marra, A., Martinelli, F., Mori, P., Saracino, A.: Implementing usage control in internet of things: a smart home use case. In: 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, 1–4 August 2017, pp. 1056–1063 (2017)
16. Martinelli, F., Mori, P.: On usage control for GRID systems. Future Gener. Comput. Syst. **26**(7), 1032–1042 (2010)
17. NIST: MQTT and the NIST Cybersecurity Framework Version 1.0 (2014). http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/cn01/mqtt-nist-cybersecurity-v1.0-cn01.pdf. Accessed 22 Jan 2017
18. Singh, M., Rajan, M.A., Shivraj, V.L., Balamuralidhar, P.: Secure MQTT for internet of things (IoT). In: 2015 Fifth International Conference on Communication Systems and Network Technologies, pp. 746–751, April 2015
19. Srivatsa, M., Liu, L.: Securing publish-subscribe overlay services with EventGuard. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, pp. 289–298. ACM, New York (2005)
20. Talaminos-Barroso, A., Estudillo-Valderrama, M.A., Roa, L.M., Reina-Tosina, J., Ortega-Ruiz, F.: A machine-to-machine protocol benchmark for eHealth applications use case: respiratory rehabilitation. Comput. Methods Programs Biomed. **129**, 1–11 (2016)
21. Thangavel, D., Ma, X., Valera, A., Tan, H.-X., Tan, C.K.-Y.: Performance evaluation of MQTT and CoAP via a common middleware. In: 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), pp. 1–6. IEEE (2014)
22. Villari, M., Celesti, A., Fazio, M., Puliafito, A.: AllJoyn Lambda: an architecture for the management of smart environments in IoT. In: 2014 International Conference on Smart Computing Workshops, pp. 9–14, November 2014