Implementing Usage Control in Internet of Things: A Smart Home Use Case

Antonio La Marra, Fabio Martinelli, Paolo Mori, Andrea Saracino
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Pisa, Italy
Email: name.surname@iit.cnr.it

***Abstract*—Internet of Things (IoT) is a paradigm which has become extremely popular, with applications spanning from e-health to industrial controls. IoT architectures are distributed and often based on constrained devices, which make challenging the task of introducing security mechanisms, in particular those requiring dynamic policy evaluation. In this paper we present UCIoT (Usage Control in IoT), a fault tolerant and adaptable framework for the enforcement of usage control policies in IoT environments. UCIoT brings the functionalities of a U-XACML-based usage control framework on a decentralized, distributed and Peer-to-Peer (P2P) architecture. In the present work, we describe an application of UCIoT in a Smart-Home environment, presenting also two possible use cases where usage control is exploited to implement a policy for energy saving and a policy for safety. A set of experiments on real devices is finally presented to report the performance of the system, measuring the overhead introduced by the UCIoT framework.**

I. Introduction

In the last years, connected objects are increasingly becoming pervasive in our daily life. After that smartphones and tablets have brought connectivity in our pockets, allowing users to be seamless connected also in mobility, connected objects are now rapidly entering our houses. "Gartner, Inc. forecasts that 8.4 billion connected things will be in use worldwide in 2017, up 31 percent from 2016, and will reach 20.4 billion by 2020. Total spending on endpoints and services will reach almost $2 trillion in 2017."[1] These devices are the core components of a new paradigm known as Internet of Things (IoT), which becomes *smart-home* when the devices including sensors and actuators are installed in a house. Smart house appliances and smart home devices are gradually entering our houses, replacing the traditional ones. The first devices hitting the shelves were smart-TVs, which have been followed by smart refrigerators, smart thermostats, smart ovens, smart cameras, smart lights, smart showers, and many others. These devices, differently from traditional ones, have the possibility of running an operative system (such as Google Android), which in several cases allows the installation of additional applications, for added functionalities and customizability.

However, the connection capabilities of smart devices, as well as the capability of installing third party software, expose devices, users and the whole environment (house in case of smart home setting) to potential security and safety issues. The security and safety risks can be mitigated by adding configurable systems for security policy enforcement. The need for the introduction of Access Control and Usage Control mechanisms in IoT and smart home environment has been anticipated by relevant and recent works in literature such as [3], [17] and [12], reporting also the challenges of introducing these mechanisms on distributed architectures and constrained devices, typical of the IoT.

In this paper we present UCIoT (Usage Control in the Internet of Things), a framework which aims at bringing the Usage Control on IoT architectures in a seamless, configurable and dynamic manner. The framework is designed for heterogeneous and distributed architectures of connected devices, evaluating and enforcing security, safety or general purpose policies by exploiting the expressive U-XACML language [5]. In particular, this work focuses on an implementation of UCIoT in a smart home environment, presenting environment specific policies and reporting experiments on a testbed comparable in performance and dimension to a smart home setting.

The contribution of this paper is summarized in the following:

- We define UCIoT, a distributed, decentralized, fault tolerant and infrastructure independent framework to implement Usage Control functionalities in IoT systems;
- We discuss an implementation of the UCIoT framework in a smart home environment, also introducing two policies for energy saving and physical safety which can be enforced through UCIoT;
- We report details on the implementation of the UCIoT framework, which exploits the CHORD-based *Apache Cassandra* Distributed Hash Table for enabling the distributed communication and data storage, reporting the challenges and implementation choices to decentralize the UCON functions.
- We present a set of experiments to measure performance of the proposed framework, to measure the performance overhead caused by the UCIoT framework on a real implementation performed on Raspberry PI-3[2] devices.

The remainder of the paper is organized as follows. In Section II describes the UCON components and workflow phases. Section III describes the UCIoT framework, detailing the

[1]Gartner Inc., http://www.gartner.com/newsroom/id/3598917

[2]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

2324-9013/17 $31.00 © 2017 IEEE
DOI 10.1109/Trustcom/BigDataSE/ICESS.2017.352

components and the workflow and implementation choices to adapt UCON components to a distributed architecture. Section IV reports the performance evaluation of UCIoT documenting the implementation on a testbed of real devices. Section V reports a set of related work on access and usage control in IoT environments. Finally Section VI briefly concludes proposing some future directions.

II. The Usage Control Model

The Usage Control (UCON) model [14], [16] extends traditional access control models introducing *mutable attributes* and new decision factors besides *authorizations*: *obligations* and *conditions*. Mutable attributes represent features of subjects, resources, and environment that change their values as a consequence of the normal operation of the system [15]. For instance, some mutable attributes change their values because the policy includes attribute update statements that are executed before (*pre-update*), during (*on-update*), or after (*post-update*) the execution of the access. For instance, the e-wallet balance is a subject attribute which could be decreased by the policy every time the subject performs a new access to a resource.

Since mutable attributes change their values during the usage of an object, the usage control model allows to define policies which are evaluated before *(pre-decision)* and continuously during the access to the object (*ongoing-decision*).

The continuous evaluation of the policy when the access is in progress is aimed at executing proper countermeasures (such as 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 attribute change by taking proper actions, e.g., by interrupting those ongoing accesses which are no longer authorized.

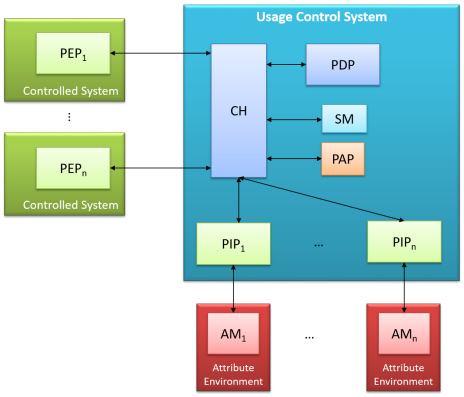

Fig. 1: Usage Control System architecture.

This paper takes into account Usage Control systems based on the XACML reference architecture [13], with particular reference to the one we presented in [2], [10], which is shown in Figure 1. In the XACML reference architecture, the Policy Enforcement Points (PEPs) embedded in the controlled system intercept the execution of security relevant operations, and they invoke the Context Handler (CH), which is the frontend of the Usage Control system. The *Policy Information Points (PIPs)* are the components invoked by the CH to retrieve the attributes required by the Policy Decision Point (PDP) for the execution of the decision process, i.e., to evaluate the policy retrieved from the Policy Store (PS). Attributes are managed by Attribute Managers (AMs), sometimes called Attribute Providers or Attribute Stores, which provide the interfaces to retrieve and, in case of mutable Attributes, to update their current values. Each specific scenario where the Usage Control system is exploited requires its own set of AMs to manage the attributes required for the policy evaluation. Hence, PIPs are properly configured in order to be able to query the specific AMs adopted in the scenario of interest for retrieving and updating attributes. In particular, each PIP implements the specific protocol required to interact with the related AM and exploits the provided mechanisms for securing the communications. The Usage Control model emphasizes the role of PIPs because it introduces the continuous policy enforcement while an access is in progress to cope with mutable attributes. In particular, the PIP is also in charge to detect when the value of an attribute changes in order to trigger the policy re-evaluation for the involved ongoing accesses, which are managed by the Session Manager (SM). To detect attribute changes, the PIP could exploit the subscription mechanism provided by the AM or the PIP must emulate it if it is not supported by the AM.

The phases of the Usage Control decision process are regulated by the interactions between the PEP and the Usage Control systems as follows (derived from [19]):

TryAccess: is the *pre-decision* phase, which begins when the Tryaccess message is sent by the PEP to the Usage Control system because a subject requests to execute the access. The TryAccess phase finishes when the Usage Control system sends the response to the PEP. The possible responses are: PERMIT, to allow the access, or DENY;

StartAccess: is the first part of the *ongoing-decision* phase, which begins when the StartAccess message is sent by the PEP to the Usage Control system because the access has just started, and finishes when the policy has been evaluated and the response has been sent back to the PEP;

RevokeAccess: this is the second part of the *ongoing-decision* phase. This phase is executed every time an attribute changes its value. This phase starts when an attribute changes its value. It finishes when the policy has been evaluate and, if a policy violation occurs, the RevokeAccess message is sent by the Usage Control system to the PEP.

When the subject s tries to execute a security relevant action a, the PEP suspends its execution and retrieves the information related to this access (subject and resource IDs, etc.). The PEP sends the *TryAccess* message with the data previously collected to the Usage Control system system, which performs the *pre-decision* process and returns the result to the PEP, which enforces it. If the execution of a is permitted, the PEP

sends the *StartAccess* message to the Usage Control system as soon as *a* is started, to start the *on-decision* phase. Again, the Usage Control system performs the first evaluation of the Usage control policy. From this moment on, as long as the action *a* is in progress, the Usage Control service evaluates the Usage Control policy every time an attribute changes its value, and we call this phase *RevokeAccess*. If the policy is violated, the Usage Control system sends the *RevokeAccess* message to the PEP, in order to take proper countermeasures.

III. The UCIoT Framework

This section describes the proposed framework which implements Usage Control on the IoT devices installed in a Smart Home scenario. Though the paradigm of Usage Control has been already successfully applied in several scenarios bringing, for example, the Usage Control System (UCS) in Cloud [2], and on Android devices [11], the present work faces the challenge of decentralizing the UCS functionalities proposing a distributed a Peer-to-Peer (P2P) architecture for constrained devices, such as the Smart Home ones. In particular, the majority of devices that we are considering, Smart Cameras, Smart TVs, Smart Thermostat, Smart Lights, Smart Oven, have limited computational power and storage capabilities, hence they are not suitable to handle and evaluate all the requests which might be issued in a smart home environment, nor to store needed information for policy reevaluation for all the active sessions. As will be detailed in the following, UCIoT partially replicates the functionalities of the UCS on each P2P node. Also, any policy evaluation can be performed by any of the nodes which can exploit the attributes collected from the other nodes. These two features enable respectively fault tolerance and higher complexity for enforced policies.

Another relevant advantage of such a distributed system concerns the fault tolerance aspects, because the failure of one (or of a number of) device(s) can be tolerated as long as the system can be executed on the remaining ones.

Moreover, being distributed and exploiting several devices, UCIoT allows the Usage Control framework to exploit the data collected from the sensors of all these devices, thus allowing to define more complex policies.

A. Architecture

The architecture of UCIoT framework, depicted in Figure 2, is a distributed P2P framework, where each node represents a smart device logically connected to all the others through a Distributed Hash Table (DHT) [1]. In particular, the DHT exploited by UCIoT is the Cassandra protocol [8], based on a modified version of the CHORD DHT, which also includes a non-relational distributed database, used by several popular applications such as Facebook.

We consider as *smart device* any house appliance, sensor, actuator, board, computer, smartphone, or other device which is running an operative system which is able to install and run third party applications. Every smart device will be thus considered a P2P node, whilst connected devices not matching these specifications will be considered as peripherals of a

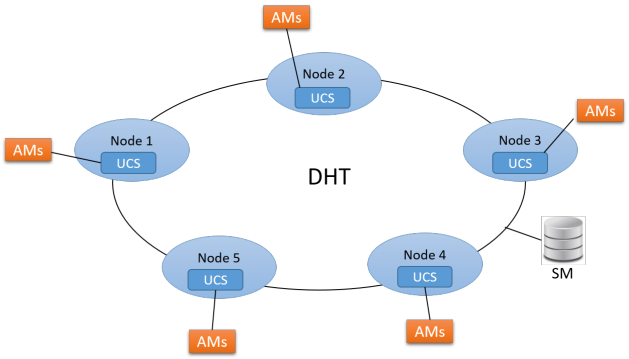

Fig. 2: Logical architecture of UCIoT.

specific node, logically incorporated in the node itself. Some elements of the smart devices environment could even have also an acceptable computational power, a consistent power autonomy, still with limited storage space. Given the capabilities of smart devices, the UCS can be installed as a third party app and run on any smart device in UCIoT.

At network and datalink layer, the used communication protocol is ad-hoc wireless which, being completely distributed, does not rely on the presence of a single router node, which might represent a single point of failure in the network. Thus, thanks to the DHT protocol and to the ad-hoc wireless, communication among smart devices will still be possible if a limited subset is switched off or malfunctioning. The network protocol is completely oblivious to the applications running on the smart devices, thus smart devices will exchange messages as if they are connected to the same Wireless-LAN.

A representation of the logical architecture components of UCIoT in a single node is shown in As anticipated, the

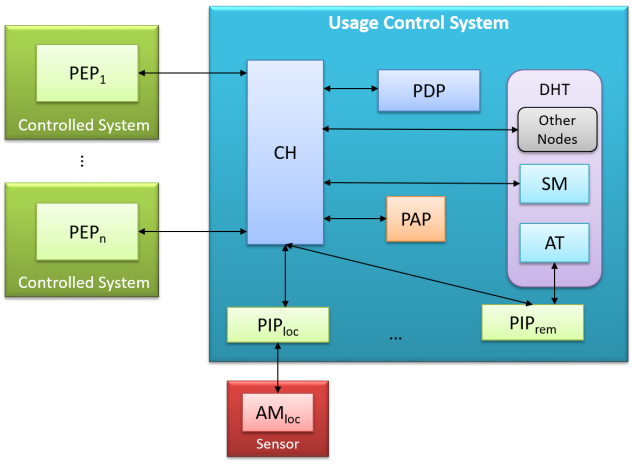

Fig. 3: Logical architecture of a UCIoT node.

functionalities of UCS are replicated on every device, which can thus decide to permit or deny the access/usage to an operation, or a resource that it controls. This request can be issued by physical users, other smart or peripheral devices and

is matched against usage control policies stored locally by the UCS. More specifically, every smart device runs (i) the Policy Administration Point (PAP) to store policies, (ii) an instance of the Policy Decision Point (PDP) to match request and policies and deciding to permit or deny the resource usage, (iii) a set of Policy Information Points (PIP) to query local attributes, (iv) a Context Handler (CH) to manage the interaction among the various components. The main difference with the UCON architecture presented in our previous works lays in the Session Manager, for which UCIoT take advantage of the distributed database offered by Cassandra. In fact, since smart devices have limited memory storage and the number of managed sessions will be likely unbalanced among devices in a smart home settings, the set of active sessions are saved in the DHT, to efficiently use the global memory storage offered by all the devices. Moreover, thanks to the configurable replication factor, the SM is not prone to single point of failure issues. The other main difference with previous UCON architectures is the presence of *remote attributes*. An attribute is considered *local* to a node when the Attribute Manager (AM) is queried directly by one of the PIPs belonging to the UCS of that node (see Figure 2). However, in UCIoT is possible that not all attributes needed to evaluate a policy are local, i.e. some attributes are local to other smart devices and are thus considered remote. To abstract this procedure to the UCS local to the evaluating device, an abstract component named *remote PIP* is added to all UCSs, and it is in charge of interacting with the remote smart device to retrieve the attribute. The remote PIP retrieves the identifier of the node physically connected to the AM by the attribute table stored in the DHT, then retrieves the attribute according to the procedure described in the following subsection. r, we defined two distinct strategies to collect the remote attributes required to perform the decision process.

B. Workflow

The operations performed by UCIoT are equivalent to the one performed by the UCON framework described in Section II. The main difference in the workflow is introduced by the presence of remote attributes and by the distributed Session Manager. UCIoT has been designed with the aim of reducing as much as possible the differences in the workflow with respect to the standard UCON workflow, despite the distributiveness of the architecture where it is applied.

As anticipated, the SM is stored on the Apache Cassandra DHT database, which brings the advantages of increased storage space, tackling thus the issue of memory constrained devices, and ensures fault tolerance thanks to the replication factor, storing data of sessions related to temporarily unavailable devices. The DHT database also masks the underlying distributed architecture of the database to the application level, leaving thus the functionalities of the SM unmodified.

On the other hand, remote attributes introduce some additional challenges, requiring thus a small alteration of the usual workflow and UCON architecture. As anticipated, the problem of remote attributes consists of collecting the value of attributes from AMs connected to a UCS node different from the one evaluating the policy. To this end, we introduce in each node a component which abstracts the procedure of collecting the remote attributes, named *PIP remote*. This PIP exploits the *Attribute Table* stored on the DHT which memorizes, for each attribute, the node whose AM is physically connected to. Hence, when a remote attribute is needed, the local CH reaches through the DHT the CH of the interested node (*CH rem* in Figure 4), which will instruct the local PIP (*PIP loc* in Figure 4) to retrieve the attribute value, which is then returned to the local CH.

The complete workflow of the retrieval of a remote attribute is depicted in Figure 4. In particular, Figure 4 represents the

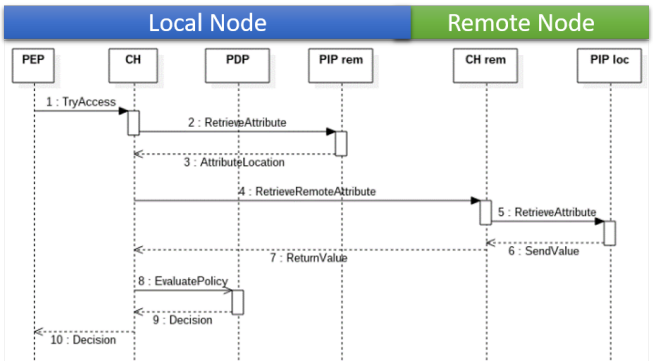

Fig. 4: TryAccess sequence diagram with remote attribute retrieval.

workflow of the TryAccess request issued by the PEP, detailing the message exchange between the various components. For the sake of simplicity and clarity of representation, no local attributes are retrieved in the represented workflow. The remote attribute retrieval is performed also for the StartAccess, which is identical to the TryAccess in workflow, except for the addition of the started session to the distributed SM and for the subscription of the remote CH to the remote mutable attributes. Thus, if a remote attribute changes its value the remote CH is notified, and the new value is sent back to the local CH for policy reevaluation and possible revocation.

C. Implementation

The current implementation of the UCIoT framework consists of a Java application, shipped in the form of a `jar` file, which can be installed on any device running a JRE. When installed, the application will instantiate on the devices the DHT for the distributed database and to handle communication with other nodes belonging to the same network. The installed application includes the full UCS, where the CH and PDP are not altered with respect to the standard UCON model. The SM is installed in the distributed database handled by the DHT, which will also handle the data replication. The distributed database also hosts the attribute table, which keeps the correspondence among the attributes and the node to which the AM is physically connected, hence it can be queried and possibly updated by any node. As it happens in the standard UCON framework, PIPs and PEPs will be device specific,

to be interfaced with actuators and sensors proper of the specific device. The single instances of the UCIoT application are dynamically configurable and the code related to specific PIPs and PEPs can be loaded at runtime by exploiting Java reflection.

D. Relevant Use Cases

Several distinct Smart Home scenarios can be successfully addressed exploiting the proposed framework. As a matter of fact, several kind of policies can be defined and enforced through the proposed framework, such as safety and security policies, energy saving policies, and so on. For example the following safety policy can be defined and enforced with the proposed system: "The burner of a smart oven can be on if the system is healthy, and if there is at least an adult in the kitchen when a child is also present."A schematic representation of this use case is shown in Figure 5. This policy aims at protecting

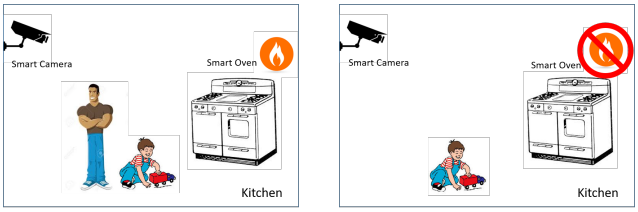

Fig. 5: Smart oven safety and parental control use case.

the house from gas leak and reducing the possibility that a child gets hurt or burned by touching the oven, due to the missing surveillance of an adult. This safety policy requires the presence of a smart oven[3], able to check that there are not physical failures such as gas leak, which enforces the usage control policy preventing the burners to be lighted and by turning off them, as soon as the policy is not matched anymore. It is also required the presence of one or more smart cameras in the kitchen, to detect who is present, which is also able to discern among children and adults. Implementing UCIoT in this specific scenario, will consider both the oven and the smart camera as nodes of the UCIoT architecture. In particular, the aforementioned policy will be evaluated and enforced on the oven node which will thus include the PEP, whose actuator is the burner controller valve. Evaluating the policy will require thus a local attribute, i.e. the health check to avoid gas leak, and a pair of remote attributes i.e. the number of people present in the kitchen and the number of children, which are provided by the camera.

An example of energy saving policy could instead be the following: "The air-cooling can be on if there are people in the house and there are no open windows." A schematic representation is again reported in Figure 6. In this case we envision as sensors a set of cameras and/or presence sensors to verify that someone is effectively present in the house, and windows sensors to check whether one or more windows are open. To implement UCIoT in this scenario, the UCS will be

[3]e.g.: http://www.dacor.com/Products/Ranges

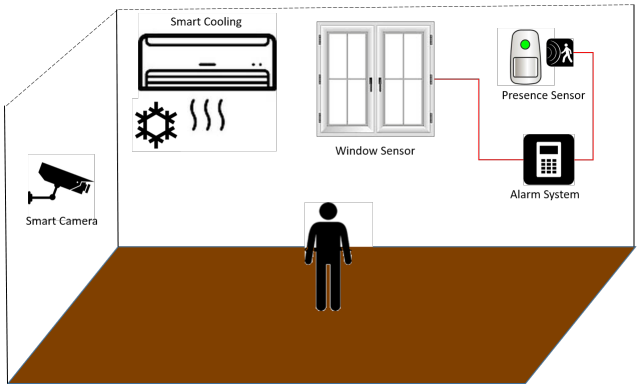

Fig. 6: Setting for energy saving policy based on smart air cooling control.

installed inside the air cooling system, the anti-theft system of the house and possibly on smart cameras. Hence, the PEP will directly interact with the air cooling controller, preventing it from switching on, or switching it off when the policy is not matched. The anti-theft will collect the information from the presence and window sensors to verify that no windows are opened and that people are effectively in the house. Additional information on the presence of people, will be provided, as anticipated by cameras. Thus, the policy will be evaluated and enforced by the UCS local on the air cooling system (air conditioner), by exploiting the remote attributes collected by the PIPs installed on the anti-theft and cameras.

IV. Experimental Evaluation

As shown, by decentralizing the UCON functionalities, we enabled the continuous evaluation and enforcement of UCON policies in constrained devices. The execution of the UCON framework on such devices and, in particular, the process of retrieving remote attributes from the other nodes and session data from the DHT, introduces overhead. To quantify such overhead and to evaluate its impact on usage in real system, we conducted on our reference testbed a set of experiments aimed at measuring the performance of the proposed system and the results are discussed in the following.

A. The testbed

As anticipated, the testbed consists of five Raspberries PI 3 Model B and is shown in Figure 7. As shown, three devices are equipped with the Raspberry SenseHat module, which adds to the simple board sensing capabilities, incorporating sensors for temperature, humidity, pressure, magnetometer, accelerometer and gyroscope. The SenseHat also features a led matrix which can be used to display messages. The two remaining devices have been equipped with a Pi Camera Module v2, adding thus the capability of collecting pictures and video. The Raspberry PI 3 Model B has the following features: 1 GB RAM, 1.2 GHz ARM processor, VideoCore IV 3D graphics core and has several interfaces for wireless communication, namely 802.11n WLAN, Bluetooth 4.0 and Bluetooth Low Energy (BLE). For

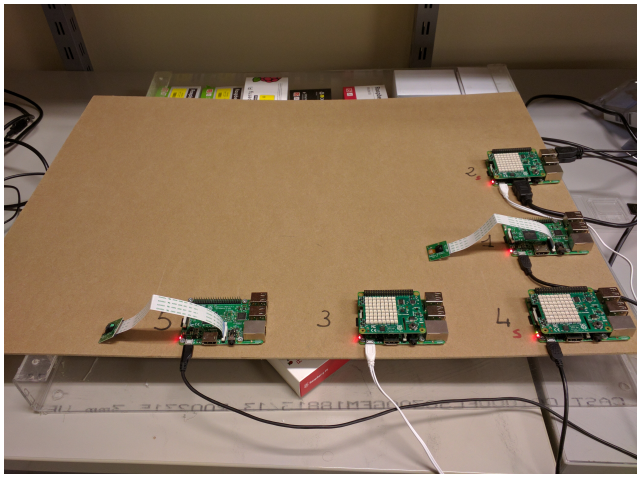

Fig. 7: Five Raspberries PI 3 used as testbed equipped with SenseHat and Pi Camera.

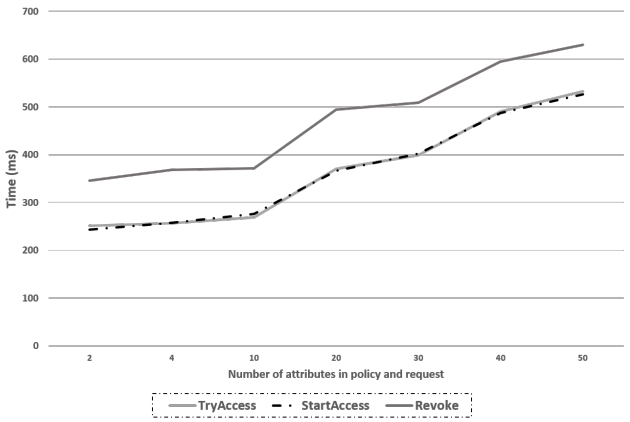

Fig. 8: Performance with local attributes.

our set of experiments, devices are interconnected through an ad-hoc WiFi network, constituting thus a WANET with AODV routing protocol [4]. The choice of using WiFi instead of Bluetooth is due to the higher reliability and greater speed of WiFi, which also does not require pairing phases as it happens in Bluetooth. Also, being all devices connected to a power supply, in the smart home use case it is not necessary to use low consumption protocols. This hypothesis is sound since in a smart home, house appliances are generally connected to power supply and do not rely on batteries. However, it is worth noting that the UCIoT framework is completely independent from the routing and data-link protocol.

Sensors and cameras are used in our testbed as AMs, providing the values for attributes which will be used in policies. The led matrix is instead used as actuator, hence is commanded by the PEP, showing a “P ”every time a session starts, i.e., PDP returns the PERMIT decision for a StartAccess and “D ”if the access is denied or revoked.

B. Local Evaluation

The first set of experiments, whose time performance is shown in Figure 8, evaluates the time required to execute the phases of the policy evaluation process, namely: TryAccess, Startaccess, and RevokeAccess (see Section II). The experiments have been performed enforcing policies including local attributes only, varying the number of the attributes required for the decision process from 2 to 50. The time of the TryAccess phase is measured from the moment when the access request is sent by the PEP to the CH to the moment when the PEP receives the response (permit/deny) to this request. This time affects the user experience because it is the delay introduced by our framework in the utilization of the smart home device. The time of the StartAccess phase is measured from the moment when the evaluation of the policy is triggered by the PEP because the access just began, to the moment when the response has been received by the PEP. The time to perform the RevokeAccess phase is measured from the moment when a given attribute changes its value to the moment when the PEP receives the revocation response. Hence, it also includes the time required by the PIP to detect that an attribute changed its value. The StartAccess and RevokeAccess phases do not directly introduce any delay in the utilization of the resource because they are executed while the access to the resource is already in progress. When a policy violation is detected, the time required by the StartAccess phase or by the RevokeAccess one represent the interval while the resource has been used without holding the related right. The X axis reports the number of local attributes, *nLA*, while the Y axis reports the time in milliseconds. For measuring the time taken by the RevokeAccess phase, we supposed that only one of the sessions stored in the DHT requires the re-evaluation of the policy.

The results of our experiments show that, in case of 2 local attributes, the time required to perform the TryAccess phase is 252 ms, the StartAccess phase takes 247 ms while the RevokeAccess phase requires 346 ms. In case of 50 local attributes, instead, the TryAccess phase takes 533, the StartAccess phase requires 528 ms, while the time to execute the RevokeAccess phase is 630 ms. First of all, we notice that the time to execute the TryAccess phase is quite the same as the time required for the StartAccess phase. The reason is that the workflows of the two phases are very similar, i.e., they perform quite the same operations in the same order. Hence, in case of local attributes, the delay introduced by the UCIoT framework would not affect the user experience, and the usage of the resource would be revoked in a short time in case of policy violation. Moreover, the difference between the time of the RevokeAccess and of the TryAccess or StartAccess, for all the attribute values, is mainly due to the time required to detect that an attribute changes its value.

C. Remote attribute retrieval

The next set of experiments, shown in Figure 9, is aimed at evaluating the time required to execute the TryAccess and RevokeAccess phases of the policy evaluation process in case

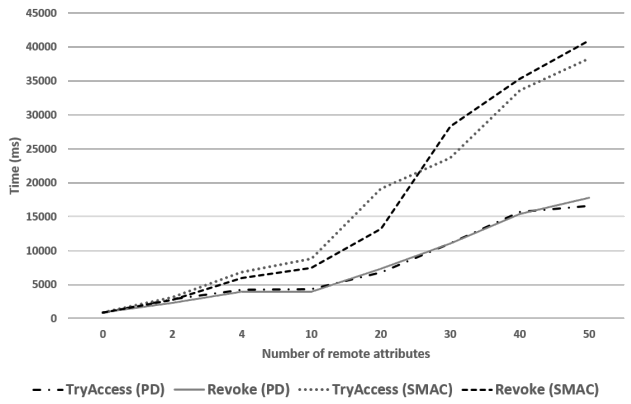

Fig. 9: Performance with local and remote attributes.

of policies including remote attributes. We omit the time required for the StartAccess phase since the results of the first set of experiments show that it is quite the same as the time of the TryAccess phase. In this set of experiments, the policy always includes 50 attributes, and we vary the ratio between local and remote attributes. The policy is written in such a way that the value of all the 50 attributes have always to be collected and evaluated to perform the decision process, i.e., the UCIoT PDP cannot perform the decision phase exploiting only a subset of the 50 attributes. We recall that real policies would embed a smaller number of attributes, hence this can be considered as a worst case test. For instance, the first policy described in Section III-D embeds 3 attributes only.

On the X axis of the graph in Figure 9 we report the number of remote attributes in the enforced policy, *nRA*. Hence, the number of local attributes *nLA* is given by $(50-X)$. In this case too, for measuring the time taken by the execution of the RevokeAccess phase, it is supposed that only one session requires the re-evaluation of the policy. The experiments have been conducted exploiting two distinct strategies for retrieving the remote attributes described, namely *Policy Driven* (PD) and *Sequential Missing Attribute Collection* (SMAC). The first strategy consists of making the CH aware of the policy to be analyzed, to have a list of the attributes to be queried by the various PIPs. Hence, the list of attributes is matched with the Attribute Table stored by the PIP remote, to find the list of remote attributes, which are then queried temporarily. The SMAC strategy instead does not perform the lookout in the attribute table, it simply send the request to the local PIPs to enrich it and then asks the PDP to evaluate the enriched request. The PDP will hence return eventual missing attributes present in the policy, which are queried by means of the PIP remote. This strategy thus, has the advantage of postponing the lookout to the attribute table, which is not performed at all if the needed attributes are all local. The lookout is instead always present in the PD strategy, which however shows better performance when the number of remote attributes is considerable. In fact, in the SMAC strategy, the PDP returns one missing attributes per time, requiring thus several consequential lookups and policy reevaluations. This difference in performances is well shown in Figure 9.

We observe that, when all the attributes are local (i.e., $nLA = 50, nRA = 0$), the time required for the execution of the TryAccess and RevokeAccess phases is, respectively, about 533 and 630 ms, with no measurable differences among the two strategies. Hence, the AT lookout time is negligible.

In case of policies with 10 remote attributes (i.e., $nLA = 40, nRA = 10$), if we adopt the PD strategy the time required for the execution of the TryAccess phase is 4,2 seconds, while the time for the RevokeAccess phase is 3,9 seconds. Adopting the SMAC strategy the times required for the execution of the two phases are, respectively, 8,8 and 7,7 seconds. Instead, when the policy embeds remote attributes only (i.e., $nLA = 0, nRA = 50$), the time required for the execution of the StartAccess and RevokeAccess phases is, respectively, about 16,6 and 17,8 seconds if we adopt the PD strategy, while it is about, respectively, 38,3 and 40,9 seconds if we adopt the SMAC strategy.

D. Network delay evaluation

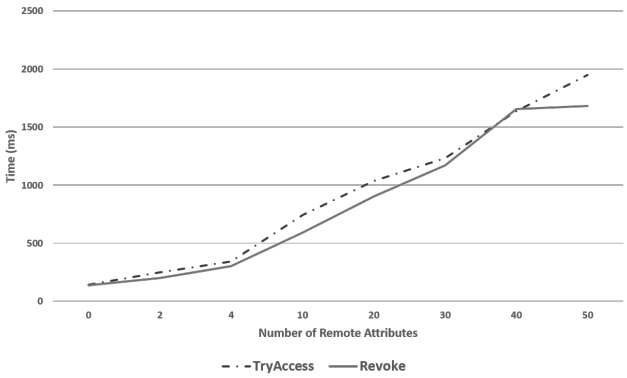

Fig. 10: Performance on emulated devices.

The aim of the third set of experiments, shown in Figure 10, is to evaluate the the overhead introduced by the communications over the network. In particular, we installed the 5 nodes on distinct Virtual Machines running on the same physical machine, and we measured the time for executing the TryaAccess and RevokeAccess phases in case of policies including remote attributes. In this experiments we adopted the PD strategy for the collection of remote attributes.

The results show that, even in case of 50 remote attributes, the time to perform the TryAccess and RevokeAccess phases is less than 2 seconds. Compared with the results of Figure 9 these experiments confirm that the time required for the communications over the network is the main factor which affects the overhead introduced by the UCIoT framework.

V. Related Work

The need of introducing access control in IoT environments has recently been discussed by V. Cerf in [3], where he also proposed some directions based on the idea of introducing

controlling and verification mechanisms on edge devices. UCIoT extends this vision, by introducing the more expressive UCON model and presenting an architecture which can be easily integrated in both edge and internal devices, given they have smart device requirements. The author of [18] present a model to include access control in constrained devices to be used in IoT. The work mainly focuses on the introduction in the COAP protocol of an overlay to ensure authenticated access. On the other hand, UCIoT focuses on usage control acting at application level, without modifying the standard communication protocol. In [17], a set of challenges for security and privacy in IoT is discussed. The paper is not proposing solutions to the presented issues, still it supports the claim that a distributed solution, like UCIoT would be more effective in the IoT environment. The work in [9] presents a distributed and decentralized architecture for data usage control. Differently from UCIoT, this work is focused on controlling the right to access and use data in multi-domain environment, attempting to ensure that policies are respected even after data are moved in a different environment with different control mechanisms. The work in [6] presents a methodology to enforce data usage control in IoT, to enable policy protected information sharing, exploiting semantic web technologies to derive the current context. However, not real implementation are presented in this work, which, differently from UCIoT is not focused on the control of operations not related to data sharing. An architecture featuring distributed UCS is presented instead in [7], where different UCS homed on different systems cooperate to find the most reliable value of a common attribute, whose AM is temporarily unavailable. In the paper is presented an application to smart cities environment, still the focus of this paper is on the reputation algorithm used to measure the reliability of exchanged information.

VI. Conclusion and Future Work

Introducing security in IoT is a challenging task, which needs to cope with the distributed, decentralized and constrained nature of the IoT architectures. In this paper we have proposed UCIoT a framework to introduce Usage Control on IoT architectures, which leverages on the distributed nature of IoT systems to provide a flexible, dynamic, fault tolerant and adaptable framework able to enforce usage control policies. The paper showed how UCON can be exploited to implement safety and energy saving policies and demonstrated the feasibility through implementation on real devices and performance experiments. Several future work direction might stem from this prototype, in particular we plan to further optimize performance to make the system able to handle policies with several remote attributes, by parallelizing the process of attribute retrieval. Furthermore additional experiments will be performed in a larger and more heterogeneous environment, evaluating performance with different routing strategy and multi-hop use cases.

Acknowledgement

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] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, February 2003.

[2] Enrico Carniani, Davide D'Arenzo, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Usage control on cloud systems. *Future Generation Comp. Syst.*, 63:37–55, 2016.

[3] V. G. Cerf. Access control and the internet of things. *IEEE Internet Computing*, 19(5):96–c3, Sept 2015.

[4] I. D. Chakeres and E. M. Belding-Royer. Aodv routing protocol implementation design. In *24th International Conference on Distributed Computing Systems Workshops, 2004. Proceedings.*, pages 698–703, March 2004.

[5] Maurizio Colombo, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. A proposal on enhancing XACML with continuous usage control features. In *Grids, P2P and Services Computing [Proceedings of the CoreGRID ERCIM Working Group Workshop on Grids, P2P and Service Computing, 24 August 2009, Delft, The Netherlands]*., pages 133–146, 2009.

[6] P. K. Das, S. Narayanan, N. K. Sharma, A. Joshi, K. Joshi, and T. Finin. Context-sensitive policy based security in internet of things. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, May 2016.

[7] Mario Faiella, Fabio Martinelli, Paolo Mori, Andrea Saracino, and Mina Sheikhalishahi. Collaborative attribute retrieval in environment with faulty attribute managers. In *11th International Conference on Availability, Reliability and Security, ARES 2016, Salzburg, Austria, August 31 - September 2, 2016*, pages 296–303, 2016.

[8] Dietrich Featherston. cassandra: Principles and application. *Department of Computer Science University of Illinois at Urbana-Champaign*, 2010.

[9] Florian Kelbert and Alexander Pretschner. A fully decentralized data usage control enforcement infrastructure. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, pages 409–430, 2015.

[10] A. Lazouski, G. Mancini, F. Martinelli, and P. Mori. Usage control in cloud systems. In *The 7th International Conference for Internet Technology And Secured Transactions,(ICITST-2012)*, pages 202–207, 2012.

[11] Aliaksandr Lazouski, Fabio Martinelli, Paolo Mori, and Andrea Saracino. Stateful data usage control for android mobile devices. *International Journal of Information Security*, pages 1–25, 2016.

[12] A. Mordeno and B. Russell. Identity and access management in the internet of things - summary guidance, 2017. https://cloudsecurityalliance.org/download/identity-and-access-management-for-the-iot/.

[13] OASIS. eXtensible Access Control Markup Language (XACML) version 3.0, January 2013.

[14] J. Park and R. Sandhu. The $UCON_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, 2004.

[15] J. Park, X. Zhang, and R. Sandhu. Attribute mutability in usage control. In *Research Directions in Data and Applications Security XVIII, IFIP TC11/WG 11.3 Eighteenth Annual Conference on Data and Applications Security*, pages 15–29, 2004.

[16] A. Pretschner, M. Hilty, and D.A. Basin. Distributed usage control. *Communications of the ACM*, 49(9):39–44, 2006.

[17] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266 – 2279, 2013. Towards a Science of Cyber SecuritySecurity and Identity Architecture for the Future Internet.

[18] Denis Sitenkov, Supervisors-Ludwig Seitz, Shahid Raza, and Göran Selander. Access control in the internet of things. *Master's thesis*, 2014.

[19] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *ACM Transactions on Information and System Security*, 8(4):351–387, 2005.