A Policy Enforcement Framework for Internet of Things Applications in the Smart Health

S. Sicari[a], A. Rizzardi[a], L.A. Grieco[b,*], G. Piro[b], A. Coen-Porisini[a]

[a] *"DISTA, Dep. of Theoretical and Applied Science", Universita' degli Studi dell'Insubria v. Mazzini 5 – 21100, Varese, Italy.*
[b] *"DEI, Dep. of Electrical and Information Engineering", Politecnico di Bari v. Orabona 4 – 70125, Bari, Italy.*

Abstract

Internet of Things (IoT) is characterized by heterogeneous technologies, which concur to the provisioning of innovative services in different application domains. Introducing efficient mechanisms for collecting, processing, and delivering data generated by sensors, medical equipment, wearable devices, and humans, is a key enabling factor for advanced healthcare services. The adoption of IoT in smart health, however, opens the doors to some security concerns. In fact, by considering the confidentiality and sensitivity of medical data, a healthcare system must fulfill advanced access control procedures with strict security and data quality requirements. To this end, a flexible policy enforcement framework, based on the IoT paradigm, is defined hereby. It is able to face security and quality threats in dynamic large scale and heterogeneous smart hearth environments. As a key feature of the proposed framework, cross-domain policies have been defined using a specification language based on XML. In this way, it becomes possible to ease the management of interactions across different realms and policy conflicts. Moreover, to demonstrate the usefulness of the proposed approach, a running example, based on a smart health application, is detailed throughout the manuscript. This helps to highlight the different facets of the

*Corresponding author: a.grieco@poliba.it
Email addresses: sabrina.sicari@uninsubria.it (S. Sicari), a.rizzardi@uninsubria.it (A. Rizzardi), a.grieco@poliba.it (L.A. Grieco), giuseppe.piro@poliba.it (G. Piro), alberto.coenporisini@uninsubria.it (A. Coen-Porisini)

conceived enforcement framework. A preliminary performance analysis also demonstrates its feasibility in large scale environments.

Keywords: Internet of Things, Smart Health, Security, Policy Enforcement

1. Introduction

During the last decade, Internet of Things (IoT) approached our lives, thanks to the availability of wireless communication systems (e.g., RFID, WiFi, 4G, IEEE 802.15.x), which have been increasingly employed as central technology for smart monitoring and control applications [1]-[3]. Nowadays, the concept of IoT is many-folded. Since it embraces many different technologies, services, and standards, it is widely perceived as the corner stone of the ICT market in the next ten years [4]-[6]. From a logical point of view, an IoT system can be depicted as a collection of smart devices that interact on a collaborative basis to fulfill a common goal. Whereas, from a technological point of view, IoT deployments may adopt different processing and communication architectures, technologies, and design methodologies, based on their target.

With reference to the smart health context, IoT can be successfully used in monitoring services and biomedical systems including patient monitoring, telemedicine, pervasive healthcare management, detection of clinical issues, management of logistics and maintenance services, and so on [7]-[10]. In such smart health environments, data sources, communication technologies, services' and users' requirements are inherently heterogeneous [10][11]. The high level of heterogeneity can be leveraged by multiple security attacks. But traditional security countermeasures and privacy solutions cannot be directly applied to IoT technologies for various reasons: their computational, memory, communication, and energy consumption requirements could not be supported by constrained devices [12]. Moreover, adaptation and self-healing play a key role in IoT infrastructures (including those related to smart health scenarios), which must be able to face normal and unexpected changes of the target environment. Accordingly, privacy and security issues should be treated with a high degree of flexibility

[13]- [15]. Together with conventional security solutions, there is also the need to provide built-in security in the devices themselves (i.e., embedded) in order to pursue dynamic prevention, detection, diagnosis, isolation and countermeasures against successful breaches [16].

As a consequence, as a first step towards the development of a comprehensive IoT-based architecture, it is mandatory to define valid security and privacy frameworks suitable for IoT applications [2],[17]-[23]. They should address: (i) the guarantee of confidentiality and integrity of data; (ii) the provision of authentication and authorization mechanisms in order to prevent unauthorized users (i.e., a nurser cannot access to sensitive data available for doctors only) from accessing the system; (iii) the assurance of anonymity of users personal information, since devices may manage sensitive information (e.g., patient details) [24].

Besides security, healthcare services should provide accurate and complete information. In many scenarios, in fact, errors or missing values might have critical impact on actions or decisions. Accordingly, an IoT-based smart health system needs to guarantee well-defined levels of data quality. Four data quality dimensions can be considered (i.e., accuracy, timeliness, completeness, and source reputation), in order to inform users of the reliability of the accessed information. This is an innovative aspect since, as pointed out in [25], current available services provide the same information to each requesting user, often without considering his/her requirements and without specifying the level of security and data quality of the provided data.

Last but not least, it is important to remember that in the smart health context, the number of violation attempts can be significant. Therefore, it is fundamental to define and develop proper enforcement mechanisms.

In literature, several works begin to address some of the issues described above. But, as emerges in [26][27], few efforts are currently made regarding the enforcement of security and data quality policies. Except for the work presented in [28] and [29], there are no specific solutions addressing policy enforcement in IoT applications. To the best of the authors knowledge, no specific en-

forcement solutions for IoT-based smart health systems are currently available. Some attempts have already been done to define the proper languages for the specification of policies (for instance, the contribution presented in [30]-[32]. A solution which addresses the definition of flexible and standardized access control mechanisms for protecting both quality and security of sensitive data across multiple, heterogeneous domains is still missing.

Based on these premises, this paper proposes a policy enforcement system for IoT that adopts a cross-domain policy specification language, able to manage the interactions among the involved entities under well-defined policies. To this end, XML syntax is used, due to its general-purpose and a highly customizable nature. The defined solution has to guarantee security and data quality in case of policy violation attempts, thus dealing with the large number of critical situations which typically characterize IoT deployments. To demonstrate the usefulness of the proposed approach, a running example, based on a smart health application, is detailed throughout the manuscript. This highlights that the adoption of enforcement mechanisms provides a flexible and effective access control to IoT resources. In addition, a preliminary performance analysis demonstrates that the conceived approach requires less than 150 Mbps of aggregate bandwidth, thus becoming feasible in large scale environments.

As a final comment, it is expected that the enforcement framework proposed hereby could be used in the future as a secure wrapper for managing policies in existing IoT architectures, such as OneM2M[1], OpenIoT[2], FIWARE[3], and MOBIUS[4], already adopted by many companies.

The paper is organized as follows. Section 2 analyzes the state of the art about the existing policy enforcement mechanisms. Section 3 provides a big picture of the conceived IoT Policy Enforcement Framework. Section 4 describes the conceptual model used for the definition of the involved entities and their

[1] `http://www.onem2m.org`
[2] `http://www.openiot.eu`
[3] `https://www.fiware.org`
[4] `http://iotmobius.com`

relationships within the IoT scenario. Sections 5 presents the reference scenario along with a running example based on a smart health application, and deeply discusses the policy enforcement framework, the policy language specification, and the adopted access control model. Section 6 analyzes storage, software/hardware, and bandwidth requirements characterizing the conceived IoT Policy Enforcement Framework. Finally, Section 7 ends the paper and provides some hints for future works.

2. Related Works

As mentioned in the introduction, IoT is revolutionizing the overall medical system by introducing novel instruments and methodologies for deploying innovative and healthcare services [7]-[11],[33]. As a result, the adoption of a large number of devices generating heterogeneous and sensitive data is transforming conventional healthcare applications. This has resulted in complex big data systems with serious security concerns, including secure storage, secure access, and secure retrieval [20].

2.1. Baseline access control schemes

Secure access to sensitive data is paramount for smart health. At the time of this writing, many solutions have been designed by leveraging or extending conventional Role-Based Access Control (RBAC) or Attribute-based Access Control (ABAC) schemes. When RBAC and ABAC are used, secure access to a given data turns around an *access policy*. The access policy defines the set of properties that a user must have to earn access to the data itself. More specifically, with RBAC, this property simply refers to the role covered by the data consumer within a smart health system (i.e., director of cardiology department). ABAC, instead, enables fine-grained access control. First of all, any data consumer may be in possession of a specific set of attributes (i.e., age, role, responsibilities, and so on). Then, an access policy encodes a combination of attributes required to earn data access, which could be dynamically adapted

according to users or providers preferences. The design of a security framework for smart health and based on RBAC is investigated in [20] and [21]. Solutions leveraging the ABAC logic are discussed in [22][23]. However, note that all of these proposals present a serious weakness: they still appear as customized solutions that cannot be directly applied to large scale and dynamic healthcare systems. This is one of the aspects that this work aims to overcome.

2.2. Policy enforcement frameworks

In general, flexible and standardized access control mechanisms could ease the protection of both quality and security of sensitive data across multiple and heterogeneous domains (one of the goals of the present contribution). The policy enforcement represents a key enabling factor in this direction. It refers to the mechanisms used to force the application of a set of defined actions in a system. More in detail, policies are operating rules which need to be enforced for the purpose of maintaining order, security, and consistency on data. Policy enforcement assures that the security tasks can only be fulfilled if they are in accordance with the underlying policies, consulting the policy decision component (see Section 3) allows an entity to access a system resource or not. With reference to IoT scenarios, literature presents neither viable solutions nor detailed analysis on this issue. Only some works describe how to manage policy enforcement. At the same time, only few contributions explicitly consider the smart health domain.

The definition of access policies through standardized languages is the first step forwards to solve these issues. The work presented in [30], for instance, proposes a security framework for modern healthcare services that combines eXtensible Access Control Markup Language (XACML), XML security, and an extension of the baseline RBAC scheme, namely Spatial Role-Based Access Control (SRBAC), to support a standardized, secure, and fine-grained access control. First, the XACML language is used to define access control policies for confidential and sensitive data related to patients. XML security is used for binding, in single document, the protected data and the related access policy.

Finally, the SRBAC approach is used by the data consumer (the doctor, for instance) to retrieve the cryptographic key useful to extract the requested content from the aforementioned XACML document. Such an approach is quite simple and does not consider the possibility to dynamically handle a heterogeneous scenario with a huge number of data producers and consumers as we aim to do in this work. Moreover, due to the fact that it uses a basic RBAC mechanism, it does not permit customization of the offered services in terms of security and data quality, as we do. The same issue arises in [31], which investigates access control mechanisms for cloud-based health information systems, in compliance with the Health Insurance Portability and Accountability Act (HIPAA) regulation.

Because it is difficult to enforce a policy across domain boundaries, it is desirable to know which policies can be supported by other domains, which are partially supported, and which are not supported. For example, in a healthcare environment, the cooperation and communication among pharmacy, hospital and medical school are essential. They have their own policy enforcement mechanisms to protect their own proprietary data and patients records. The problem is that there are lots of collaborations and communications among these domains. Therefore a cross-domain policy enforcement becomes an essential component. The same problem also exists in social networking environments. To cope with such issues, [32] includes, in a proper simulation environment, a semantic model mapping and translation mechanism for policy enforcement across domain boundaries by means of Web Ontology Language (OWL). When compared with our solution, the work discussed in [32] presents two main disadvantages. First, the aim of our work is to propose a unifying language to be adopted in IoT applications, able to interoperate with different data sources and technologies. At the same time, we act as a cross-domain middleware, without the need of a translation system, as the one proposed in [32] could lead to scalability issues. Second, the translated/mapped languages (e.g., WS-Policy, XACML), besides being supported by most Web service platforms, are complex and, with their central rule processing engine, may be a bottleneck for a poten-

tially large amount of authorization requests. Note that XACML also requires a cache system for improving its efficiency, while in this paper we carry out an enforcement framework based on a distributed IoT architecture.

Expressing security policies to govern distributed systems (such as IoT) is a complex and error-prone task. Because of their complexity and of the different degrees of trust among locations in which code is deployed and executed, it is challenging to make these systems secure. Moreover, policies are hard to understand, often expressed with unfriendly syntax, making it difficult for security administrators and business analysts to create intelligible specifications. In [34] a Hierarchical Policy Language for Distributed Systems (HiPoLDS) is introduced. HiPoLDS has been designed to enable the specification of policies in distributed systems in a concise, readable and extensible way. HiPoLDS design focuses on decentralized execution environments under the control of multiple stakeholders. It represents policy enforcement through the use of distributed reference monitors, which control the flow of information among services (i.e., SOAs) and have the duty to put into action the directives output by the decision engines. But, it gives only the main implementation directions, conceived for service-oriented architectures, not reasoning about the scalability and the robustness towards malicious attacks of the proposed solution.

[35], [36] and [37] enforce only access control policies by means of a proper framework named Policy Machine (PM) and of a semantic web framework, respectively. But they do not refer to a distributed nature of the proposed solutions, which is a pivotal requirement in IoT applications.

Finally, the enforcement solution presented in [28] is based on a model-based security toolkit, named SecKit, which is integrated with the MQTT protocol layer - a widely adopted technology to enable lightweight communications among constrained IoT devices. In this work, authorizations and obligations are identified and a specific module acts as a connector to intercept the messages exchanged in the broker with a publish-subscribe mechanism. The main drawbacks of such an approach are that: (i) the enforcement operations are executed at the broker level, thus hindering the efficiency of the whole system; (ii) the

broker is also vulnerable to violation attempts, which could compromise all the activities taking part into the system itself.

2.3. Final considerations

To conclude, except for the work presented in [28], there are no specific solutions available in the literature that address policy enforcement in IoT applications. The identification of the enforcement mechanisms suitable for IoT context, instead, is fundamental for reaching an equilibrium between the guarantee of proper security requirements and computing efforts. This is even more true in a smart health context, where heterogeneous devices and services are jointly offered.

Some attempts have already been done to define the proper languages for the specification of policies, but a standard which addresses specifically the IoT paradigm is still missing. To address such issues, we propose herein an enforcement framework integrated in a distributed IoT middleware architecture. It is able to manage the defined policies and the interactions among the involved entities in a general smart health scenario (presented in the model in Section 4) through a general-purpose and cross-domain language. The proposed policy definition identifies the minimal set of policies required for management of a typical IoT-based application, thus reducing computational overhead. Moreover, users who make use of services provided by IoT system should be allowed to express their needs in terms of reliability and quality of the data they receive.

In the future, it is expected that the proposed framework could be integrated in existing IoT architectures (like OneM2M, OpenIoT, FIWARE, and MOBIUS) as an orthogonal/additional security component. This kind of integration is not new in the literature. A first attempt in this direction, in fact, has been done in [38], where a security plugin for managing access operations has been added to the original OneM2M system.

3. Policy Enforcement Framework

An enforcement framework should be able to cope with access control and service provisioning under well-defined requirements. Functions should be configurable in order to make new access control decisions, and enforcing the expressed policies. In this section, we present an IoT Policy Enforcement Framework for smart health. It should not be considered as an extension of any existing model or framework. But it is intended to represent a re-definition of access control and data exchange in terms of a standardized and generic set of functions and roles suitable for IoT applications.

To better describe the advantageous capabilities of the conceived IoT Policy Enforcement Framework, a use case referring to the provisioning of medical care and cafeteria services within the department of cardiology in a hospital will be taken into account. It is introduced in this Section and further investigated in Section 5. A use case diagram is sketched in Figure 1. It includes three main components (i.e., *Nodes*, *Users*, and *IoTPlatform*) and all the main functionalities offered to them.

First of all, *Nodes* embrace all the devices able to generate and disseminate data within the smart health environment. In general, they may include: wearable devices that track activities and biological parameters of patients; sensors that measure environmental conditions (e.g., temperature, humidity, brightness) of operating and waiting rooms, corridors, bathrooms, etc.; sensors that recognize the presence of people in a given tracking area; RFID tags that announce the presence/position of medical equipments and drugs; as well as other kind of instruments (e.g., cardiac, hemodynamic, respiratory, neurological, blood glucose, childbirth, and body temperature monitoring devices) able to capture heterogeneous information during the time. The use case considered in this work integrates the following nodes:

- **ECG**: a medical instrument that checks for problems with the electrical activity of the patient's heart. It acts as a registered node, connected to the *IoTPlatform* through a wired cable. The medical instrument must

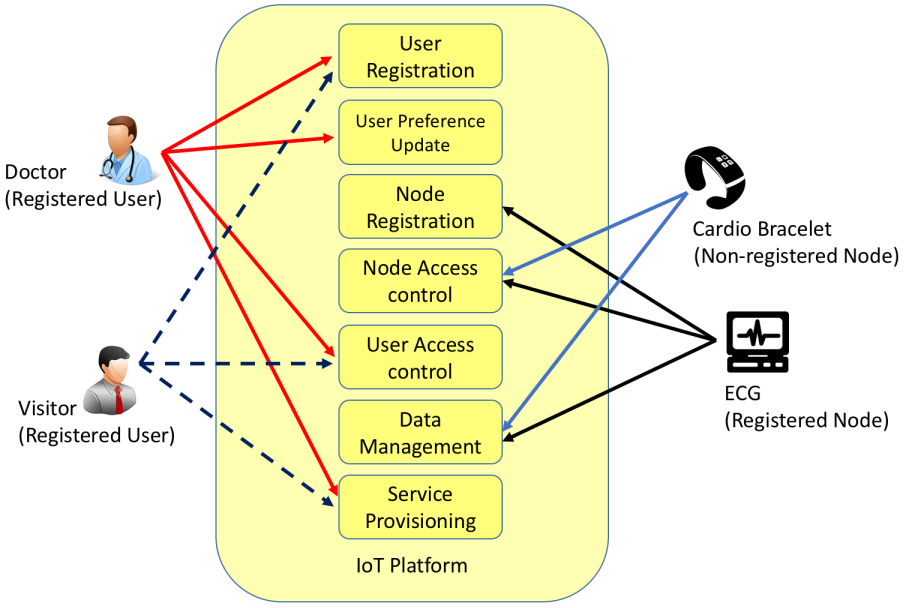


Figure 1: Use case diagram

generate trusted data. By considering their sensitivity level, in fact, any external violation must not be accepted because it will negatively influence the effectiveness of a medical examination or intervention.

- **Cardio bracelet**: a wearable device monitoring biological activities of a patient (e.g., heart rate). It acts as a non-registered node, connected to the *IoTPlatform* through Bluetooth communication. Thus, it may generate data with a lower security level since it could be more easily compromised.

Users are people requesting various kind of services. Without loss of generality, it is possible to assume that users may download an ad-hoc advanced application on his/her smartphone, possibly equipped with technologies such as Bluetooth, ZigBee and/or NFC, and connected to the Internet through Wi-fi or 4G. The user may interact with physical objects and obtain specific digital services. Note that users have different profiles (e.g., doctors, nurses, patients,

visitors, and so on) and each one can be interested in different services with respect to the others. For example, nurses may be interested in information related to the alert generated by instruments monitoring patients' activities; a doctor may want to be informed about a group of patients to be examined; patients may want to request medical care or communicate food preferences to the canteen; and visitors may want to find public bathrooms, coffee machines, or any other recreational space. Also, the application running on the users personal device is in charge of providing the user with the expected customized contents. As a consequence, services should be offered in relation to their profiles and preferences. The use case considered in this work integrates the following users:

- **The doctor**: a specialist in cardiology, in charge of assisting patients during the morning. It acts as a registered user. It has attributes for accessing medical care and cafeteria services. With reference to the medical care service, it is only interested in receiving alerts generated by registered nodes (e.g., ECG in the proposed example); therefore, data with high levels of security.

- **The visitor**: a man that is just interested to know the availability of a coffee dispenser. It acts as a registered user and it is associated only with the attributes for accessing the cafeteria service. Therefore, in the case it tries to access to data generated by medical devices (ECG and cardio bracelet in the proposed example), the platform will deny the service request.

Finally, the *IoTPlatform* mainly integrates the IoT Policy Enforcement Framework, which continuously manages service provisioning and access control activities. Specifically, it handles all the tasks related to node/user registration, data processing, dynamic and adaptive service provisioning, resource access control, etc. *IoTPlatform* hosts databases for policies, user profiles, and data storage servers, as well as logical entities and interfaces exposing the set of functions of the IoT Policy Enforcement Framework (see Section 3.2 and Section 3.1 for more details).

The following subsections provide a thorough description of the IoT Policy Enforcement Framework, its components interaction, and the considered access control model. A pragmatic explanation of actions with the support of a running example, based on the smart health context just described, will be presented in Section 5, preceded by a formal representation in Section 4.

3.1. Components interaction

In the IoT Policy Enforcement Framework, *Nodes*, *Users*, and *IoTPlatform* interact each other through the following main procedures:

- **User registration.** At the beginning, each user performs the registration phase and communicates some personal information (i.e., first name, last name, age, role within the hospital, etc.) to the *IoTPlatform*. Such details are used by the *IoTPlatform* for creating the user profile. At the end of the registration phase, the user receives a message of confirmation, delivering many parameters (including the credentials to access the system itself). For the users, the registration phase is mandatory (note that a user who does not want to provide personal information can register with a minimal profile). In fact, user interaction with *IoTPlatform* has to be controlled in order to customize the provided services and also to protect the exchanged information. Once the users are registered to *IoTPlatform* and have an associated profile, then they can request the services provided by the IoT system in compliance with the access policy defined by the framework.

- **User preferences update.** After the registration phase, the user may specify his/her preferences which further customize their profile. These preferences also refer to the expected levels of data quality and security. In fact, the data received by *IoTPlatform* are processed and analyzed according to well-defined metrics related to data quality and security, thus letting the users choose, for example, only accurate information with a high confidentiality degree.

- **Node registration.** Similar to users, a registration phase occurs for the data source. In this case, the node may communicate some technical information (i.e., kind of source it is, type of communication mode, data type, encryption scheme, and so on). Based on these details, the *IoTPlatform* is able to execute a preliminary quality and security assessment. Quality and security levels, however, can be further updated in the future, e.g., when the source starts to send data. At the end of the registration phase, the node receives a message of confirmation, delivering many parameters (including the credentials to access the system itself).

  Different from the users, the registration phase is not mandatory for nodes, therefore there would be one or more sources for which the system has no information, but are allowed to send data. Such information will be managed by *IoTPlatform* depending on the levels of security and quality requested by the users.

- **Node access control.** This action defines the beginning of the interaction between node and *IoTPlatform*. The node simply communicates its intention to send data. The *IoTPlatform* possibly verifies the node's credentials and opens a session.

- **User access control.** This action defines the beginning of the interaction between user and *IoTPlatform*. The user simply communicates its intention to request services and specifies its attributes (i.e., neurologist doctor, nurse of the department of cardiology, and so on). Then, *IoTPlatform* verifies the user's credentials and determines access to the information provided by the system. A session is opened also in this case.

- **Data management.** When the sensor node sends data to *IoTPlatform*, they are stored in a repository as raw data, waiting to be analyzed. If the node is registered, data are encrypted with credentials obtained during the registration phase. Otherwise, data are sent in clear. Data received by registered node are processed by the framework through *data evaluation* and are the object of the *score assessment* action (specifically, data are

sent to specific modules for security and quality score assessment). Score assessment is also performed for the data received by unregistered sources. Note that in such a situation the data is not discarded, but the lower reputation of the source will influence the use in the user service provision.

- **Service provisioning.** A user, who has registered, can ask for the services made available by *IoTPlatform*. The request is made in a secure manner (i.e., the information is encrypted with credentials sent in the registration phase). In such a context, the user (i.e., the doctor, the patient, or others) has to be aware of the security and the quality of the received and transmitted data, as well as the accuracy, completeness and integrity of the retrieved information. The definition of specific policies, according to the users consensus, aims at guaranteeing the desired level of security and data quality. The policies are stored in *IoTPlatform* that guarantees the access control, the key management, and the enforcement of the policies themselves (Section 3.2). Note that the proposed solution preserves the user privacy, since, without an ad-hoc approach, it is possible to derive sensitive data by analyzing user behavior and learning habits. Finally, the services are provided to the users taking into account user's attributes, user's preferences (expressed in terms of quality and security levels), and access policy.

3.2. The framework

Usually, an enforcement framework includes a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), and a Policy Administration Point (PAP) [39]. PEP is the point which intercepts the requests of access to resources from users, and makes a decision request to the PDP in order to obtain the access decision (i.e., approved or rejected). Each data access request is then routed by the PEP to the PDP, which evaluates such requests against the authorization policies, before taking the access decisions. To this end, the PDP queries a policies store. Once the PDP completes the evaluation, it returns a response to the PEP. Based on this decision, PEP either permits or denies access to the

user/resource. The authorization policies are finally administered through the PAP, which allows the runtime change/update of policies. The aforementioned functions are generally performed in real systems by an application.

In the considered smart health domain, each *IoTPlatform* includes a policies store, a PEP, a PDP and a PAP, while each user has an application representing an interface for the user device and *IoTPlatform*. More in detail, a user logs on the application running on his/her device using the provided GUI. A session is opened, during which the user can request for the services provided by *IoTPlatform* on the basis of the accessible resources. All application components interact with the underlying PEP. As regards nodes, a separate discussion has to be made, since the system has to be able to deal both with registered and unregistered nodes (i.e., data sources), while users are always registered. Security among the involved components (i.e., *IoTPlatform*, users, nodes) is guaranteed by means of encryption mechanisms.

The way PEP, PDP, and PAP modules interact with the proposed smart health architecture is highlighted in Figure 2. Although the figure shows only one *IoTPlatform*, the enforcement framework is expected to run in a distributed manner on multiple platforms, since a typical IoT application may involve a large area and a large number of users and data sources. In fact, *IoTPlatform* aims to create a middleware layer able to process the data provided to the IoT system closer to the sources, in order to cope both with scalability and computational issues. Hence, the distribution of policies and their update and synchronization processes should be taken into account in a real *IoTPlatform* implementation. In the smart health context, it is possible to have multiple platforms interacting with each other. For example, hospitals, clinics, and pharmacies that cooperate within the same geographical area. In this case, platforms should share the same security policies (in the same IoT context) and each of them should have their own policy enforcement component. For the sake of simplicity, in the following discussion we refer to only one *IoTPlatform*.

We remark that an important advantage of the adopted policy-based control is that the controlling unit of the system (i.e., the enforcement framework)

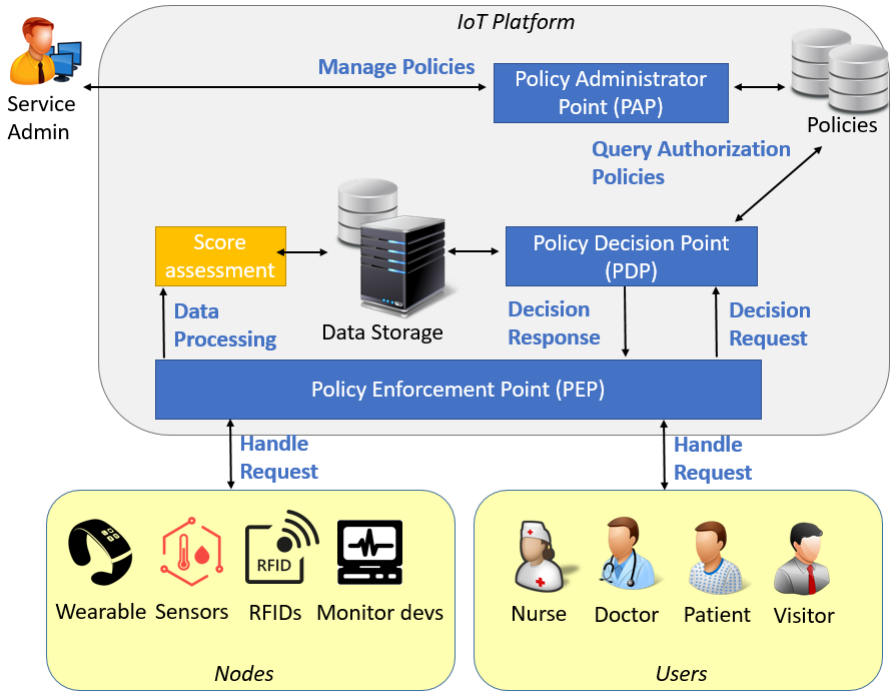


Figure 2: Enforcement architecture for the smart health

is kept decoupled from other management components (i.e., analysis and processing operations, request handler, resource handler). As a consequence, the *IoTPlatform* behavior can be managed and/or changed without modifying the software or the user/node interfaces. As detailed in the following sections, all the interactions happening with *IoTPlatform* are regulated by policies, which specify the rules interpreted and enforced by the framework itself. Hence, if the conditions change or new services or blocks are added, only the corresponding policy rules have to be adapted without the need to do the updates off-line. In fact, policies can be loaded at runtime through the PAP into the policy store.

More in detail, the policy store contains a list of configurations that are used for a policies' composition. Such configurations are represented in a format, according to the adopted policy specification language, described in Section 3.3. A policy is associated to a particular interaction that happens within

the system (which acts as a *primary key* for establishing the operations to enforce) and the policy satisfaction will depend on the attributes associated to the user/node involved in such an action. Such a feature increases the flexibility of the framework and makes it suitable for smart health applications, which require a high degree of availability, especially in large scale environments.

3.3. Access control model and policy language

Even if many services may coexist in a smart health environment, users may not be allowed to access all of them. The services are provided to the users taking into account user's profile, user's preferences, and access policy. However, in order to handle possible violation attempts, proper rules should be defined for identifying the actions to be performed, guaranteeing the correct application of policies.

Once the components of the enforcement framework and its scope are defined, the main challenge is the identification of a minimal set of primitives able to specify, and subsequently enforce, a large variety of attribute-based rules, on the basis of the interactions taking part into the smart health system (Section 3.1). To this end, *IoTPlatform* generates a series of policies in the form of XML syntax, in order to guarantee satisfactory and trustworthy services. Through the presented language, policies are expressed and enforced, in order to provide a unifying framework to support a wide range of attribute-based policies. Due to the large number of possible smart health applications, such a language should be flexible enough to represent the analyzed contexts, both in a general-purpose and in a customizable way. For such reasons, we decided to adopt XML and define proper tags for specifying data input and operations to be controlled and enforced. XML allows to express the whole set of policies for each involved entity, which, as specified in Section 4, are mainly distinguished in: *User*, *Node*, and *IoTPlatform*. Each of them has specific attributes, as described in the next sections, which are stored in the policy store of the *IoTPlatform*. According to the defined attributes, each entity is allowed to perform different actions.

As regards the mentioned attributes, it is worth noting that the access con-

trol model adopted in this paper is inspired by the Attribute Based Access Control (ABAC) mechanism [40]. It offers scalable, flexible and fine-grained access control. With ABAC, the subjects that want to access to the resources, the subjects that expose the resources, and the resources themselves are described by means of specific attributes. These attributes should be properly defined for handling access permissions in a healthcare scenario. Moreover, to jointly address heterogeneous smart health environments, these attributes must be configured by means of the adopted policy specification language, based on XML.

The main attributes considered in this work refer to security and quality levels of the data managed by *IoTPlatform*, In fact, the final goal of the proposed framework is to provide security- and quality-aware information to the end users, possibly taken into account users preferences. In this way, the enforcement framework forces the system to respect the desired security and quality requirements of authorized users. The policies are distinguished depending on the actions currently performed by the entities within the smart health scenario.

4. IoT Model

The design of the presented IoT Policy Enforcement Framework originally started from the definition of a general UML conceptual model, just published in [41]. Such a model represents the components (*Node*, *User*, *Service*, *IoTPlatform*) involved in the conceived IoT Policy Enforcement Framework, and their interactions, as well as the levels of security and data quality. The model is suitable for heterogeneous IoT applications and architectures, including those devised for smart health contexts. It is detailed in the following sections in order to point out a formal representation of the entities, their attributes, functions and roles, and the performed actions and obligations. This helps clarify the contribution of the IoT Policy Enforcement Framework.

4.1. Node

From a technological point of view, the different technologies involved in the acquisition of information from the environment, such as Wireless Sensor Networks (WSN), RFID, nanodevices, actuators and so on, are represented by the class *Node*. Such a class is extended by the sub-classes representing the mentioned technologies, as shown in Figure 3.

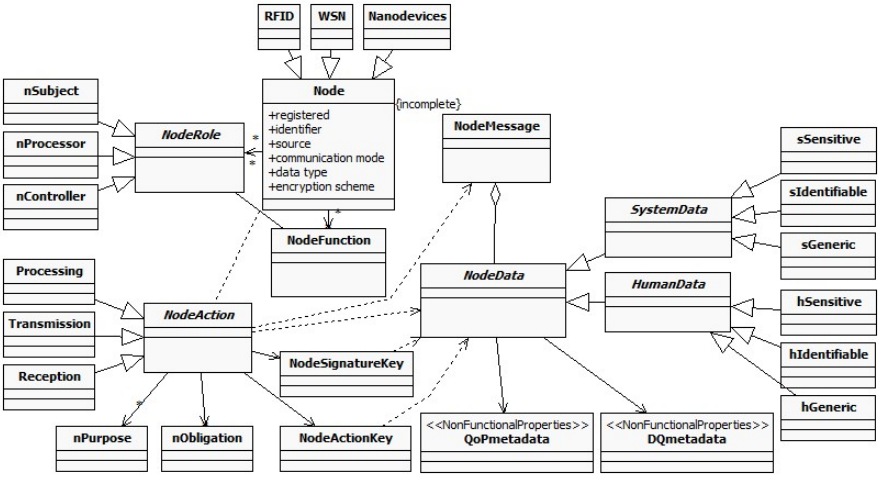


Figure 3: Class Diagram - Node

Each instance of the class *Node* is characterized by a pair function-role, identified by *NodeRole* and *NodeFunction* classes. *NodeRole* is strictly related to the privacy concept [42] and introduces three sub-classes:

- *nSubject* represents the node that senses or generates the data
- *nProcessor* represents the node which processes data by executing some actions (i.e., forwarding, aggregation)
- *nController* represents the node which verifies that the actions perfomed on data satisfy the defined policies [43] [44] [45].

As regards *NodeFunction*, which represents the task performed by a node, the UML model does not specify any sub-class, since the functions depend on the

specific IoT application context in which the system is employed (e.g., shopping retail, hospital, university).

The class *NodeAction* is associated with the pair function-role, and specifies the set of actions which can be undertaken by the node itself. The identified actions are:

- *Processing*, when a node executes some operations on data
- *Trasmission*, when a node sends data to another node
- *Reception*, when a node receives data from another node.

An action is executed under some purpose (*nPurpose* class) that specifiies the reasons under which it is possible to handle data (i.e., marketing purpose, research purpose, health purpose). *NodeAction* is also associated with one or more obligations (*nObligation* class), in order to model the fact that the execution of a set of mandatory actions is guaranteed by the processor and/or the controller at the end of the processing activities. An example of obligation is whenever an inconsistency or a violation is detected, then some countermeasures, such as generation of an error/alert message, must be taken.

In order to guarantee authentication, integrity, confidentiality and non repudiation, two groups of keys, named *NodeSignatureKey* and *NodeActionKey* are associated to *NodeAction*. From one side, message integrity and peer authentication are offered through asymmetric encryption. Therefore, *NodeSignatureKey* embraces the public key (i.e, *NodeSignatureKey.public*) and the private key (i.e, *NodeSignatureKey.private*) of the node. While the public key must be delivered to the platform during the registration phase, the private key must be kept secret. Indeed, to authenticate and guarantee the integrity of its messages, the node can encrypt specific fields of these messages with the *NodeSignatureKey.private*. The platform is able to verify the validity of received messages by decrypting the aforementioned fields with *NodeSignatureKey.public*. From another side, data confidentiality is offered through symmetric encryption, where sensitive data are protected with *NodeActionKey*.

Finally, the joint adoption of *NodeSignatureKey* and *NodeActionKey* also ensures non-repudiation. In fact, a node that authenticates and protect messages with *NodeSignatureKey.private* and *NodeActionKey*, respectively, cannot deny to have generated these messages. Since no other entities in the system are in possession of these cryptographic material, in fact, the messages authenticated and protected with *NodeSignatureKey.private* and *NodeActionKey* are surely generated by the node.

Communications among nodes occur by exchanging instances of the class *NodeMessage*. A message is composed of several heterogeneous kinds of data (e.g., numbers, text, multimedia), which can contain different information, depending on the application context. Furthermore, the instances of abstract class *NodeData* may be distinguished as:

- *SystemData*, which represents the data generated by the IoT system (e.g., the information provided by a locator tag related to the movements of a particular user or object in a target tracking application)
- *HumanData*, which includes the data produced by users, for example by means of a social network

Moreover, both system and human data are classified in three different categories, represented as extensions of *SystemData* and *HumanData* classes:

- *sIdentifiable* and *hIdentifiable*, which represent the information used to uniquely identify the nodes (i.e., node identifiers)
- *sSensitive* and *hSensitive*, which include information that should not be freely accessible because they may reveal private acquired data
- *sGeneric* and *hGeneric*, which represent other generic information not included in the previous classes

Each instance of *SystemData* or *HumanData* is associated *QoPmetadata* and *DQmetadata* information, which represent the non functional properties related, respectively, to the levels of security and data quality of the data themselves.

4.2. User

Besides nodes, another fundamental entity is represented by the class *User*. It concerns all humans that could interact with the IoT system, for example by means of their personal devices (e.g., smartphone, NFC, tablet). Note that users are distinguished from nodes due to the fact that the former require one or more services from the infrastructure, while the latter acquire the necessary information to provide such services.

In addition, in order to provide each user with the best services, he/she requires a personal *Profile*, as represented in Figure 4. More in detail:

- *Profile* includes an aggregation of *PreferenceOnService*, used to customize the services on the basis of user requirements in terms of security (i.e., *PreferenceOnSecurity*) and quality (i.e., *PreferenceOnQuality*), as well as the communication keys.
- *Profile* also concerns *UserData*, which can be further classified as:
  - *Identifiable*, including data referred to the user identity (i.e., first and last name)
  - *Sensitive*, containing information related to user's private life and habits, such as health conditions, food intolerances, religious beliefs and so on
  - *Generic*, regarding general information not belonging to the previous classes.

Classes *UserRole*, *UserFunction*, and *UserAction* are intended in the same way as for nodes. In particular, as regards *UserAction*:

- A user, before interacting with the IoT system, has to register himself/herself (*Registration* class)
- Then, the user has to accept the agreement with the service provider (i.e., the consent to handle user personal data for specific purposes and under some obligations - class *Consent*)

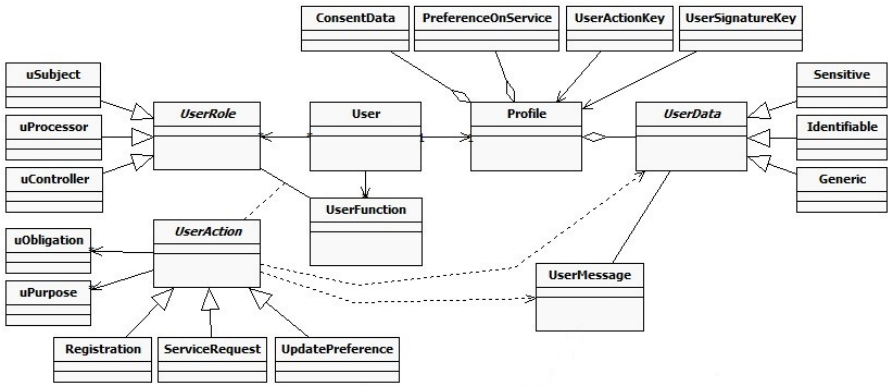


Figure 4: Class Diagram - User

- *ServiceRequest* represents the user requests for services

- *UpdatePreference* represents the capability of the users to change at any time their preferences expressed in their own profile.

Furthermore, the communications occur by means of packets, which are instances of *UserMessage* class. Note that, in an IoT scenario, the number of nodes and users varies over the time.

4.3. IoTPlatform

The heterogeneous data, acquired by nodes and users, are handled by *IoTPlatform*. *IoTPlatform* plays different roles (*IoTPlatformRole* class), and functions (*IoTPlatformFunction* class), in relation to the current action and the application domain. As regards *IoTPlatformAction*, besides the obvious *DataProcessing* and *ServiceProvision*, the main tasks, shown in Figure 5, are:

- *UserProfileDefinition*, which models the acquisition of the users data and preferences

- *ConsentAcquisition*, which represents both the user acceptance of the agreement with the requested service, and user data management, according to the established policies

- *AccessControl*, which represents the execution of the access control operations, in order to restrict the access to the system only to authorized and pre-registered users
- *PolicyDefinition*, which represents the definition of a policy for ensuring user privacy and, in particular, anonymity
- *Enforcement* is required in order to force compliance with the defined policies, under which the user has given the consent to handle his/her data

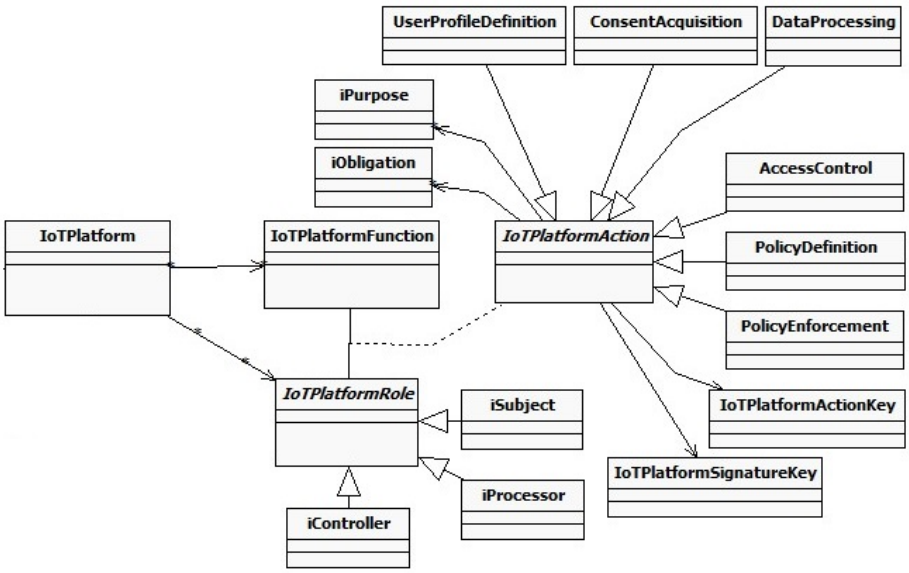


Figure 5: Class Diagram - IoT Platform

IoTPlatform deals in particular with encryption and decryption keys. As already discussed before, different pools of keys are used for different purposes. First, *IoTPlatform* is in possession of its own public (i.e., *IoTPlatformSignatureKey.public*) and private (i.e., *IoTPlatformSignatureKey.private*) keys to use for authenticating itself and ensuring the integrity of messages sent to both Nodes and Users. During the registration phase, the public key of the platform is shared with each registered entity, without incurring to security flaws. On

the contrary, the corresponding private key must be kept secret (as explained in Section 5 and Appendix Appendix A). Second, *IoTPlatform* stores the public keys of all registered Nodes and Users, delivered by them during the registration phase and consequently used for authentication purposes (see Section 5 and Appendix Appendix A). Third, the platform also shares with each registered entity a unique symmetric key. The term unique means that a symmetric key cannot be used by more than one registered user, for security reasons. This key is generated by the platform, delivered to the registered entity during the registration phase, and used to encrypt data that will be exchanged within the platform in the future (as explained in Section Section 5 and Appendix Appendix A). Based on these considerations, asymmetric keys used for authentication and integrity services are defined within the class *IoTPlatformSignatureKey*. Symmetric keys used for confidentiality purposes are defined within the class *IoTPlatformActionKey*. It is important to highlight that while each user/node has an unique signature key, which represents an access credential, the action key is directly related to the pair function-role currently played by the user/node. A similar system is adopted in [43] and [44] as regards WSN field. Such a behavior ensures the privacy compliance of the transmitted information, both from users/nodes towards *IoTPlatform* and from *IoTPlatform* towards the requesting users. Each user/node encrypts its data with its proper action key. When *IoTPlatform* receives a request of data from a user, it performs some queries in order to establish, first, if the user is authorized to get such data and, second, the user's preferences in relation to the security and quality properties of the requested services. Note that the defined solution supports any kind of encryption technique and any key distribution mechanism, although the details related to such issues are out of the scope of this work.

4.4. Service

Finally, *Service* is another core class of the model, since all the IoT system activities turn around the request and provision of services. The users give information related to their identity, life-style, interests, and preferences, in

order to obtain customized services. Such a service has to guarantee several non-functional properties, shown in Figure 6, which include:

- Data Quality (DQ), since data are collected from different sources, the state of consistency, validity, timeliness, and accuracy of the provided data has to be modelled
- Quality of Protection (QoP), which regards the insurance of well-defined levels of security.

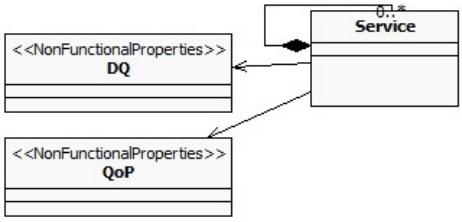


Figure 6: Class Diagram - Service

A service is simple or atomic if the users are enabled access to one source, whereas it is composite if the information provided by multiple sources are integrated. The set of services can be directly and dynamically configured by a network administrator and can be orchestrated by a remote server through standard Web services approaches.

Summarizing, Figure 7 represents the entities just separately described in a unified way, along with their relationships.

5. Example Case Study

While considering the formal model of the conceived framework described in Section 4, the use case presented in Section 3 is further investigated. Specifically, the interaction among framework's components is explained through a running example.

As described in Figure 8, the interaction between *Nodes* and *IoTPlatform* embraces: nodes registration, node access control, and data management. The

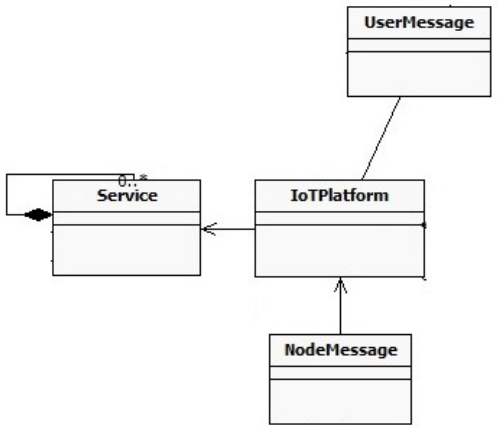


Figure 7: Class Diagram - IoT System

interaction between *Users* and *IoTPlatform*, instead, include the following procedures (see Figure 9): user registration, profile management, user access control, and service provisioning.

All the messages reported in both Figure 8 and Figure 9 are exchanged under a secure connection, through HTTPs. More technical details are discussed below.

5.1. Nodes registration

In line with the model described in Section 4 (see *Node* class), the *IoTPlatform* identifies a node through the following properties and attributes:

- *Registered* (possible values yes/no): is an attribute which clarifies if the considered node is registered or not to *IoTPlatform*. Note that the system accepts data both from registered and unregistered sources, but it will manage the provided information in a different manner with respect to user preferences
- *Identifier*: is an identifier given by *IoTPlatform* to the sources
- *NodeSession*: is the session identifier currently owned by the node (i.e., the identifier of the last session in which the node interacted with *IoTPlatform*)

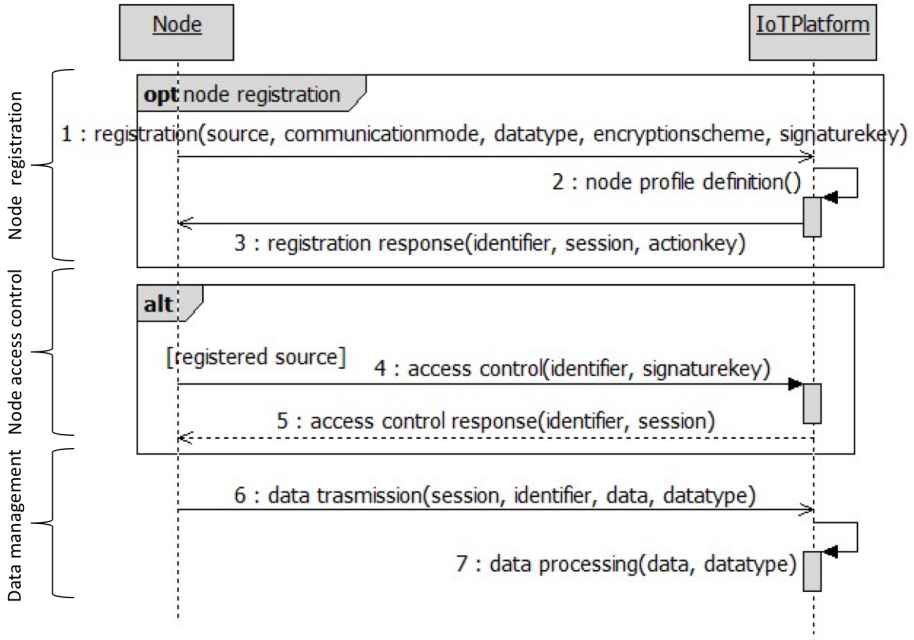


Figure 8: Node interactions with IoTPlatform

- *Source*: identifies the kind of source (e.g., sensor node, RFID, actuator)
- *CommunicationMode*: identifies the mean of communication used in order to transmit the data (e.g., WiFi, 3G, Ethernet)
- *DataType*: represents the kind of data provided by the source. Note that a source may transmit multiple kinds of data (e.g., a double for a temperature and a string for a reference area)
- *EncryptionScheme*: represents the encryption technique used by the source for its communications with *IoTPlatform*; in case of unregistered sources, the corresponding fields related to the encryption scheme and keys are marked as undefined
- *NodeSignatureKey*: is the public key of the node, sent to *IoTPlatform* during the registration phase as already described in the previous Sec-

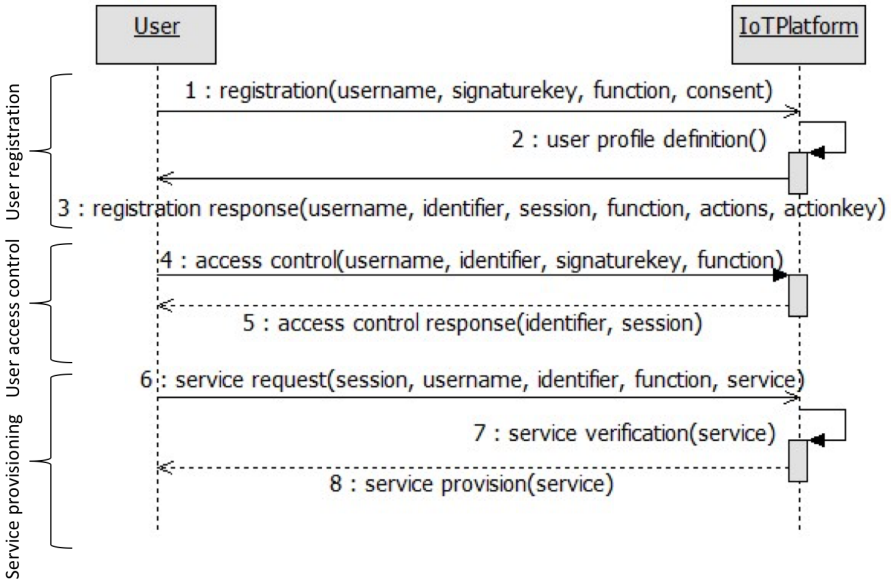


Figure 9: User interactions with IoTPlatform

tion, it is used to offer message integrity and peer authentication. It it important to remark that this public key can be delivered by means of a X.509 certificate. In this way, the whole system becomes robust against man-in-the-middle and impersonation attacks.

- *NodeActionKey*: is the symmetric key given to the node by *IoTPlatform* after the registration phase, which is used to encrypt the information exchanged between node and *IoTPlatform*. Note that a node could have more than one action key, in relation to the function played at a specific moment (e.g., a node may acquire the transmitted data or only forward a data received by another source)

- *QoPmetadata*: when the source sends data, *IoTPlatform* begins to perform an analysis of such information, in terms of confidentiality, integrity, authentication, and privacy. It also takes into accout the kind of source,

the communication mode (e.g., a wifi channel is considered less secure than a wired one), and the adopted encryption scheme (i.e., its level of robustness). As a result, a set of metadata is associated to the information provided by each source (both registered and non registered), which will allow the users to filter the data they want to receive

- *DQmetadata*: as for security features, *IoTPlatform* analyzes the data provided by each source in terms of completeness, accuracy, timeliness, and reputation, considering both historical feedback about the source behavior and the frequency of the information update.

Both *QoPmetadata* and *DQmetadata* are evaluated during data processing activities. Such scores are in the range [0,1]. The definition of proper algorithms for the assessment of the security and data quality requirements is out of the scope of this work, since it is subject of another work [46].

In the considered use case, only ECG represents a node that registers itself to *IoTPlatform*. To this end, it sends a request, as presented in Listing 1 in Appendix A. In the request, it specifies what kind of source it is, what type of communication mode it will use, the kinds of data it will transmit, and, finally, the encryption scheme to be used for authentication purposes, performed with *NodeSignatureKey*.

IoTPlatform replies to the node request by providing a message which contains its public key (i.e., *IoTPlatformSignatureKey.public*) and*NodeActionKey*, as well as the session and the node identifier, established by *IoTPlatform* (see Listing 2 in Appendix A). Note that session and the node identifier are encrypted with the private key of the platform (i.e., *IoTPlatformSignatureKey.private*) for authentication reasons.

The registration process performed by the ECG node is depicted in Figure 10.

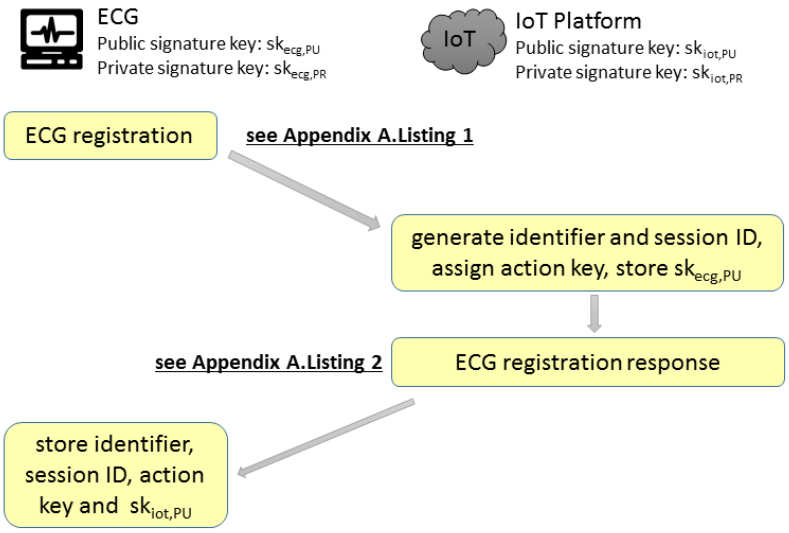


Figure 10: Registration phase for ECG

5.2. Nodes access control

Registered and non-registered sources can start at any time to interact with the IoT system. When the interaction between node and *IoTPlatform* begins, the action is classified as *access control*. If the node is already registered, as in this case, the task to be performed by *IoTPlatform* is the node authentication by means of the public-key algorithm previously introduced. The node sends its identifier and a digital sign generated with *NodeSignatureKey.private*. Then, the platform verifies the digital sign by using *NodeSignatureKey.public*. In the case the node is not registered, instead, the identifier is transmitted in clear and the authentication procedure is not executed. As, said, the registration phase is not mandatory for nodes, therefore there would be one or more sources for which the system has no information, but which are allowed to send data. Such information will be managed by *IoTPlatform* depending on the levels of security and quality requested by the users. If a source has already interacted with *IoTPlatform*, it will be recognized by the identifier previously assigned. Instead, it is still an unknown source and *IoTPlatform* assigns a new identifier

in response (which is stored in *IoTPlatform* as described in Listing 7 in Appendix A).

Listings 3 and 4 in Appendix A describe this behavior in XML syntax, considering the registered ECG previously described. Listings 5 and 6 in Appendix A, instead, refer to the cardio bracelet, which acts as a non-registered node. Note that the response includes the session for all kinds of sources. In such a way, *IoTPlatform*always knows the nodes currently connected to it, since in an IoT context nodes continuously join and leave the network.

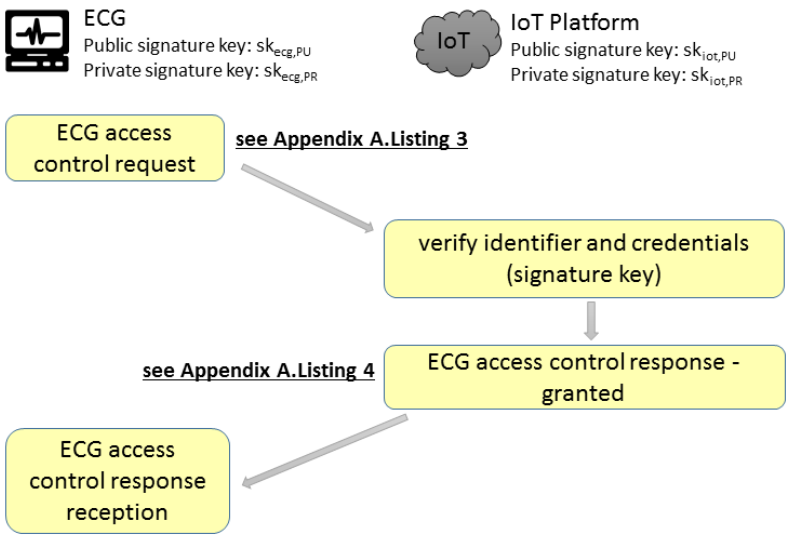
To conclude, within *IoTPlatform*, the registered and unregistered nodes are stored as presented in Listing 7 in Appendix A. As regards the no registered source, an identifier is also assigned and stored, since *IoTPlatform* is able to recognize a node which had already sent some data. *QoPmetadata* and *DQmetadata* may be further assigned by *IoTPlatform* during its activity, when sources start to send data. Note that, presumably, a registered source will present higher levels of security and quality for its data with respect to a unregistered one.

The access control process for both ECG and cardio bracelet nodes is depicted in Figure 11.

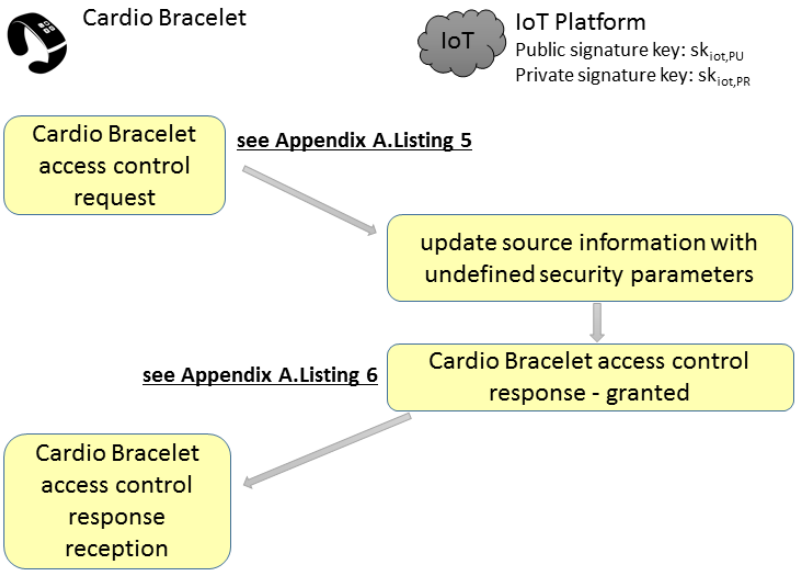
5.3. Data management

As already anticipated in Section 3.1, data management integrates both transmission and processing tasks.

Data transmission: ECG and cardio bracelet send data to *IoTPlatform*. Data are stored in a repository as raw data, waiting to be analyzed. The kind of action performed is defined as transmission. *IoTPlatform* receive, as input from the node its identifier, the data, and the type of data which it is transmitting. If the source is registered, the data and the type of data are encrypted with the proper *NodeActionKey*. In addition, and similarly with the access control phase, the identifier is transmitted along with its digital sign, obtained through the public key of the node (i.e., *NodeSignatureKey.public*. At this stage, no data verification has to be performed, since the data is not analyzed yet. In the case the source is not registered, both data and identifier are transmitted in clear.



(a)



(b)

Figure 11: Access control for (a) ECG and (b) cardio bracelet.

Listing 8 in Appendix A represents the XML syntax of the described behavior for the ECG instrument. Listing 9 in Appendix A, instead, represents the XML syntax of the described behavior for the cardio bracelet.

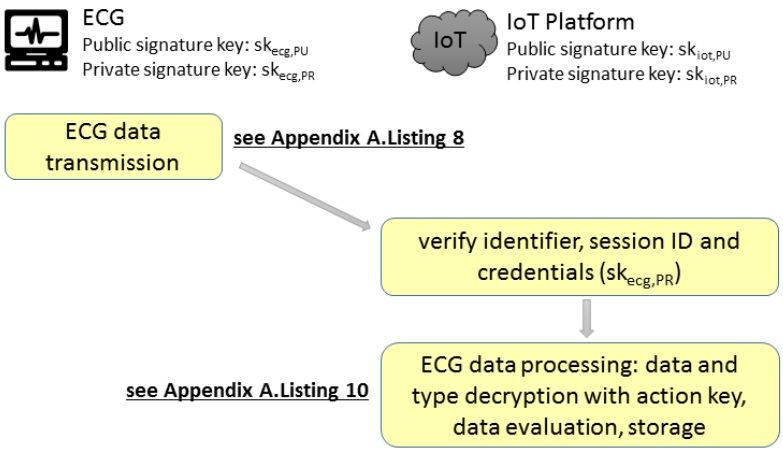
Data evaluation and security/quality score assessment: Once the data are stored in the raw data repository, they are sent to the enforcement framework, which performs the *data evaluation* task. For registered sources, *IoTPlatform* decrypts the received encrypted data d and the data type dt using the key *NodeActionKey* and the encryption algorithm stored by *IoTPlatform* after the registration phase (Listing 1 in Appendix A). Then the data are subject of the *score assessment* action and sent to specific modules for security and quality score assessment.

Score assessment is also performed for the data received by unregistered sources; they have no *NodeActionKey*, therefore no decryption is performed on data. In such a situation the data is not discarded, but the lower reputation of the source will influence the use in the user service provision.

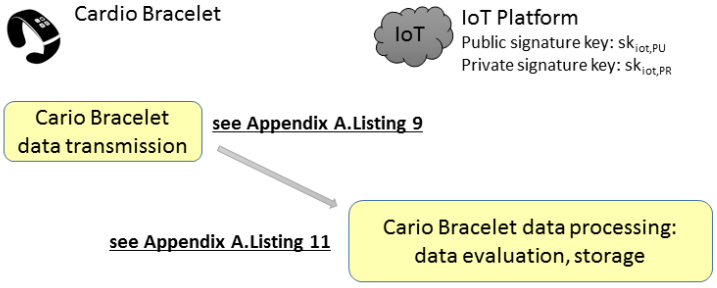
Listing 10 and Listing 11 in Appendix A report the XML representation of data processing for the ECG instrument and the cardio bracelet, respectively. Moreover, the data transmission process is also depicted in Figure 12.

5.4. Users registration and profiles management

IoTPlatform requires user's personal information (i.e., first name, last name, age, role within the smart health environment) and also provides him/her the credentials to access the system itself: *UserSignatureKey* (only the public key) and *UserActionKey*. Information (e.g., first/last name and phone number) are classified as identifiable data, whereas age and nationality are classified as generic data, since they can be exploited only for statistical purposes. An important step is that each user has to give the consent for the processing of personal information in order to let the system use his/her preferences to provide customized services. Such personal information will be used according to the established policies (e.g., information can be used for a specific purpose and under a proper obligation). All data belonging to the user profile are stored also



(a)



(b)

Figure 12: Data transmissionfor (a) ECG and (b) cardio bracelet.

by *IoTPlatform*. Summarizing, the following attributes are considered, in order to define the user profile:

- *Username*: is the nickname chosen by the user when he/she registers himself/herself to *IoTPlatform* (it can be considered, with reference to the model in Section 4, an attribute of *Profile*). Note that the username could be conceived as a pseudonym for the users, thus improving their level of privacy.
- *UserIdentifier*: is an identifier given by *IoTPlatform* to the users.
- *UserSession*: is the session identifier currently owned by the user (i.e., the identifier of the last session in which the user interacted with *IoTPlatform*).
- *UserFunction*: is the function assigned by *IoTPlatform* to the registered user. For example, a user could be a consumer of the services (i.e., a simple customer) offered by *IoTPlatform* or an administrator of the resources and information involved in the IoT infrastructure (e.g., for statistics or monitoring purposes).
- *UserSignatureKey*: is the public key of the user, sent during the registration phase. It is used for authentication purposes (as already described for the node).
- *UserActionKey*: is the symmetric key given to the user by *IoTPlatform* after the registration phase, which is used to encrypt the information exchanged between user and *IoTPlatform*. Note that a user could have more than one action key, in relation to the function played at a specific moment (e.g., a user may act both as a customer and an administrator in a certain domain).
- *UserAction*: is the action currently executed by a user (e.g., registration, login, service request) Each action is coupled with further attributes:

- *Registration* (possible values yes/no): points out if the user is regularly registered or not. If not, further interactions with the IoT system must be prevented
- *Consent* (possible values yes/no): points out if the user gave the consent to *IoTPlatform* to handle his/her personal data
- *UpdatePreference* (possible values not-specified/updated): points out if the user has specified the preferences related to the IoT services (in this case this attribute is set to *updated*) or not (in this case the user does not specify any preferences, therefore no customization is provided by *IoTPlatform*)
- *ServiceRequest* (possible values enabled/disabled): points out if the user is enabled to request services to *IoTPlatform* (e.g., a user may be disabled for the misuse of the services).

- *PreferencesOnService*: are divided in:
  - *PreferencesOnSecurity*: specifies the required security requirements and their priority, which is specified by means of the *order* attribute (i.e., in the form of a decreasing scale from 1 to 4). The available properties are confidentiality, integrity, privacy, and authentication; to each of the user can specify a score in the range [0,1]. The meaning of such scores will be clarified later
  - *PreferencesOnQuality*: specifies the required quality requirements and their priority, which is specified by means of the *order* attribute. The available properties are completeness, timeliness, accuracy, and source reputation. Note that, as well as for *PreferencesOnSecurity*, it is possible to assign the same order to different properties and also a score. The order will be used by *IoTPlatform* when a user makes a request for a service. The information provided by *IoTPlatform* will be compliant with the desired order (e.g., the integrity of the information takes the priority with respect to privacy and confiden-

tiality).

Doctor registration and profile update: As an example, the doctor registers himself to *IoTPlatform* in order to utilize the services provided, and chooses *doctor* as his username. The request for registration is an action performed by users and denoted by registration in Listing 12 in Appendix A; it includes the following information as input: the chosen username, the public key (i.e., *UserSignatureKey.public*), the function for which he wants to register (e.g., doctor, cardiology), and the consent for handling his data.

IoTPlatform responds with a confirmation message (the performed action is denoted by *registration response*) to the requesting user. Such a response includes a summary of the registration (i.e., it confirms the received username and function) and the successful actions performed, which are: (i) the assignment of an identifier and a session; (ii) the request for the consent for interacting with the IoT system (action *consent acquisition* in the model in Section 4); (iii) the registration within *IoTPlatform*; (iv) the enabling for the request of services; (v) the public key of the platform (i.e., *IoTPlatformSignatureKey.public*). Identifier a session are transmitted together with their digital signs, obtained with *IoTPlatformSignatureKey.private*).

The registration process performed by the doctor is depicted in Figure 13.

Furthermore, *IoTPlatform* informs the user about the possibility of specifying his preferences about the security and quality levels of the information provided to him in future interactions (at this time, the preferences are set to *not specified*). Finally, *IoTPlatform* provides the user with the proper *UserActionKey* (see Listing 13 in Appendix A). In other words, the service provider gives the smartphone the required credentials, which aim at guaranteeing access to the service itself by means of secure communications.

After the registration phase, the doctor may specify his preferences, which are elaborated by *IoTPlatform* in the way presented in Listing 14 in Appendix A referred to a doctor profile. In this example, the doctor requires a high level of confidentiality (*order* 1 and score equal or greater than 0.5), and gives less

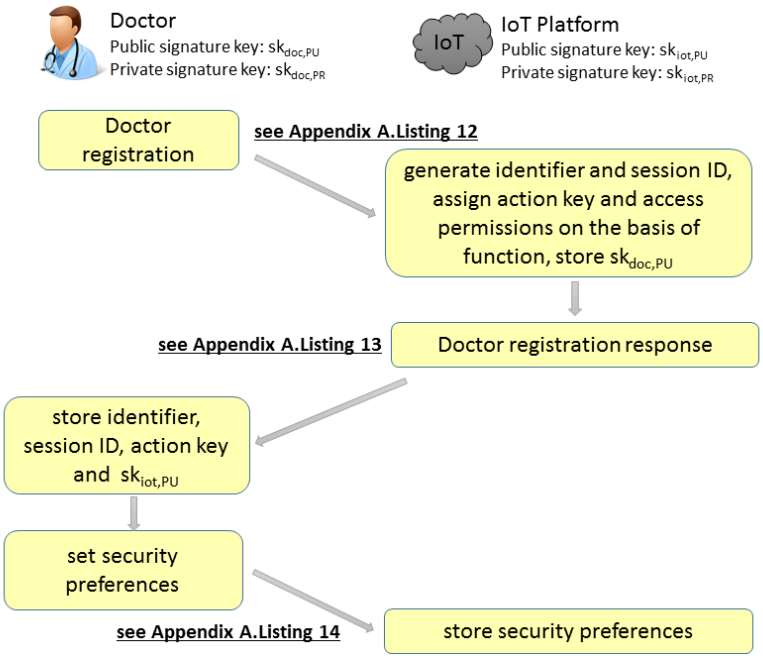


Figure 13: Registration phase for the Doctor

importance to integrity (*order* 2 and score equal or greater than 0.6), privacy and authentication (*order* 3 and 4, and scores equal or greater than 0.8). As regards data quality, he requires a high level of completeness (*order* 1 and score equal or greater than 0.8), timeliness (*order* 2 and score equal or greater than 0.5), accuracy (*order* 3 and score equal or greater than 0.8), and source reputation (*order* 4 and score equal or greater than 0.6).

Note that the attribute related to the preferences is set to *updated* and the user profile definition is the action performed by *IoTPlatform*, as presented in the model in Section 4. Summarizing, after the registration, the users can connect themselves to *IoTPlatform* through their personal devices using the exchanged credentials, then communicate with *IoTPlatform*. The services are provided to the users taking into account both *Profile* and *PreferenceOnService*, which can always be modified by the user through the downloaded application itself.

Visitor registration and profile update: The visitor registers himself to *IoTPlatform* in order to exploit the services provided, and chooses *visitor* as his username. The request for registration is described in Listing 15 in Appendix A. The corresponding confirmation message is reported in Listing 16 in Appendix A. The visitor does not provide any further details useful to update his profile.

The registration process performed by the visitor is depicted in Figure 14.

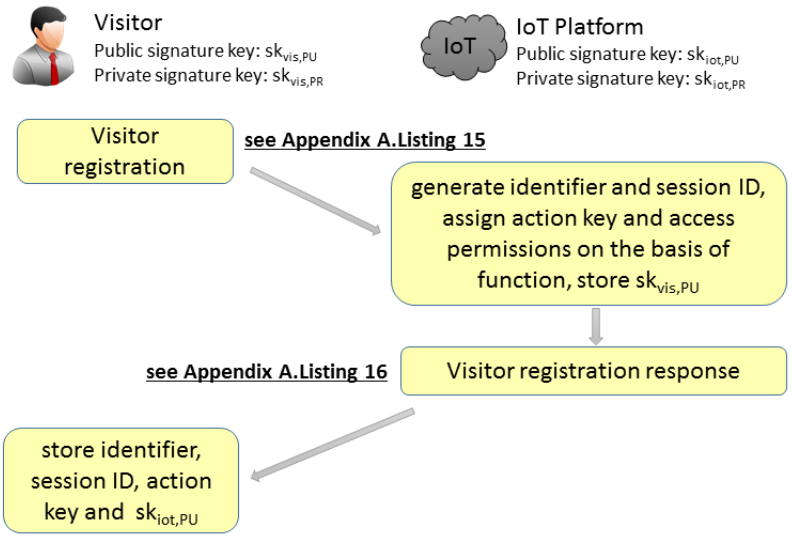


Figure 14: Registration phase for the visitor

5.5. User access control

Once the users are registred to *IoTPlatform* and have an associated profile, they can request the services provided by the IoT system in compliance with the owned function defined during the registration phase. In this context, access control is the first operation to be executed.

In the considered use case, the doctor accesses the IoT system through his username and identifier. The digital sign of the identifier, generated by using *UserSignatureKey.private* is provided too. Moreover, the doctor registers himself for a session playing a specific function (in this case, cardiology), therefore

the credentials have to match also with the desired function, since further authorization is derived by *IoTPlatform* from this information. Also in this case, the task performed by *IoTPlatform* is the verification of the digital sign of the identifier.

The procedure is presented in Listing 17 in Appendix A. The first performed action is the evaluation of the existence of the user with username "doctor" with the specified cardiology function and identifier in the repository. Note that the service request and provision depend on the function currently performed by the user himself. To provide further insight, the visitor access operation is presented in Listing 18 in Appendix A.

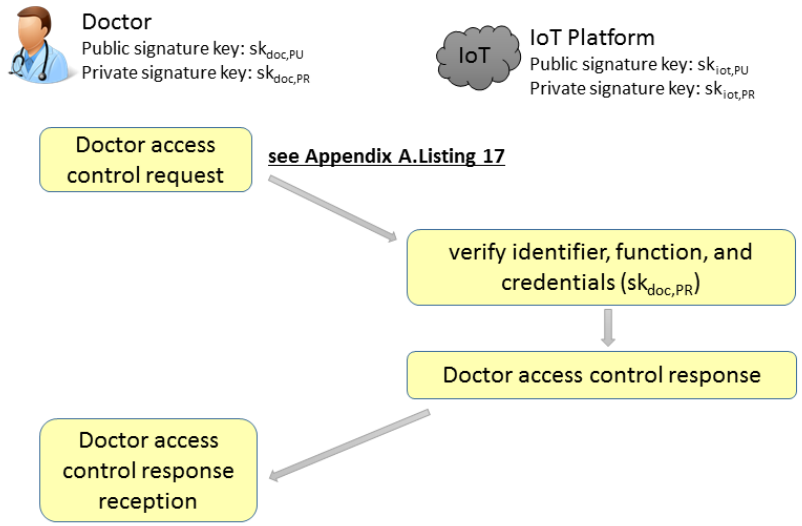
The access control process for both doctor and visitor is depicted in Figure 15.

5.6. Service provisioning

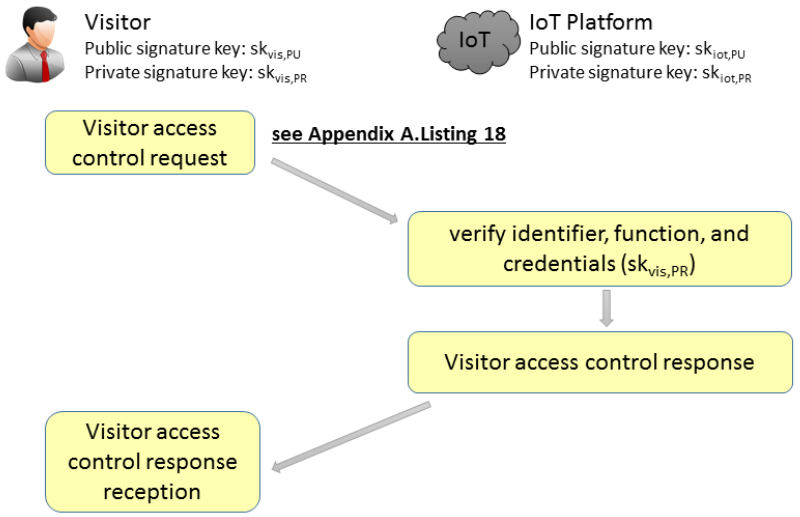
A registered user can ask for the services made available by *IoTPlatform*. Note that user interactions with the IoT system are complex, since data management has to be customized depending on user functions and preferences in terms of security and data quality.

Medical care service allowed to the doctor. Suppose that a doctor wants to be aware about the alerts generated by registered monitoring devices with high levels of security (the ECG machine in the presented example). The request of this service is made in a secure manner (i.e., the information are encrypted with *UserActionKey* sent in the registration phase). *IoTPlatform* receives as input his username, his function, and the requested service *CardiologyAlerts*. Listing 19 in Appendix A shows the corresponding XML representation.

In order to obtain remote services and resources, the user access attempts are checked against the policies considering: the requester's current session id, the activated functions, and the available permissions. Moreover, as regards the service invocation, we treat the services as object instances identified by a hierarchical name space (e.g. a URI). A service is conceived as a piece of



(a)



(b)

Figure 15: Access control for (a) doctor and (b) visitor.

software able to fulfill a specific task managing the available data. There is no direct interaction among users and services, but a well-defined programming interface is needed through a software application. Services can be accessed by users once they are published as object instances.

Now, the service request, previously encrypted with the proper *UserActionKey*, has to be decrypted, in order to verify if it is a valid request. If not, the request is discarded by the system. In order to verify the validity of the request, *IoTPlatform* compares the function played by the requesting user and the function provided by the system itself (i.e., *doctor, cardiology*): if the requesting user is not a doctor with the cardiology attribute, then the request is discarded. However, if the requesting user corresponds to a doctor with cardiology attribute, then *IoTPlatform* proceeds with the service verification action, in which the service *CardiologyAlerts* request is decrypted with the proper *UserActionKey*. At this point, the system retrieves (through the function *getData*) the data corresponding to the requested service *CardiologyAlerts* (temporarily stored in the variable named *listOfData*) and filters them on the basis of user preferences in terms of security and quality levels (specified in Listing 14 in Appendix A). In Listing 20 in Appendix A, the XML code expresses this behavior, defining a function named *dataComplianceForUser* with two parameters (the source of the data and the username). Such a function retrieves the security and quality scores related to the source and compares them with the user preferences. As a result, only the compliant data are sent to the interested user. Note that the proposed language is independent from the implementation of the presented functions (e.g., *verification*, *dataComplianceForUser*).

To conclude, Figure 16 shows the interaction between doctor and platform during the service provisioning process.

Medical care service denied to the visitor: Since the goal of an enforcement mechanism is to force the system to be compliant with the defined policies, then it has to take the proper countermeasures when a violation is detected (e.g., a user tries to request services or perform actions denied to him/her).

For example, the visitor is only allowed to enjoy cafeteria service. Therefore,

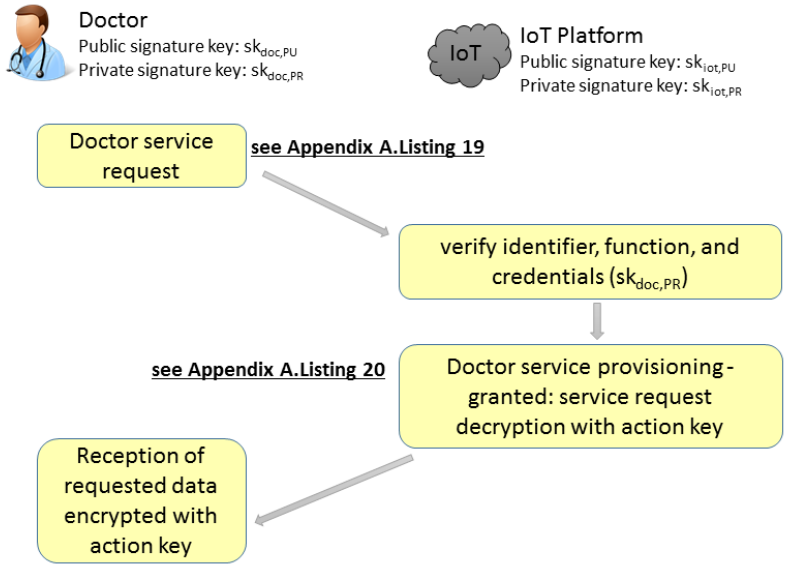


Figure 16: Medical care service allowed to the doctor

in the case it tries to access the data generated by ECG and cardio bracelet, the enforcement mechanism has to prevent the disclosure of the requested data. These are only available, for instance, to the doctors of the cardiology department.

Also in this case, the request of this service is made in a secure manner (i.e., the information are encrypted with *UserActionKey* sent in the registration phase). *IoTPlatform* receives in input his username, his function and the requested service *CardiologyAlerts*. Listing 21 in Appendix A shows the corresponding XML representation.

According to the access policy, the visitor's request is denied. Listing 22 in Appendix A shows the *IoTPlatform* behavior in such a situation. Note that, with respect to Listing 20, *IoTPlatform* does not proceed with the retrieval of the data corresponding to the requested service, since the username-function does not match with the policy established for such a service. Moreover, the enforcement framework may log such kinds of events in order to inform the system administrators of violation attempts.

To conclude, Figure 17 shows the interaction between visitor and platform during the service provisioning process.

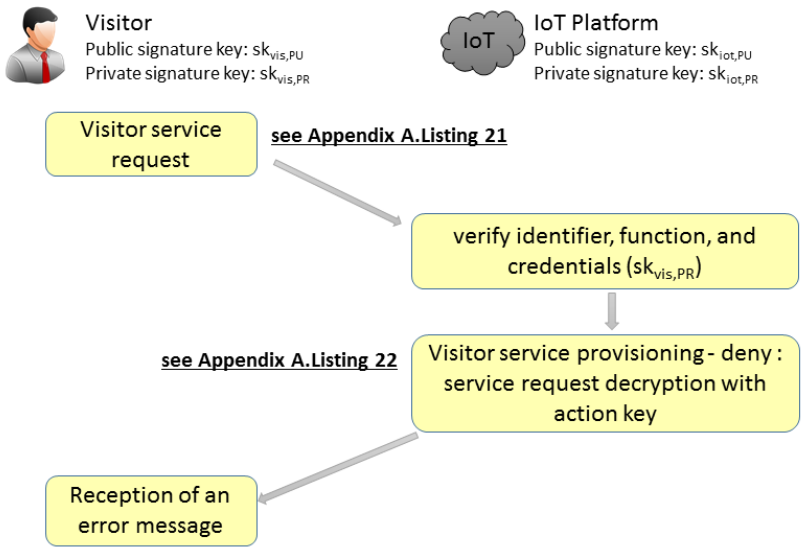


Figure 17: Medical care service denied to the visitor

We remark that this represents an important example of application of the proposed enforcement framework. It may be easily adapted to other application scenarios by defining new functions for the users and new kinds of data provided by the nodes.

5.7. Final considerations on security

The discussed solution aims at mitigating and/or counteracting possible violation attempts or attacks to the IoT system. More in detail, the main task performed by the enforcement framework is the verification of the correct access to the resources. To this end, authentication is required by the interested users, and the services' information is provided in an encrypted way. In this case, an impersonation attack from an illegitimate entity may hinder the confidentiality of the transmitted data. However such a malicious entity, in order to succeed in the attack, should know both the signature key to authenticate itself to the

IoT platform and the action key to decrypt the received data. This mechanism, based on two separated keys, certainly improves the reliability of the system. Also, a man-in-the-middle attack, that may intercept the information if the adopted encryption mechanism is weak, is mitigated by the use of two keys. A more robust system could be obtained by introducing X.509 certificates in order to validate the signature keys at the registration phase for both registered sources and users. Moreover, the encryption of the timestamp associated to the transmitted message allows prevetion of potential replay attacks. Note that we did not point out a specific encryption schema, since it can be freely chosen by the administrators of the IoT platform according to the domain needs.

The main issue arises with the unencrypted data sent by unregistered sources. In this case, the integrity of such information may be compromised by malicious entities. The enforcement framework does not prevent this behavior, but low scores will be assigned to security and quality levels of the data received by such sources, thus informing the users of the nature of these data.

Whereas, the enforcement framework does not deal with attacks to network resources (e.g., denial of service), but we are considering, as future work, to introduce an intrusion detection system, able to recognize various kinds of attacks and, thus, put in place proper protection mechanisms.

6. Discussion

To provide further insight, this section discusses some important requirements that should be taken into account in the case the proposed IoT-based policy enforcement system will be integrated into concrete smart health architectures.

6.1. Storage requirements

As regards storage, the proposed IoT Platform requires storing the following information, possibly in proper repositories (e.g., on a dedicated server, or on a cloud):

- The data related to registered sources
- The users' profiles
- The policies
- The data received by the IoT Platform (e.g., the raw data collection).

Note that the data transmitted to the IoT Platform should not be permanently stored on the platform itself, because an IoT system is conceived as a mean for providing services in real time with current data, thus proper routines can be developed in order to remove obsolete information and avoid congestion states.

Therefore, the actual storage depends on the number of connected users, and the number of active policies. These are further established on the basis of the kinds of services offered by the IoT platform itself and the quantity of users' attributes used for users' profile definitions (e.g., the number of users' functions made available).

6.2. Software/hardware requirements

With reference to the nodes, the described example reveals that the registered devices (e.g., the ECG) should be equipped with a proper interface for communications within the IoT platform. Such an additional module may consist of a small hardware component to be integrated into the nodes, or in a piece of software or an app for mobile devices. Therefore, the costs of deployment among medical equipment of such network interfaces and/or the development of mobile apps must be taken into account. In fact, the involved users, including doctors, nurses, patients, and visitors, must download the smart health app in order to access the services provided by the IoT platform and manage their profiles. Note that communication will take place, for example, by means of wired cables, Wi-Fi, 4G, or Bluetooth.

The enforcement framework acts as a wrapper, able to filter all the interactions among such entities, and is fully installed on the IoT platform. It is worth

noting that the aforementioned interfaces are indeed exposed by the enforcement framework (i.e., the PEP detailed in Section 3), because all the communications among the IoT platform and nodes/users must be controlled.

6.3. Bandwidth requirements

Among all the procedures described in Section 5, data transmission and service provisioning are those that mainly influence the bandwidth consumption. Nodes/users registration and access control are handled with a low frequency with respect to the other tasks.

To investigate bandwidth requirements, a scenario composed by a number of registered nodes and users is taken into account. Let λ_N and λ_U be the data generation rate and the request generation rate handled by the system, respectively. Moreover, S_{Data} represents the size of the data generated within the platform. The conducted study aims at estimating the upper bound of the aggregate data rate that *IoTPlatform* must handle in different load conditions. To this end, the following parameters setting was considered: $\lambda_N \in [1 : 1000]$ transmission/s, $\lambda_U \in [1 : 1000]$ request/s, and $S_{Data} \in [10 : 1000]$ B.

Bandwidth consumption is due to many contribution, that are described below (some of them were evaluated by using Wireshark and postman tools; the others are set as input variables):

- At the application layer, nodes/users and platforms interact with the HyperText Transfer Protocol over transport layer Security (HTTPS) protocol. The establishment of a Transport Layer Security (TLS) connection requires the exchange of 4365 B (Bytes) of information, as summarized in Figure 18;
- The TCP/IP protocol introduces an overall overhead of 100 B (66 B for the TCP header with additional flags, 20 B for the IP header, and 14 B for the IEEE 802.3 header). That overhead should be considered for application messages, TCP Acknowledgment, as well as for the for way handshake closing the TCP connection;

	Source	Destination	Protocol	Length (Byte)	Info
TCP Handshake	ECG	IoT Platform	TCP	78	63560 > 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=627992018 TSecr=0 SACK_PERM=1
	IoT Platform	ECG	TCP	74	443 > 63560 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1058102401 TSecr=627992018 WS=128
	ECG	IoT Platform	TCP	66	63560 > 443 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=627992019 TSecr=1058102401
TLS	ECG	IoT Platform	TLSv1.2	624	Client Hello
	IoT Platform	ECG	TCP	66	443 > 63560 [ACK] Seq=1 Ack=559 Win=30080 Len=0 TSval=1058102401 TSecr=627992019
	IoT Platform	ECG	TLSv1.2	1514	Alert (Level: Warning, Description: Unrecognized Name), Server Hello
	IoT Platform	ECG	TLSv1.2	1279	Certificate
	ECG	IoT Platform	TCP	66	63560 > 443 [ACK] Seq=559 Ack=2662 Win=129088 Len=0 TSval=627992039 TSecr=1058102406
	ECG	IoT Platform	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
	IoT Platform	ECG	TLSv1.2	340	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
	IoT Platform	ECG	TCP	66	63560 > 443 [ACK] Seq=685 Ack=2936 Win=130784 Len=0 TSval=627992041 TSecr=1058102407

Overhead due to the TCP handshake (Byte)	**218**
Overhead due to the TLS handshake (Byte)	**4147**
Total overhead (Byte)	**4365**

Figure 18: Bandwidth consumption due to the initialization of a TLS connection

- Data are transmitted to the URI

  https://[ip address of the platform]/datatransmission,

  through a HTTP POST. By assuming 10 B for session, identifier, and datatype fields (Listing 8 in Appendix A), the XML message has a size equal to $(160 + S_{Data})$ B. The encrypted HTTP payload, instead, registers a size of $(454 + S_{Data})$ B.

- Service requests are set to the URI

  https://[ip address of the platform]/servicerequest,

  through a HTTP POST. By assuming 10 B for session, username, identifier, function, and service fields (see Listing 19 in Appendix A), the XML message has a size equal to 177 B. The encrypted payload registers a size of 469 B.

- service requests are set to the URI

  https://[ip address of the user]/serviceprovisioning,

  through a HTTP POST. By assuming the transmission of only one data for each request, the XML message has a size equal to $(36 + S_{Data})$ B. The encrypted payload, instead, registers a size of $(333 + S_{Data})$ B.

Results are reported in Figures 6.3-6.3. As expected, bandwidth requirements increase with the amout of data transmissions and service requests generated in the unit of time. At the same time, it is little influenced by the size of the data. S_{Data} provides a limited contribution to the entire overhead, if compared with respect to HTTPS and TCP/IP protocols. In any case, it emerges that the overall communication overhead does not exceed 150 Mbps. Therefore, it is possible to conclude that the proposed approach is feasible also in large-scale environments.

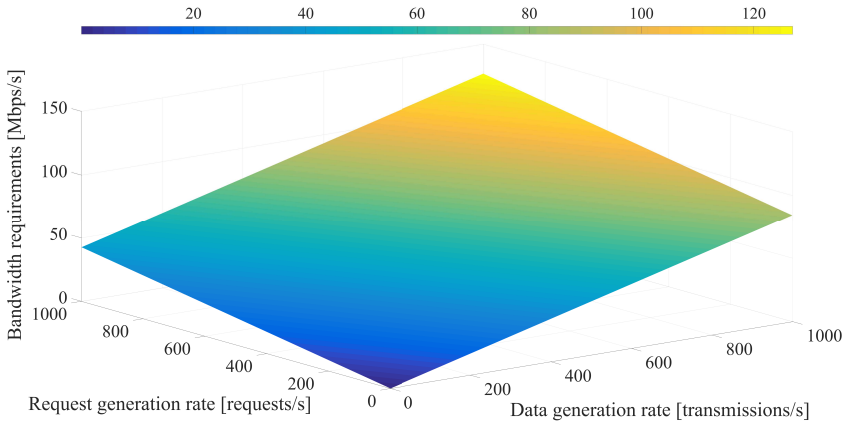


Figure 19: Bandwidth requirements, $S_{Data} = 10$ B

7. Conclusions and future works

Security threats risk to refrain the development of smart health applications in large scale heterogeneous scenarios. To this end, a flexible security

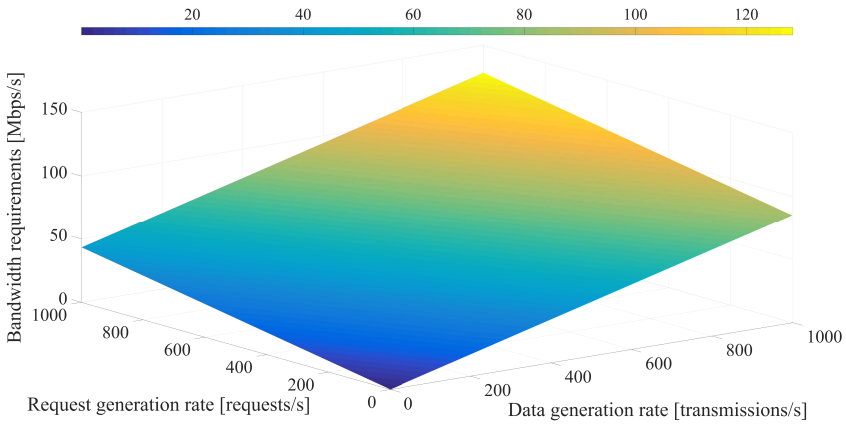


Figure 20: Bandwidth requirements, $S_{Data} = 100$ B

enforcement framework has been proposed in this manuscript along with a policy definition language, based on XML formalism. The proposed framework is able to provide policy enforcement primitives suitable across different administrative domains in general, or specifically related to healthcare infrastructures. Its effectiveness has been demonstrated through a running example in a concrete use case, tailored to a smart health application. As regards other future works, we will focus on the deployment of the framework in an ad-hoc prototype in order to test its real robustness in a smart health distributed environment. Moreover, proper mechanisms for enabling policies' combination and conflicts' resolution will be investigated, taking into account that multiple policy administrators may belong to the IoT application area, as well as different smart health environments may coexist (e.g., hospitals, clinics, pharmacies) and require conflicting policies on data. In this case, the actual centralized approach should be overcome in order to consider a distributed policy management and handle proper mechanisms for policy propagation and synchronization.

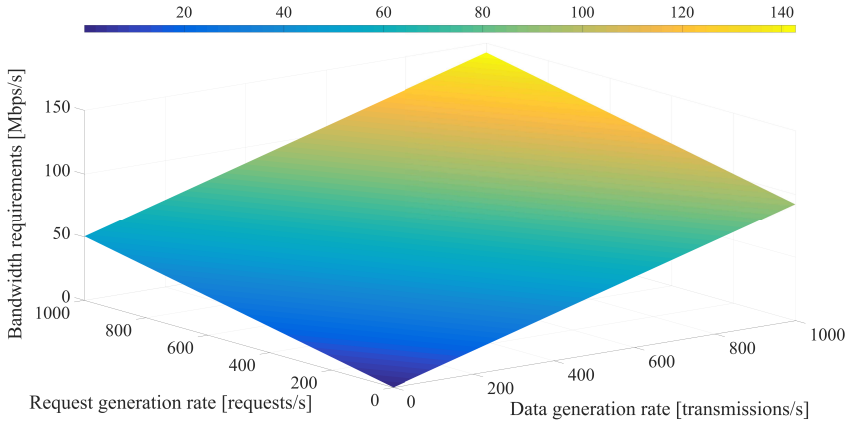


Figure 21: Bandwidth requirements, $S_{Data} = 1000$ B

References

[1] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805.

[2] D. Miorandi, S. Sicari, F. De Pellegrini, I. Chlamtac, Survey internet of things: Vision, applications and research challenges, Ad Hoc Netw. 10 (7) (2012) 1497–1516.

[3] M. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. Grieco, G. Boggia, M. Dohler, Standardized protocol stack for the internet of (important) things, Communications Surveys Tutorials, IEEE 15 (3) (2013) 1389–1406.

[4] B. Emmerson, M2M: the Internet of 50 billion devices, Huawei Win-Win Magazine Journal (4) (2010) 19–22.

[5] D. Boswarthick, O. Elloumi, O. Hersent, M2M Communications: A Systems Approach, 1st Edition, Wiley Publishing, 2012.

[6] O. Hersent, D. Boswarthick, O. Elloumi, The Internet of Things: Key Applications and Protocols, 2nd Edition, Wiley Publishing, 2012.

[7] L. Catarinucci, D. de Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, L. Tarricone, An iot-aware architecture for smart healthcare systems, IEEE Internet of Things Journal 2 (6) (2015) 515–526.

[8] P. A. Laplante, N. Laplante, The internet of things in healthcare: Potential applications and challenges, IT Professional 18 (3) (2016) 2–4.

[9] B. Xu, L. D. Xu, H. Cai, C. Xie, J. Hu, F. Bu, Ubiquitous data accessing method in iot-based information system for emergency medical services, IEEE Transactions on Industrial Informatics 10 (2) (2014) 1578–1586.

[10] Y. YIN, Y. Zeng, X. Chen, Y. Fan, The internet of things in healthcare: An overview, Journal of Industrial Information Integration 1 (2016) 3 – 13.

[11] K. Ullah, M. A. Shah, S. Zhang, Effective ways to use Internet of Things in the field of medical and smart health care, in: Proc. of IEEE International Conference on Intelligent Systems Engineering (ICISE), 2016, pp. 372–379.

[12] D. Altolini, V. Lakkundi, N. Bui, C. Tapparello, M. Rossi, Low power link layer security for IoT: Implementation and performance analysis, in: Proc. of Int. Wireless Communications and Mobile Computing Conf. (IWCMC), 2013, pp. 919–925.

[13] S. Bandyopadhyay, M. Sengupta, S. Maiti, S. Dutta, A survey of middleware for internet of things, in: Third International Conferences, WiMo 2011 and CoNeCo 2011, Ankara, Turkey, 2011, pp. 288–296.

[14] M. A. Chaqfeh, N. Mohamed, Challenges in middleware solutions for the internet of things, in: 2012 International Conference on Collaboration Technologies and Systems (CTS), Denver, CO, 2012, pp. 21–26.

[15] Q. M. Ashraf, M. H. Habaebi, Autonomic schemes for threat mitigation in internet of things, Journal of Network and Computer Applications 49 (0) (2015) 112 – 127.

[16] S. Babar, A. Stango, N. Prasad, J. Sen, R. Prasad, Proposed embedded security framework for internet of things (iot), in: 2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace and Electronic Systems Technology, Wireless VITAE 2011, Chennai, India, 2011, pp. 1 – 5.

[17] R. H. Weber, Internet of things - new security and privacy challenges, Computer Law & Security Review 26 (1) (2010) 23–30.

[18] R. Roman, J. Zhou, J. Lopez, On the features and challenges of security and privacy in distributed internet of things, Computer Networks 57 (10) (2013) 2266–2279.

[19] Z. Yan, P. Zhang, A. V. Vasilakos, A survey on trust management for internet of things, Journal of Network and Computer Applications 42 (0) (2014) 120 – 134.

[20] U. Premarathne, A. Abuadbba, A. Alabdulatif, I. Khalil, Z. Tari, A. Zomaya, R. Buyya, Hybrid cryptographic access control for cloud-based ehr systems, IEEE Cloud Computing 3 (4) (2016) 58–64.

[21] M. F. F. Khan, K. Sakamura, Fine-grained access control to medical records in digital healthcare enterprises, in: Proc. of International Symposium on Networks, Computers and Communications (ISNCC), 2015, pp. 1–6.

[22] M. Barua, X. Liang, R. Lu, X. Shen, Peace: An efficient and secure patient-centric access control scheme for ehealth care system, in: Proc. of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2011, pp. 970–975.

[23] H. S. G. Pussewalage, V. A. Oleshchuk, An attribute based access control scheme for secure sharing of electronic health records, in: Proc. of IEEE International Conference on e-Health Networking, Applications and Services (Healthcom), 2016, pp. 1–6. doi:10.1109/HealthCom.2016.7749516.

[24] A. Coen Porisini, P. Colombo, S. Sicari, Privacy aware systems: from models to patterns, igi global Edition, 2011.

[25] D. Barbagallo, C. Cappiello, A.Coen-Porisini, P. Colombo, M. Comerio, F. D. Paoli, C. Francalanci, S. Sicari, Towards the definition of a framework for service development in the agrofood domain: A conceptual model, in: WEBIST 2012, Porto, Portugal, 2012.

[26] S. Sicari, A. Rizzardi, L. A. Grieco, A. Coen-Porisini, Security, privacy and trust in internet of things: The road ahead, Computer Networks 76 (2015) 146–164.

[27] S. Sicari, S. Hailes, D. Turgut, S. Sharaffedine, D. U., Security, Privacy and Trust Management in the Internet of Things era- SePriT, 11th Edition, Special Issue of Ad Hoc networks, Elsevier, 2013.

[28] R. Neisse, G. Steri, G. Baldini, Enforcement of security policy rules for the internet of things, in: Proc. of IEEE WiMob, Larnaca, Cyprus, 2014, pp. 120–127.

[29] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, A. Coen-Porisini, Security policy enforcement for networked smart objects, Computer Networks 108 (2016) 133–147.

[30] A. A. A. El-Aziz, A. Kannan, Access control for healthcare data using extended xacml-srbac model, in: Proc. of International Conference on Computer Communication and Informatics, 2012, pp. 1–4.

[31] M. Anwar, A. Imran, Access control for multi-tenancy in cloud-based health information systems, in: Proc. of IEEE International Conference on Cyber Security and Cloud Computing, 2015, pp. 104–110.

[32] Z. Wu, L. Wang, An innovative simulation environment for cross-domain policy enforcement, Simulation Modelling Practice and Theory 19 (7) (2011) 1558–1583.

[33] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, K. S. Kwak, The internet of things for health care: A comprehensive survey, IEEE Access 3 (2015) 678–708. doi:10.1109/ACCESS.2015.2437951.

[34] M. Dell'Amico, M. S. I. G. Serme, A. S. de Oliveira, Y. Roudier, Hipolds: A hierarchical security policy language for distributed systems, Information Security Technical Report 17 (3) (2013) 81–92.

[35] D. Ferraiolo, V. A. ans S. Gavrila, The policy machine: A novel architecture and framework for access control policy specification and enforcement, Journal of Systems Architecture 57 (4) (2011) 412–424.

[36] J. Rao, A. Sardinha, N. Sadeh, A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources, Web Semantics: Science, Services and Agents on the World Wide Web 7 (1) (2009) 40 – 56.

[37] J. Rao, A. Sardinha, N. Sadeh, A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources, Web Semantics: Science, Services and Agents on the World Wide Web 7 (1) (2009) 40–56.

[38] S. Sicari, A. Rizzardi, A. Coen-Porisini, L. A. Grieco, T. Monteil, Secure om2m service platform, in: Autonomic Computing (ICAC), 2015 IEEE International Conference on, 2015, pp. 313–318.

[39] N. Ulltveit-Moe, V. Oleshchuk, Decision-cache based XACML authorisation and anonymisation for XML documents, Computer Standards & Interfaces 34 (6) (2012) 527 – 534.

[40] V. Goyal, O. Pandey, A. Sahai, B. Waters, Attribute-based encryption for fine-grained access control of encrypted data, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, 2006, pp. 89–98.

[41] S. Sicari, A. Rizzardi, A. Coen-Porisini, C. Cappiello, A NFP model for internet of things applications, in: Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on, IEEE, Larnaca, Cyprus, 2014, pp. 265–272.

[42] Q. Ni, A. Trombetta, E. Bertino, J. Lobo, Privacy-aware role based access control, in: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, ACM, New York, USA, 2007.

[43] S. Sicari, L. A. Grieco, G. Boggia, A. Coen-Porsini, Dydap: A dynamic data aggregation scheme for privacy aware wireless sensor networks, Elsevier Journal of Systems and Software 88 (1) (2012) 152–166.

[44] S. Sicari, L. Grieco, A. Rizzardi, G. Boggia, A. Coen-Porisini, SETA: A secure sharing of tasks in clustered wireless sensor networks, in: 9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2013, Lyon, France, 2013, pp. 239–246.

[45] S. Sicari, A. Coen-Porisini, R. Riggio, Dare: evaluating data accuracy using node reputation, Elsevier Computer Networks 57 (15) (2013) 3098–3111.

[46] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, A. Coen-Porisini, A secure and quality-aware prototypical architecture for the I nternet of Things, Information Systems 58 (2016) 43–55.

Appendix A. XML representations

This appendix collects all the XML listings related to the transactions presented in Section 5. In the XML codes presented in this work, the tag <*input*> identifies the information obtained, while the tags <*security*> and <*verification*> highlight the performed security tasks.

ECG registration.

```
<action type=='registration'>
  <input>
```

```
    <node>
       <source>ecg</source>
       <communicationmode>ethernet</communicationmode>
       <datatype>integer</datatype>
       <encryptionscheme>RSA</encryptionscheme>
        <signaturekey>NodeSignatureKey.public</signaturekey>
  </node>
  </input>
</action>
```

Listing 1: ECG registration

ECG registration response.

```
<action type=='node registration response'>
    <node>
      <session>encrypt(33333, IoTPlatformSignatureKey.private)</
    session>
      <identifier>encrypt(id003, IoTPlatformSignatureKey.private)</
    identifier>
      <signaturekey>IoTPlatformSignatureKey.public</signaturekey>
      <actionkey>*****</actionkey>
  </node>
</action>
```

Listing 2: ECG registration response provided by IoTPlatform

ECG access control.

```
<action type='access control'>
  <input>
    <node>
      <identifier>id003, encrypt(id003, NodeSignatureKey.private)</
    identifier>
      <signaturekey>NodeSignatureKey.private</signaturekey>
    </node>
  </input>
  <security>
    <verification>
      <if registration='registered=='yes' '>
        <signaturekey>NodeSignatureKey.public</signaturekey>
```

```
        <identifier>decrypt(id003, NodeSignatureKey.public)</
    identifier>
      <else/>
        <signaturekey>undefined</signaturekey>
        <identifier>id003</identifier>
      </if>
    </verification>
  </security>
</action>
```

Listing 3: ECG access control

ECG access control response.

```
<action type='access control response'>
    <node>
       <session>33333, encrypt(33333, IoTPlatformSignatureKey.
    private)</session>
       <if registration='registered=='yes' '>
           <access>yes</access>
       <elseif registration='registered=='no' '>
           <access>yes</access>
        <else>
            <identifier>id003</identifier>
       </if>
  </node>
</action>
```

Listing 4: ECG access control response

Cardio bracelet access control.

```
<action type='access control'>
  <input>
    <node>
      <identifier>id004</identifier>
    </node>
  </input>
  <security>
    <verification>
      <if registration='registered=='yes' '>
        <signaturekey>NodeSignatureKey.public</signaturekey>
```

```
        <identifier>decrypt(id004, NodeSignatureKey.public)</
    identifier>
      <else/>
        <signaturekey>undefined</signaturekey>
        <identifier>id004</identifier>
      </if>
    </verification>
  </security>
</action>
```

Listing 5: Cardio bracelet access control

Cardio bracelet access control response.

```
<action type='access control response'>
    <node>
       <session>44444</session>
       <if registration='registered=='yes' '>
           <access>yes</access>
       <elseif registration='registered=='no' '>
           <access>yes</access>
        <else>
            <identifier>id004</identifier>
       </if>
  </node>
</action>
```

Listing 6: Cardio bracelet response

Node definition.

```
<nodes>
  <node registered='yes'>
       <identifier>id003</identifier>
       <source>ecg</source>
       <communicationmode>ethernet</communicationmode>
       <datatype>integer</datatype>
       <encryptionscheme>RSA</encryptionscheme>
        <actionkey>*****</actionkey>
        <signaturekey>NodeSignatureKey.public</signaturekey>
        <securitymetadata>
    <confidentiality>0.8</confidentiality>
```

```
    <integrity>0.8</integrity>
    <authentication>0.6</authentication>
    <privacy>0.4</privacy>
       </securitymetadata>
        <qualitymetadata>
    <completeness>0.7</completeness>
    <accuracy>0.8</accuracy>
    <timeliness>0.7</timeliness>
    <reputation>0.6</reputation >
       </qualitymetadata>
    </node>

    <node registered='no'>
       <identifier>id004</identifier>
       <source>cardio bracelet</source>
       <datatype>double</datatype>
     <encryptionscheme>undefined</encryptionscheme>
        <actionkey>undefined</actionkey>
        <signaturekey>undefined</signaturekey>
        <securitymetadata>
    <confidentiality>0.2</confidentiality>
    <integrity>0.3</integrity>
    <authentication>0</authentication>
    <privacy>0.2</privacy>
       </securitymetadata>
        <qualitymetadata>
    <completeness>0.5</completeness>
    <accuracy>0.4</accuracy>
    <timeliness>0.8</timeliness>
     <source reputation>0.4</ source reputation>
       </qualitymetadata>
    </node>
</ nodes>
```

Listing 7: Node storage within *IoTPlatform*

For each node, *IoTPlatform* stores the related information with the tags <*node*>. As a consequence, a block <*node*> is included for all the nodes in the parent tag <*nodes*>.

In the following listings, the tag <*message*> identifies the content of the transmitted packet.

ECG data transmission.

```
<action type='transmission'>
  <input>
    <message>
       <session>33333, encrypt(33333, NodeSignatureKey.private)</session>
      <node>
      <identifier>id003, encrypt(id003, NodeSignatureKey.private)</identifier>
      </node>
      <data>encrypt(d, actionkey)</data>
      <datatype>encrypt(dt, actionkey)</datatype>
    </message>
  </input>
</action>
```

Listing 8: ECG data transmission

Cardio bracelet data transmission.

```
<action type='transmission'>
  <input>
    <message>
       <session>44444</session>
      <node>
      <identifier>id004</identifier>
      </node>
      <data>d </data>
      <datatype>dt </datatype>
    </message>
  </input>
</action>
```

Listing 9: Cardio bracelet data transmission

ECG data processing.

```
<action type='data processing'>
  <input>
```

```
    <message>
       <session>33333, encrypt(33333, NodeSignatureKey.private)</
    session>
      <node>
      <identifier>id003, encrypt(id003, NodeSignatureKey.private)</
    identifier>
      </node>
      <data>encrypt(d, actionkey)</data>
      <datatype>encrypt(dt, actionkey)
          </datatype>
    </message>
  </input>
  <data evaluation>
    <if cond='node.actionkey != undefined'>
        <data>decrypt(d, actionkey)</data>
        <datatype>decrypt(dt, node.actionkey)
          </datatype>
       <score assessment><data>d</data></score assessment>
    <elseif cond='actionkey == undefined'>
        <score assessment><data>d</data></score assessment>
    <else/>
      <discard/>
    </if>
  </data evaluation>
</action>
```

Listing 10: ECG data processing

Cardio bracelet data processing.

```
<action type=='data processing'>
  <input>
    <message>
       <session>44444</session>
      <node>
          <identifier>id004 </identifier>
      </node>
      <data>d </data>
      <datatype>dt
          </datatype>
```

```
    </message>
  </input>
  <data evaluation>
    <if cond='node.actionkey != undefined'>
        <data>decrypt(d, actionkey)</data>
        <datatype>decrypt(dt, node.actionkey)
          </datatype>
       <score assessment><data>d</data></score assessment>
    <elseif cond='actionkey == undefined'>
        <score assessment><data>d</data></score assessment>
    <else/>
      <discard/>
    </if>
  </data evaluation>
</action>
```

Listing 11: Cardio bracelet data processing

Doctor registration.

```
<action type=='registration'>
  <input>
    <user>
      <username>doctor</username>
      <signaturekey>UserSignatureKey.public</signaturekey>
      <function>doctor, cardiology</function>
      <consent>yes</consent>
  </user>
  </input>
</action>
```

Listing 12: Doctor registration

Doctor registration response.

```
<action type=='registration response'>
    <user>
      <username>doctor</username>
      <identifier>id001, encrypt(id001, IoTPlatformSignatureKey.
    private)</identifier>
      <session>11111, encrypt(11111, IoTPlatformSignatureKey.
    private)</session>
```

```
      <function>doctor, cardiology</function>
      <actions consent='yes' registration='yes'
                preferences='not specified'
                servicerequest='enabled'>
        </actions>
      <actionkey>*****</actionkey>
      <signaturekey>IoTPlatformSignatureKey.public</signaturekey>
  </user>
</action>
```

Listing 13: Doctor registration response

Doctor profile definition.

```
<users>
  <user>
    <username>doctor</username>
    <identifier>id001</identifier>
        <function>doctor, cardiology</function>
        <medical care>
          <department>cardiology</department>
        </medical care>
        <cafeteria service>
          <intolerance>gluten</intolerance>
          <intolerance>milk products</intolerance>
        </cafeteria service>
        <actionkey>*****</actionkey>
        <signaturekey>UserSignatureKey.public</signaturekey>
        <actions consent='yes' registration='yes'
                  preferences='updated'
                  servicerequest='enabled'>
        </actions>
        <preferenceonsecurity>
    <confidentiality order='1'>0.5</confidentiality>
    <integrity order='2'>0.6</integrity>
    <privacy order='3'>0.8</privacy>
    <authentication order='4'>0.8</authentication>
       </preferenceonsecurity>
        <preferenceonquality>
    <completeness order='1'>0.8</completeness>
```

```
    <timeliness order='2'>0.5</timeliness>
    <accuracy order='3'>0.8</accuracy>
    <source reputation order='4'>0.6
            </ source reputation>
       </preferenceonquality>
    </user>
</ users>
```

Listing 14: Doctor profile definition within *IoTPlatform*

Note that, for each registered user, *IoTPlatform* stores the information within the tags <*user*>. As a consequence, a block <*user*> is included for all the registered users in the parent tag <*users*>.

Visitor registration.

```
<action type='registration'>
  <input>
    <user>
      <username>visitor</username>
      <signaturekey>UserSignatureKey.public</signaturekey>
      <function>visitor</function>
      <consent>yes</consent>
  </user>
  </input>
</action>
```

Listing 15: Visitor registration

Visitor registration response.

```
<action type='registration response'>
    <user>
      <username>visitor</username>
      <identifier>id002, encrypt(id002, IoTPlatformSignatureKey.
    private)</identifier>
      <session>22222, encrypt(22222, IoTPlatformSignatureKey.
    private)</session>
      <function>visitor</function>
      <actions consent='yes' registration='yes'
                preferences='not specified'
                servicerequest='enabled'>
```

```
      </actions>
      <actionkey>*****</actionkey>
      <signaturekey>IoTPlatformSignatureKey.public</signaturekey>
  </user>
</action>
```

Listing 16: Visitor registration response

Access control of the doctor.

```
<action type='access control'>
  <input>
    <user>
      <username>doctor</username>
      <identifier>id001, encrypt(id001, UserSignatureKey.private)</
    identifier>
      <signaturekey>UserSignatureKey.public</signaturekey>
      <function>doctor, cardiology</function>
  </user>
  </input>
  <security>
    <verification>
      <signaturekey>UserSignatureKey.public</signaturekey>
      <function>doctor, cardiology</function>
      <identifier>decrypt(id001, UserSignatureKey.public)</
    identifier>
    </verification>
  </security>
</action>
```

Listing 17: Doctor access control

Access control of the visitor.

```
<action type='access control'>
  <input>
    <user>
      <username>visitor</username>
      <identifier>id002, encrypt(id002, UserSignatureKey.private)</
    identifier>
      <signaturekey>UserSignatureKey.public</signaturekey>
      <function>visitor</function>
```

```
  </user>
  </input>
  <security>
    <verification>
      <signaturekey>UserSignatureKey.public</signaturekey>
      <function>visitor</function>
      <identifier>decrypt(id002, UserSignatureKey.public)</
    identifier>
    </verification>
  </security>
</action>
```

Listing 18: Visitor access control

Doctor's service request.

```
<action type=='service request'>
  <input>
    <message>
      <session>11111, encrypt(11111, UserSignatureKey.private)</
    session>
      <user><username>doctor</username>
      <identifier>id001, encrypt(id001, UserSignatureKey.private)</
    identifier>
      <function>doctor, cardiology</function></user>
      <service>encrypt(CardiologyAlerts, actionkey)</service>
    </message>
  </input>
</action>
```

Listing 19: Doctor's request for medical care service

Note that the tag <*discard*> indicates a discarded request, while the tag <*result*> includes the information disclosed to the user during the service provision (an empty <*result*> tag means that no data can be transmitted).

Doctor's service provision.

```
<action type=='service provision'>
  <input>
    <message>
```

```
      <session>11111, encrypt(11111, UserSignatureKey.private)</
    session>
      <user><username>doctor</username>
      <identifier>encrypt(id001, UserSignatureKey.private)</
    identifier>
      <function>doctor, cardiology</function></user>
      <service>encrypt(CardiologyAlerts, actionkey)</service>
    </message>
  </input>
  <security>
  <if cond='user.function != 'doctor, cardiology' '>
      <discard/>
    <else/>
      <serviceverification>
  <service>decrypt(CardiologyAlerts, actionkey)</service>
      </serviceverification>
    </if>
  </security>
    <results>
      <declare name='listOfData'
        function='getData' service='CardiologyAlerts'/>
      <foreach item='data' items='listOfData'>
        <if  function='dataComplianceForUser'
        data='CardiologyAlerts.data' user='doctor'>
          <result>data</result>
        <else/>
          <result></result>
        </if>
    </results>
</action>
```

Listing 20: Medical care service provision - doctor

Visitor's service request.

```
<action type=='service request'>
  <input>
    <message>
      <session>22222, encrypt(22222, UserSignatureKey.private)</
    session>
```

```
      <user><username>visitor</username>
      <identifier>id002, encrypt(id002, UserSignatureKey.private)</
    identifier>
      <function>visitor</function></user>
      <service>encrypt(CardiologyAlerts, actionkey)</service>
    </message>
  </input>
</action>
```

Listing 21: Visitor's request for medical care service

Visitor's service request violation.

```
<action type=='service provision'>
  <input>
    <message>
      <session>22222, encrypt(22222, UserSignatureKey.private)</
    session>
      <user><username>visitor</username>
      <identifier>id002, encrypt(id002, UserSignatureKey.private)</
    identifier>
      <function>visitor</function></user>
      <service>encrypt(CardiologyAlerts, actionkey)</service>
    </message>
  </input>
  <security>
    <if cond='user.function != 'doctor, cardiology' '>
      <discard/>
    <else/>
      <serviceverification>
        <service>decrypt(CardiologyAlerts, actionkey)</service>
      </serviceverification>
     </if>
  </security>
  <results>
        <result>Requested data are not available for a user
    registered as a visitor</result>
  </results>
</action>
```

Listing 22: Service request violation