UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

DOCTORAL THESIS

Security in Internet of Things: Networked Smart Objects

Ph.D Thesis of:
Alessandra RIZZARDI

Advisor:
Ch.mo Prof. Sabrina SICARI

Ph.D. Degree Program in
Computer Science and Computational Mathematics

Dipartimento di Scienze Teoriche e Applicate

Academic Year 2015-2016

"Sometimes it's the very people who no one imagines anything of who do the things no one can imagine"

Alan Turing - The Imitation Game

Abstract

Internet of Things (IoT) is an innovative paradigm approaching both industries and humans every-day life. It refers to the networked interconnection of every-day objects, which are equipped with ubiquitous intelligence. It not only aims at increasing the ubiquity of the Internet, but also at leading towards a highly distributed network of devices communicating with human beings as well as with other devices. Thanks to rapid advances in underlying technologies, IoT is opening valuable opportunities for a large number of novel applications, that promise to improve the quality of humans lives, facilitating the exchange of services.

In this scenario, security represents a crucial aspect to be addressed, due to the high level of heterogeneity of the involved devices and to the sensibility of the managed information. Moreover, a system architecture should be established, before the IoT is fully operable in an efficient, scalable and interoperable manner.

The main goal of this PhD thesis concerns the design and the implementation of a secure and distributed middleware platform tailored to IoT application domains. The effectiveness of the proposed solution is evaluated by means of a prototype and real case studies.

Acknowledgements

I wish foremost to thank prof. Sabrina Sicari for her guidance and for the interesting opportunities she gave me during this PhD course.

I am also really grateful with prof. Alberto Coen-Porisini and with the people who collabotated with our research group, in particular Dr. Daniele Miorandi, prof. Roberto Passerone, prof. Luigi Alfredo Grieco and Dr. Cinzia Cappiello, for their precious advice.

I am also thankful to prof. Stephen Hailes and prof. Gianluca Dini, the reviewers of this PhD thesis, for their valuable comments for improving my work.

Finally, a special thanks to my family, to Emanuele, and to all my friends for encouraging me all the time.

Contents

List of Figures

List of Tables

List of Algorithms

List of Equations

List of Abbreviations

IoT	**I**nternet **o**f **T**hings
NOS	**N**etw**O**rked **S**mart object
WSN	**W**ireless **S**ensor **N**etwork
RFID	**R**adio **F**requency **ID**entification
NFC	**N**ear **F**ield **C**ommunication
NF	**N**on **F**unctional
QoP	**Q**uality **o**f **P**rotection
DQ	**D**ata **Q**uality
UML	**U**nified **M**odeling **L**anguage
JSON	**J**ava**S**cript **O**bject **N**otation
XML	e**X**tensible **M**arkup **L**anguage
MQTT	**M**essage **Q**ueue **T**elemetry **T**ransport
CoAP	**Co**nstrained **A**pplication **P**rotocol
QoS	**Q**uality **o**f **S**ervice
AUPS	**AU**thenticated **P**ublish&**S**ubscribe system
PEP	**P**olicy **E**nforcement **P**oint
PDP	**P**olicy **D**ecision **P**oint
PAP	**P**olicy **A**dministration **P**oint
ABAC	**A**ttribute **B**ased **A**ccess **C**ontrol
RBAC	**R**ole **B**ased **A**ccess **C**ontrol
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
HTTPS	**H**yper**T**ext **T**ransfer **P**rotocol over **S**ecure Socket Layer
SSL	**S**ecure **S**ockets **L**ayer
TLS	**T**ransport **L**ayer **S**ecurity
DTLS	**D**atagram **T**ransport **L**ayer **S**ecurity
REST	**RE**presentational **S**tate **T**ransfer
SOA	**S**ervice-**O**riented **A**rchitecture
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
WSDL	**W**eb **S**ervices **D**escription **L**anguage
M2M	**M**achine **to** **M**achine
KMS	**K**ey **M**anagement **S**ystem
AES	**A**dvanced **E**ncryption **S**tandard
RSA	Ronald **R**ivest, Adi **S**hamir, Leonard **A**dleman
MD5	**M**essage **D**igest **5**
SHA-1	**S**ecure **H**ash **A**lgorithm**-1**
PKC	**P**ublic **K**ey **C**ryptography
PKI	**P**ublic **K**ey **I**nfrastructure
ECC	**E**lliptic **C**urve **C**ryptography
DHKE	**D**iffie-**H**ellman **K**ey **E**xchange
DoS	**D**enial **o**f **S**ervice
DBMS	**D**ata**B**ase **M**anagement **S**ystem
DSMS	**D**ata**S**tream **M**anagement **S**ystem
RDBMS	**R**elational **D**ata**B**ase **M**anagement **S**ystem

To Emanuele

Chapter 1

Introduction

1.1 Internet of Things

Internet of Things (IoT) is an emerging paradigm that includes a large number of technologies and research disciplines, aimed at enabling the Internet to reach out into the real world of physical objects. The result is an interconnected smart world, where humans and devices interact with each other, establishing a smart environment in which the exchange of data and services is continuous.

During the last decade, IoT approached our lives silently and gradually, thanks to the availability of wireless communication systems (e.g., RFID, WiFi, 4G, IEEE 802.15.x), which have been increasingly employed as technology drivers for crucial smart monitoring and control applications [11] [149].

An encouraging factor for the spread of the IoT paradigm is that many industry-leading manufacturers, service providers, and software and systems developers are making investments in the IoT future world vision. In fact, *Forbes* market estimatesd that 2015 has been the year of changing direction of IoT from a purely theoretical design to more anchored development [63]. Figure 1.1 shows the 2015 hype cycle for emerging technologies identified by *Gartner* society [65].

Cisco predicts that the global IoT market will be \$14.4 trillion by 2022, with the majority invested in improving customer experience, as shown in Figure 1.2. Additional areas of investment include: reducing the time-to-market (\$3T), improving supply chain and logistics (\$2.7T), cost reduction strategies (\$2.5T) and increasing employee productivity (\$2.5T). *Cisco* also found that 50% of IoT activity today is in manufacturing, transformation, smart cities and consumer markets [41].

Moreover, software and services are expected to be a \$600B market by 2019, attaining a 44% CAGR (*Compounded Average Growth Rate*) from 2015 to 2019. *BI Intelligence* also predicts the number of devices connected via IoT technologies will grow at a 35% CAGR from 2014 to 2019, as shown in Figure 1.3.

Analysts at *McKinsey* institutes [128] expect that the installed base for IoT devices will grow from around 10 billion connected devices today to as many as 30 billion devices by 2020, which corresponds to an uptick of about 3 billion new devices per year, as Figure 1.4 shows.

Nowadays, as can be seen from the analysis presented above, the concept of IoT is multi-dimensional, since it embraces many different technologies, services, and standards. Furthermore, it is widely perceived as the cornerstone of the ICT market in the next ten years, as also advocated in [26]

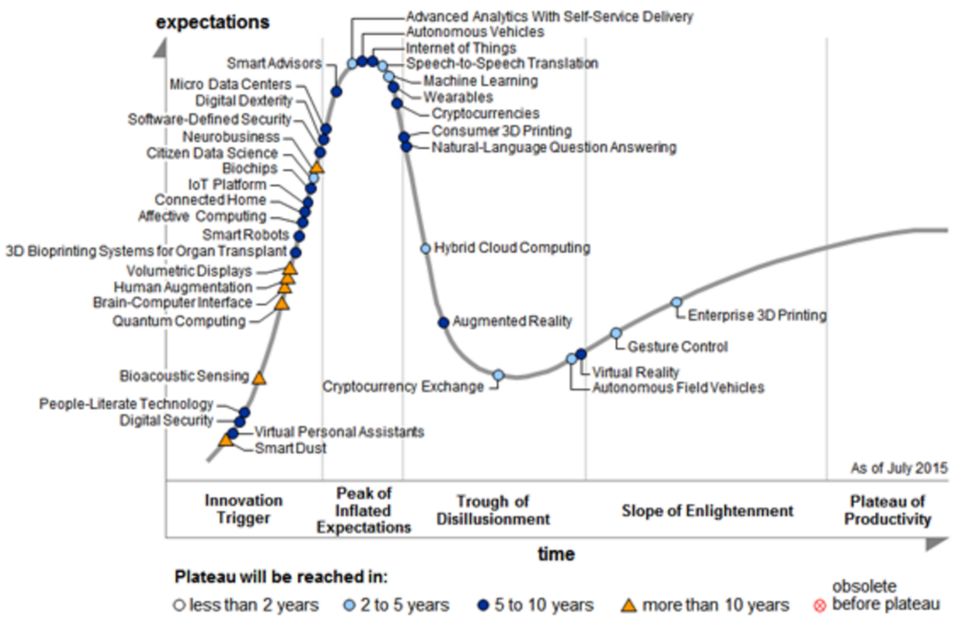


FIGURE 1.1: Gartner 2015 hype cycle for IoT emerging technologies

[53] [81]. In fact, IoT deployments may adopt different processing and communication architectures, technologies, and design methodologies, based on the target context. Examples of IoT scenarios include health equipment for patient monitoring, connected cars in vehicular networks, surveillance devices, wearable sensors, smart home systems, and so on. In such contexts, it is fundamental to define how the involved *things* could efficiently communicate and exchange information among themselves and with remote servers.

The term *"Internet of Things"* is becoming widely used for broadly defining a future in which objects equipped with sensing and actuation capabilities get connected to a global networked infrastructure, able to bridge the gap between the physical and digital realms [130]. An IoT system can be depicted as a collection of smart devices which interact on a collaborative basis to fulfill a common goal, acquiring data from and acting upon the environment in which they are.

From a technological point of view, the term *things* refers to various physical everyday objects that embed the enabling IoT technologies (e.g., wireless sensor nodes, actuators, RFIDs, and so on) to make them *smart* and suitable to support the provisioning of innovative and customized services to individuals and businesses in different application domains.

The IoT revolution is properly turning everyday objects into *smart* ones. All these devices are able to acquire data from the environment in which they are placed and/or to provide different types of information to other devices belonging to the network. Such a behavior generates a distributed network, in which heterogeneous sources of data can cooperate by querying the different information, retrieved both from the environment and from the users, in order to satisfy the user needs. Note that some recent interpretations of IoT extend the term *things* to the individuals that, through the adoption of appropriate devices, can broadcast the status of the context

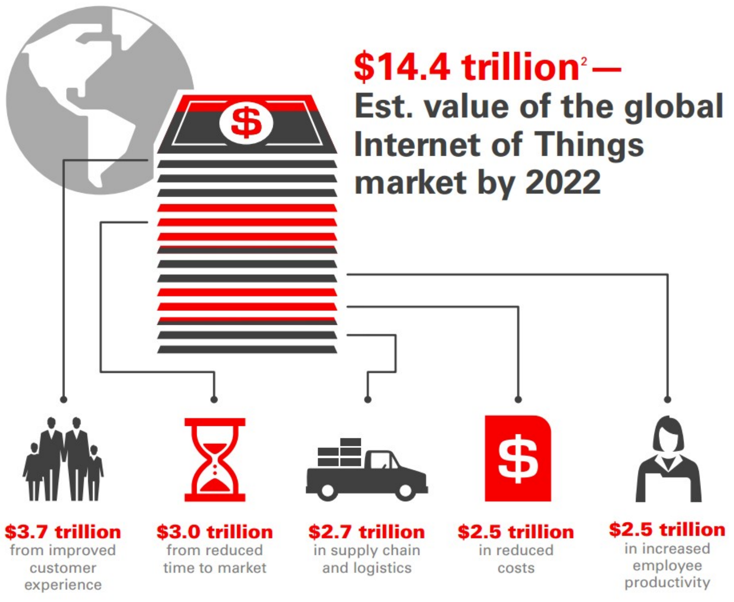


FIGURE 1.2: Cisco IoT market predictions

in which they are at a specific time. Therefore, users become sensors able to provide useful and usable data.

1.2 Open challenges

From the presented scenario, several issues naturally emerge. In particular, IoT deployments are characterized by large heterogeneity in terms of adopted technologies and communication protocols; some of them may include a huge number of devices, introducing scalability and interoperability issues. Therefore, one of the most crucial challenge in building an IoT system lies in the lack of a common and standardised software framework, able to manage all the involved entities in an efficient manner.

Of course, this high level of heterogeneity, along with the wide scale of IoT systems, is expected to magnify the security threats of the current Internet, which is being increasingly used to let humans, machines, and robots interact, in any combination. In more detail, traditional security countermeasures cannot be directly applied to IoT technologies due to their limited computing power; moreover, the high number of interconneted devices raises scalability issues. At the same time, to reach full acceptance by users, it is mandatory to define valid security, privacy and trust models suitable for the IoT application context [207] [59] [170] [10].

As far as security and privacy are concerned, data anonymity, confidentiality and integrity need to be guaranteed, as well as authentication and authorization mechanisms, in order to prevent unauthorized users (i.e., humans and devices) from accessing the system. In particular, both data protection and the confidentiality of users' personal information must be ensured, since devices may manage personal and/or sensitive information

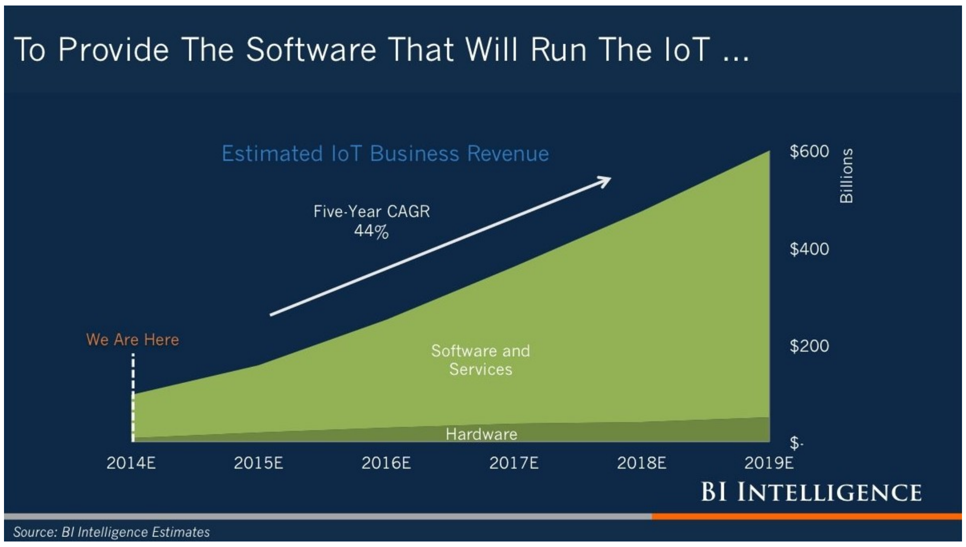


FIGURE 1.3: Software and services vs hardware IoT growth

(e.g., user habits). Finally, trust is a fundamental issue since the IoT environment is characterized by different devices that have to process and handle the data in compliance with user needs and rights.

Note that adaptation and self-healing play a key role in IoT infrastructures, which must be able to face normal and unexpected changes of the target environment. Accordingly, security issues should be treated with a high degree of flexibility, as advocated in [16] [38]. Together with the 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, as underlined in [12].

An IoT-based environment may suffer of different kinds of attack, which can be distinguished into three groups: (i) attacks against the IoT devices (including tampering or physical attacks); (ii) attacks against the communications (including monitoring and altering messages or routing attacks); (iii) attacks against the masters of the devices (i.e., the platform that manages the IoT data provided by the IoT devices). Proper countermeasures must be put in place to address each of them.

Along with security, a second fundamental aspect that needs to be taken into account, for ensuring the effectiveness of IoT services, is data quality. IoT services should provide correct, complete and updated information; in fact, in some scenarios, errors or missing values might have a critical impact on actions or decisions.

As IoT-enabled services and applications may make use of different data sources, the user (or the application itself) has to be aware of the security and quality level of the data being accessed, in order to take informed decisions about their usage.

Figure 1.5 summarizes the open challenges in the IoT field.

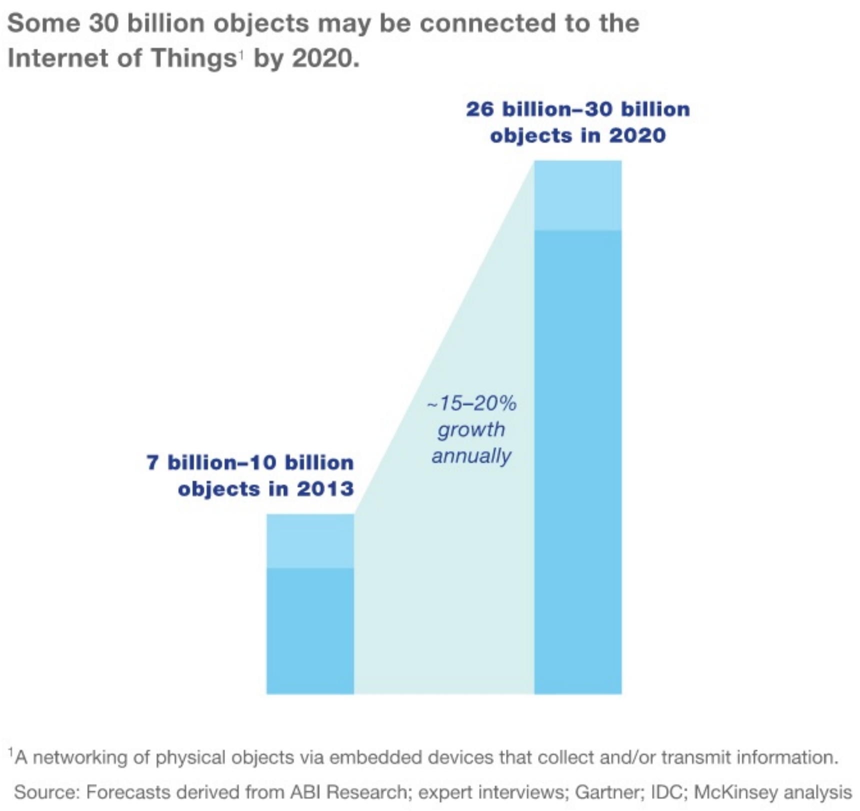


FIGURE 1.4: McKinsey IoT devices growth prediction

1.3 Proposed solution

In order to address the aforementioned issues, a system able to deal with heterogeneous data sources and to evaluate security and quality of information being collected, processed and transmitted should be defined.

Therefore, the main goal of this thesis is the design and development of a flexible and distributed IoT architecture. It should be based on the idea of bringing processing, security and data qualification closer to the actual data sources, in order to face scalability issues as well as to perform an assessment of the managed information.

As a result, a middleware, named *NOS (NetwOrked Smart Object)*, is presented. The main innovative contribution concerns the implementation of NOS highly modular, cross-domain and lightweight system architecture conceived for IoT applications, with the following functionalities:

- Provision of proper interfaces towards heterogeneous data sources, in order to cope with interoperability issues
- Methods for the distributed and autonomic management and runtime optimization of NOS middleware platform itself
- Key management systems for the key distribution and replacement towards the entities involved in secure interactions with NOS platform

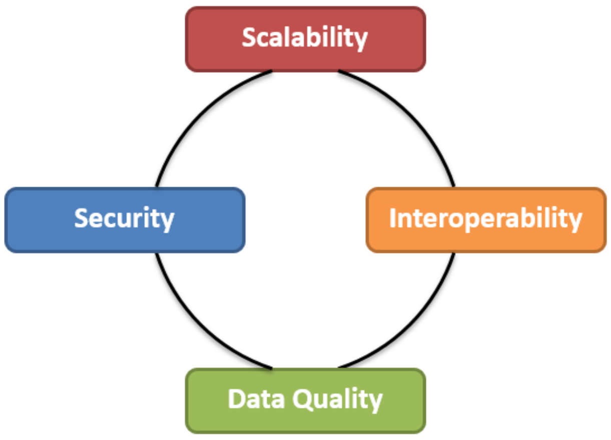


FIGURE 1.5: Open challenges in IoT field

- Mechanisms for the automatic assessment of data quality and security by means of well-defined algorithms, along with the provision of standardized interfaces and data models for applications/services to access qualified IoT information, where raw data are enriched with metadata specifying their security and quality levels
- Capability for users and applications to dynamically specify, for the required services, the level of security and data quality suitable for their own needs
- Integration with a policy enforcement framework, able to manage the rules defined by NOS middleware and/or by the specific application domains, in order to regulate the access to resources, and to handle violation attemps
- A lightweight and secure publish/subscribe messaging protocol, extended by means of key management and policy enforcement framework, in order to guarantee the authentication and identification of entities and the authorization controls over the available resources
- A synchronization system for guaranteeing the correct application of the policies across different realms in real-time

NOS middleware, along with the functionalities listed above, is implemented in a real prototype, whose code is openly accessible as open source under a permissive license at `https://bitbucket.org/alessandrarizzardi/nos.git`.

Note that, in real application scenarios, multiple NOSs are deployed in a distributed manner, managing a different number of sources and users, depending on their proximity.

The thesis is organized as follows:

- Chapter 2 provides a detailed overview of the state of the art about security in IoT
- Chapter 3 presents the high-level definition of NOS architecture, starting from a UML general conceptual model, the modules and the interfaces which compose NOS prototype

- Chapter 4 discusses NOS functionalities in detail, including the key management systems, the algorithms for the data assessment, the policy enforcement framework, the authenticated publish/subscribe mechanism, and the synchronization system
- Chapter 5 shows the performance evaluation of NOS middleware, in order to validate the proposed solution in a real IoT application context
- Chapter 6 ends the thesis and provides hints for future work.

Chapter 2

State of the Art

2.1 Internet of Things Security Open Issues

In order to better clarify the innovative contribution of this thesis, a deep analysis of the state of the art regarding IoT and, in particular, the available solutions existing in literature about security (i.e., integrity, confidentiality, authentication), privacy, and trust is conducted. Also proposals regarding security middleware and secure solutions for mobile devices, as well as ongoing international projects on this subject are studied. The main topics are shown in Figure 2.1. The results of this investigation were published in a survey paper [187] and are presented in the following, for motivating the choices and the solutions proposed in Chapters 3 and 4.

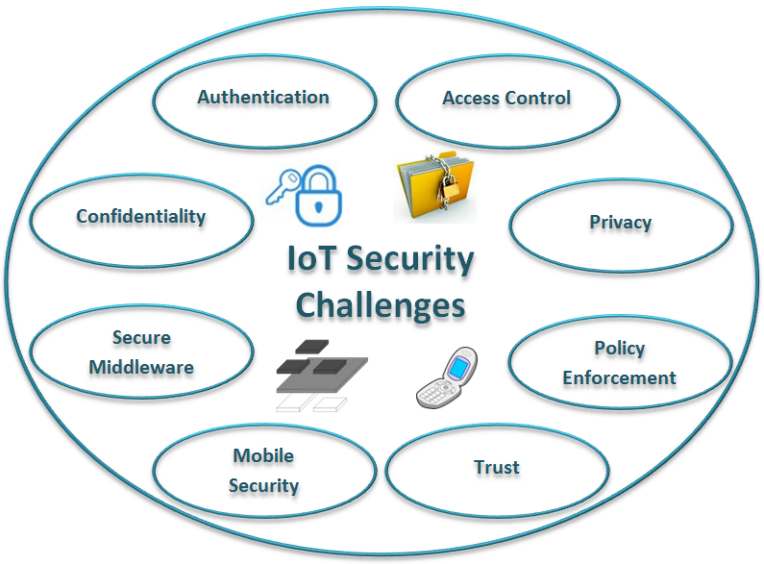


FIGURE 2.1: Main security issues in IoT field

In literature, other surveys deal with issues related to the IoT paradigm: [11] analyzes the IoT enabling technologies and existing middleware, also from an application point of view, and presents open issues in security and privacy together with those from standardization, addressing, and networking; [207] considers the security and privacy challenges from a specifically legislative point of view, with particular attention to the European Commission directives; [130] discusses the main research contexts (i.e., impact areas, projects, and standardization activities) and challenges in IoT, dealing also with data confidentiality, privacy, and trust as regards the security requirements; [47] is on Internet of Underwater Things and presents

TABLE 2.1: Contribution of available surveys on IoT security

	[11]	*[207]*	*[130]*	*[47]*	*[170]*	*[75]*	*[216]*	*[187]*
Security	yes	no	yes	yes	yes	yes	no	yes
Privacy	yes	yes	yes	no	yes	yes	no	yes
Trust	no	yes	yes	no	yes	no	yes	yes
Middleware	yes	no	no	no	no	no	no	yes
Mobile	no	no	no	no	no	no	no	yes
Projects	no	no	yes	no	no	no	no	yes

only few hints about security issues; [170] investigates the advantages and disadvantages of centralized and distributed architectures in terms of security and privacy in IoT with an analysis of the main attack models and threats; [75] provides a general overview on various IoT aspects, such as the technologies involved, the applications, the cloud platforms, the architectures, the energy consumption and security issues, the quality of service and data mining implications; [216] focuses only on the specific issue of trust management in IoT.

A comparison of the aforementioned surveys with [187] is given in Table 2.1. [187] clearly addresses the full breadth of security issues in this field.

The following paragraphs are organized as follows: Paragraph 2.1.1 analyzes the available approaches regarding confidentiality and access control in IoT; Paragraphs 2.1.2 and 2.1.3 deal with privacy and trust issues, respectively; Paragraph 2.1.4 shows the security and privacy policies enforcement in IoT applications; security frameworks and middlewares are discussed in Paragraph 2.1.5; Paragraph 2.1.6 addresses security in mobile IoT devices; finally, Paragraph 2.1.7 refers to the ongoing international projects on IoT security.

2.1.1 Authentication, confidentiality and access control requirements in IoT

This paragraph analyzes in depth three main security requirements: authentication, confidentiality, and access control, with a special focus on IoT systems. IoT, in fact, enables a constant transfer and sharing of data among things and users in order to achieve particular goals. In such a sharing environment, authentication, authorization, access control and non-repudiation are important to ensure secure communications. New approaches should be introduced or existing techniques should be tailored to the new IoT environment. In the following, the seminal contributions in such a field are illustrated together with a critical review of open issues that deserve further investigation.

Authentication and confidentiality

As regards authentication, some preliminary works adopt lightweight encryption mechanisms. For example, the approach presented in [223] makes use of a custom encapsulation mechanism, combining cross-platform communications with encryption, signature, and authentication, in order to improve IoT application development capabilities by establishing a secure communication system among different things. Likewise, [109] adopts a lightweight encryption method based on XOR manipulation for anti-counterfeiting and privacy protection, in order to cope with constrained IoT devices. Also exploiting XOR and hash computations, particularly suitable for resource-constrained architectures, [197] proposes an user authentication and key agreement scheme for Wireless Sensor Network (WSN). It enables a remote user to securely negotiate a session key with a sensor node, using a lean key agreement protocol. In this way, it ensures mutual authentication among users, sensor nodes, and gateway nodes, although the latters are never contacted by the user. Instead, the authentication and access control method presented in [218] aims at establishing the session key on the basis of *Elliptic Curve Cryptography (ECC)*. This scheme defines attribute-based access control policies, managed by an attribute authority, enhancing mutual authentication among the users and the sensor nodes, as well as solving the resource-constrained issue at the application level. Such works only represent starting points towards the development of more complex and robust schemes. More structured solutions are presented in the remainder of this paragraph.

In [102] the first fully implemented two-way authentication security scheme for IoT is introduced; it is based on existing Internet standards, specifically the *Datagram Transport Layer Security (DTLS)* protocol, which is placed between transport and application layer. This scheme is based on *RSA* and it is designed for *IPv6* over *Low power Wireless Personal Area Networks (6LoWPANs)* [149]. Although, the extensive evaluation, based on real IoT systems, shows that such an architecture provides a certain level of message integrity, confidentiality, and authenticity with enough affordable energy, end-to-end latency, and memory overhead.

As regards confidentiality and integrity, [171] analyzes how existing key management systems could be applied to the IoT context. *Key Management System (KMS)* protocols can be classified into four major categories: key pool framework, mathematical framework, negotiation framework, and public key framework. In [171] the authors argue that most of the *KMS* protocols are not suitable for IoT. In fact:

- Key pool approaches suffer insufficient connectivity
- Mathematical approaches make use of the deployment knowledge to optimize the construction of their data structures, but such an approach cannot be used in IoT since client and server nodes are usually located in different physical locations
- Combinatorics-based *KMS* protocols suffer both connectivity and scalability/authentication issues
- Negotiation approaches make use of the wireless channel and its inherent features to negotiate a common key; however, they cannot be

suitable for IoT because client and server nodes usually belong to different networks and they should route the information through the Internet in order to be able to talk with each other.

Hence, the *KMS* protocols that might be suitable for some IoT scenarios are the *Blom* [49] and the polynomial schema [116], whose computational overhead is quite low in comparison to *Public Key Cryptography (PKC)* operations (i.e., public key framework). However, for such schemes, several countermeasures are required in order to manage device authentication and face man-in-the-middle attacks. For example, [143] and [164] present a framework for IoT based on *Public Key Infrastructure (PKI)*, but much work needs to be done yet to obtain a secure and efficient solution.

In general, what emerges is that a robust *KMS* should further secure the communications among data sources and IoT middleware.

Several approaches have been designed for WSN scenarios. In particular, many solutions have been proposed aiming to overcome traditional predistribution approaches, such as [175] [54] [36] [213] [189]. Among them, the algorithms proposed by Dini et al. [44] and Di Pietro et al. [162] gained popularity due to their efficiency in terms of resource consumption and resilience towards malicious attacks, with respect to the aforementioned approaches (i.e., key pool, mathematical, combinatorics, negotiation). For such reasons, they will be detailed in Paragraph 4.2, since they are adopted in the solution proposed in this thesis for securing the communications among IoT entities.

As regards IoT context, a limited contribution in the key management field actually exists.

[141] provides a classification of existing protocols relying on key distribution mechanisms to establish a secure communication channel among the nodes belonging to the network, underlining security requirements and open challenges. [141] concludes that symmetric approaches are still not the default choice for IoT. Instead, public key cryptography is likely to be increasingly recommended, provided that the associated asymmetric techniques are properly optimized. In [141] opinion, a trusted third party will take a more active role to secure IoT and to adapt to its heterogeneous nature. Additionally, security protocols should take into account the resource-constrained feature of the devices involved in the IoT scenarios.

[3] proposes a lightweight key management protocol for e-health applications. It is based on collaboration to establish a secure end-to-end communication channel among highly resource-constrained nodes and remote entities. To achieve this goal without affecting in a relevant way the network consumption, third parties are in charge of executing the cryptographic primitives. However, the evaluation showed that high overhead and the use of third parties and of a certificate authority heavily impacts scalability and efficiency.

[201] and [82] adopt a group key management distribution scheme for WSN in a IoT scenario in which the sensor nodes are organized in a hierarchical structure. Note that, it is preferable to not assume a hierarchy among the sources since, in typical IoT scenarios, each device is independent from the others.

A more practical approach [210] proposes a transmission model based on signature-encryption schemes, which addresses several IoT security requirements (i.e., anonymity, trustworthy and attack-resistance) by means of *Object Naming Service (ONS)* queries. *Root ONS* is able to authenticate the identities and platform creditability of *Local ONS* servers (*L-ONS*) by a *Trusted Authentication Server (TAS)*. The *TAS* gives a temporary certificate to validated *L-ONS*, which can apply for inquiry services many times, using the certificate in the validated time. A security *ONS* query with anonymous authentication provides credentials only to authorized and trusted *L-ONS*, preventing the illegal *ONS* to enquire information from things. In the transmission process, a *Remote Information Server of Things (R-TIS)* wraps the information of things into multiple encryption layers with the routing node public key. The encrypted data are decrypted at each routing node, until the *Local Information Server of Things (L-TIS)* receives the plain text. Meanwhile, the nodes can check the integrity of the received data and the creditability of routing path in the transmitting procedure. Such a transmission model is very weak in terms of attack-resistance, mainly due to the adoption of a hop-by-hop encryption/decryption approach.

From the work discussed above, it appears that a unique and well-defined solution able to guarantee confidentiality in a IoT context is still missing, as also asserted in [163]. It is worth noting that many efforts have been conducted in the WSN field [6] [35] [74] [111] [219] [222] , but several questions arise, such as:

- Are the WSN proposals adaptable to the IoT environment, considering both the heterogeneity of the involved devices and the different application contexts? Some of the solutions presented are specifically targeted to the problem of lightweight ciphering in pervasive environments, but are these proposals enough secure and efficient?
- How and at which network layer to handle authentication?
- Is it feasible to reuse the traditional security mechanisms (e.g., encryption algorithms) or it is better to start from new solutions?
- How to handle the different keys?
- Which kind of key distribution mechanism is the most suitable?
- How to ensure an end-to-end integrity verification mechanism in order to make the system more resilient to malicious attacks?

Therefore, further efforts are required to complement the existing lean mechanisms with standardized protocols for authentication and a clear definition of one or more authorities, aimed at guaranteeing the expected confidentiality within the IoT infrastructure.

Access control - stream data

Access control refers to the permissions in the usage of resources, assigned to the different actors of a wide IoT network. The two subjects identified in [7] are the data holders and the data collectors. Users and things, as data holders, must be able to feed data collectors only with the data regarding a specific target. At the same time, data collectors must be able to identify or authenticate users and things as legitimate data holders, from which the information are collected.

In IoT, is usually necessary to deal with processing of streaming data and not, as in traditional database systems, with discrete data. The main critical issues in this context refer to performance and temporal constraints, since access control for a data stream is more computationally intensive than in traditional *DBMS (DataBase Management System)*. In fact, queries have to be directly executed on incoming streams, which can be made of large volumes of data that might arrive at unpredictable rates. Several works deal with these aspects in database management field.

For example, [8] develops an architecture that aims at ensuring data integrity and confidentiality, starting from a prototype query processing engine for data streams, called Nile [80]. Such a mechanism is based on *FT-RC4*, an extension of the *RC4* algorithm, which represents a stream cipher encryption scheme, to overcome possible decryption fails due to desynchronization problems. [80] is focussed on shared processing of window joins over data streams, in order to enhance the performance and the scalability of the *DBMS*.

An approach that addresses the authentication problem of outsourced data streams can be found in [151] and in [152] with *CADS (Continuous Authentication on Data Streams)*. In this scenario, the presence of a service provider that collects data from one or more data owners is assumed, together with authentication information, and, at the same time, that processes queries originating from many clients. The service provider returns the query results to the clients, as well as verification information, which allow them to verify the authenticity and the completeness of the received results, on the basis of the authentication information provided by the data owner.

[153] also focuses on data outsourcing. In particular, due to the large amout of streaming data, companies may not acquire the resources for deploying a *Data Stream Management System (DSMS)*. Therefore they could outsource the stream storage and delegate its processing to a specialized third-party with a stronger *DSMS* infrastructure. Naturally, this gives rise to a trust issue: the third-party may act maliciously to increase profit. The solution is to adopt a method for stream authentication, in order to enable clients to verify the integrity and the freshness of the streaming results received from the server. Such a solution should be very lightweight for all parties involved (e.g., WSN applications). To this end, [153] represents streams as linear algebraic queries, allowing to authenticate dynamic vector sums and dot products, as well as dynamic matrix products, by means of hash operations, modular additions/multiplications and cryptographic security functions. Such techniques may be very suitable for IoT entities, which are characterized by resource constraints in terms of energy consumption, computation and storage.

[114] proposes a semi-distributed approach, in which a framework and an access control model to secure the *DSMS* are proposed. They extend the *Borealis* data stream engine [2] with some security requirements. The framework exploits an owner-extended version of *RBAC (Role Based Access Control)* [177], called *OxRBAC*. Users have to prove their identity through a login process, consequently a session is created and a role is established

for the user to perform authorized tasks. As a result, the authorization is checked by analyzing the user-session couple. The system itself provides each user with the access permissions to objects; therefore users can see only the catalogue of the objects they are allowed to view. Since there can be many output streams, the system filters the tuples in order to give to the users only permitted results. Such an approach does not consider the adoption of any encryption algorithms for data streams. Note that this framework uses a single node system and not a totally distributed data stream engine. Clearly, a distributed approach would create new issues: the output streams might be on different nodes and the use of identifiers to uniquely recognize and filter the tuples to be managed might create conflicts.

Two works, [138] and [139], exploit metadata in order to guarantee the security of the tuples in the stream. In [138] a stream-centric approach is proposed, in which the security constraints are directly embedded into data streams and not stored on the *DSMS* server. In more detail, security metadata tuples are interleaved with the data tuples in the streams, in order to reduce the overhead. In [138], no new access control model is defined, but there is an enforcement mechanism suitable for streaming data, exploiting query processing. Note that, either *RBAC*, *DAC (Discretionary Access Control)* or *MAC (Mandatory Access Control)* can be cast in such a solution. In [138] policies on data streams are stated by the user owning the device producing the data streams itself. This makes a user able to specify how the *DSMS* has to access his/her personal information (e.g., location, health conditions).

In [139], an extended approach is proposed, which enriches data streams with metadata called streaming tags. In this way, users are able to use a free vocabulary to add information to reported events. It supports a variety of tagging granularities; therefore, users could tag streams, tuples, attributes or specific data values. A framework based on the *CAPE* engine [225] is implemented and tested, after the definition of a proper and novel tag query language, but this solution may present some overhead and memory issues, as reported by simulation results.

The work in [33] presents an extension of the solution provided in [34] as regards access control of streaming data, based on the *Aurora* data model [1]. This framework supports two types of privileges, named "read" and "aggregate", and also two temporal constraints, named "general" and "window". The subjects (i.e., the users) are specified according to a role-based approach, therefore permissions are associated with roles and not directly with subjects, as in *RDBMS (Relational DataBase Management System)*. Another idea, taken from *RDBMS*, is the definition of an independent representation language for the managed object, similar to the concept of "view", in order to model the high granularity levels required by IoT applications. Queries are registered in the stream engine and continuously executed on the incoming tuples. Whenever a user submits a query, a specific component, called *Query Rewriter*, checks the authorization catalogues, where permissions are specified, to verify whether the query can be partially or totally executed or should be denied. In case of partially authorized queries, it is rewritten in such a way that it only contains authorized data. In order to support the query rewriting task, a set of secure operators is defined, which filter out from the results of the corresponding non secure operators those tuples/attributes that are not accessible, according to the specified access control policies.

In [32] the authors extend the two previous works, just presented ([33] and [34]), in order to make the solution independent from the stream engine. Note that, in general, each *DSMS* adopts its own language; to overcome such an issue and to allow the interaction among different *DSMS*, in [32] a common query model is defined and then the most used operations are translated by the *Deployment Module* into the specific engine query language. The results have been compared with other proposals. For example, with respect to [114], which has the drawback of wasting computation time when unauthorized queries are performed, it represents a better solution. [138], as [32], focuses on access control requirement for data streams; however, in [138] access control is considered from a different point of view: the privacy protection. This is due to the fact that in [138] the privacy policies on data streams are stated by the users themselves; while, in [32], policies are specified by the system administrator. Moreover, in [138] access control policies are not stored in the *DSMS*, but they are encoded via security constraints and directly embedded into the data streams; this also represents an important difference with respect to [32]. In [139] a set of operators is defined, able to enforce security constraints, but it implements them only in the *CAPE* engine [225]; in contrast, [32] proposes a framework able to work across a wide range of *DSMS*s.

While the previous works propose extended versions or acquire some features of *RBAC*, in [77] the authors affirm that authorization frameworks like *RBAC* and *ABAC (Attribute Based Access Control)* do not provide sufficiently scalable, manageable, and effective mechanisms to support distributed systems with many interacting services and the dynamic and scaling needs of IoT context. According to the authors of [77], a problem, common to *ACLs (Access Control Lists)*, *RBAC* and *ABAC* is that, in these systems, the principle of least privilege access is hard to enforce. In order to overcome such a limit, *Capability Based Access Control (CapBAC)* has been developed, within the *European FP7 IoT@Work* project [57]. *CapBAC* is able to manage the access control processes to services and information with least-privilege operations. In *CapBAC*, the user has to present his/her authorization capability (and demonstrate he/she is the owner of it) to the service provider, while in a traditional *ACL* system the service provider has to check if the user is, directly or indirectly (e.g., via a role owned by the user), authorized to perform the requested operation on the resource. The authorizations are given by the owner of a certain resource/service to the desired users, which, as a consequence, can prove their capability to access to the resource or benefit of the service. However, it is not clear how to make possible to revoke capabilities and to set validity conditions under which the authorization is available or not; such actions are, instead, supported by *RBAC* and *ABAC*. Note that *ABAC* is adopted in this thesis, showing its capability to manage the dynamic needs of IoT domain. Another crucial point is how to establish the degree of independence for users and data owners in taking autonomous decisions on the rules and negotiation activities; then, a middleware entity should be introduced to better regulate such interactions, as this thesis does.

Access control - specific contexts

Besides solutions based on *DSMS*, other work tries to address the access control issue in various contexts.

For example, in [121] the attention is focused on the layer responsible for data acquisition, which is the direct responsible for the collection of information. In such a layer, a large number of nodes is required to sense a wide range of different data types for authorized users, possibly in accordance with privacy and security levels. Therefore, [121] presents a hierarchical access control scheme for this layer, which considers the limited computational and storage capacity of the nodes. In fact, only a single key is given to each user and node; the other necessary keys are derived by using a deterministic key derivation algorithm, thus increasing the security (since the key exchange is limited) and reducing the node storage costs. However, the presented approach seems not to be sufficiently robust for large-scale IoT environments.

Instead, focusing on emergency situations, note that the location of a user should be made available, while under normal circumstances, the user location information is confidential. In this scenario, [83] presents an identity framework, consisting of the following primitives: registration, user authentication, policy, and client subsystems. The system evaluates the identity of the user through the user authentication subsystem and gets the level of the emergency through the policy subsystem. Then it can make sure that user location information can be accessed only by authorized users and only when it is needed.

Multicast communication is secured in [201] by adopting a common secret key, denoted the group key, shared by multiple communication endpoints. Such keys are managed and distributed with a centralized batch-based approach. Note that such a mechanism reduces the computational overhead and network traffic due to group membership changes, caused by user joins and leaves, as happens in a typical IoT context. Such a protocol can be applied to two several relevant scenarios: (i) secure data aggregation in IoT and (ii) *Vehicle-to-Vehicle (V2V)* communications in *Vehicular Ad hoc Networks (VANETs)*. However, it is not suitable, at this stage, to wide IoT environments, since the actual solution requires the presence of a centralized server provider, thus compromising the scalability, for example, in presence of multiple application scenarios with heterogeneous data and different policies to be managed.

Other works again couple different kinds of authentication mechanims with lightweight protocols for information dissemination. In fact, concerning access control in publish/subscribe environments, several solutions address such an issue by means of: hierarchical role-based rules based on subscriber privileges [23]; attribute-level policies concerning multi-domain environments and involving the use of shared keys for the interaction with untrusted brokers [161]; policy integration and distribution along with performance implications [211].

[192] and [191] investigate secure information sharing in the healthcare context; since medical data are sensitive, they must be protected and released only to authorized parties. The solution proposed by the authors involves a publish/subscribe middleware, which is in charge of handling the access control policies on the data transmitted by the healthcare structures

by customizing policies depending on the actual events. The defined policy rules are local to each administrative domain and proper mechanisms are provided to assist administrators to maintain a consistent policy set. However, the data disclosure and policy management are centralized, thus compromising the scalability of the whole system. Moreover, automatic updates and revocation of policies are not considered.

Another work targeted to the medical sector is [108]. It presents a tool developed for the expression of security policy controls aimed at governing electronic healthcare records. In particular, the authors introduce the theory behind the use of knowledge management for automatic and consistent security policy assertion, using a new formalism, called *Secutype*. Integration with an existing medical record system (*EHR Standards*) is proposed, in order to efficiently provide protection for discrete and sensitive medical data. Many open issues remain, such as: (i) how to formally define rules tailored to a clinical purpose; (ii) how to model the security concepts; (iii) how to enable interoperability among different collaborators.

The work in [97] describes a context-aware access control architecture for the provision of e-services based on an end-to-end web services infrastructure. The proposed architecture is able to control access permissions to distributed Web services through an intermediary server in a completely transparent way both to clients and resources. The access control mechanism is based on *RBAC* model, in order to enable the enforcement of complex rules and the inclusion of context information in the authorization decisions. Contexts are classified on the basis of the requirements imposed by the provision of e-services to the industrial domain. The architecture is entirely based on open Web standards: *HTTP*, *XML*, *SOAP* and *WSDL*. What is missing is the effective portability of such a mechanism in a more general IoT context, taking into account the data sources management and user interactions with the system.

A scheme for the asynchronous discovery of topics in distributed publish/subscribe settings, based on *Java Message Service*, *WS-Eventing* and *WS-Notification* infrastructures, is presented in [150]. Every interaction within the system is secured and requires the presence of credentials before any actions can take place. A topic advertisement is secured by encrypting the advertisement with a symmetric key and by securing this advertisement key with the creator's public key. In this work, no platform is defined, able to assess the behaviour of the presented mechanism in a wide IoT scenario, with a proper threat model.

[64] aims to demonstrate that a standardized federated, dynamic, user-directed authentication and authorization model can be adapted from the web to be used in IoT, while preserving privacy for information and devices. The authors explore the use of *OAuth*, built on top of *HTTPS*, for IoT systems that, instead, use the lightweight *MQTT* protocol. In particular, they use *OAuth* 2.0 to enable access control to information shared via *MQTT*. Nevertheless, this work does not clarify how to allow the re-use and the integration of the proposed mechanism with IoT devices and the standard protocols.

In [24], a category-based metamodel for access control in distributed (federated) environments is presented. A framework for the specification and the enforcement of global access control policies that take into account the local policies specified by each member of the federation is described.

Such a framework provides mechanisms for specifying heterogeneous local access control policies, for defining policy composition operators, and for using them to define conflict-free access authorisation decisions. In this framework, distributed access control policies can be easily specified and manipulated by means of local policy specification mechanisms and definitions of policy composition operators. Anyway, no real application case-studies or implemented tools have been proposed by the authors, thus limiting the contribution to a theoretical approach.

In [202] an architecture for open networks is proposed, aiming to allow "things", with limited or no user interfaces, to provide a high level of data security by delegating trust to a trusted third party (i.e., a provisioning server). Such a third party helps the device to determine which users, devices or services are authorized to perform a given operation on the device itself in a secure manner. Such a solution uses an existing, open and standardized transport protocol for communication, named *Extensible Messaging and Presence Protocol (XMPP)*, for guaranteeing interoperability and scalability. *XMPP* supports the most commonly communication patterns necessary for IoT, such as request/response, asynchronous messaging and publish/subscribe. It is based on message brokers to solve the security issues concerning user identities, enforcing secure user authentication, and message authorization. The architecture requires zero-configuration for operators and manufacturers, without compromising security or ease-of-use for end-users. It is also scalable and can be used both in local environments such as cars, homes, offices, buildings, industry plants, with local provisioning servers and local message brokers, as well as in global environments, with global provisioning servers connected to global message brokers. An implementation of the provisioning server is also provided. Note that the authors have chosen to use *XMPP* protocol rather than *MQTT* or *CoaP*, as explained in [99]. However, the scalability claimed by the authors is not clear, since most of the operations are brought to external entities and devices seem unaware about what they are or are not allowed to do. Moreover, no performance analysis is provided to verify the overhead on devices of such a solution.

Two theoretical approaches are presented in [39] and [148], which address secure data sharing among different applications domains. In particular, [39] proposes a knowledge access control policy language model able to identify the knowledge access control and sharing rules across enterprises members. Such a language is based on an ontology, which aims to solve knowledge heterogeneity issues within companies and workers belonging to different areas. This is achieved by establishing relationships among the topics of interest of the parties involved, thus allowing a timely response to access authorization requests, also in case of changes in the business environment. Instead, in the IoT and big data fields, [148] proposes the policy-carrying data mechanism, with the aim of establishing access control rules for data consumers. Several drawbacks of such a work are the following: (i) the introduction of a complex language for policy definition; (ii) the need to a centralized entity able to evaluate the behavior of producers and consumers (which may represent a bottleneck); (iii) solely a theoretical presentation of the proposed solution.

Access control - hardware solutions

In order to provide a complete overview, it is important to present a sketch of the hardware solutions, conceived for access control in IoT applications.

The authors of [96] propose the use of Sensor *PUF*s to address the challenge of data provenance and integrity. Note that traditional *PUF* produces the response based on the challenge, while Sensor *PUF* produces the response based on the challenge as well as the sensed physical quantity. Sensor *PUF*s and *PUF*s can be used for identity management; along with hardware performance counters, they can be used for trust management. Lightweight encryption algorithms can be further used to provide confidentiality and privacy to the users.

[40] and [201] solutions are also tailored to IoT. In particular, [40] proposes an authorization scheme for constrained devices, combining *Physical Unclonable Functions (PUF)* with *Embedded Subscriber Identity Module (eSIM)*. The former provides cheap, secure, tamper-proof secret keys to authenticate constrained devices; while the latter provides mobile connectivity guaranteeing scalability, interoperability and compliance with security protocols. The main drawback of this mechanism is that IoT devices have to be physically equipped with an *eSIM* card, which, at the moment, is not yet a standard technology.

Also the authors of [5] utilizes *PUF* for guaranteeing a higher level of privacy and, at the same time, a constant identification time for *things*. *PUF* is exploited as a secure storage to keep secrets of the tag and defend against compromising attacks (i.e., an adversary compromising one tag can reveal the secrets of other tags). The target of this work is that of *RFID* systems, which play an important role in the IoT scenarios for solving the identification issues of things cost-effectively.

An overview of *PUF*-based hardware security solutions for IoT is given in [79], pointing out open security challenges in this field.

[180] develops a secure and efficient user search engine, named *SecDS* based on *EPC* Discovery Services (*EPCDS*) for *EPCglobal* networks. In more detail, the authors analyzed the requirements of access control for *EPCDS* and proposed an extended *ABAC* model to meet the requirements of enforcing the access control on resources; to achieve this goal, *ABAC* policies have been transformed to *FGAC (Fine-Grained Access Control)* policies. The final aim is to provide a bridge among different partners of supply chains to share information while enabling them to quickly find who is in possession of an item.

Finally, few hardware solutions are based on *Field Programmable Gate Array (FPGA)*. For example, [165] couples *FPGA* with the *Blowfish* algorithm with the scope of reducing the encryption time and achieve greater throughput for data transmissions among IoT devices; while [68] proposes an *FPGA*-based edge device, which uses *FPGA* technology to offload critical features of the communication stack to the dedicated hardware, aiming to increase the overall system performance.

Access control - summary

Starting from this discussion, the cited solutions can be grouped to a number of macro-categories, which are summarized in Table 2.2: (i) methods

TABLE 2.2: Summary of related works on access control

Kind of approach	Works
DBMS/DSMS	[8] [80] [151] [152] [153] [114] [138] [139] [33] [34] [32] [77]
Specific application contexts	[121] [83] [40] [201] [108]
Publish/subscribe	[23] [161] [211] [202] [97] [150] [192] [191]
Federation	[64] [24]
Theoretical solutions	[39] [148]

targeted to stream data in database management systems; (ii) access control frameworks tailored to specific application contexts; (iii-iv) works conceived for web service platforms and adopting publish/subscribe or federation mechanisms; (v) theoretical approaches.

The major challenges that emerge from this analysis, related to access control in an IoT scenario, are:

- How to guarantee the access permission in an environment where not only users, but also "things" could be authorized to interact with the system?
- It is more effective to adopt a centralized or distributed approach or a semi-distributed one (e.g., in which some tasks are performed in a distributed way and others by means of a centralized entity) in order to manage the scalable IoT architecture?
- How to handle the huge amount of transmitted data (i.e., in the form of stream data) in a common recognized representation?
- What are the access control features to be specifically tailored for distributed systems in order to bring proper decision regarding the development of an authentication system?
- How to support the identification of entities?

In fact, as regards identification, today one of the main changes is the increase in mobility of portable and powerful wireless devices. Identity requirement is not yet adequately met in networks, especially given the emergence of ubiquitous computing devices. Addressing identity issue requires to reformulate the architecture for naming, addressing and discovery, and the development of specific identity management framework for IoT [123]. Rather few solutions have been proposed, that relate to such an issue. Furthermore:

- To manage access control, how could an IoT system deal with the registration of users and "things" and the consequent issuance of credentials or certificates by authorities?
- Could the users/things present these credentials/certificates to the IoT system in order to be allowed to interact with the other authorized devices?

- Could a further step be the definition of specific roles and functions within the IoT context, in order to manage the authorization processes?

2.1.2 Privacy

IoT finds application in many different fields, for example: remote patient monitoring, energy consumption control, traffic control, smart parking systems, inventory management, production chains, customization of the shopping at the supermarket, civil protection. For all of them, users require the protection of their personal information related to their movements, habits and interactions with other people. In a single term, their privacy should be guaranteed. In literature, there are some attempts to address such an issue.

In [58] data tagging for managing privacy in IoT is proposed. Using techniques taken from information flow control, data representing network events can be tagged with several privacy properties; such tags allow the system to reason about the flows of data and preserve the privacy of individuals. Exploiting tagging within resource-constrained sensor nodes may not be a viable solution because tags may be too large with respect to the data size and sensitivity; therefore they generate an excessive overhead. Clearly, in this case it is not suitable for IoT.

In [85] a user-controlled privacy-preserving access control protocol is proposed, based on context-aware k-anonymity privacy policies. Note that privacy protection mechanisms are investigated: users can control which of their personal data are being collected and accessed, who is collecting and accessing them, and when this happens.

In [158] privacy protection of location-based services is addressed by proposing *UniLO*. It is an obfuscation operator, also able to provide service differentiation by means of three *UniLO*-based obfuscation algorithms, which offer multiple contemporaneous levels of privacy. In more detail, each user, before sending his/her location to a service provider, applies the obfuscation operator, in order to reduce the precision of the measurements and guarantee a certain privacy level. Such a precision is defined by the user according to his/her requirements in terms of privacy. Moreover, a user may require different privacy levels for different services.

The same authors of [158] present, in [45], a middleware architecture, named *LbSprint*, for location-based services, which integrates different privacy mechanisms by means of the standard *XACML* language. In fact, the first assumption of [45] is that a single privacy mechanism is not sufficient for meeting privacy requirements for heterogeneous services. To cope with such issues, *LbSprint* secures all communications in terms of confidentiality, authenticity and integrity, supporting various localization technologies (e.g., GPS, WiFi positioning, RFID). Every location receiver first authenticates to *LbSprint*, and then asks for and receives location notifications. Note that different receivers have different rights for accessing location data. The related rules are managed by a filter engine, on the basis of user-customizable policies, written in *XACML*, aimed at protecting user privacy. The *LbSprint* protocol is built over *SOAP* and is a centralized architecture.

In [30] *Continuously Anonymizing STreaming data via adaptive cLustEring (CASTLE)* is presented. It is a cluster-based scheme that ensures anonymity, freshness, and delay constraints on data streams, thus enhancing those privacy preserving techniques (e.g., k-anonymity) that are designed for static data sets and not for continuous, unbounded, and transient streams. In more detail, [30] models k-anonymity on data streams and defines k-anonymized clusters exploiting the quasi-identifier attributes of tuples in order to preserve the sensitive data privacy.

In [217], the traditional privacy mechanisms are divided into two categories: discretionary access and limited access. The former addresses the minimum privacy risks, in order to prevent the disclosure or the cloning of sensitive data; whereas the latter aims at limiting the security access to avoid malicious unauthorized attacks.

[205] analyzes the privacy risk that occurs when a static domain name is assigned to a specified IoT node. In this work, the authors propose a privacy protection enhanced *DNS (Domain Name System)* for smart devices, which can authenticate the original users' identity and reject illegal access to the smart device. The scheme is compatible with widely used *DNS* and *DNSSEC (Domain Name System Security Extensions)* protocols.

In [7] a fully decentralized anonymous authentication protocol for privacy-preserving target-driven IoT applications is presented. Such a proposal is based on a multi-show credential system, where different showings of the same credential cannot be linked together, therefore avoiding the discovering of the generating keys. The system defines two possible roles for participant nodes: (i) users, which represent the nodes originating the data and (ii) data collectors, which are responsible for gathering the data from authorized users. Users can anonymously and unlinkably authenticate themselves in front of data collectors proving the owning of a valid *Anonymous Access Credential (AAC)* encoding a particular set of attributes, established by the system itself. The protocol is divided into three phases: set-up; user registration, during which users obtain *Anonymous Access Credentials*; credential proving, during which users prove the possession of valid *AAC* to a data collector. Such a protocol guarantees: user anonymity, *AAC* unlinkability (no data collector or set of colluding data collectors can link two transactions to the same user), resistance to user impersonation, faulty and selfish nodes, nodes hindering the efficiency, and adversary controlling the data collectors. Moreover, such a system relies on a fully distributed approach, thus avoiding single point of failure issues. This work represents a valuable starting point for the development of a secure IoT distributed architecture; however, at this stage, no solution about to disseminate and protect the available resources is provided. Furthermore, the data should be distinguished as sensitive/identifiable and "normal", in order to guarantee both a certain level of information availability and privacy for users.

In this direction, [198], starting from the privacy preserving data mining techniques, aims at minimizing the sensitive data disclosure probability and the sensitive content analysis. In such a work, the user privacy awareness issue is addressed, proposing a privacy management scheme that enables the user to estimate the risk of sharing sensitive data. It also aims at developing a robust sensitivity detection system, able to quantify the privacy content of the information.

[22] focuses on e-commerce applications (e.g., *eBay*). In particular, it

claims that there exist two main paradigms to protect the customer privacy: one relies on the customer trustworthiness; the other one insists on the customer anonymity. The proposed paradigm hides the customer's real identity and only data that cover the actual resources he/she is looking for are allowed to circulate. Such data will be orchestrated through the network to raise potential matches, and each node will use a certified email to send the customer a matching offer in a standardized format.

Other works handle the privacy issues with the use of various mechanisms based on encryption schemes.

For example, [204] analyzes in depth the performances of the two major types of *Attribute-Based Encryption (ABE)*: *Key-Policy Attribute-Based Encryption (KP-ABE)* and *Ciphertext-Policy Attribute-Based Encryption (CP-ABE)*. Simulations are carried out on different classes of mobile devices, including a laptop and a smartphone, in order to establish under what conditions *ABE* is better suited for IoT. *ABE* potentially provides a public key encryption scheme which enables fine-grained access control, scalable key management, and flexible data distribution.

Another approach that uses an attribute-based signature scheme to guarantee privacy in IoT is presented in [194]. Here, a novel *Attribute-Based Signature (ABS)* scheme, named *ePASS*, uses an attribute-tree and expresses any policy consisting of AND/OR operators, which are unforgeable for the computational *Diffie-Hellman* assumption. In fact, users cannot forge signatures with attributes they do not own, and the signature provides assurance that only a user with appropriate attributes satisfying the policy can endorse the message. Moreover, the legitimate signers remain anonymous and are indistinguishable among all users whose attributes satisfy the policy, which provides attribute privacy for the signer.

Focusing on the privacy protection in IoT, [157] puts forward a key-changed mutual authentication protocol for WSN and RFID systems. Such a protocol integrates a random number generator in the tag and the reader, and adopts a one-way hash function, key refresh in real time, and key backup as mechanisms to reduce the risks of replay, replication, denial of service, spoofing and tag tracking.

To summarize, privacy requirements in IoT are currently only partially covered and there is a wide space of research issues to be investigated. Those include to the need to define privacy policies starting from a well-defined model and the corresponding development, dealing with scalability and the dynamic environment that characterizes IoT scenarios, and allowing different management for sensitive or identifiable data. In fact, capturing privacy requirements in the very early stages of development is essential for creating sufficient public confidence and facilitating the adoption of novel IoT systems.

2.1.3 Trust

The trust concept is used in various contexts and with different meanings. Trust is a complex notion about which no definitive consensus exists in the scientific literature, although its importance is widely recognized. One significant problem with many approaches towards trust definition is that they do not lend themselves to the establishment of metrics and evaluation methodologies. Moreover, the satisfaction of trust requirements are strictly related to the identity management and access control issues.

[18] and [19] focus on trust level assessment of IoT entities. The authors assume that most smart objects are human-carried or human-related devices, so they are often exposed to public areas and communicate through wireless, hence are vulnerable to malicious attacks. Smart objects have heterogeneous features and need to work together cooperatively. The social relationships considered are: friendship, ownership and community, since users are friends among themeselves (i.e., friendship), users own the devices (i.e., ownership) and the devices belong to some communities (i.e., community). Malicious nodes aim to disrupt the basic functionality of IoT by means of trust-related attacks: self-promoting, bad-mouthing and good-mouthing. The trust management protocol for IoT proposed in [18] is distributed, encounter-based, and activity-based: two nodes that come into contact with each other or are involved in a mutual interaction can directly rate each other and exchange trust evaluation about the other nodes, so they perform an indirect rate which seems like a recommendation. The reference parameters to trust evaluation are: honesty, cooperativeness, and community-interest. Therefore such a dynamic trust management protocol is capable of adaptively adjusting the best trust parameter setting in response to dynamically changing environments, in order to maximize application performance.

A similar approach to provision of a trustworthiness evaluation is carried out in [144] in the so called *Social Internet of Things (SIoT)*. This paradigm derives from the integration of social networking concepts into IoT, due to the fact that the objects belonging to the IoT infrastructure are capable of establishing social relationships in an autonomous way with respect to their owners. The challenge addressed in [144] is to build a reputation-based trust mechanism for the *SIoT* which can effectively deal with certain types of malicious behaviors aimed at misleading other nodes, in order to drive the use of services and information delivery only towards trusted nodes. A subjective model for the management of trustworthiness is defined, and built upon the solutions proposed for peer-to-peer networks, such as those proposed in [94] [214] [179] [220] [113]. Each node computes the trustworthiness of its friends on the basis of its own experience and on the opinion of the common friends. As a consequence, a node chooses the provider of the service it needs on the basis of the highest computed trustworthiness level.

In relation to the social network context, in [105] the authors propose a secure distributed ad-hoc network; it is based on direct peer-to-peer interactions and communities creation in order to grant a quick, easy and secure access to users to surf the Web; thus it is close to the social network concept. Each node (i.e., device) and community have an identity in the network and modify the trust of other nodes on the basis of their behavior, thus establishing a trust chain among users. The analyzed parameters are: physical proximity, fulfillment and consistency of answer, hierarchy on the trusted chain, similar properties (e.g., age, gender, type of sensor), common goals and warrants, history of interaction, availability, interactions. Chains of confidence will allow the establishment of groups or communities and unique identities (for the communities) for the access to services as well as for the spreading of group information. Therefore, security is established when the users access the network through the trust chain, generated by nodes, which he/she crosses.

In [124] the traditional access control models are considered as unsuitable for the decentralized and dynamic IoT scenarios, where identities are not known in advance. A trust relationship between two devices helps in influencing the future behaviors of their interactions. When devices trust each other, they prefer to share services and resources. This is the same as the idea that emerged in [144] and [18]. [124] presents a *Fuzzy* approach to the *Trust Based Access Control (FTBAC)*. The trust scores are calculated by the *FTBAC* framework from factors like experience, knowledge and recommendation. Such trust scores are then mapped to permissions, and access requests are accompanied by a set of credentials which together constitute a proof for allowing the access or not.

FTBAC framework is composed of three layers:

- *Device Layer*, which includes all IoT devices and communication among these devices
- *Request Layer*, which is mainly responsible for collecting experience, knowledge and recommendation information and calculating fuzzy trust values
- *Access Control Layer*, which is involved in the decision making process and maps the calculated fuzzy trust value to the access permissions, with the principle of least privilege.

The simulation results show that this framework guarantees flexibility and scalability and it is energy efficient. In fact, a solution based on cryptographic protection can achieve access control by increasing the trust level, but it creates extra overhead in terms of time and energy consumption; instead, according to authors, the fuzzy approach is easier to integrate in utility-based decision making.

In [93] another fuzzy approach to trust evaluation, based on three layers, is presented: sensor layer, core layer, and application layer. The sensor layer includes physical devices (e.g., RFID, WSN and base stations); the core layer mainly includes access network and Internet; the application layer includes various distributed networks (e.g., peer-to-peer, grid, cloud computing), application systems and interfaces. From the users' point of view, the IoT system is regarded as a *Service Provider (SP)* and the trust management aims to provide an auxiliary service that assists the IoT to provide more qualified service to any *Service Requester (SR)*. The relationship is bidirectional as the trust mechanism has effects both on the *SR* (for privacy protection) and *SP*. Such a trust management model includes three steps: trust extraction, trust transmission, and trust decision-making. Requested information service and trust based service coexist in this model. Trust management should act as a self-organizing component in order to deal with the information flow and preventing the privacy information from leaking to un-trusted *SR*. The authors in [93] make use of fuzzy set theory and a formal semantics-based language to perform the layered trust mechanism, evaluated by using specific layer attributes (i.e., efficiency, risk, history). The user has access to the IoT only if security credential satisfies security policies, which are defined by means of a decision-making function according to user trust value.

Note that, such a work discusses no concrete trust models, but establishes only a general framework, in which the well-defined trust models could be integrated.

[119] and [118] propose a trust model to protect the user security by combining location-aware and identity-aware information and authentication history; as a consequence, the users can obtain the trustworthiness for the requested services. Three trust regions are considered, each one having high, medium, and low ranks, respectively. For each rank, the authentication approach is different. In the high rank case, no extra key is needed. For medium rank, users have to offer their PIN for login. Low rank means that users need to provide biometric information, such as face image, fingerprint or iris scan, which may be not convenient for its complexity and hardware constraints. The goal is to make a classification of the provided services, in order to evaluate the sensitivity of the trasmitted information (i.e., on the basis of the type of application or the host in which the application is executed); to achieve this, a fuzzy approach is used.

Other proposals are not based either on the social networking concept nor on fuzzy methods. For example, in [208] the authors propose a hierarchical trust model for IoT, able to effectively detect malicious organizations from the behavior of their neighboring nodes. A *Verifiable Caching Interaction Digest (VCID)* scheme is introduced for the purposes of monitoring object-reader interaction, and a long-term reputation mechanism is used to manage the trust of organizations.

[176] proposes a trust management system for IoT, able to assess the trust level of a node from its past behavior, in distinct cooperative services. The main goal of this solution is to manage cooperation in a heterogeneous IoT architecture taking into account the different capabilities of nodes by exploiting a decentralized approach. Such a model considers both first-hand information (i.e., direct observations and own experiences) and second-hand information (i.e., indirect experiences and observations reported by neighboring nodes) to update trust values. Different phases are involved, in which the trust management system: (i) gathers information about the trustworthiness of the available nodes; (ii) sets up a collaborative service with the requesting nodes; (iii) learns from its past operation by performing self-updates aimed at improving its future operations; (iv) assigns a quality recommendation score to each node after each interaction during the learning phase.

In [48], the authors make an attempt to design an attack-resistant trust management model for distributed routing strategy in IoT. Such a model can evaluate and propagate reputation in distributed routing systems and it is then proposed to establish reliable trust relations between self-organized nodes and defeat possible attacks in distributed routing systems.

[117] starts from WSN and defines trust management for IoT, consisting of an identity-based key agreement; this agreement occurs by means of a distributed self-organizing key negotiation process. Such a protocol aims at preventing attacks from outside the network and recognizing malicious nodes. Thus it can reduce communications with malicious nodes to improve security and extend network lifetime.

[126] presents an identity-based network protocol aimed at identifying network nodes, which move themselves from host to host during the handover processes. Therefore, it needs to decouple identifiers and locators in

order to separate the node identification from host addressing. The mutual authentication of network nodes is achieved by the validation of the identity attributes and then by attaching a signature to each attribute, emitted by a trusted signing entity. Access to non public identity information is regulated by policies defined by the owner of the information. Thus, it is only disclosed to the authorized subjects by using the same attribute-based authorization method. Nodes and a domain trusted entity are connected to each other to build a globally trusted infrastructure by the pre-sharing of cryptographic certificates and ensuring the confidentiality and authentication of their exchanges by means of encryption and signature mechanisms.

As pointed out in [196], current trust and reputation management approaches usually offer rigid and inflexible mechanisms to compute reputation scores, which hinder their dynamic adaptation to the current environment where they are deployed. At most, they provide certain parameters that are configurable or tunable. This seems not enough for the heterogeneous and dynamic IoT context. Therefore, [196] has designed and prototyped a flexible mechanism to select the most suitable trust and reputation model in heterogeneous environments. Such a mechanism can be applied on-the-fly, among a pool of predefined ones, considering the current system conditions (e.g., number of users, allocated resources).

Another trust system is proposed in [120], based on node behavior detection. The metrics periodically evaluated are recommended trust and history statistical trust. They are calculated by evidence combination and *Bayes* algorithm, respectively.

This overview shows that the available solutions exploit different techniques in order to handle the trust issue in the IoT scenario. Proposals include hierarchical models, reputation mechanisms, approaches derived from social networking, fuzzy techniques, mechanisms based on nodes past behavior or on routing strategies (a scheme concerning the analyzed works is shown in Table 2.3). Literature seems mature enough in the field of trust management, but the definition of a fully distributed and dynamic approach suitable for the scalable and flexible IoT context is still missing, as confirmed in a complete survey on trust management in IoT, provided in [216]. Further missing items are the definition of globally accepted certification authorities and of a common-accepted trust negotiation language. To sum up, the following issues are still open in IoT-trust management:

- The introduction of a well-defined trust negotiation language supporting the semantic interoperabilty of IoT context
- The definition of a proper object identity management system
- The development of a trust negotiation mechanism in order to handle data stream access control.

2.1.4 Enforcement in IoT

The definition of security and data quality policies on data may be not sufficient for satisfying the requirements of an IoT system, because violation attempts should also be considered. This requires the inclusion of policy enforcement mechanisms, which define how the system shall react in such

TABLE 2.3: Summary of related works on trust assessment

Adopted technique	Works
Social networking	[18] [19] [144] [105]
Fuzzy technique	[124] [93] [119] [118]
Cooperative approach	[208] [176] [48] [120]
Identity-based method	[117] [126]

cases. In more detail, policies are operating rules that need to be enforced for the purpose of maintaining data order, security, and consistency. The policy enforcement assures that the security tasks can only be fulfilled if they are in accordance with the underlying security policies, as determined by consulting the policy decision component and deciding whether to allow an entity to perform an operation on a system resource. This aspect is poorly covered in existing literature, which mostly focuses on how to manage policy enforcement.

[209] presents a simulation environment for various policy languages, such as *WS-Policy (Web Services-Policy)* and *XACML (eXtensible Access Control Markup Language)*, used in different systems. In fact, low-level enforcement mechanisms can vary from system to system. Thus, it is difficult to enforce a policy across domain boundaries or over multiple domains. Before applying policies 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 schools are essential. They have their own policy enforcement mechanisms to protect their own proprietary data and patient records. The problem is that there are many collaborations and communications among these domains; therefore, cross-domain policy enforcement becomes an essential component. However, in most cases, these domains use different policy languages to define the rules that are executed on their own platforms. When a new cooperation or communication is required between two separate domains, we do not know how many rules from one domain can be enforced by current enforcement mechanisms. Then, in most cases, the technical departments from these two domains have to work together to evaluate whether or not it is possible to make their systems interoperate. The same problem also exists in social networking environment (e.g., *Facebook, MySpace, LinkedIn*). Most existing social networking sites have privacy configurations based on their own enforcement mechanisms. When two social networking sites or two healthcare domains need to communicate or collaborate with each other, they have to rebuild or reconfigure their systems to make sure these activities are consistent with their own and their partners' policies. To cope with such issues, [209] 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), which can be used to model both policy languages and enforcement mechanisms. Therefore, a configurable middle-level component is provided for the mapping process among such different domains. The main disadvantage of the approach presented in [209], in comparison with our solution, is two-fold. On the one side, 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 and, at the same time, acting as a cross-domain middleware, without the need of a translation system, as the one proposed in [209], which could also lead to scalability issues. On the other side, 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 potentially large amount of authorization requests. Note that *XACML* also requires a cache system for improving its efficiency.

Also regarding policy definition languages, in [52] they are classified into two categories. On the one hand, there are policy enforcement languages, which generally simplify the specification and interpretation of policies; however, they lack the formal semantics needed to allow the verification of the policies themselves by means of formal proofs. On the other hand, there are policy analysis languages, which allow the formal policies analysis and the expression of a large variety of obligations. In [52], the authors introduce a policy language that aims to combine the advantages of both approaches. Then, policies are enforced using reference monitors, and a set of active rules specifies the set of actions that should be executed after the detection of particular events, if some conditions are met. However, such a language does not provide the operational semantics needed to dynamically enforce and handle obligations in a policy managed system.

As just emerged, expressing security policies to govern distributed systems, like 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 for business analysts to create intelligible specifications. In [43] a *Hierarchical Policy Language for Distributed Systems (HiPoLDS)* is introduced; it 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., *SOA*) 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.

A theoretical approach is presented in [107]. It introduces a formal and modular framework allowing one to enforce policies on a given concurrent system. In fact, one of the important goals of the software development process is to prove that the system always meets its requirements. To deal with this problem, two different approaches are proposed. The former is a conservative enforcement: the program should be terminated as soon as it violates the policy even if the current run could be partially completed. The latter is a liberal enforcement: the execution of the process is not aborted if it could be partially satisfied. With this approach, more properties are enforced than with the conservative one, but the program may terminate without fully satisfying the policy. Therefore the conservative enforcement will generate false negatives, while the liberal enforcement will generate false positives, and neither of them will reach the desired result. In [107] the liberal enforcement is developed, which can be further extended to handle the conservative one. In more detail, an extended version of *Algebra for Communicating Process (ACP)* [13], designed for specifying concurrent systems behavior, and *Basic Process Algebra (BPA)* language for policy specification are adopted. *ACP* is further enhanced with an enforcement operator, whose actions run in parallel within the system, in order to monitor the requests and the satisfaction of the related policies.

Further theoretical systems are the following. [122] provides an overview of network security, security policies, policy enforcement and firewall policy management systems. As far as policy enforcement is concerned, it proposes to use security services such as authentication, encryption, antivirus software and firewalls in order to protect data confidentiality, integrity, and availability. In contrast, the authors of [155] present a framework able to prove whether the code implementing access control respects access control policy specifications. Furthermore, the authors of [190] state that the application logic, embodied in the system components, should be separated from the related policies. Therefore, they propose an infrastructure that can enable policy, representing high-level (i.e., users) or systems entities, able to drive the system functionality in a distributed environment. To this end, a middleware, able to support a secure and dynamic reconfiguration and to provide a policy enforcement mechanism across system components, is introduced. Nevertheless, neither a case study nor a working implementation is provided.

In [60] a novel access control framework, named *Policy Machine (PM)*, is proposed. It is composed by the following basic entities: authorized users, objects, system operations, and processes. Users may be either human beings or system users; objects specify system entities which are controlled under one or more policies (e.g., records, files, e-mails); operations identify the actions that can be performed on the contents of objects (e.g., read, write, delete); finally, users submit access requests through processes. Policies are grouped into classes according to their attributes and, therefore, an object may be protected under more than one policy class, and, similarly, a user may belong to more than one policy class. *PM* is able to configure many types of access control policies, and it is independent from the different operating systems and applications; users need only to login to *PM* in order to interact with the secure framework. [60] demonstrates the *PM* ability to express and enforce the policy objectives of *RBAC* [177], *Chinese Wall* [28], *MAC* and *DAC* models [25]. Moreover, *PM* is able to face many Trojan horse attacks, to which *DAC* and *RBAC* are vulnerable.

Similarly, [166] introduces a semantic web framework and a meta-control model to orchestrate policy reasoning with the identification and access of the information sources. In fact, in open domains, enforcing context-sensitive policies requires the ability to opportunistically interleave policy reasoning with the dynamic identification, selection, and access of relevant sources of contextual information. Each entity (i.e., user, sensor, application or organization) relies on one or more policy enforcing agents responsible for applying relevant policies in response to incoming requests. The framework is suitable to a number of domains where policy reasoning requires the automatic discovery and access of external sources.

Note that [60] and [166] only enforce access control policies and do not refer to a distributed nature of the proposed solutions, which is a pivotal requirement in IoT applications.

Another aspect to be considered is how information and policy sharing is regulated within distributed systems. In fact, in the IoT context, multiple application domains may be involved with heterogeneous data and different required policies. Therefore, an efficient and effective IoT architecture should be able to manage both the data acquisition and provision and the synchronization of policies belonging to the actual realms. In this direction, the approaches of [101] and [71] are presented below.

[101] developed an information security policy process model, based on a methodology involving qualitative techniques, able to evaluate the external and internal influences that can impact on organizational security against cyber threats. Such a model uses a data-centered approach that allows to identify the primary policy processes, the main environmental and organization influences, and the relevant linkages among them. Note that such a process model represents a generalized framework rather than a specific model for a single company. Therefore, it does not consider the peculiar aspects of each organization in a way that the model may not equally apply to all organizations. Moreover, it does not address exceptional situations that may warrant a temporary violation of predefined policies. However, it would be better to to go beyond such limitations by means of a distributed middleware, able to manage the information of heterogeneous application domains and update the policies in real time, in accordance with the organizational or users' actual requirements.

Also, the work presented in [71] highlights that security mechanisms are often enforced in a separated way from each other (e.g., no interactions or integrations among different companies or organizations), thus limiting the kinds of policies that can be enforced in distributed and heterogeneous settings. To cope with such an issue, [71] introduces the concept of a *Security Service Bus (SSB)*, representing a dedicated communication channel among the applications and the different security mechanisms. It allows the enforcement of advanced policies by providing uniform access to application-level information. Nevertheless, such a security infrastructure is not flexible and scalable enough for the breadth of IoT scenarios, since the dedicated channel may become a bottleneck for the interactions among security services and bounded applications.

Finally, the enforcement solution presented in [140] 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 activites taking part into the

TABLE 2.4: Summary of related works on policy enforcement

Target	**Works**
Cross-domain policy issues	[209] [103] [190]
Policy language definition	[52] [43]
Access control	[60] [166]
Formal proof	[107] [122]
Policy sharing	[101] [71]

system itself.

Summarizing the state of the art, except for the work presented in [140], there are no specific solutions addressing policy enforcement in IoT applications, although they are essential to ensure a safe deployment of IoT systems.

In fact, [209], [103] and [190] mainly address cross-domain policy issues, [52] and [43] focus on a policy language definition, [60], as well as [166], enforces only access control policies, [107] and [122] are about a formal proof of the correct behavior of a system with poorly defined security rules, [101] and [71] cope with policy sharing issue. Such considerations are summarized in Table 2.4.

The identification of the enforcement mechanisms suitable for the IoT context is fundamental, finding an equilibrium between the guarantee of proper security requirements and the computing efforts. Some attempts have already been made to define the proper languages for the specification of policies, but a standard that addresses the IoT paradigm specifically is still missing. Note that an efficient and secure solution is expected to be suitable for integration in existing IoT architectures, such as *OneM2M*[1], *OpenIoT*[2], *FIWARE*[3], and *MOBIUS*[4], already adopted by a multitude of companies. In particular, an enforcement framework should be conceived as an orthogonal secure extension of such architectures. A first attempt in this direction has been made in [185], a work orthogonal to this thesis, where a security plugin for managing access operations has been added to the original *OneM2M* system.

2.1.5 Secure middlewares for IoT

One of the main factors limiting the growth and take-up of IoT is the lack of a set of standardised tools, platforms and interfaces able to provide interoperability across different vendors of hardware and software solutions as well as across diverse vertical domains.

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

In the last few years, many initiatives tried to bridge this gap, reusing concepts, techniques and protocols from the Internet domain. For example, in recent years, the widespread adoption of web services has provided a standard framework to enable systems interoperability according to the principles of *SOA*. *Service-Oriented Communications (SOC)* technologies manage web services by creating a virtual network and adapting applications to the specific needs of users rather than users being forced to adapt to the available functionality of applications [154] [221]. Although the trend towards the adoption of *SOA* architectural paradigm in the IoT domain is shared by the majority of the scientific community, at the moment the state of the art in this area is still somehow limited [156] [159].

Due to the very large number of heterogeneous technologies normally in place within the IoT paradigm, several types of middleware layer are employed to enforce the integration and the security of devices and data within the same information network. Typically, they enforce data to be exchanged according to strict protection constraints. Heterogeneity of devices and communication technologies in IoT has to be accounted for in the design of such a middleware architecture. In fact, while many smart devices can natively support IPv6 communications [149] [14], existing deployments might not support the IP protocol within the local area scope, thus requiring ad hoc gateways and middlewares [26].

Both the networking and security issues have driven the design and the development of *VIRTUS* [42], an IoT middleware relying on the open *eXtensible Messaging and Presence Protocol (XMPP)* to provide secure event-driven communications within an IoT scenario. Leveraging the standard security features provided by *XMPP*, the middleware offers a reliable and secure communication channel for distributed applications, protected with both authentication (through *TLS* protocol) and encryption (*SASL* protocol) mechanisms. For client-to-server based communications, it is not clear from the description which method is actually implemented; while for server-to-server communications the use of *SASL* is specified to ensure full server federation. The *VIRTUS* model does not describe the challenges of implementing a personal instance of middleware for single users or devices; however a concept of edge computing is presented, where some interactions may happen within an edge domain (e.g., within a house) and lower security is required. The main drawback is that the capability to process and/or filter data from the edge network before sharing it is not discussed, except in very granular terms.

[69] proposes an *AmI* framework, called *Otsopack*. This solution provides two core features: (i) it is designed to be simple, modular and extensible and (ii) it runs in different computational platforms, including *Java SE* and *Android*. The underlying interface is based on HTTP and uses a *REpresentational State Transfer (REST)* interface. Different implementations can provide only certain features (e.g., data access) and still interact with each other. In this way, it is possible to embed it in other devices. This gateway platform only supports *Python* and requires a partial *ad hoc* implementation. Note that such aspects represent a limitation of the proposed solution in [69]. It uses a *TSC (Triple Space Computing)*, that is a coordination paradigm which promotes the indirect communication style and uses semantic data. The way it works is simple: each application writes semantically annotated information in a shared space, and other applications or

nodes can query for it. As regards security, given the data-centric nature of the framework, there are two core requirements: (i) a data provider may only grant access to data to a certain set of users and (ii) a data consumer may trust only a set of providers for certain set of acquired data. A derived issue is how to authenticate each other in such a dynamic scenario. In order to support the first requirement, an *OpenID*-based solution has been built. An identity provider securely identifies data consumers to the data providers. Data providers can establish which graphs can be accessed by which users. Therefore, the provider will return a restricted graph only if the valid user is requesting it. In other words, the same application can get different amounts of information depending on whether it provides credentials or not. As far as security is concerned, [69] is not yet mature work. In this thesis, the goal is to go beyond actual implementations towards the definition of a middleware able not only to guarantee an efficient access control system, but also a complete and accurate assessment of the managed data. In this direction, *Otsopack* is far from being portable and security-aware.

In [90], a framework for enhancing security, privacy and trust in embedded system infrastructures is proposed. The authors suggest the use of lightweight symmetric encryption (for data) and asymmetric encryption protocols (for key exchange) in *Trivial File Transfer Protocol (TFTP)*. To this end, two schemes are implied: *AES*, for protecting personal and sensitive data, and *DHKE (Diffie-Hellman Key Exchange)*, for exchanging cryptographics keys between two entities that do not know each other. The target implementation of *TFTP* is the embedded devices such as WiFi *Access Points (AP)* and remote *Base Stations (BS)*, which should be attacked by malicious users or malwares with the installation of malicious code (e.g., backdoors). [90] emphasizes finding a solution for strengthening the communication protocol among *AP* and *BS*. To verify this proposal, the authors decided to use *UBOOT (Universal Boot loader)*. As a result, the solution proposed in [90] has a limited target, without concerning scalability issues.

In [115] a *Naming, Addressing and Profile Server (NAPS)* as a middleware to bridge different platforms in IoT environments is presented. Given a large number of heterogeneous devices deployed across different platforms, *NAPS* serves as key module at the back-end data center to aid the upstream, the content-based data filtering and matching and the downstream from applications. [115] proposes a novel naming convention for devices and device groups. While previous research efforts only focus on a specific standard or protocol, the authors aim to design a middleware component serving dynamic application needs. Therefore, an IoT *Application Infrastructure (IoT-AI)* was designed, the technical components of which are: application gateway, service registration portal, *Real-time Operational DataBase (RODB)*, and protocols like *Universal Plug and Play (UPnP)*. The interfaces provided are based on the *RESTful* design style where standard HTTP request/response is used for data transport. When device profile information is registered either manually or automatically from each IoT platform, an identifier is automatically generated. The system deals with *Authentication, Authorization and Accounting (AAA)*. Although it is not the focus of this work, the design can largely leverage the *Network SEcurity Capability (NSEC)* SC in *ETSI M2M* service architecture. Note that the device domain is organized on a tree structure. It uses a key hierarchy, composed of root key, service key and application keys. The root key is used to derive service keys through authentication, and key agreement between the device or gateway and the *M2M SC*s at the *M2M Core*. The application key, derived from the service key, is unique for every *M2M* applications. Nevertheless, security aspects are not deeply investigated, and neither is energy consumption. Moreover, it is not clear if the reference architecure is conceived as a centralized or distributed one.

OneM2M [147] proposes a global service layer platform for *M2M* communications. It aims at unifying the *Global M2M Community*, by enabling the interoperability of different *M2M* systems, across multiple networks and topologies on top of IP. The presented middleware is able to support a sort of secure end-to-end data transmissions among the *M2M* devices and the customer applications. Such a goal is obtained by means of authentication, encryption, connectivity setup, buffering, synchronization, aggregation and device management, but it does not represent a complete solution. As discussed in Paragraph 2.1.4, as an orthogonal work of this thesis, a secure extention of *OneM2M* architecture is provided in [185], in order to deeply cope with security issues.

[66] deals with the problem of task allocation in IoT. In more detail, the cooperation among nodes must be performed in an interoperable way towards collaborative deployment of applications, able to take into account the available resources, such as energy, memory, processing, and object capability to execute a given task. In order to address such an issue, a resource allocation middleware for the deployment of distributed applications in IoT is proposed. Starting from this component, a consensus protocol for the cooperation among network objects in performing the target application is added, which aims to distribute the burden of the application execution, so that resources are adequately shared. Such a work exploits a distributed mechanism and demonstrates better performance than its centralized counterpart. Note that this solution presents an orthogonal aspect of IoT system management, where data sources, besides acquiring data, have an active role in executing some functionalities, depending on the specific application domain.

Finally, the theorical work presented in [206] defines a method to deduce the process for the systematic construction of a general-purpose middleware for IoT. The middleware is generated starting from high level algebraic structures, which are then mapped into building components depending on the underlying computing infrastructure. Therefore, it is supposed to be adaptable to heterogeneous systems.

What emerges from this paragraph is that middleware currently lacks a unified vision, able to respond to all the IoT requirements, both in terms of security and network performance. Moreover, interoperability is becoming a fundamental challenge, in order to allow an independent development of distributed components, able to interact and cooperate with each other and also to exchange data on the basis of standards. It should be borne in mind that IoT involves not only data provided by devices/machines, but also by users, and so the interactions are machine-to-machine and also among users and machines and among users and users. Therefore, the design and development of a middleware have an impact on the system architecture (i.e., scalability, coupling among components). To design an effective solution, it is necessary to deal with several important questions:

- How heterogeneous devices and users can dynamically interact and agree on the same communication protocols, ensuring also security and privacy?
- How to make the solution suitable for different platforms and therefore not dependent either on the exploited interfaces or protocols?

2.1.6 Mobile security in IoT

Mobile nodes in IoT often move from one cluster to another, in which cryptography based protocols are required to provide rapid identification, authentication, and privacy protection. An ad-hoc protocol is presented in [125] that is activated when a mobile node joins a new cluster. Such a protocol contains a valid request message and an answer authentication message, which rapidly implements identification, authentication, and privacy protection. It could be robust towards replay attack, eavesdropping, and tracking or location privacy attacks. Compared to other similar protocols such as a basic hash protocol, it has less communication overhead, and more security and privacy protection properties.

[91] analyzes the security challenges for the *HIMALIS (Heterogeneity Inclusion and Mobility Adaptation through Locator ID Separation)* architecture regarding features from IoT and the id/locator management messages, vulnerable to attacks. This work proposes a secure and scalable mobility management scheme that considers the IoT constraints, solving the possible security and privacy vulnerabilities of the *HIMALIS* architecture. The proposed scheme supports scalable interdomain authentication, secure location update, and binding transfer for the mobility process.

Furthermore, RFID systems, which are one of the enabling IoT technologies and are based on *EPC (Electronic Product Code) Network Environment*, automatically identify tagged objects, using RF signals without direct contact. In [215], a mobile RFID network based on *EPC* is explained and the threats of the mobile RFID system are analyzed. Such an architecture guarantees certain levels of security and efficiency.

Moreover, for the security and privacy of mobile RFID systems, another security and privacy model is proposed about IoT in [224]. The model does not only take into account the privacy of tags and readers, but also supports tags corruption, reader corruption, multiple readers and mutual authenticated key exchange protocols.

Powered by location based services, IoT systems have the potential to enable systematic mass surveillance and to violate the personal privacy of users, especially their location privacy. [51] overviews some of the existing location privacy issues found in mobile devices. Particular attention is paid to the current access permission mechanisms used on the Android, iPhone, and Windows Mobile platforms. Note that the actual privacy issues in mobile platforms should be inherited by IoT and integrated with other static platforms.

In [112] a secure handshake scheme among mobile nodes is proposed in an intelligent transportation system. In more detail, a mobile node verifies, over an insecure communication channel, the legitimacy of an ordinary sensor node by a private negotiation of the handshake attributes; in this way, a mobile hierarchy is established in order to query a deployed WSN in a secure manner.

[95] points out that a secure healthcare service creates new demands for mobile solutions. To protect the privacy and security of patients in a healthcare context using an IoT infrastructure, a proper mechanism is proposed. From a trustworthiness point of view, service providers must get authentication from a public authority, which is also responsible for handover cryptography credentials to each actor, in order to allow secure communication among the end-devices and the application brokers; the goal is to establish trusted IoT application market, where information on end-devices can be exchanged to establish a secure connections between market and users.

In [70], a security architecture deployable on mobile platforms is defined for mobile e-health applications. In particular, RFID tag identification in medical context and structured and secured IoT solutions are combined, in order to enable ubiquitous and easy access to medical related records, while providing control and security to all interactions.

Also, in [224] and [145], the mobile RFID technology is exploited to solve the following security and privacy issues: not all existing tags support hash functions in designing RFID protocols and channels between readers and servers are not always secure in a mobile context. Therefore, an ultralightweight and privacy-preserving authentication protocol for mobile RFID systems is defined, using only bitwise XOR and several special constructed pseudo-random number generators. This approach provides several privacy properties (e.g., tag anonymity, tag location privacy, reader privacy, mutual authentication) and avoids suffering from a number of attacks (e.g., replay attacks, desynchronization attacks).

In [92] an efficient and secure *mobile-Intrusion Prevention Systems (m-IPS)* is proposed for business activities using mobile devices for human-centric computing. This system checks user temporal and spatial information, profiles and role information to provide precise access control.

[67] designs a mobile information collection system based on IoT, implementing an access gateway by smart mobile devices. Moreover, besides the authentication of the mobile terminals through the gateway, a pivotal role is played by the collection strategy, which exploits the historical data movement paths, in order to reduce the problem of over long device connections, improving the efficiency of information transmission.

In [104] special attention is paid to security and mobility in IoT. In fact, people and companies want to secure their data using firewalls, which inevitably leads to a challenging conflict between data security and usability. Since lots of products are becoming increasingly mobile, the authors of [104] design a *Quantum Lifecycle Management (QLM)* messaging standard in order to provide generic and standardized application-level interfaces to guarantee a two-way communications through any type of firewall, for example to perform real-time control.

A *Mobile Sensor Data Processing Engine (MOSDEN)* is presented in [160], which is a plug-in-based IoT middleware for resource-constrained mobile devices (until now built on the *Android* platform), which allows one to collect and process sensor data without programming efforts. It supports both push and pull data streaming mechanisms as well as centralized and decentralized (e.g., peer-to-peer) data communication.

[46] focuses on security for mobile devices, which adopt various permission systems for installing the applications. In more detail, the defintion of proper contracts, able to regulate the actions performed by such applications towards the user data, is addressed. In fact, probability aspects are introduced into the workflow of two contract-based approaches, specifically developed for mobile devices, namely *Security-by Contract* and *Security-by-Contract-with-Trust* frameworks. Such approaches integrate several security techniques to build a chain of trust, in order to ensure that the downloaded applications will execute only security actions allowed by user policy. Note that current models only permit the definition of a set of allowed actions, but more expressive policies are required, able to take in account a possible action history. To this end, a probabilistic automata-based model is proposed; in this way, developers/users can define more expressive contracts/policies through probabilistic clauses.

Hence, since a large number of IoT devices is likely to be mobile, a mobility management protocol is required in order to maintain IP connectivity, for example through the 6LoWPAN standard, as proposed in [133]. Other works, such as [173], deal with the efficient video dissemination in mobile multimedia IoT applications, while [55] studies the interaction of smart things with the traditional web technologies by means of a mobile Bluetooth platform. Social relationships are investigated in [9] by means of a cognitive model, in IoT mobile nodes; while the use of *NFC (Near Field Communication)* for payments with mobile devices in the so called *Web of Things (WoT)* is studied in [73], which proposes a lightweight architecture based on *RESTful* approach.

Summarizing, also if the security issues of mobile devices (i.e., devices identification and authentication, key and credential storage and exchange) are under investigation by the scientific community, the available solutions partially address these needs, thus requiring further efforts in order to allow the integration with the other IoT technologies.

2.1.7 Ongoing projects

Security and privacy in IoT are an object of interest to the *European Commission*. In fact, there are many projects addressing such issues in IoT field.

Butler [29] is *European Union FP7* project; its purpose is to enable the development of secure and smart life assistant applications by means of a context and location-aware, pervasive information system. It focuses on the following scenarios: smart-cities, smart-health, smart-home/smart-office, smart-shopping, smart-mobility/smart-transport. As regards security and privacy requirements, the *Butler* project aims to allow users to manage their distributed profile; this implies the control of data duplication and of identity sharing over distributed applications. The final purpose is to implement a framework able to integrate user dynamic data (i.e., location, behavior) in privacy and security protocols.

[98] presents an *Intrusion Detection System (IDS)* framework for IoT systems empowered by IPv6 over low-power personal area network (6LoWPAN) devices, which is a protocol suitable for resource constrained IoT environments. 6LoWPAN devices are vulnerable to attacks inherited from both the wireless networks and Internet protocols. The proposed *IDS* framework, which includes a monitoring system and a detection engine,

has been integrated into the network framework developed within the *EU FP7* project *EBBITS* [57].

The *Hydra* project [86] develops a middleware for networked embedded systems, based on a *SOA*. It is co-funded by the *European Commission*. *Hydra* contemplates distributed security issues and social trust among the middleware components. Such a middleware allows developers to incorporate heterogeneous physical devices into their applications by offering easy-to-use web service interfaces for controlling any type of physical device without relying on the various network technology involved, such as Bluetooth, RF, ZigBee, RFID, WiFi, etc. *Hydra* incorporates the means for device and service discovery, semantic model driven architecture, peer-to-peer communication and diagnostics.

The *uTRUSTit (Usable Trust in the Internet of Things)* [200], *EU-funded FP7* project, aims at creating a trust feedback toolkit in order to enhance the user trust perception in a IoT context. *uTRUSTit* enables system manufacturers and system integrators to express underlying security concepts to users in a comprehensible way, allowing them to make valid judgments on the trustworthiness of such systems.

iCore project [88] provides a management framework as a wider IoT eco-system, able to be used by different kinds of users and stakeholders and across different applications domains. The *iCore* proposed solution is a cognitive framework including three levels of functionality: virtual objects (*VO*s), composite virtual objects (*CVO*s), and functional blocks, for representing the user/stakeholder perspectives. Of particular importance are *VO*s, which are cognitive virtual representations of real-world objects (i.e., sensors, devices, everyday objects) and hide the underlying technological heterogeneity. Whereas *CVO*s are cognitive mashups of semantically interoperable *VO*s, delivering services in accordance with the user/stakeholder requirements. The difference between a real or digital object and a virtual object is that the former may be owned or controlled by a particular stakeholder, whereas the latter can be owned or controlled by particular service providers. *CVO*s may be owned or controlled by yet another provider who adds value by combining different virtual objects and providing these combinations to users. This leads to a hierarchical structure and therefore to a complex eco-system, which is hidden from the different stakeholders and opens new opportunities. The *iCore* solution shall be equipped with essential security protocols/functionalities, which span all levels of the framework and take into account the ownership and privacy of data and the access to objects. It will guarantee the secure distribution and aggregation of information exchanged among the architecture components, as well as between physical and virtual world. To test the effectiveness of such proposals, *iCore* addresses the following use-cases: ambient-assisted living, smart-office, smart-transportation and supply chain management.

Beyond Europe, other countries concur with several projects to deal with security issues in IoT. In US, in 2012 *DARPA* announced the *High Assurance Cyber Military Systems* program *(HACMS)* [78], which is trying to patch the security vulnerabilities of IoT. The agency wants to make sure that military vehicles, medical equipment, and even drones cannot be hacked from the outside. *HACMS* aims at providing the seeds for future security protocols, allowing IoT to get off the ground, achieveing sufficient standardization and security. In the future, some of the software tools that emerge from the *HACMS* program could be used in a civilian context. Another institute interested in security in the cyber-physical systems is the *National Science Foundation (NSF)* [135]. It financed the *Roseline* project [174], which aims to find robustness solutions for cyber-physical systems to accurately and securely interact with time; in fact, the coordination of the activities within the infrastructure, the control of communications and the knowledge of time to infer location emerge as being critical issues for real-time security. The *Roseline* project is targeted to a variety of sectors, such as smart grids, aerospace systems, safety systems and autonomous vehicles. Other multi-institutional projects, included in the *NSF Future Internet Architectures (FIA)* program, are: *XIA-NP (Deployment-Driven Evaluation and Evolution of the eXpressive Internet Architecture)* [212], *NDN-NP (Named Data Networking Next Phase)* [136], *NEBULA* [137], and *MobilityFirst-NP (Next-Phase MobilityFirst-NP Project)* [131]. They aim at exploring novel network architectures and networking concepts, such as new communications protocols, able to extend beyond current networking components, mechanisms and application requirements. They also consider the larger societal, economic and legal issues that arise from the interplay between Internet and society, providing support for mobility and enhancing the cyber-security. In more detail, *XIA-NP* [212] addresses the growing diversity of network models, the need for trustworthy communication, and the growing set of stakeholders who coordinate their activities to provide Internet services. *XIA-NP* defines the application programming interface (API) for communication and the network communication mechanisms, guaranteeing the integrity and the authentication of the communication itself. In fact, *XIA-NP* enables flexible context-dependent mechanisms for establishing trust among the communicating devices. *NDN-NP* [136] addresses the technical challenges, including routing scalability, fast forwarding, trust models, network security, content protection and privacy. *NEBULA* [137] provides an architecture dealing with cloud computing; in such a project the data centers are connected by a high-speed, extremely reliable and secure backbone network, aiming at developing new trustworthy data, control and core networking approaches to support the emerging cloud computing model of always-available network services. The architecture proposed by *MobilityFirst-NP* [131] uses generalized delay-tolerant networking (*GDTN*) to provide robustness even in presence of link/network disconnections. *GDNT* is integrated with self-certifying public key addresses, providing a trustworthy network. Dealing with mobility, *MobilityFirst-NP* allows functionalities like context and location-aware services to naturally fit into the network. Such a project focuses on the tradeoffs between mobility and scalability and on opportunistic use of network resources to achieve effective communications among mobile endpoints.

The *National Basic Research Program of China* [84] raises the problem of security protection during the interaction process among the network entities, focusing on the information representation and balancing between efficiency and energy consumption. Europe collaborates both with China and Korea in the realization of an IoT architecture within the *Future Internet Research and Experimentation (FIRE)* project [61] [62], which aims at finding solutions for the deployment of IoT technologies in several application areas (e.g., public safety, social security, medical and health services, urban management, people's livelihood) with particular attention to information

TABLE 2.5: Contribution of ongoing projects in IoT security field

	Butler	*EBBITS*	*Hydra*	*uTRUSTit*	*iCore*	*HACMS*	*NSF*	*FIRE*	*EUJapan*
Authentication	x			x	x	x	x	x	
Confidentiality	x	x	x		x	x	x	x	x
Access Control	x	x		x	x	x	x	x	
Privacy	x				x		x	x	x
Trust				x	x		x		
Enforcement									
Middleware		x	x		x				
Mobile	x						x		

security, privacy and intellectual property rights. Likewise, the *EU-Japan ICT Cooperation* [56] carries on a collaboration between Europe and Japan as regards the so called *Future Internet*. Its main drivers are: the establishment of common global standards to ensure seamless communications and common ways to store and access information, the guarantee of highest security, and energy efficiency standards.

As regards worldwide projects, there are several attempts that address IoT requirements in terms of security, privacy and trust in order to develop a unified framework or middleware. The open IoT security issues faced by each project are summarized in Table 2.5. At the moment the efforts are aimed at specific application contexts and the impact of these proposals on a mass-scale market still needs to be checked.

It is worth remarking that other projects exist, which do not explicitly deal with security issues. Among them, the *FP7 COMPOSE (Collaborative Open Market to Place Objects at your Service)* project [57] aims to design and develop an open marketplace for IoT data and services. The basic concept underpinning such an approach is to treat smart objects as services, which can be managed using standard service-oriented computing approaches and can be dynamically composed to provide value-added applications to end users.

A dynamic architecture for service orchestration and self-adaptation is proposed in *IoT.EST (Internet of Things Environment for Service Creation and Testing)* [89]. The project defines a dynamic service creation environment that gathers and exploits data from sensors and actuators making use of different communication technologies and formats. Such an architecture deals with issues such as the composition of business services based on reusable IoT service components, the automated configuration and testing of services for "things" and the abstraction of the heterogeneity of underlying technologies to ensure interoperability.

The *Ebbits* project [50] designs a *SOA* platform based on open protocols and middleware, effectively transforming IoT subsystems or devices into web services with semantic resolution. The goal is to allow businesses to integrate IoT into mainstream enterprise systems and to support interoperable end-to-end business applications.

2.2 Research directions to secure IoT applications

The broad overview provided in Paragraph 2.1 raises many open issues, and sheds some light on research directions in the IoT security field. A unified vision regarding the satisfaction of security and privacy requirements in such a heterogeneous environment, involving different technologies and communication standards is still missing.

Suitable solutions need to be designed and deployed, which should be able to guarantee: confidentiality, access control, and privacy for users and things, trustworthiness among devices and users, compliance with defined security and privacy policies. Research efforts are also required to achieve the integration of IoT and communication technologies in a secure middleware, able to cope with the defined protection constraints. Another research field focuses on IoT security in mobile devices, increasingly widespread today.

Much effort is spent by the worldwide scientific community to address the aforementioned topics, but there are still many open issues to be faced. This thesis aims to cope with some of them. In particular, the solutions proposed in the next chapters concern:

- The modelling of a system architecture able to include all the entities involved in the IoT environment along with their relationships, as expressed in the requirements specified in Paragraphs 2.1.1 and 2.1.2, which would represent a starting point for the other steps towards the realization of a real and complete IoT platform
- The design and development of a secure distributed middleware conceived for the general-purpose IoT domain and able to manage secure interactions among heterogeneous devices and users, in response to the questions pointed out in Paragraph 2.1.5
- The definition of proper methods for evaluating the security level (e.g., confidentiality, integrity, privacy) of the transmitted/processed information, as underlined in Paragraphs 2.1.1 and 2.1.2, to improve the resilience of the system in the presence of malicious attacks or disallowed accesses, as well as to guarantee end-to-end security
- The adoption of an effective key management system in order to improve the system resilience to external or malicious attacks, as pointed out in Paragraph 2.1.1
- The integration of a policy enforcement framework tailored for IoT applications, as specified in Paragraph 2.1.4
- The definition of an authentication mechanism in order to control access to resources in a secure and efficient way, as required in Paragraph 2.1.1

2.3 Data quality issues

Finally, a brief discussion has also to be made as regards data quality issues. As far as data quality is concerned, it is not the main focus of this thesis, the primary goal of which is related to security. However, some details are also provided in order to better understand the capabilities of the solutions proposed in the following chapters. Note that several scientific publications recognize the pivotal role of data quality in the IoT research landscape.

In fact, the volume of data and the variety of the information sources lead to new challenges, besides security, in the data quality field. In particular, the aim of researchers and practitioners is the evaluation of the "fitness for use" of data sets [203].

Traditional data quality approaches mainly focus on the following activities:

- The definition and the assessment of proper metrics, capable of relevating the fitness of specific structured or semi-structured data for its intended use
- The identification of the objects, in order to recognize whether data from one source or another represent the same objects in the real world [21]
- The definition of improvement methods, such as data cleaning, ora a process of analysis for the detection of the root causes of poor data quality.

Most of the existing methods and techniques are based on two main assumptions: (i) data are structured and (ii) the purposes for which data are used are known. Clearly, these two assumptions are not valid in the IoT environment, where heterogeneous sources are available and the data are not related to a predefined set of processes, but they can be used to satisfy various requirements.

In such a scenario, new assessment techniques should be investigated and additional data quality metrics should be defined. Data quality may be affected by both intrinsic metrics or by the data provenance, such as the credibility and/or reputation of the sources themselves.

The former ones describe the gathered values by evaluating the following metrics:

- *Accuracy*, which is conceived as the extent to which data are correct (i.e., the gathered values stored, for example, in a database correspond to real-world values [203] [15])
- *Completeness*, which is defined as the degree to which a given collection includes data describing the corresponding set of real-world objects
- *Timeliness*, which corresponds to the extent to which the age of data is appropriate for the actual task. In more detail, the temporal validity associated to the data is defined by two components: age and volatility. Age or currency is a measure of how old the information is, based on how long ago it was recorded. Volatility is a measure of information instability intended as the frequency of change of its values [27].

As regards the latter ones, a feature to be considered is the concept of trust, which, as regards data quality, is associated with source reputation and, thus, reliability. In particular, trust can be defined as the probability for data to be suitable for inclusion in a specific process. The source or the sources, used to gather the information, may influence the data reliability; for example, information extracted from the web have, in general, a degree of trustworthiness lower than the information provided by a certified source.

This all argues for the importance of enriching data with information about their quality emerges. As an example, in [76], authors claim the need for control over data sources to ensure their validity, accuracy and credibility. Data accuracy is also covered in [129], where the authors observe that the presence of many data sources raises the need to understand the quality of such data. In particular, they state that the data quality dimensions to consider are accuracy, timeliness and the trustworthiness of the data providers. Moreover, anomaly detection techniques are widely employed in various scenarios to remove noise and inaccurate data in order to improve data quality.

Also data integration is a relevant issue in IoT. In order to support data fusion, data quality plays a fundamental role, addressing issues related to object identification. Note that data fusion may be also driven by the assessment of the data values, together with other techniques able to manage the, possibly, huge data volume. In this way, it should be possible to provide support for the selection of the suitable sources, which can satisfy a specific request.

Note that the huge number of data sources in IoT is considered a positive aspect for data fusion and for the extraction and provisioning of advanced services. Besides temporal aspects (i.e., currency) and data validity, a related work adds another important dimension such as availability [110]. The authors of this work define new metrics for the aforementioned quality dimensions in IoT context and evaluate the quality of the real-world data available on the open IoT platform *Cosm*. In particular, they show that data quality problems are frequent and they should be adequately tackled or, at least, users should be aware of the poor quality of the data sources they use.

Chapter 3

Internet of Things Middleware

3.1 Introduction

Open issues related to both security and data quality may hinder the large-scale adoption and diffusion of IoT applications, as can be seen from the state of the art in Chapter 2. Furthermore, as discussed in the preceeding chapters, IoT deployments include a large heterogeneity of technologies and communication protocols, leading to scalability issues. Hence, the most crucial challenge in building a dynamic and efficient IoT system lies in the definition of a cross-domain and interoperable software framework, which is still missing. In order to fill this gap, the following steps have been carried out during this thesis:

- The modeling of an IoT platform by means of the definition of a UML (Unified Modeling Language) general conceptual model, including all the entities involved in the IoT context, their relationships and the Non Functional (NF) properties related to security and data quality
- The definition of a high-level reference architecture, named NOS, representing the modules and the interfaces that compose the new IoT middleware
- The development of a real working prototype of NOS, by means of the state of the art technologies emerging in the IoT scenario.

The three aspects presented above will be detailed in Paragraphs 3.2, 3.3 and 3.4, respectively. Figure 3.1 sketches the steps towards the development of a general-purpose IoT middleware.

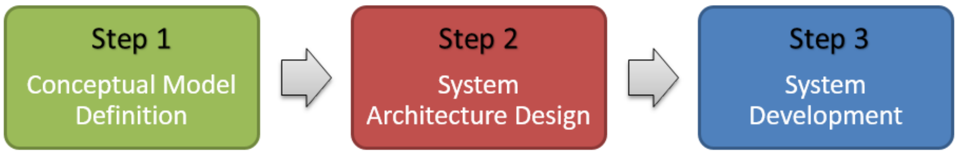


FIGURE 3.1: Steps towards IoT middleware development

3.2 Non functional property model

As a starting point for the development of IoT privacy-aware solutions, exploiting data with a well-defined quality, a UML conceptual model of a general-purpose IoT platform is defined hereby and published in [181].

It aims to support the other phases towards the development of an IoT secure middleware, which are: (i) the architecure design, described in Paragraph 3.3 and (ii) the real prototype, presented in Paragraph 3.4. To this end, it is important to identify all the entities involved and consider the relationships among them. In particular, the following elements have been identified:

- The *nodes*, intended as data sources that provide information from the environment into which they are placed
- The *services*, made available by the IoT platform integrating the data received from the nodes
- The *users*, interested in exploiting the services provided by the IoT platform.

Due to the heterogeneity of such entities and of the possible IoT domains, the model has to be suitable for general-purpose contexts with a variable number of entities, thus not limiting its application to single case studies. Along with this feature, the model has also to deal with the following main NF properties:

- A *Quality of Protection (QoP)* property, embracing the requirements of security, such as data integrity, confidentiality, privacy, and authentication (Figure 3.2)
- A *Data Quality (DQ)* property, regarding the accuracy, the precision, the timeliness and the completeness of the information handled by the IoT platform itself (Figure 3.3).

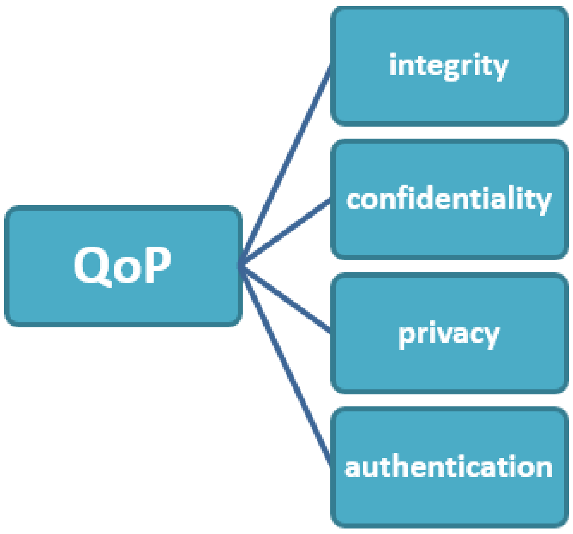


FIGURE 3.2: Quality of Protection (QoP) requirements

Such requirements arise for two reasons. On the one hand, the IoT system may manage both environmental and personal information; thus preserving a user's private life becomes a pivotal goal. The IoT platform

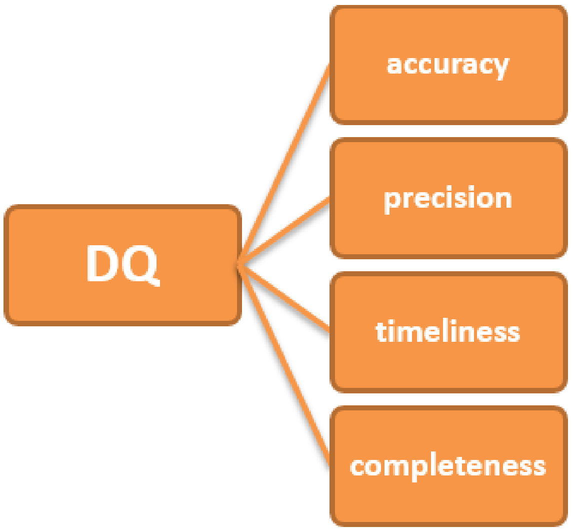


FIGURE 3.3: Data Quality (DQ) requirements

should guarantee that only authorized devices or users are allowed to access the different types of information. The risk of violation is further increased by the use of wireless communications, which may lead to malicious attacks such as eavesdropping and masking. On the other hand, there is the need to evaluate the quality of the data coming from different, and sometimes unknown, sources, for which there is no *a priori* knowledge on the accuracy or on the rate of updates. Hence, the services made available by the IoT platform should provide data with well-defined quality levels. In fact, in many scenarios, errors or missing values might have a critical impact on actions or decisions. An efficient and robust IoT system should allow the users to be aware of the reliability of the accessed information. This is an innovative aspect since, as pointed out in [20], current available services provide the same information to each requesting user, often without considering his/her requirements and without specifying the level of QoP and DQ of the data themselves. Indeed, the services must be customized according to users' preferences and habits.

A further specification is due with regards to QoP properties. In particular, integrity represents the maintenance of the consistency and the trustworthiness of the data from the time when they are acquired to the time when they are received by the final user; in more detail, data must not be changed during their transmissions and proper countermeasures have to be taken to ensure that data are not altered by unauthorized users or devices (e.g., by means of verification systems). In this context, authentication is the act of verifying a claim of identity. Lots of mechanisms are available for this scope, with different degrees of strength, such as the username/password based systems, identification documents, biometrics, and so on. Hence, confidentiality involves the measures undertaken for preventing information from reaching unauthorized users or devices; note that this not implies that information are modified in any way (as for integrity checks). Data encryption is a common method of ensuring confidentiality. Finally, privacy is formally the right of a person to keep his/her data anonymous and to determine whether, when, how, and to whom, his/her personal or organizational information are to be revealed. Therefore, privacy is referred, for example, to the protection of personally identifiable information (e.g., name, address, telephone number), which can be used to identify a person as an individual.

Bearing in mind the entities and the NF properties introduced above, in the following paragraphs the components of the conceptual model are detailed by means of UML diagrams.

3.2.1 Node component

The different adopted technologies involved in the acquisition of information from the environment, such as WSN, RFID, nanotechnologies, actuators and so on, are represented, in the UML diagram of Figure 3.4, by class *Node*.

Such a class is extended by the sub-classes representing the individuated technologies (the note *incomplete* means that there are *other types of such nodes which have not been mentioned*). Each instance of class *Node* is characterized by a pair function-role, identified by *NodeRole* and *NodeFunction* classes. *NodeRole* [142] is a concept strictly related to the privacy issue. Therefore, three classes extending *NodeRole* are introduced:

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

NodeFunction represents the tasks performed by a node in the network in which it operates; the UML model does not specify any sub-class for *NodeFunction*, since the functions depend on the specific application context in which the system is employed (e.g., shopping retail, health-care, university, factories, building automation).

Class *NodeAction* is associated with the pair function-role, which specifies the set of actions that can be undertaken by the node itself. The identified actions are: *Processing*, when a node performs some elaborations on data; *Trasmission* or *Reception*, when the node sends or receives data to/from another node. An action is executed under *nPurpose* that specifies the reason under which it is possible to handle the data (i.e., marketing purpose, research purpose, health purpose). *NodeAction* is also associated with one or more *nObligation*s, in order to model the fact that the execution of a set of actions is guaranteed by the processor and/or the controller at the end of the elaboration activities (e.g., whenever an inconsistency or a privacy violation is found, some countermeasures, such as generation of an error/alert message, have to be taken). In order to guarantee integrity, confidentiality and non repudiation, two kinds of keys, named *NodeSignatureKey* and *NodeActionKey* are associated with *NodeAction*.

Communications among nodes happen by exchanging instances of class *NodeMessage*. A message is composed of several heterogeneous types of

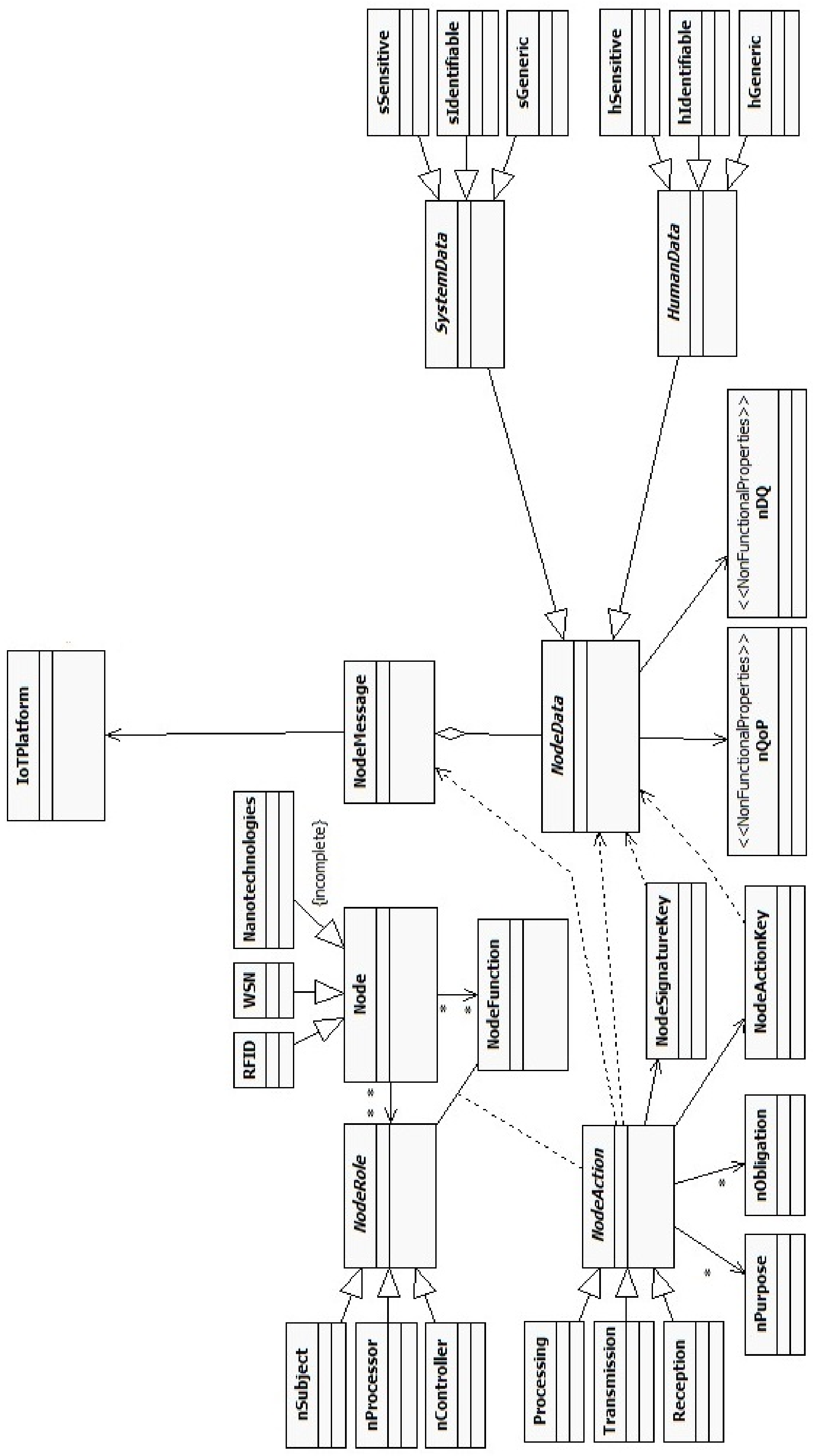


FIGURE 3.4: Node component - Conceptual Model

data (e.g., numbers, text, multimedia), which can contain different information depending on the generating source; therefore, 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 sensor nodes or by locator tags related to the activities of users or devices)
- *HumanData*, which includes the data produced by users themselves, for example by means of a social network.

Both system and human data are classified into three different categories, represented in the UML diagram as extensions of *SystemData* and *HumanData* classes:

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

Each instance of *SystemData* or *HumanData* is associated with *nQoP* and *nDQ* information, which represent the NF properties regarding the level of QoP and DQ associated with the data.

3.2.2 User component

Besides nodes, another fundamental actor is represented by class *User*, shown in the UML diagram of Figure 3.5.

Such a class concerns all humans that could interact with the IoT system, for example by means of their personal devices (e.g., smartphones, NFC, tablet) or interested applications. 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, a personal *Profile* is required. *Profile* includes the following information:

- An aggregation of *ConsentData*, which represents the acceptance of the agreement established with the service provider (i.e., the consent to handle user personal data for specific purposes and under proper obligations)
- An aggregation of *PreferenceOnService*, used to customize the services on the basis of user requirements
- The keys exploited to address QoP issues, distinguished in *UserSignatureKey* and *UserActionKey*.

Profile also concerns *UserData*, which can be further classified as:

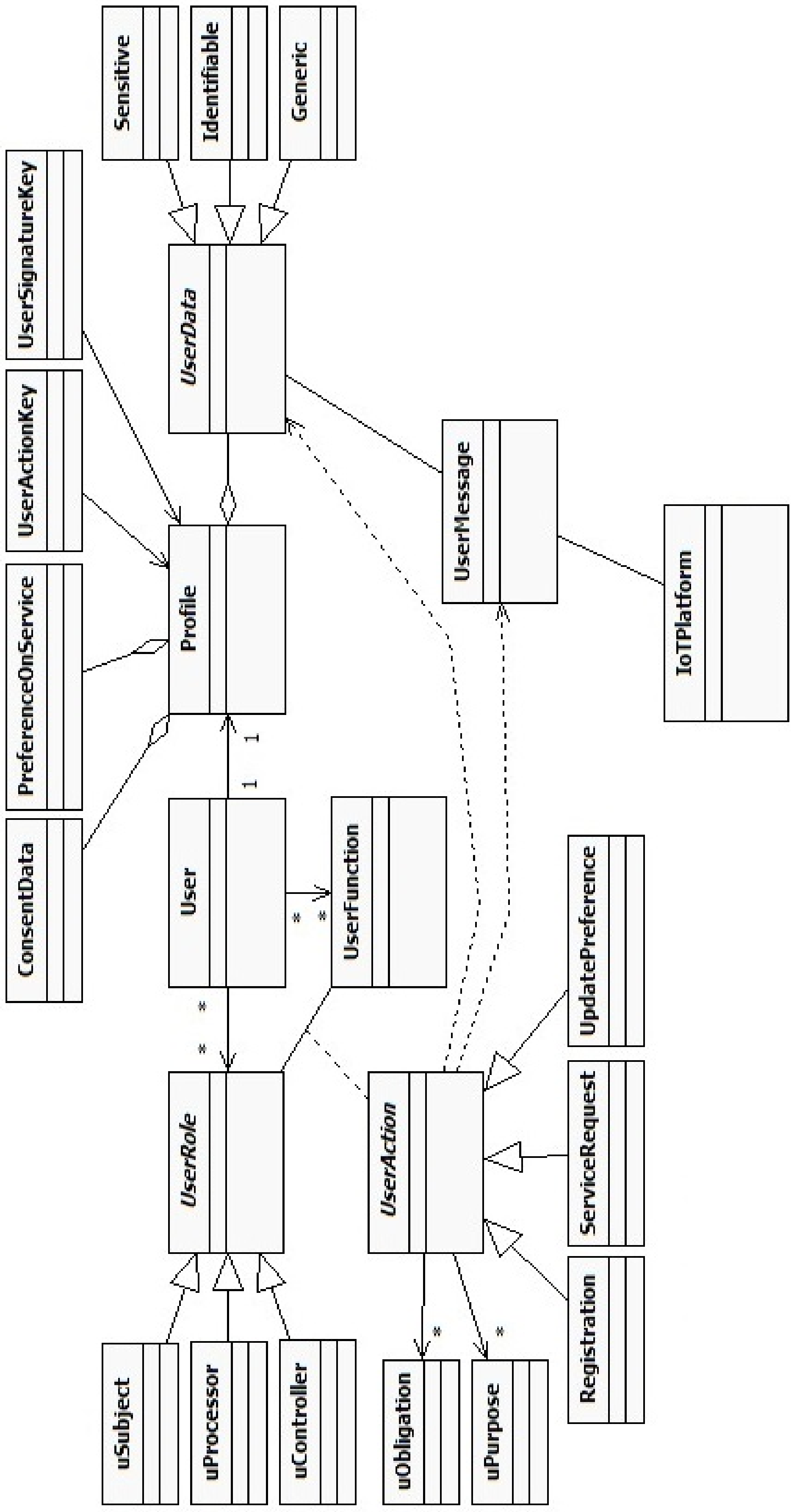


FIGURE 3.5: User component - Conceptual Model

- *Identifiable*, including data that refers to the user identity (i.e., first and last name)
- *Sensitive*, containing information related to user private life and habits, such as health conditions, food intolerances, religious beliefs and so on
- *Generic*, regarding general information not belonging to the two previous classes.

Each user is associated with a pair function-role. Class *UserRole* is extended in a similar way as class *NodeRole*, because a user could represent *uSubject*, *uProcessor* or *uController* of the transmitted or provided data. In detail:

- *uSubject* is the owner of the data
- *uProcessor* is the subject who handles *uSubject* data
- *uController* represents the subject who verifies the compliance with the defined policies, related to the user profile.

As regards *UserFunction*, it strictly depends on the application context. Each pair function-role is also associated with the respective *UserAction*, which includes a set of actions. In fact, a user, before interacting with the IoT system, has first to register himself/herself (*Registration* class). *ServiceRequest* represents the user's requests in terms of the information of interest. *UpdatePreference* represents the capability of the users to change at any time their preferences, expressed in their own profile. As for nodes, specific *uPurpose* and several *uObligation*s are associated with each action; for example, users, before requesting services, must register themselves. Furthermore, the communications among users and IoT platform occur by means of packets, which are instances of *UserMessage* class. Note that, in a typical IoT scenario, the number of nodes and the users varies over time.

3.2.3 IoT Platform

IoTPlatform, presented in the UML diagram of Figure 3.6, has a crucial role and performs different tasks described by class *IoTPlatformAction*. Like the nodes and the users, *IoTPlatform* also plays different roles, represented by *IoTPlatformRole* class, and functions, represented by *IoTPlatformFunction* class, in relation to the current action and the application domain. *IoTPlatformRole* is extended by the *iSubject*, *iProcessor* and *iController* classes, which represent, respectively, the owner of the data, the processor that executes some actions on data, and the verifier of the performed actions. As far as *IoTPlatformAction* is concerned, the main *IoTPlatform* tasks are:

- *UserProfileDefinition*, which models the acquisition of the user data and preferences, required for the execution of the best suited services
- *ConsentAcquisition*, which represents both the user acceptance of the agreement with the requested service, and the user data management, according to the established policies

- *AccessControl*, which represents the execution of the access control operations towards the users registered within the IoT system. Note that such an action allows the access to the system to be restricted only to authorized and pre-registered users
- *PolicyDefinition*, which represents the definition of policies in order to ensure user privacy (in particular, anonymity), and, in general, the satisfaction of security and data quality requirements
- *PolicyEnforcement*, which is required in order to force the compliance with the defined policies, under which the user has given the consent to handle his/her data.

For *IoTPlatform*, several *iPurpose*s and *iObligation*s may be specified to guarantee that the proper actions are performed only when particular conditions are verified. In particular, *IoTPlatform* deals with encryption and decryption keys; in fact, when a node (i.e., a device) or a user enters the system, *IoTPlatform* has to provide the credentials and the keys to allow future secured interactions with the offered services. *IoTPlatform* owns both *IoTPlatformSignatureKey* and *IoTPlatformActionKey* required to perform the access control, authentication and privacy control operations, and the signature and action keys belonging to the nodes and the users, as described in Paragraph 3.2.1 and 3.2.2, respectively.

On the basis of the action to be performed, nodes and users may exploit a defined encryption mechanism on the transmitted data. When a new node or user begins to interact with the IoT system, *IoTPlatform* sends the proper signature and action keys that will be used for securing the communications. Note that the transmission of such keys is performed in a secure way, for example by means of HTTPS protocol. Each user/node has a unique signature key, which represents an access credential; while the action key is directly related to the pair function-role, currently played by the user/node and it is used to exchange message and to encrypt sensitive data referring to the instances of *sIdentifiable*/*nIdentifiable* data. 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, firstly, if the user is authorized to get such data and, secondly, which are the user preferences in relation to the NF properties of the requested services. The defined solution supports any kind of encryption technique and any key distribution mechanism.

3.2.4 Service component

After processing the data received by nodes, *IoTPlatform* may provide atomic or composite services. The former ones simply provide users with the access to one source; while the latter ones integrate information gathered by multiple sources and allow users to access data by using advanced queries. *Service* is one of the core class of the model, since all the IoT system activities turn around the request and provision of services. In fact, the users give information related to their identity, life style, interests and preferences in general, under well-defined QoP and DQ policies, in order

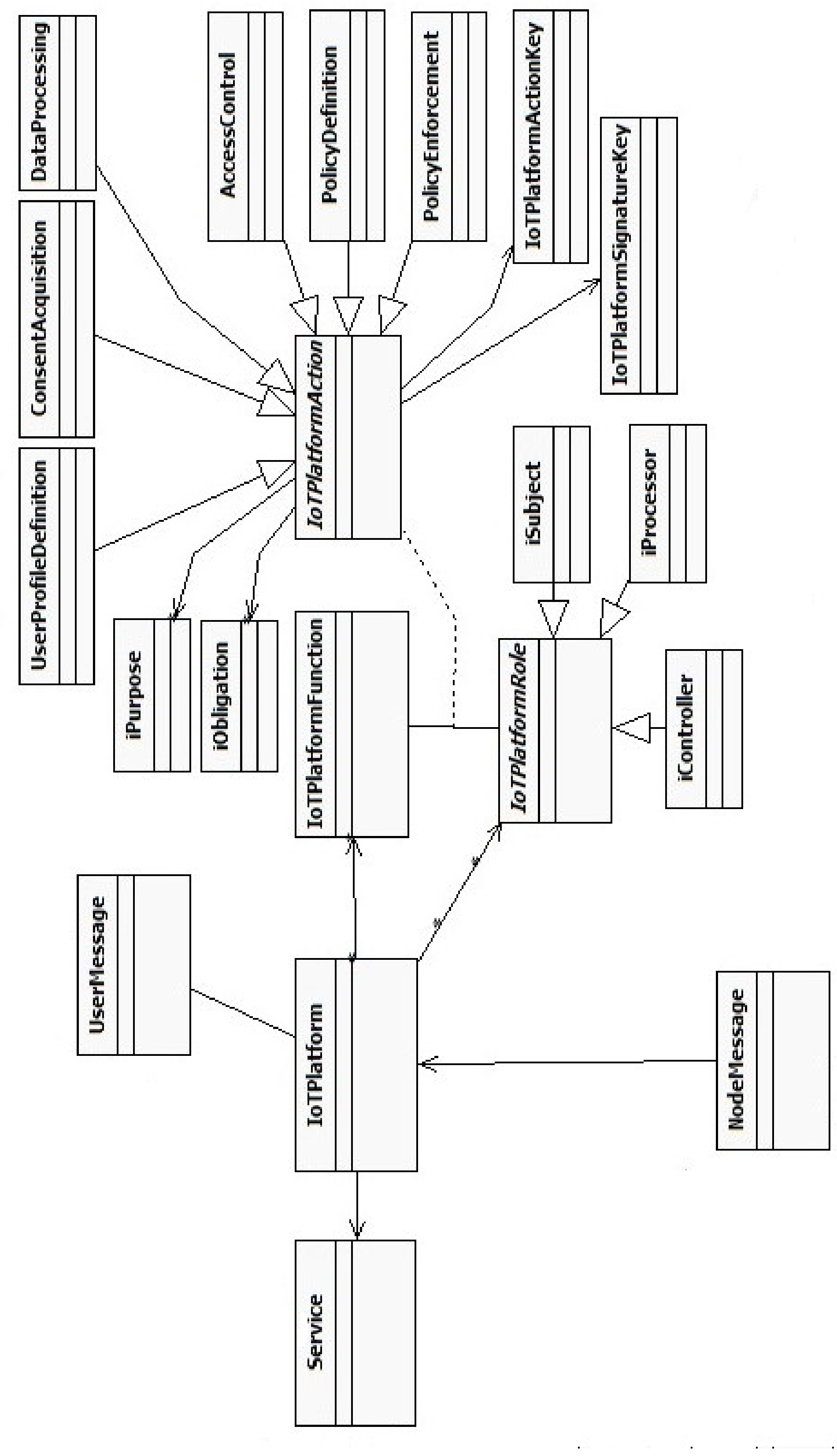


FIGURE 3.6: IoT Platform - Conceptual Model

to obtain customized services, accordingly. Such services have to guarantee the required NF properties, represented in the UML diagram of Figure 3.7.

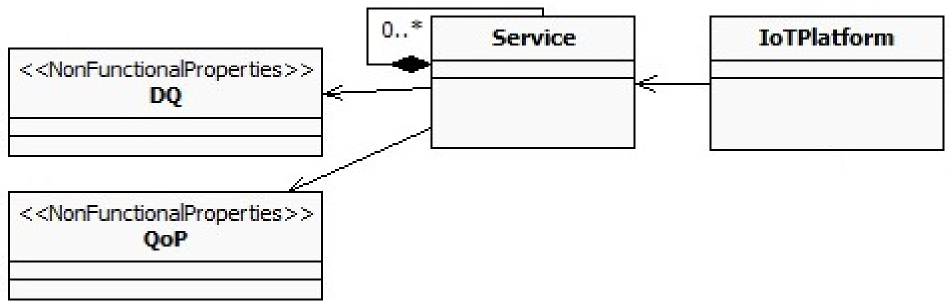


FIGURE 3.7: Service component - Conceptual Model

An overall representation of the UML class diagram, including all the entities involved in the model, is given in Figure 3.9. Note that *IoTPlatform* is directly connected to: (i) *Service* class; (ii) *NodeMessage* on the node side; (iii) *UserMessage* on the user side. Such a solution points out the middleware role of *IoTPlatform*, as sketched in Figure 3.8.

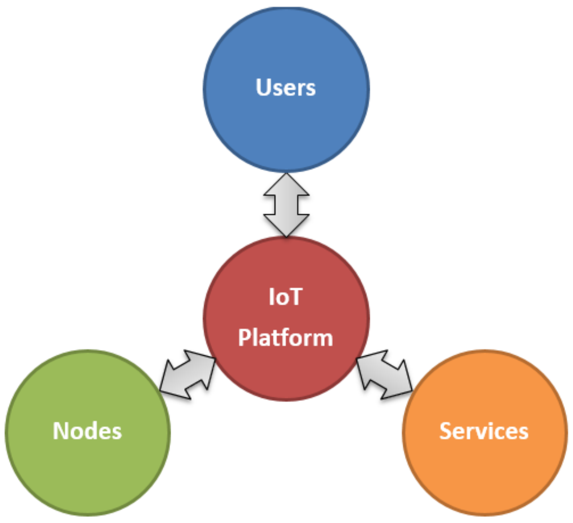


FIGURE 3.8: IoT Platform interactions

A detalied case study regarding the application of the aforementioned model is provided in Appendix A.

3.3 Networked Smart Object architecture

From the general conceptual model including the entities, the actions and the QoP and DQ requirements involved in a general-purpose IoT application domain, a reference system architecture has been derived [182] and is presented in this paragraph. The outcomes of this stage have been published in [169].

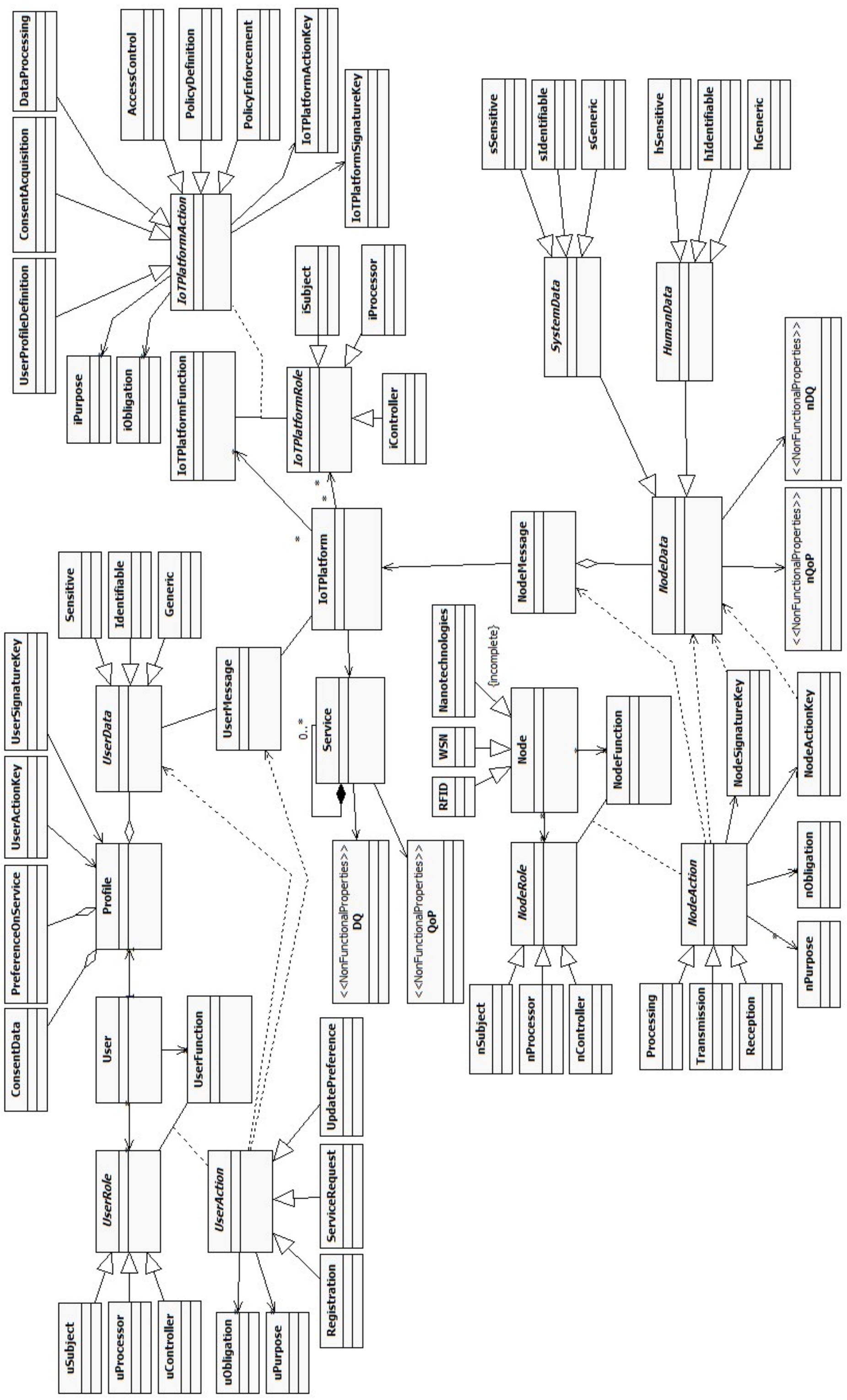


FIGURE 3.9: IoT general conceptual model - UML class diagram

3.3.1 High-level IoT reference architecture

Data are provided from the environment in which the IoT system is deployed by a set of heterogeneous technologies that are referred to with the term *E-Nodes*. Such technologies may require different communication systems (e.g., NFC, UWB, IEEE802.15.x, Bluetooth) and include nodes that generate data automatically (i.e., WSN, RFID, smart tags and so on), referred to *System Data*, and nodes providing data by user devices (i.e., social networks), referred to *Human Data*.

In order to handle such a heterogeneous and huge amount of information, the NOS layer is introduced, which corresponds to *IoTPlatform* defined in Paragraph 3.2. NOSs are conceived as computationally powerful smart devices, without strict constraints in terms of resources and computational capabilities, connected to create a distributed processing and storage layer, able to process the data acquired from large-scale IoT deployments close to the actual data sources. NOSs act as a middleware for collecting the data generated by nearby IoT devices (i.e., *E-Nodes*) and processing them. Such a middleware includes provisionings for users and applications to dynamically specify the levels of QoP and DQ suitable for their own purposes. The distributed architecture automates the deployment of adequate filters to ensure that only qualified data is being used by the actual services. This represents a clear innovation over conventional one-size-fits-all approaches, which provide the same information to all consumers, often without considering their requirements in terms of QoP and DQ.

NOSs are structured as the following three internal phases:

- *Analysis*
- *Data Normalization*
- *Integration*.

In more detail, NOSs acquire the raw data provided by *E-Nodes* and process them according to *Analysis* and *Data Normalization* phases shown in Figure 3.10, in order to provide in output a normalized representation of the gathered information, called *IoT Data*. For each incoming data, the following information can be extracted:

- *Data source*, that, due to its possible heterogeneity, is classified as *System Data* and *Human Data*, as previously described; note that NOSs is able to derive the type of node
- *Data communication mode*, that is the way in which data are collected (e.g., discrete or streaming communication)
- *Data schema*, that consists of the type (e.g., text, numbers, multimedia), the format and the metric of the data attributes (if available and if needed - on the basis of the application context)
- *QoP metadata*, that specifies the nature of data (sensitive or not) and which QoP properties are guaranteed. In particular, QoP metadata provide details about confidentiality, authentication, integrity and privacy

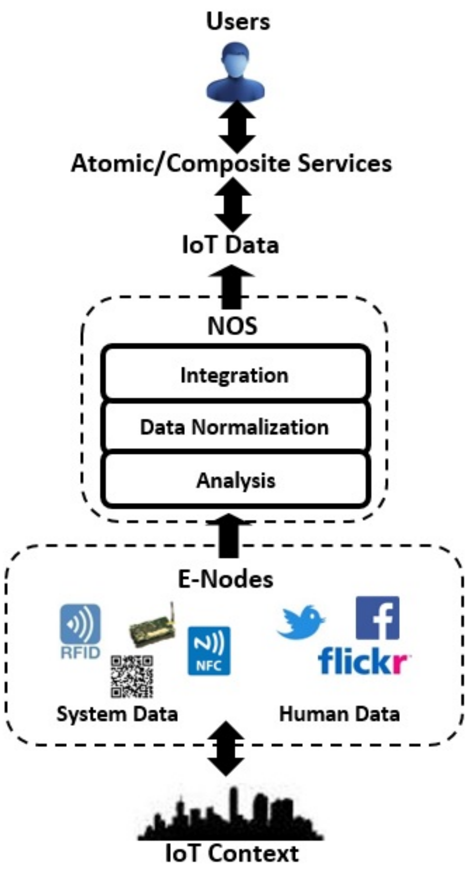


FIGURE 3.10: Reference system architecture

- *DQ metadata*, that provides information about the DQ level of the data and the related sources. Data trustworthiness can be measured in terms of timeliness (that depends on the data volatility and currency), data completeness, data accuracy and precision
- *Timestamp*, that describes when the data have been received by NOS.

The *Analysis* phase is responsible for the assessment of QoP and DQ levels, which will be detailed in Paragraph 4.3, along with the function of the *Event Monitor* component. The *Data Normalization* phase represents the information obtained from the *Analysis* phase according to a well-specified syntax and includes also a semantic description of the data content. In particular, the data are annotated with a set of metadata, including a score in the range [0:1] for each QoP and DQ requirement, as shown in Figure 3.11. Note that the choice to provide a score for each QoP and DQ requirement makes NOSs flexible for smart integration in different application scenarios. In fact, a solution in which only general security and quality scores are considered, without details about all the different QoP and DQ dimensions, would not allow the system to precisely identify weaknesses and strengths of the different input data sources. For example, there may be application scenarios in which it is necessary to use data that have a high level of accuracy and timeliness, but there is no interest to satisfy privacy requirements.

After *Data Normalization* phase, the processed information may be integrated according to the application needs within a specific IoT context. In fact, the *Integration* phase permits data coming from different sources to be merged according to the services provided by the IoT platform. Such a phase takes into account the QoP and DQ levels offered by *Data Normalization* phase, in order to choose the data that better satisfy user needs in the different application scenarios. In more detail, the *Integration* phase is aware of the characteristics of the sources and related data, thus the associated metadata could be used for two different purposes: knowledge level and/or operating level. The former refers to the possibility of providing the integrated information annotated with the QoP and DQ scores in order to let the users be aware of data trustworthiness. The latter refers to some integration procedures, which might be affected by the data provided by different levels. For example, QoP and DQ levels can be used for the selection of sources (if there are alternatives) to be integrated. In particular, if the application domain aims at providing a service characterized by error-free data and high confidentiality scores, the *Integration* phase should select sources that are able to satisfy these requirements. Summarizing, the final goal of NOS layer is enabling access to *IoT Data* through a properly engineered lightweight data discovery mechanism. Note that NOSs may be connected to IP-based networks (i.e., Internet, Intranet), allowing nodes and end-users to access the offered services, provided at the end of *Integration* phase. A service may be:

- *Atomic*, if the users are enabled to access to the information belonging to one source
- *Composite*, if data coming from multiple sources are integrated.

Such a set of services, made available by NOSs, may be directly and dynamically configured by a network administrator and may be orchestrated by a remote server through standard Web services approaches. Since NOSs include self-organizing features, they can be deployed where and when needed. In fact, NOSs, through their interfaces with both enterprise platforms and IoT enabling technologies, can be used to enrich software platforms, making them able to interact in a standardised way with the physical world.

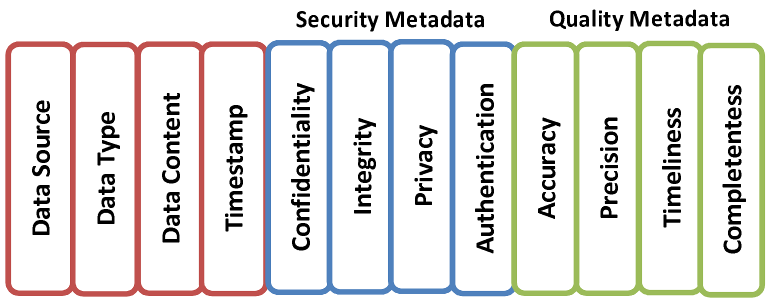


FIGURE 3.11: IoT annotated data

3.3.2 Modular IoT middleware

The reference architecture presented in Paragraph 3.3.1 is just a high-level representation of NOS structure and points out its middleware position between *E-Nodes*, services, and users. The workflow of information is bottom-up, from the data sources to the processing phases within NOS and, then, to the service provision, as shown in Figure 3.12. More in depth, each NOS exposes interfaces both towards data sources and user service provision (highlighted with green boxes in Figure 3.12). In the remainder of this paragraph, through a bottom-up analysis, the specific modules composing NOS are detailed. The description includes:

- The *southbound interfaces*, which are used to interact with the data sources
- The *processing units*, which are in charge of performing QoP and DQ evaluation
- The *northbound interfaces*, which include the mechanism used for sharing the information through services.

The next discussion will show the modular nature of NOS, which makes it flexible and extensible, allowing the easily addition of new modules or removing/updating the existing ones. In general, NOS would act as a gateway with built-in processing capabilities, able to manage a dynamic number of data sources. Multiple NOSs may co-exist, each of them serving a subset of the devices belonging to the environment.

Southbound interfaces

NOSs are able to collect data transmitted by different kinds of devices. The system has been designed to support both registered sources as well as non registered (i.e., anonymous) ones, each of which characterized by different communication technologies and providing different QoP and DQ levels for the gathered data. NOSs know, for all the sources, the kinds of the data provided. As regards the registered nodes, NOSs provide a service for source registration by means of HTTP protocol; the related information are stored in *Sources* data structure and, in particular, registered sources are associated with:

- An *identifier*
- Optionally, a *geographical position*
- Optionally, an *encryption scheme*, including the proper keys for interactions with NOSs.

The HTTP communication protocol is also used among NOSs and the sources for the data transmission, thus also including the non registered ones. As regards them, NOSs can only keep track by assigning pseudo-random identifiers after the first interaction, but no encryption scheme can be shared.

Since received data are highly heterogeneous, each NOS initially stores them in *Raw Data* collection, and, periodically, processes them, in a batch way, according to the two-phase structure shown in Figure 3.12, which includes *Data Normalization* and *Analysis*. These two phases have been investigated in Paragraph 3.3 and have the final goal to obtain an uniform representation of information, enriched with relevant metadata.

Initially, the data stored in *Raw Data* are converted according to the format, specified in Figure 3.11 by *Data Normalization* module, which stores them in *Normalized Data* collection. This represents a sort of pre-processing phase in which the unnecessary information are removed, so that later processing stages can access the information in a unified way. At this stage, QoP and DQ metadata fields are still empty. Then, a second module, consisting of a set of *Analyzers* (i.e., *Security Analyzer* and *Quality Analyzer*), periodically extracts the normalized data from the storage unit *Normalized Data* and elaborates them, computing relevant QoP and DQ indicators (detailed in Paragraph 4.3). Within such a step, as shown in Figure 3.11, data are annotated with a set of metadata in the form of a score for each QoP and DQ property. The processed data are used for providing services to the target users, as described in Paragraph 3.3.2.

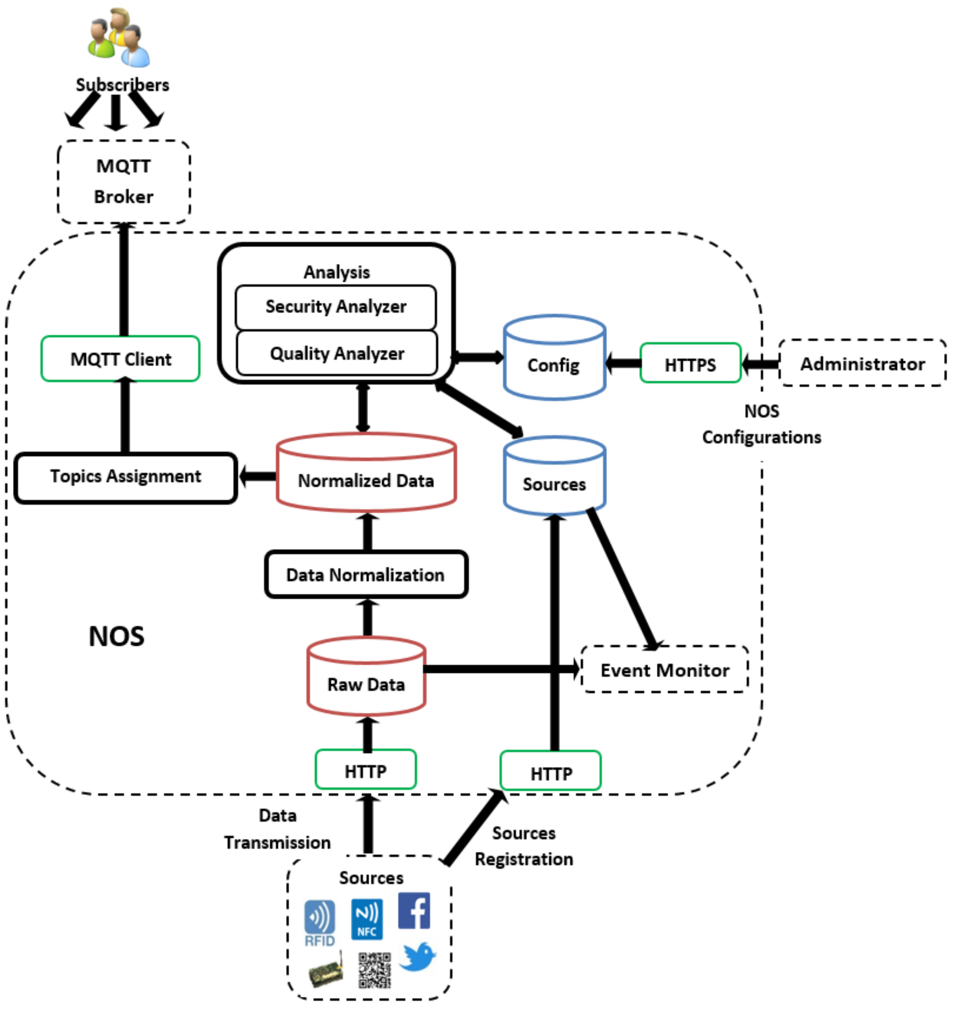


FIGURE 3.12: NOS architecture

Northbound interfaces

While NOS southbound interfaces are based on the HTTP protocol, the northbound interface is based on the Message Queue Telemetry Transport (MQTT) protocol [87]. MQTT is a lightweight publish/subscribe protocol specifically designed for resource constrained devices. In the IoT context, it is widely used to enable communication among devices using a publish/subscribe messaging approach. In particular, it aims at making the processed data available for interested applications and/or users. In fact, the system allows the subscription of both users and external applications, which authenticate to NOS and may then make requests to the services made available. In case of application registration, multiple users may register to such an application, instead of registering to NOS.

An MQTT client, like that contained in NOSs, exchanges messages with an MQTT broker by means of publications and subscriptions to topics. Such a mechanism is adopted to support interactions among services and IoT devices. NOSs include a module, named *Topics Assignment*, in charge of assigning data items to the corresponding topic and to publish them on an MQTT broker, as depicted in Figure 3.12. The mapping of data to a specific topic depends on the application domain and may require the usage of an ontology able to represent the semantic of the managed resources. In general, topics are multi-level structures separated by a forward slash, similar to a directory structure. An example of a topic for publishing temperature information of a sensor with identifier *sensorId* could be `sensor/temperature/sensorId`. Note that subscribers may register for specific topics at runtime and NOSs provide a mechanism for dynamic subscription and unsubscription to topics. The publish/subscribe mechanism adopted by NOSs will be further detailed in Paragraph 4.5.

Configuration

So far the NOS system has been introduced as a set of interfaces and running modules without specifying how such modules behave. Actually, the actions undertaken by NOSs are regulated by well-defined rules; among them, the most important concerns are:

- How to assess the QoP and DQ levels of the incoming information and, in particular, which features about data and/or sources to consider in performing such an evaluation
- The policies to be applied to control the access to data
- The topic assignment strategy.

The whole set of rules is stored in a proper format in the storage unit, named *Config*, also represented in Figure 3.12. *Config* contains all the configuration required for the correct management of NOS system. Rules in the *Config* store can be dynamically configured at run time by system administrators connecting remotely to NOS over a secure connection (e.g., HTTPS, SSL) without the need to re-start NOS services. The usage of a secure communication protocol is required in this case, as the policy adopted by NOS for processing *IoT data* has to be protected against external attacks.

As regards access control policies and topic assignment, they will be discussed in Paragraphs 4.4 and 4.5, respectively. Instead, concerning QoP and DQ assessment, *Security Analyzer* and *Quality Analyzer* practically query *Config* store to retrieve a list of the operations they are intended to carry out. However, it is worth remarking that the assignment of QoP and DQ scores provides a handle for letting the users filter the data processed by NOS, according to their personal preferences. In fact, as pointed out in Paragraph 3.3, the choice to provide an evaluation for each QoP and DQ dimension makes an NOS-based approach extremely flexible and able to adapt to very different application scenario requirements. For example, there exist application scenarios (e.g., factory floor automation) in which only data with a high level of privacy and confidentiality can be used, but there is no need to satisfy integrity requirements. Another application domain may aim to provide a service characterized by error-free data and high confidentiality scores; therefore the data to be selected are those provided by sources able to satisfy these requirements. In many situations, in fact, no description, either about the sources or the acquired data, may be available *a priori*; at the moment this requires labour-intensive search and selection of which data sources to use. The NOS approach automates such a task, leaving the system administrators to define scoring policies and to the service provider to specify the requirements on the data to be used. The ability to support automatic reasoning about QoP and DQ is what makes the NOS system able to deal properly with the scale and heterogeneity of IoT contexts. Note that the QoP and DQ assessment rules will be discussed in Paragraph 4.3.

3.4 Prototype

NOS system, presented in Paragraph 3.3, has been implemented as a prototypical service middleware platform, able to manage a large amount of data from heterogeneous devices with lightweight modules and interfaces working in a non-blocking manner to perform data analysis, discovery, and query. In a real scenario, one or more NOSs can be deployed in a distributed manner. This represents an important step beyond conventional *ad-hoc* centralized IoT solutions. Note that, from an analysis of NOS functionalities, there is no need for a peer-to-peer management of NOSs. In fact, NOSs are able to:

- Independently handle the connected data sources, without the need to inform other NOSs of their active and past interactions
- Be independently re-configured by IoT system administrators through *Config* interface
- Independently assign topics and publish data on the basis of the defined rules.

The existing IoT deployments are often barely reconfigurable [127], since they are conceived for very specific applications, based on a vertical silo-based approach. NOS middleware supports dynamic reconfiguration and can be remote orchestrated through Internet/Intranet protocols, which are based on open standards (see Paragraphs 3.3.2 and 3.3.2).

Besides being compliant to the architecture presented in Paragraph 3.3, the NOS implementation is based on the following components/technologies/libraries:

- *Node.JS* platform [146] for NOS core functionalities
- *MongoDB* [132] for the data management part
- *Mosquitto* [134] as open-source MQTT broker.

The code is accessible as open source under a permissive license at `https://bitbucket.org/alessandrarizzardi/nos.git`. NOS modules interact among themselves through *RESTful* services, allowing the addition of new modules or modification/removal of the existing ones at runtime, since they are able to work in a parallel manner. The non-relational nature of *MongoDB* also allows the data model to evolve dynamically over the time. Consequently, such an implementation is independent both from the data model and from the application domain. Data and rules handled by NOS are formatted in JSON (JavaScript Object Notation) language, which was chosen due to its flexibility as lightweight data-interchange format.

In more detail, NOS modules can be divided into: node interfaces, processing modules, and service interfaces. The node interfaces include a specific *REST* endpoint for handling source registration and acquire data from IoT devices. As described in Paragraph 3.3.2, these correspond to the southbound NOS interfaces. The following endpoints are exposed:

- **POST data/**, used for handling transmission of data from the nodes to NOS. Messages shall be formatted as valid JSON nodes
- **POST registration/**, used by nodes for registering to NOS. Messages shall be formatted as valid JSON nodes. The following fields are mandatory: `NodeId, NodeType, CommunicationMode`. Optional field: `EncryptionScheme`. The response includes node credentials.

The NOS system also includes additional modules, namely *Data Normalization* and *Analyzers*. Such processing modules are daemons that start along with NOS and periodically extract data from *Raw Data* or *Normalized Data* storage units. Since they are internal to NOS, no specific APIs are provided.

As far as northbound interfaces are concerned, the following endpoint is exposed:

- **mqtt.Client#publish(topic,payload,[options])**, used by NOS for publishing processed data to the MQTT broker. Mandatory parameters: `topic` (channel on which the message is to be published), `payload` (message to be published). Optional parameters (see Paragraph 4.5): `[options]` (QoS, retain flag, callback). The address of the broker is specified at client initialization time.

Notice that the broker is a module that runs separately from the NOS system, therefore it acts as an intermediary among NOS and the subscribers.

TABLE 3.1: NOS storage collections

Collection	Type
Raw Data	Non-Persistent
Normalized Data	Non-Persistent
Sources	Persistent
Config	Persistent

Given the overall system architecture, it is worth remarking that NOS does not require data persistence for IoT-generated data. Rather, data is temporarily cached on NOS while being processed and before being submitted to the MQTT broker. Accordingly, NOS uses the in-memory capability of *MongoDB* for two of its databases, namely *Raw Data* and *Normalized Data*, whereas the databases *Config* and *Sources* must be persistent. Table 3.1 summarizes the nature of the storage collections used. A routine runs on NOS in order to remove from *Raw Data* the data already normalized and from *Normalized Data* the data already published. Such an approach greatly improves NOS performance, significantly lowering the effort for query and read/write operations.

3.5 Security threats

Once the NOS system has been presented, a more detalied analysis can be done with regards to the security threats that may occur. As introduced in Paragraph 1.2, multiple kinds of attack may be put in place in an IoT environment. They mainly include attacks against the IoT devices, attacks against the communications among IoT devices and NOSs, and attacks against NOSs themselves.

Attacks against the IoT devices concern physical disruption, tampering or information theft (e.g., certificates or encryption keys). To cope with such issues, an authentication system is necessary, which would allow NOSs to recognize the authorized devices from the misbehaving ones.

Attacks against the communications can be addressed by means of encryption mechanisms. Note that, such threats can be represented by any kind of DoS, routing or man-in-the-middle (e.g., spoofing, sniffing) attack. In this case, not only can the data content be compromised, but the identities of the IoT devices are also at risk. Moreover, network resources are wasted, thus causing serious damage to the supply of services.

In the following chapters, some solutions will be provided in order to, at least partially, deal with the aforementioned threats.

Finally, in this thesis, attacks against NOSs are not considered, since NOSs are assumed to be trustworthy. If this is not true, the information they managed and data received by the IoT devices would be insecure; furthermore, the tasks performed by NOSs would also be unreliable, thus discouraging the use of the services provided by NOSs by the users. In the case in which a single NOS is compromised, actions could be undertaken in order to: (i) recognize that a NOS is under attack; (ii) isolate the compromised NOS and restore the communications with the IoT devices that it managed. Such aspects are left as a future extension.

Chapter 4

Security Solutions

4.1 Introduction

NOS architecture and components were described in Chapter 3 along with their prototypical implementation. In particular, the following aspects were specified:

- The middleware role performed by NOS in the IoT context
- The presence of two other entities interacting with NOS, namely the nodes (i.e., data sources) and the users
- NOS main functionalities (i.e., processing units), which are data *Analysis*, *Data Normalization* and service *Integration*
- NOS southbound interfaces towards the nodes and northbound interfaces towards the users.

NOS is conceived as a modular architecture, in order to ease the integration with new modules. Moreover, NOS has to operate in the dynamic and heterogeneous IoT environment, thus requiring the adoption of strict security measures, in particular to guarantee wide acceptance by people and industries of IoT service paradigm. In order to improve NOS behavior in terms of security features (e.g., resilience towards malicious attacks, secure data assessment, policy enforcement), NOS has naturally evolved with the following functionalities:

- Use of a key distribution and management system, in order to improve the robustness of the adopted encryption mechanisms for information exchanges among NOS and nodes/users
- Introduction of new algorithms for the automatic reasoning about the security (QoP) and the quality (DQ) levels, associated with the data received by IoT devices, in order to make users aware of the reliability of the services made available by the IoT system
- Adoption of policy enforcement mechanisms able to handle violation attempts
- Integration of the aforementioned policy enforcement mechanisms with a lightweight authenticated publish/subscribe system for securing the data provided to users
- Establishment of a synchronization system for ensuring the correct application of policies across different application domains, since an IoT system may include various stakeholders.

The NOS functionalities just presented are summarized in Figure 4.1 and will be detailed in the following paragraphs.

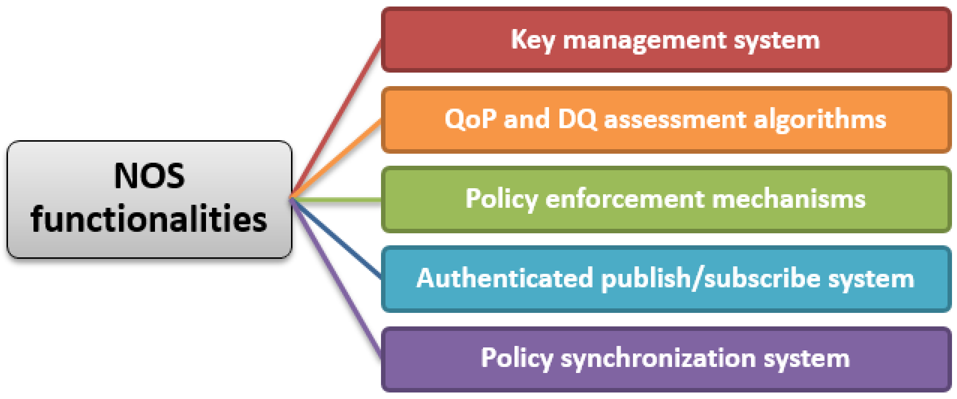

NOS functionalities
Key management system
QoP and DQ assessment algorithms
Policy enforcement mechanisms
Authenticated publish/subscribe system
Policy synchronization system

FIGURE 4.1: NOS functionalities

4.2 Key distribution and management

A secure system includes the use of keys for encrypting the transmitted information and/or authenticating an entity with respect to another. Starting from this assumption, within each secure system, a key management system should be integrated. In this way, a clever control over key distribution, replacement and revocation can be put in place, improving the resilience of the system and of the involved entities towards malicious attacks either on transmitted data or on sensitive information.

As pointed out in the state of the art in Chapter 2, the key management in IoT is not a mature field at all, even though many solutions have been proposed for the WSN context. In this thesis, two popular and robust key management systems, named Dini et al. [44] and Di Pietro et al. [162], conceived for WSN, have been integrated into the NOS architecture. The results obtained are published in [183].

As a first insight, the algorithm proposed by Dini et al. includes the generation of the keys and their subsequent propagation, in order to create a secure communication channel; while Di Pietro et al. adopts a different approach, in which each node belonging to the network generates its session keys, by means of values (i.e., initial seeds) assigned by NOS in the initial registration phase.

Since NOSs support both registered and non registered sources (see Paragraph 3.3.2), note that only the former are interested in the mechanism of key distribution. The latter may send their data to NOSs in clear (or otherwise they are discarded, if ciphered with an unknown mechanism), because no agreement has been taken through the proper HTTP source registration interface, about key exchange or the encryption scheme (e.g., RSA, AES) to be adopted. In particular, registered sources are associated with a unique identifier, that allows NOSs to recognize one source with respect to another. In addition, during the registration phase, some credentials may be provided by NOS to the interested source. In Chapter 3, the concepts of *SignatureKey* and *ActionKey* were introduced in a general way. In this paragraph, the generation and the use of such keys will be investigated.

In fact, the key management algorithms proposed by Dini et al. [44] and Di Pietro et al. [162] will be detailed along with their integration into NOS functionalities in paragraphs 4.2.1 and 4.2.2, respectively.

4.2.1 Key management system by Dini et. al

The distribution system of the cryptographic keys presented by Dini et al. [44] responds to the requirements of dynamism and mobility of the large scale IoTenvironment. This approach does not assume a fixed network topology, but considers the possibility that the nodes may dynamically join and leave the network. Moreover, the resource constraints of the nodes are taken into account, thus reducing the network traffic and the processing operations. The approach of Dini et al. represents a node-to-node distributed key agreement, which supports the creation of secure communication channels among the nodes. Such channels connect sequences of adjacent nodes sharing the same keys. This result is achieved by propagating the key connecting the start node and the first subsequent node to all the other nodes in the channel. Note that all the communication channels are bidirectional, allowing one to deliver reply messages as well as to transmit a message from/to any intermediate node. Once a secure channel has been established, no hop-by-hop encryption/decryption is required, thus improving the algorithm performance in terms of overhead and reliability.

Two set of keys are stored in each node (source):

- The *global keys*, which are assigned by NOS during the initial registration phase and are used to legitimate the encrypted communications among the sources within the IoT system
- The *local keys*, which are also directly assigned by NOS to a source or indirectly by a source to another one (just owning the same global key) and are used for encrypting the data transmitted to NOSs.

In particular, global keys are only aimed at the propagation of the local keys, as specified in the following, where each key is denoted by a name and a value. The name *K* of a local key consists of two components:

- K_{node}, which is coded in the *d* most significant bits of *K*, and is equal to the name of the NOS that generated the key
- An incremental number K_{incr}, which is coded in the least significant bits of *K*.

Since the name of a local key corresponds to the name (i.e., the identifier) of the generating NOS, it is always possible to identify, by means of the component K_{node}, the NOS belonging to the communication channel that established the local key for a pool of sources.

In a similar way, the global keys have a value and a name. The latter is the order number of that key in the pool of key values generated by each NOS by means of the algorithm of Eschenauer and Gligor [54]. In fact, as regards the initialization phase of Dini et al., the algorithm of Eschenauer and Gligor [54] is adopted; it implies a random key pre-generation scheme before any source can start to send or receive data to or from NOS. During the registration phase, each incoming source receives from NOS a global key, from the large pool of keys generated by NOS itself after this preliminary task.

Bearing in mind that multiple NOSs could act in the same IoT environment, each NOS N stores three tables aimed at preserving information about keys and connections with registered sources:

- The connection table CT_N, which contains one entry for each source S connected to the NOS N; in more detail, the entry $CT_{N,S}$ contains the list of the local keys K_i generated by N (i.e., K_{node} is equal to N) shared with S, which allows multiple connections $(N, S)K_i$
- The global key table GT_N, including one entry for each global key G_i generated by NOS N (i.e., K_{node} is equal to N)
- The local key table KT_N, containing an entry for each local key K generated by another NOS (K_{node} is not equal to N). The information included in KT_N are required in case a source begins to send its data to another NOS, which does not correspond to the NOS to which it was registered before.

Figure 4.2 provides an example with two NOS, namely NOS_A and NOS_B, and three data sources (i.e., S_1, S_2, S_3). Sources S_1 and S_2 send data to NOS_A, while S_3 provides data both to NOS_A and NOS_B. Figure 4.2 also shows the contents of the three aforementioned tables for NOS_A and NOS_B, along with the global and local keys owned by the three sources.

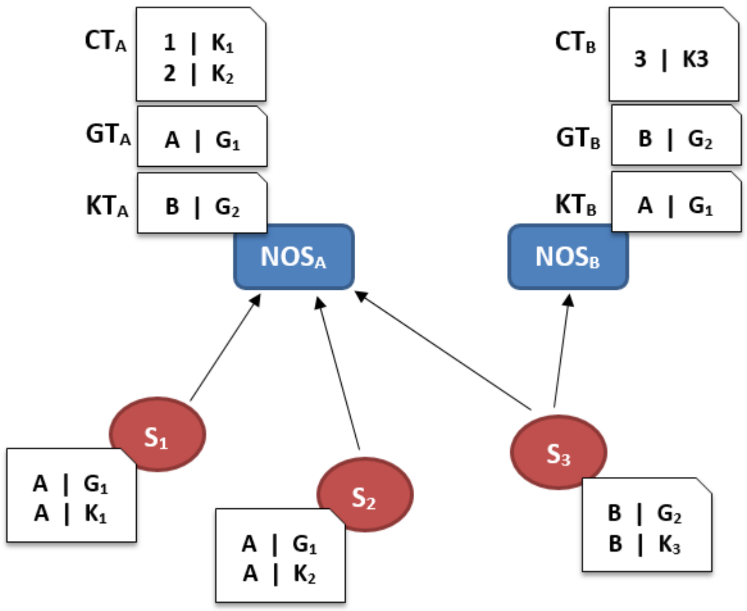


FIGURE 4.2: Key management in Dini et al.

Note that, in the case of a single NOS, no local key table is required. Instead, since data sources do not generate local keys by themselves, they store only two local tables: the connection table, including their active connections with NOS and other sources, and the global key table, containing the global keys received by NOS. The number of global keys stored by a source depends on the specific application domain, due to the fact that a source could provide data belonging to different contexts and, therefore, register to NOS with multiple credentials. Such a case is not considered yet, but the potentialities of the algorithm of Dini et al. are not be limited to a single application scenario.

Algorithm 4.1 outlines the steps of the key generation and distribution proposed by Dini et al. Steps 1-3 point out that an entry $CT_{N,S}$ already exists; therefore, the source *S* and the NOS *N* are connected and can use the local key *K* for encrypting and decrypting the data *D* to be exchanged. In contrast, if such an entry does not exist, *N* searches for a global key *G* shared with *S* (steps 4-5). If such a global key exists, then *N* generates a new local key *K*, thus establishing the connection between *S* and *N* (steps 36-41). Whereas, if the global key is not found, but *S* communicates with an adjacent node *M*, connected to NOS *N* and sharing the local key *K*, then *M* could act as an intermediate source for setting a new connection between *S* and *N* (steps 6-13). Instead, if a connection between *M* and *N* exists by means of the local key K' (steps 15-19), different from *K*, or the global key *G* (steps 21-25), then a connection between *S* and *N* could also be created with *M* as an intermediate source. Otherwise, if no global key *G* is shared by *M* and *N*, another source M' could be used as intermediary (steps 27-32). If no connection succeds through neighboring sources (i.e., no key is valid), *S* cannot take part in secure communcations with NOS, since no registration has been performed.

Dini et al. also provides a mechanism for the local key replacement. The new key value has to be updated in the connection key table of each node which shares this key. This is achieved by propagating a key replacement message to all the sources directly connected to NOS. Such a procedure is periodically started by NOSs for the local keys, in order to improve the robustness of the communications with the registered sources, or in case of malicious node detection; in fact, if a network attack is recognized by one or more NOSs, as explained in Paragraph 4.3, then it/they immediatly initiates/initiate the key replacement algorithm. For further detail about the procedure, refer to [44], but, in general, it exploits the correspondence between the actual name of the local key and the identifier of the generating NOS. If, otherwise, a global key has been compromised, then it should be discarded from the network and further updated. In this case, NOS will send an invalidation message containing the name of the compromised global key; therefore, the sources will discard such a key to prevent its use in establishing other connections (e.g., with possible malicious nodes). Finally, a new global key must be established.

4.2.2 Key management system by Di Pietro et. al

The Di Pietro et al. [162] scheme is composed of two main phases:

- First phase: the new session key is autonomously generated by each source
- Second phase: the new session key is synchronized among all the registered sources.

Algorithm 4.1 Dini et al. key distribution algorithm

```
1: if $(N,S)K\ exists\ in\ CT_S$ then
2:     $S$ sends $D$ to $N$ encr with $K$
3:     $N$ decrypts $D$ with $K$
4: else
5:     $G$ = findGlobalKey($N$, $S$)
6:     if ($G ==$ “) then
7:         $S$ communicates with $N$ through adjacent node $M$ with connection $(S,M)K$
8:         $S$ sends $D$ to $M$ encr with $K$
9:         if $(M,N)K\ exists$ then
10:            $M$ sends $D$ to $N$ encr with $K$
11:            $N$ decrypts $D$ with $K$
12:            $S$ and $N$ update their $CT$
13:            connection $(S,M,N)K$ set
14:        else
15:            if $(M,N)K'\ exists,\ where\ K\,!=\ K'$ then
16:                $M$ sends $D$ to $N$ with couple $K = (K_{node}, K_{incr})$ received by $S$ encr with $K'$
17:                $N$ sends an ack to $M$
18:                $M$, $S$ and $N$ update their $CT$
19:                connection $(S,M,N)K$ set
20:            else
21:                if $(M,N)G\ exists$ then
22:                    $M$ sends $D$ to $N$ with couple $K = (K_{node}, K_{incr})$ received by $S$ encr with $G$
23:                    $N$ sends an ack to $M$
24:                    $M$, $S$ and $N$ update their $CT$
25:                    connection $(S,M,N)K$ set
26:                else
27:                    does not exist any $G$ shared by $M$ and $N$
28:                    $M$ communicates with $N$ through adjacent nodes
29:                    while connection with $N$ set or adjacent nodes available do
30:                        execute searchAdjacentNodes()
31:                    end while
32:                    comment: if such procedure fails, $S$ is disconnected from the network
33:                end if
34:            end if
35:        end if
36:    else
37:        $N$ generates a new local key $K$
38:        $S$ sends to $N$ the couple $K = (K_{node}, K_{incr})$ encr with $G$
39:        $S$ sends an ack to $N$
40:        $N$ and $S$ update their $CT$
41:        $S$ can send $D$, encr with $K$, to $N$ and can decrypt messages from $N$ with $G$
42:    end if
43: end if
```

In fact, the algorithm guarantees that each source generates and shares the same key. The following definitions are required:

- *MSG* denotes a message that a source wants to send
- $E_k(MSG)$ indicates the encryption algorithm *E* employing the key *k*
- $E_{k^{-1}}(MSG)$ represents the decryption of *MSG*
- *q* denotes he length in bits of key *k*
- *H* and *G* are one-way hash functions (e.g., SHA-1 or MD5), which do not need to be kept secret.

Each source is provided by NOS, during the registration phase, with two random seeds, *S1* and *S2*, each *q* bits long. These two seeds are the only critical secrets that the tamper-resistance property has to preserve. Furthermore, each source can store an integer counter representing the sequence number of the current session key, and has enough memory to store a limited, constant number of session keys. The final value of the key is represented by the XOR boolean function applied to the results of the hash operations on the two seeds *S1* and *S2*.

Figure 4.3 clarifies that NOS_A stores the information related to the seed pairs (in this example, the couple *i, j*), while each data source (i.e., N_1 and N_2) stores a seed pair, the current session identifier, the current key and the previous key (whose role is explained in the following). Note that N_2 can send a message MSG_1 to N_1, and, then, N_1 can both decrypt the message and/or forward it to NOS_A, always encrypted with the current key.

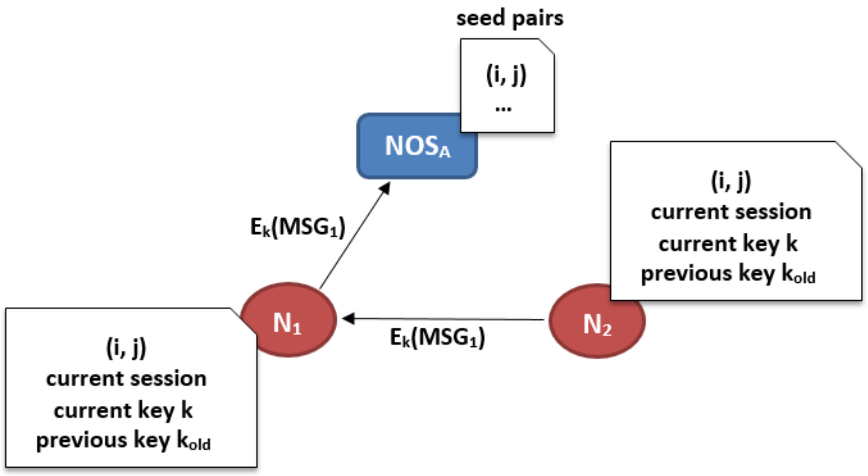


FIGURE 4.3: Key management Di Pietro et al.

Algorithm 4.2 outlines the steps of the solution proposed by Di Pietro et al. for the key generation and distribution; while Algorithm 4.3 defines the key update function.

There exist two different scenarios for the application of this scheme. The former requires a single central entity acting as a synchronizer for performing the re-keying of the sources; while the latter is a completely distributed approach, in which the sources should rely on themselves to achieve synchronization in the re-keying process. A distributed approach fits better NOS's aims. In particular, a scenario in which multiple NOSs manage the configuration activities among the registered sources, which, in turn, do not directly initiate any re-keying process. Note that this differs from the original Di Pietro et al. approach, in which each node could initiate the re-keying activity. In more detail, each NOS has to interact with its managed sources in order to invoke the command to generate the new keys.

With regards to the re-keying commands, NOS invokes them periodically. When a source receives from NOS a message that requires to update the current session key, it first saves the current value of the key, and then updates the session key. The source must save the value of the previous key, since it could receive messages that have been previously encrypted with the old key (for instance the session key has been updated while some messages were still on the fly). NOS encrypts the command to generate the new key with the last key generated by the sources themselves. Note that such

Algorithm 4.2 Di Pietro et al. key generation algorithm

```
Initialization
currentSession = 0
S1′ = H(S1)
S2′ = G(S2)
k = S1′ XOR S2′
while true do
    $MSG$ reception
    if $MSG.session == currentSession$ then
        $M = E_{k^{-1}}(MSG)$
    else
        $M = E_{oldk^{-1}}(MSG)$
    end if
    if $MSG.type == updateSession$ then
        execute updateKey() see Algorithm 4.3
    end if
    continue with normal execution
end while
```

Algorithm 4.3 Di Pietro et al. key update function

```
updateKey()
S1′ = H(S1)
S2′ = G(S2)
$k_{old}$ = k;
k = S1′ XOR S2′
currentSession = currentSession + 1
```

a key could be compromised by a malicious entity; therefore, a more resilient mechanism, such as *DHKE (Diffie-Hellman Key Exchange)* [90], should be adopted in order to secure the just mentioned operation and counteract the man-in-the-middle attack.

In Paragraph 5.3, the performance of Dini et al. and Di Pietro et al. key management systems is evaluated in terms of storage occupancy, overhead, delay, and robustness towards malicious attacks.

4.3 QoP and DQ assessment

The adoption of an efficient and robust key management system is only one of the steps required for the development of a secure IoT middleware. Since the final goal of NOS is to provide security and quality aware services to the end-users, a number of supporting algorithms for the evaluation of the information as well as of the data sources should also be integrated. The final result is published in [193].

Security and privacy are widely acknowledged to represent crucial issues in the IoT heterogeneous context, as explained in Chapter 1. On the one hand, the confidentiality and the integrity of the transmitted and stored information has to be guaranteed, and authentication and authorization mechanisms have to be provided to prevent unauthorized users or devices to improperly access the system. On the other hand, privacy of users, in terms of ability to support data protection and users anonymity, has to be ensured, which represents a critical aspect in particular in the presence of personal and/or sensitive information. Beyond security, also data quality represents an essential requirement for a large adoption of IoT services. The information provided should be accurate, timely and complete, since, in some scenarios, errors or missing values might have a critical impact on the actions or decisions upon the IoT system itself. Indeed, since IoT-enabled services and applications may make use of different data sources, the user (or the application itself) has to be aware of the security and quality level of the data being accessed, in order to take informed decisions about their usage. Such features have been presented as NF properties in Chapter 3, namely, QoP and DQ requirements.

What emerged is the need for a system able to deal with heterogeneous data sources and to assess both the QoP and the DQ of the information being collected, processed and transmitted, possibly in real-time and in an automatic manner. Furthermore, such a system must be able to work in the absence of *a priori* complete knowledge of the sources themselves, since IoT environments are highly dynamic and different kinds of attack may occur. In fact, in such a scenario, data may be compromised by rogue devices, hindering the correctness and confidentiality of the information (e.g., data integrity violation, man-in-the-middle attacks, packet sniffing). Source authentication issues (e.g., compromised keys, session violation) shall also be accounted for.

In order to deal with such threats, novel mechanisms for the assessment of DQ and QoP have been integrated into NOS, aimed at analyzing the data sources as well as the data they generate over the time. Paragraphs 4.3.1 and 4.3.2 provide more insights about the features and the behavior of the algorithms for the data quality and security evaluation, respectively.

4.3.1 Data quality evaluation

As regards the data quality analysis, a score in the range $[0, 1]$ is assigned to timeliness, completeness, accuracy and precision levels [31] [100] by the *Quality Analyzer* (see Paragraph 3.3.2). In particular:

- *Timeliness* is defined as the temporal validity of data and is calculated on the basis of the freshness of data and on the frequency of data updates; it is usually measured as a function of two variables: currency and volatility:

$$Timeliness = \max\left(1 - \frac{Currency}{Volatility}, 0\right), \tag{4.1}$$

  where $Currency$ is defined as the interval from the time when the value was sampled to the time instant at which data are received by NOS

- $Volatility$ is static information that indicates the amount of time units (e.g., seconds) during which data remains valid; it is usually associated with the type of phenomena that the system has to monitor and depends on the timescale of its dynamics

- *Completeness* is calculated as the number of collected values over a given time interval divided by the number of expected values:

$$Completeness = \frac{collectedValues}{expectedValues}. \tag{4.2}$$

  Note that missing values can be caused by sensor inefficiencies or communication issues

- *Accuracy* is usually defined as the degree of similarity of a measured quantity to its true value; it is also related to *precision*, which is the degree to which further measurement or calculations return the same or similar results. Ideally, a sensor shall be both accurate and precise, with all the measurements close to a reference value representing the true figure. In continuous value monitoring, accuracy and precision can be used as the features able to reveal errors or changes in the monitored process. Moreover, precision is often specified in terms of the standard deviation of the measured values: the smaller the standard deviation, the higher the precision. Formally, the accuracy of a value can be retrieved by calculating the error resulting from the difference between the sensed value v_n and a reference value v_{ref}. The acceptable measurement error can be defined as ϵ_{acc} and the measure is considered accurate if:

$$|v_n - v_{ref}| < \epsilon_{acc} \tag{4.3}$$

  Considering such a constraint, each value is associated with boolean metadata: the value 1 is assigned to accurate values while the value 0 is assigned to inaccurate values. In this way, considering the streaming of values, the accuracy of the received values can be calculated as the ratio of the number of inaccurate values over the number of collected values:

$$Accuracy = 1 - \frac{wrongValues}{collectedValues}. \tag{4.4}$$

- *Precision* can be defined as the inverse of variance. A measure is considered precise if:

$$1/n \cdot \sum_{n=1}^{N} (v_n - \mu)^2 < \epsilon_{prec} \tag{4.5}$$

  where μ is the average of the sequence v_n of sensed values, while N is the maximum number of measurements considered in a specific time interval. Accordingly, an aggregate measure of precision can be computed as:

$$Precision = 1 - \frac{notPreciseValues}{collectedValues}. \tag{4.6}$$

Precision is a value that is mainly used to better understand the accuracy measure. Indeed, situations in which values are incorrect but precise should be thorough analyzed since inaccuracy can be caused by changes in the monitored phenomenon or by faulty sensors [31] .

Quality assessment is performed periodically on the basis of a window-based approach [100].

4.3.2 Security evaluation

For security purposes, data sources are allowed to register with NOS, as already explained. The registration gives various advantages, since it allows NOS to have a complete knowledge of the source itself and to establish an encryption scheme along with the proper keys and identifiers to be used. Some registered sources may use neither authentication credentials nor encryption. NOS also accepts data from anonymous sources; also in this case a security evaluation has to be performed.

The *Security Analyzer* (see Paragraph 3.3.2) must be able to access to the *Sources* storage unit since, in order to analyze the received data, it may need information regarding the sources registered to NOS. To this end, NOS exploits an algorithm valid for both registered and anonymous sources, which aims to associate a score in the range $[0, 1]$ with the security metrics (i.e., confidentiality, integrity, privacy, authentication), detailed in Paragraph 3.2. As in the IoT context, NOS may have to manage sensitive data. Such security scores are intended to represent levels of *confidentiality* and *integrity* of the information transmitted to NOS, *privacy* of the transmitting source (i.e., idenfiable information related to data sources) and *authentication* (i.e., the robustness of the source authentication towards NOS). Note that malicious devices may be represented both by registered (for example, compromised) and non registered sources, thus sending corrupted data to NOS or executing malicious actions on data transmitted by non malicious sources (e.g., spoofing, sniffing).

The proposed security assessment algorithm takes into account two sets of parameters:

- A set of threats/attacks a_n, which includes the attacks that may be carried out towards the sources or the data transmitted to NOS (e.g., data violation, unauthorized access, masking, impersonation)
- A set of security countermeasures c_m, which regards the countermeasures made available by NOS in order to face the attacks included in a_n (i.e., encryption, authentication, key pre-distribution).

The security model considered by the algorithm links the attacks of a_n with the corresponding countermeasures in c_m. The taxonomy of the security attacks and the related countermeasures is retrieved from [172]. This work is appropriate as a mean of identifying a set of attacks and countermeasures for the IoT environment. In fact, it is not closely related to computers and networks threats, as are most of the existing works on taxonomies, but focuses on embedded devices, which are strictly related to the IoT technologies. It refers to:

- Data confidentiality
- Authentication and integrity
- Key management protocols
- Reputation schemes.

In detail, it considers each countermeasure to present a degree of resistance to a violation or to an attack attempt. The relationship among attacks and countermeasures is many-to-many, because an attack can be tackled through a plurality of countermeasures and a countermeasure can face more than one attack.

Each relationship is associated with a weight w_{a_i,c_j} in the range $[0 : 1]$, which represents the level of robustness of the countermeasure c_j with respect to the attack a_i (see Figure 4.4). It is worth to remark that there only exists a couple a_i/c_j with the associated weight w_{a_i,c_j}; such information have a general meaning, therefore they are not established on a per source basis. Furthermore, the same couple a_i/c_j can be used in different contexts, with the same security levels.

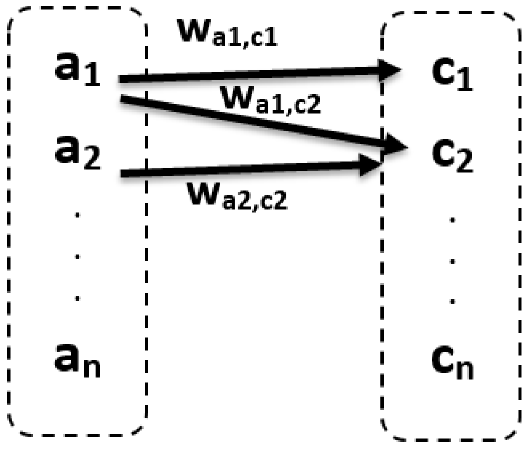


FIGURE 4.4: Weighed relationships among attacks and countermeasures

The identified relationships are clustered into four groups, one for each security metric to be analysed (i.e., confidentiality, integrity, privacy, authentication). The assignment of each attack-countermeasure pair to a group is made by NOS administrators in an early phase of system configuration, but it can be updated at runtime. Hence, at this initial stage, there are four groups of sets of attacks-countermeasures, which are named as follows:

- g_{conf} for attacks-countermeasures related to data confidentiality
- g_{int} for the pairs related to data integrity
- g_{pri} for privacy issues
- g_{auth} for the pairs concerning source authentication.

TABLE 4.1: Pairs attack-countermeasure

Attack	Countermeasure	Group
1) Packet sniffing	Data content encryption	g_{conf}, g_{pri}
2) Password attack	Complex password generation	g_{conf}, g_{auth}, g_{pri}
3) Man-in-the-middle attack	Authentication	g_{auth}
4) Session hijacking attack	Secure session establishment	g_{int}, g_{auth}
5) Identity spoofing	Identity encryption	g_{auth}
6) Key impairment	Secure key distribution scheme	g_{conf}, g_{auth}, g_{pri}

Note that such groups are not necessarily disjoint, since a pair may belong to one or more groups, as shown in Figure 4.5. The information related to the groups are put in the storage unit named *Config*, just presented in Paragraph 3.3.2.

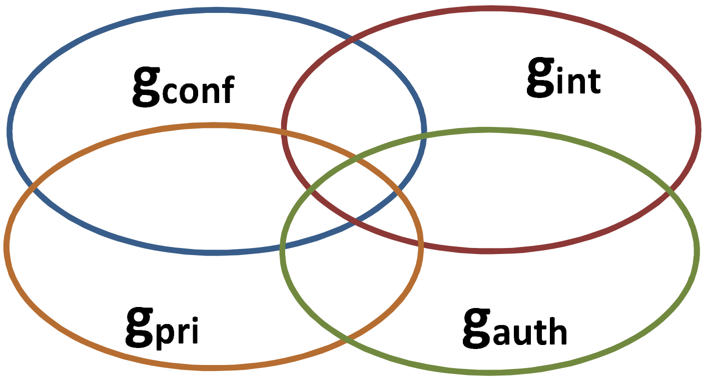


FIGURE 4.5: Attacks-countermeasures groups

Table 4.1 shows some examples of attack-countermeasure pairs derived from the used taxonomy and classified into the groups described above. Note that in Table 4.1 the countermeasures are presented in a generic way, since, as regards, for example, data encryption, a source can adopt different encryption schemes (e.g., RSA, AES), and, as a consequence, the corresponding robustness level varies over the time. In more detail, there is a weight for each relationship among, for example, the man-in-the-middle attack and the different encryption schemes which can be adopted (e.g., man-in-the-middle attack and RSA, and attack man-in-the-middle and AES). The same considerations apply to password generation and secure session establishment techniques.

Once the groups are defined, NOSs can update the weight corresponding to the relationships among attacks and countermeasures depending on the sources' behaviour and, consequently, on the basis of the received data. In this way, the system is able to evaluate the robustness of the countermeasures to particular kinds of attack during the normal IoT system operations, since NOS recognizes the possible malicious activities that can occur within the IoT system.

When the NOS platform is deployed in an IoT context, the initial weights are all conservatively set to 1. During operations, NOS should be able to detect the following events, for example by means of a proper intrusion detection system, which corresponds to the *Event Monitor* component in Figure 3.10:

- Data confidentiality and/or integrity violations; for example, a source may share some proper keys with NOS in order to encrypt its data; then, NOS may verify the integrity of received data, for example using a hashing technique, as [188]; if the data is recognized as corrupted, then an integrity attack on the encryption technique adopted by the source has been successful
- Source privacy violations
- Unauthorized access to the system (e.g., password violation, so an unauthorized device has accessed the system)
- Robustness of key management protocols in relation to key length (bits), deterministic or probabilistic generation of keys, encryption scheme adopted
- Replay or routing attacks which can hinder the freshness and the availability of the data received from the different sources.

Weights can vary over time in a dynamic way; such variations depend on the events described above or on context changes such as a source changing the length of its keys or the adopted encryption scheme. Such a process of automatic adjustment is performed by means of a well-known learning approach, namely difference temporal learning [195]. It is suitable for the learning in dynamic environments and is able to make predictions about specific features on the basis of temporal differences observed during system activity. In this way, the weights may decrease over the time with the observations of system or data violations, but they may also increase if a certain countermeasure turns out to be more resilient or if an attack is no longer performed.

The following equation regulates the update of weights:

$$\Delta w_t = \alpha \cdot \sum_{k=1,t} \nabla_w \cdot w_t \tag{4.7}$$

Where the weight variation Δw_t at time t depends on: (i) the learning rate α, which is a linear decreasing function of time; (ii) the sum $(\sum_{k=1,t} \nabla_w \cdot w_t)$, which is the sum of the gradients taking into account all the previous predictions until the time t.

After the update, performed at time t on the i-th pair attack/countermeasure, the corresponding weight w_{t_i,c_i} is updated according to Equation 4.8 (note that w corresponds to a weight w_{t_i,c_i}). Δw_t could assume negative values, but the resulting weights must be values in the range $[0:1]$.

$$w_{t+1} = w_t + \Delta w_t \tag{4.8}$$

Once the attack/countermeasure model is defined, the algorithm computes, for each incoming data, the related security scores, on the basis of the actual weights and of the data source src. Equations 4.9, 4.10, 4.11 and 4.12 show how the score corresponding to the level of confidentiality (sec_{conf}), integrity (sec_{int}), privacy (sec_{pri}) and authentication (sec_{auth}), respectively, is determined. Note that a score for each QoP property is calculated for every source.

$$sec_{conf} = \frac{a_{conf,src}}{a_{g_{conf}}} \cdot \frac{\sum\limits_{i\epsilon a_{conf,src}, j\epsilon c_{conf,src}} w_{i,j}}{c_{conf,src}} \tag{4.9}$$

Where: $a_{conf,src}$ is the number of confidentiality attacks to which the source src could suffer; $a_{g_{conf}}$ is the total number of attacks included in the model in the group g_{conf} for any kind of sources (not only those related to src); $c_{conf,src}$ is the number of countermeasures adopted by src in relation to the attacks on confidentiality included in $a_{conf,src}$. The sum of the weights considers only the weights between the attacks in $a_{conf,src}$ and the countermeasures in $c_{conf,src}$. For example, suppose that the source src adopts AES for encrypting its data; moreover, it also adopts an 8-bit length password as a credential for ensuring both confidentiality and authentication. As shown in Table 4.1 (points 1 and 2), AES is a countermeasure associated with the g_{conf} group; while the password is associated with both g_{conf} and g_{auth} groups. The steps performed by NOS to assess the confidentiality score sec_{conf} are the following:

- The initial weights corresponding to the two pairs attack-countermeasure (i.e., AES-packet sniffing, 8-bit password-credential violation) are set to 1; therefore, the corresponding confidentiality score sec_{conf} is 1. Equation 4.9 is initially evaluated as shown in Figure 4.6. For simplicity, $a_{conf,src}$ is considered, in this example, equal to $a_{g_{conf}}$. $a_{conf,src}$ is composed of two elements (i.e., packet sniffing and credential violation), and also $a_{conf,src}$ (i.e., AES and 8-bit password)

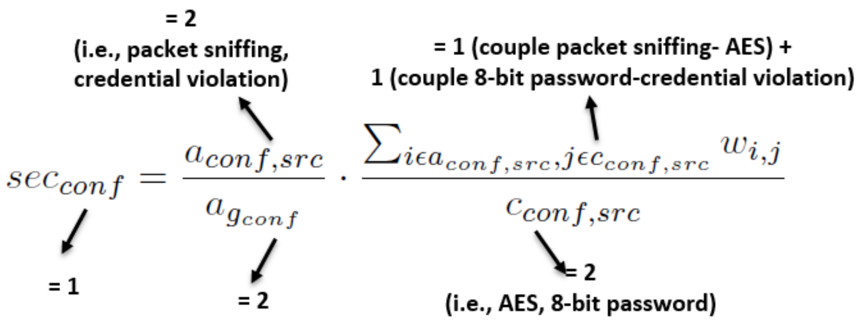

= 2
(i.e., packet sniffing, credential violation)
= 1 (couple packet sniffing- AES) +
1 (couple 8-bit password-credential violation)
$sec_{conf} = \frac{a_{conf,src}}{a_{g_{conf}}} \cdot \frac{\sum_{i\epsilon a_{conf,src}, j\epsilon c_{conf,src}} w_{i,j}}{c_{conf,src}}$
= 1
= 2
= 2
(i.e., AES, 8-bit password)

FIGURE 4.6: Confidentiality score assessment - initial stage

- During the system operations, NOS recognizes no violated packets from the source src, but its password has been intercepted (e.g, through brute-force attack) several times, as detected by the *Event Monitor*
- As a consequence, the weight related to the pair 8-bit password-credential violation decreases; for example, it is updated to 0.3 by the learning algorithm (Equations 4.7 and 4.8)

- The new data obtained from the source *src* will receive a lower confidentiality score sec_{conf}, which is recomputed to 0.65, as shown in Figure 4.7.

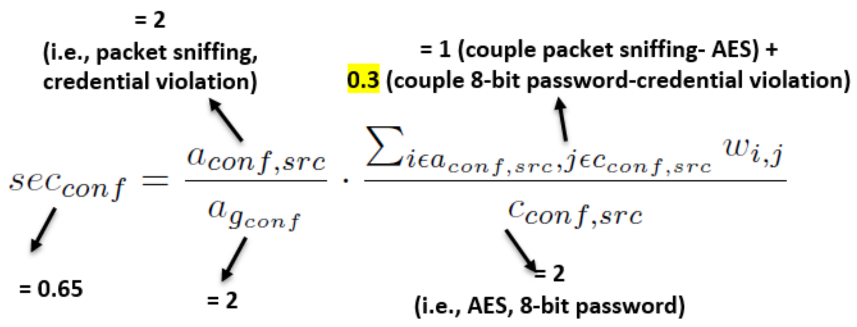

= 2
(i.e., packet sniffing, credential violation)
= 1 (couple packet sniffing- AES) +
0.3 (couple 8-bit password-credential violation)
$sec_{conf} = \frac{a_{conf,src}}{a_{g_{conf}}} \cdot \frac{\sum_{i \epsilon a_{conf,src}, j \epsilon c_{conf,src}} w_{i,j}}{c_{conf,src}}$
= 0.65
= 2
= 2
(i.e., AES, 8-bit password)

FIGURE 4.7: Confidentiality score assessment - update

As a consequence, a user who wants to receive data from the source *src* will be aware that they have a level of confidentiality not greater than 0.65, so there is a 35% risk of a confidentiality attack.

Instead, considering a malicious device which tries to execute a man-in-the-middle attack among two registered sources src_1 and src_2, the scope of the proposed security algorithm is to evaluate the robustness of the countermeasures, adopted by such sources (i.e., the robustness of the data content encryption scheme adopted, as shown in Table 4.1), are. For example, in this case, the affected score is that of integrity sec_{int}. Therefore, NOS executes the following steps:

- The initial weights corresponding to the attack-countermeasure pair for the two sources, src_1 and src_2, are both set to 1, but src_1 adopts a 128-bit key for encrypting its data, while src_2 uses a 256-bit key
- If NOS analyzes the level of robustness of such a countermeasure in terms of integrity against a possible man-in-the-middle attack, then it considers the difference in the bit-length of the keys used, and, following Equation 4.10, the result will be that $sec_{int_{src_1}}$ is less than $sec_{int_{src_2}}$.

Therefore, a user may choose to receive data only from the source src_2, since it presents a greater level of integrity with respect to the source src_1.

$$sec_{int} = \frac{a_{int,src}}{a_{g_{int}}} \cdot \frac{\sum_{i \epsilon a_{int,src}, j \epsilon c_{int,src}} w_{i,j}}{c_{int,src}} \tag{4.10}$$

The same considerations apply to the following metrics:

$$sec_{pri} = \frac{a_{pri,src}}{a_{g_{pri}}} \cdot \frac{\sum_{i \epsilon a_{pri,src}, j \epsilon c_{pri,src}} w_{i,j}}{c_{pri,src}} \tag{4.11}$$

$$sec_{auth} = \frac{a_{auth,src}}{a_{g_{auth}}} \cdot \frac{\sum_{i \epsilon a_{auth,src}, j \epsilon c_{auth,src}} w_{i,j}}{c_{auth,src}} \tag{4.12}$$

Note that, as stated above, the algorithm presented is suitable for both registered and non registered sources. It is very remarkable that the proposed algorithm allows one to perform a security analysis and obtain a valid score (not trivially set to 0 or to "undefined") also for non registered sources, with which NOS does not share, for example, any encryption scheme. This result is achieved by analysing the data they provide over time and the node behaviour within the IoT system. Obviously, non registered sources will be associated with low scores, thus the data provided will be filtered accordingly, and a user who requires reliable information, will be prevented from receiving the data obtained from such sources.

Concluding the discussion about the security evaluation, Algorithm 4.4 summarizes the steps performed by the security assessment scheme. Note that NOS is conceived as a modular architecture, therefore such a mechanism can be enabled or disabled on the basis of the current application needs.

Algorithm 4.4 Security assessment algorithm

```
Require: $Relationships\ among\ a_n,\ c_n\ from\ the\ taxonomy$
1: for all $a_i, c_i$ do
2:     $Assigment\ to\ groups\ g_{conf},\ g_{int},\ g_{pri},\ g_{auth}$
3:     $Initialization\ of\ the\ weight\ w_{a_i,c_i}\ to\ 1$
4: end for
5: while $Learning\ is\ enabled$ do
6:     $Events\ monitoring$
7:     $Weights\ adjustment\ (Eqs\ 4.7\ and\ 4.8)$
8:     $Scores\ update\ (Eqs\ 4.9,\ 4.10,\ 4.11,\ 4.12)$
9: end while
```

The rules, just presented for the assessment of the QoP and DQ scores of data, are stored in a proper format in *Config* storage unit. Such a collection, as introduced in Paragraph 3.3.2, contains all the configuration parameters required for the correct management of the IoT system (e.g., how to calculate quality properties, which attacks or security countermeasures to consider), represented in JSON format. It can also be configured at any time by an IoT system administrator through a secure connection (e.g., via HTTPS) depending on the requirements of the specific deployment, without the need to re-start NOS system. The communication protocol to be used is HTTPS, since the policy adopted by NOS for processing IoT data must be protected against external attacks. *Analyzers* periodically query the *Config* storage unit in order to know which rules to use.

Summarizing, NOS architecture has been designed to provide a score for each QoP and DQ requirement; in this way, a possible application scenario can be easily integrated depending on its purposes and on the specific context, and can benefit from high level of flexibility. For example, some applications require the use of data with a high level of privacy and confidentiality, but there is no interest in integrity issues. Other application domains, on the contrary, may aim to provide a service characterized by error-free data and high confidentiality scores, therefore the data to be selected are those provided by sources able to satisfy these requirements.

Note that searching and selecting data sources when there is no description, either about the sources or the acquired data, is a very challenging task. Although NOS is not able to directly counteract malicious devices (e.g., if NOS recognizes that a set of sources are being violated for a certain period, it only lowers the scores associated, but no other action is performed), it is able to recognize that the data provided by a source is corrupted or it presents a poor level of confidentiality or privacy and discard it as unsuitable. However, after repeatedly receiving bad data from the same source, NOS could even block it.

As a further remark, the scope of the algorithm presented is to assign a level of robustness to each data source according to the aforementioned security features (i.e., integrity, confidentiality, authentication system, and privacy). Therefore, this solution does not directly tackle the security attacks, but aims at minimizing the associated risks by letting users and applications be aware of the QoP level of the requested data.

By means of the algorithms presented above, NOS is able to perform automatic reasoning about QoP and DQ and, then, allow the users to filter information and deal properly with the massive amount of data received by IoT services, as shown in Paragraph 5.4, along with an example of application in a real context, by means of NOS prototype, presented in Paragraph 3.4.

4.4 Policy enforcement

The security and data quality assessment algorithms are based on rules, stored in *Config* collection, as specified in Paragraph 4.3.2. Such rules can be translated into the form of policies to be applied in an automatic manner within the IoT system along with other kinds of policy.

As pointed out in Chapter 2, traditional security countermeasures and privacy solutions cannot be directly applied to IoT scenarios for various reasons, including, but not limited to, energy and computing constraints, scalability etc. Moreover, adaptation and self-healing play an important role in IoT infrastructures, which must be able to face sudden and unexpected changes in the operational environment. Accordingly, privacy and security issues should be treated with a high degree of flexibility [16] [38]. Together with the 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 [12].

Note that, one instrumental aspect concerns the ability of the system to preserve QoP and DQ in presence of external attacks. It is important to remark that in the IoT context, the number of violation attempts is high. In such a scenario, the integration of the flexible NOS middleware, able to handle a large number of data streams and of interconnected devices, with a flexible policy enforcement framework is needed. In this direction, the solution proposed hereby aims to ease the management of interactions across different realms and policy conflicts.

In other words, in order to deal with the huge amount of critical situations typical of the sharing approach of IoT paradigm, it is fundamental to adopt well-defined enforcement mechanisms able to successfully tackle them. Furthermore, IoT deployments are characterized by a high degree of heterogeneity in terms of architectures and technologies, so that a suitable security framework should be highly flexible in order to adapt to various deployment features.

The NOS middleware presented in foregoing paragraphs does not define supporting mechanisms for: (i) controlling the access of both users and data sources; (ii) the data provision to users. An enforcement system would allow to overcome such limitations. To address this shortcoming, NOS is integrated with a policy enforcement system specifically tailored to IoT, able to manage the interactions among the involved entities, as presented in Paragraph 3.2, under well-defined policies. The proposed solution is able to guarantee adequate QoP and DQ. Such an approach automates this task, leaving to the system administrator to define scoring policies for QoP and DQ assessment, and to the service provider to specify the requirements on the data to be used, as will be clarified in the next paragraphs. The outcomes of the solution presented are published in [186].

4.4.1 Policy enforcement integration

In order to effectively manage the available resources and to handle possible violation attempts, NOSs are provided with a set of well-defined policies, specifying the behavior and the actions to be taken in a given situation. Accordingly, a fundamental role is played by the enforcement framework integrated in the NOS system, as it guarantees that the specified policies are correctly applied. In the case of NOS, policies refer, in particular, to the control of access to IoT data and manage communications. This comes from the requirement to protect both data resources and user sensitive information.

Security among the involved components (i.e., NOS, users, nodes) is guaranteed through the adoption of suitable encryption mechanisms, as presented in Paragraph 4.2.

Technically, the main challenge to be faced is how to integrate an enforcement mechanism in the existing NOS architecture, without affecting the existing functionalities. As specified in Paragraph 3.4, NOS is conceived as a modular architecture, but, in this case, the enforcement mechanism is not limited to the integration of one decoupled component; instead, it concerns all the interactions that happen during NOS activities and processing operations. For such reasons, the enforcement functionality is embedded in a wrapper layer, as shown in Figure 4.8, able to control NOS operations, but without requiring major system-level modifications.

Another challenge is represented by the identification of a minimal set of primitives, able to specify and enforce a large variety of attribute-based QoP and DQ policies.

An important feature of the presented policy framework is that it also supports the loading of new policies at runtime, without disrupting service operations. Such a feature increases the flexibility of the framework and makes it particularly suitable for IoT applications, which require a high degree of availability.

Another advantage of the adopted policy-based control is that the controlling unit of the system (i.e., the enforcement framework) is kept decoupled from other management components (i.e., *Data Normalization* and *Analysis* phases). As a consequence, the system administrator can manage and change the system behaviour without modifying the software or the

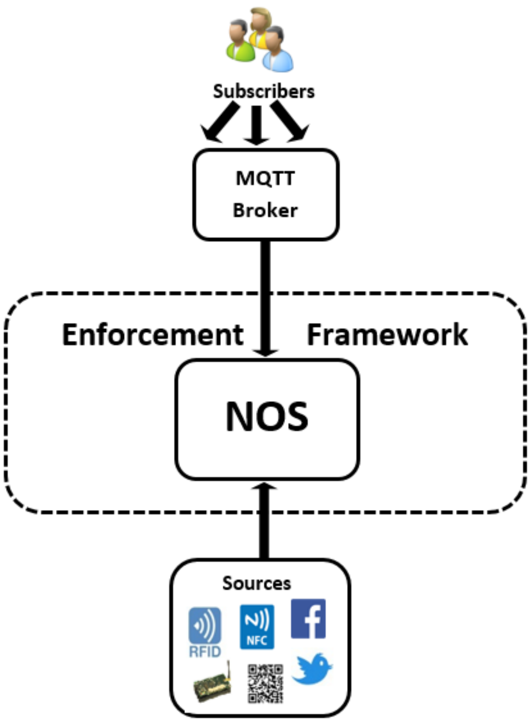


FIGURE 4.8: NOS policy enforcement integration

user/node interfaces. Furthermore, the entire system is controlled by policies that specify the rules interpreted and enforced by the proper framework. Hence, if the conditions change or if new services or applications are added, only the corresponding policy rules have to be adapted. Within NOS, all the security related tasks are executed seamlessly so that services are not required to have explicit knowledge of the security policies.

In the remainder of this paragraph the enforcement functionality is analysed in detail.

Enforcement components and policy language

The enforcement framework is in charge of handling access control and service provisioning under well-defined QoP and DQ requirements. The framework is defined so as to represent a redefinition of access control and data exchange in terms of a common set of functions and roles suitable for IoT applications. Functions and roles are dynamically configurable in order to provide the required level of flexibility to cover different application scenarios.

Conventional access control enforcement frameworks include a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), and a Policy Administration Point (PAP) [199]. In more detail:

- PEP is in charge of intercepting any request for access to resources from users, and of making a decision request to PDP in order to obtain the access decision (i.e., approve or reject). Whenever a user or an application requests access to data, this is routed through a PEP and transferred to a PDP for evaluation and an authorization decision

- PDP evaluates the access requests against the authorization policies in order to decide whether the request should be accepted. To this end, the PDP refers to, and queries, a policies store. When the PDP completes the evaluation, it returns a response to the PEP. Based on such a decision, PEP either permits or denies access to the user/resource
- The authorization policies are finally administered through a "centralized" PAP.

The functions just described are usually performed by application software. In the NOS case, where communication is based on the MQTT protocol (see Paragraph 3.3.2), all requests are handled via the MQTT broker.

The architecture underlying the framework may comprise one or more NOS and a huge number of nodes, which act as data sources, and users, who act as data consumers (either directly or mediated by applications/services), as clarified in Paragraph 3.4. In particular:

- Each NOS includes a PEP, a PDP and a PAP
- Each user has an application representing an interface for the user's personal device and NOS. Users may be directly registered to NOS or to another application, which is further registered to NOS. In the latter case, the application itself manages all the interactions with NOS and establishes the levels of security and quality for the data to be provided to the interested users. While, in the former case, a user, besides logs on the application running on his/her device using the GUI provided, opens a session, during which he/she can request for the services provided by NOS on the basis of the accessible resources.
- As far as nodes are concerned, a separate discussion has to be made, since the system has to be able to deal both with registered and non registered nodes (i.e., data sources).

All the introduced components interact with the underlying PEP on NOS.

The structure of the enforcement framework is sketched in Figure 4.9, where the collection, named *Policies*, may correspond to *Config* storage unit. Although the figure shows only one NOS, the framework may be executed in a distributed manner on multiple NOSs, whereby each NOS runs its own framework and a single application/service may interact with a plurality of NOSs.

Therefore, the distribution of policies, their update and synchronization have to be considered (this means, roughly speaking, to synchronize the content of the various instances of *Config* store on multiple NOSs). It is assumed that, in the case of multiple NOSs interacting with each other, all NOSs within the same administrative domain share the same security policies and each of them has its own policy enforcement component. More insights about policy synchronization will be provided in Paragraph 4.6.

The approach adopted follows the ABAC model [72]. In such a mechanism, both the subject who wants to access or to provide the resources, and the objects (i.e., data), which represent the resources themselves, are described by means of specific attributes, which are used for the policy definition. Attributes can be based on the metadata fields natively supported

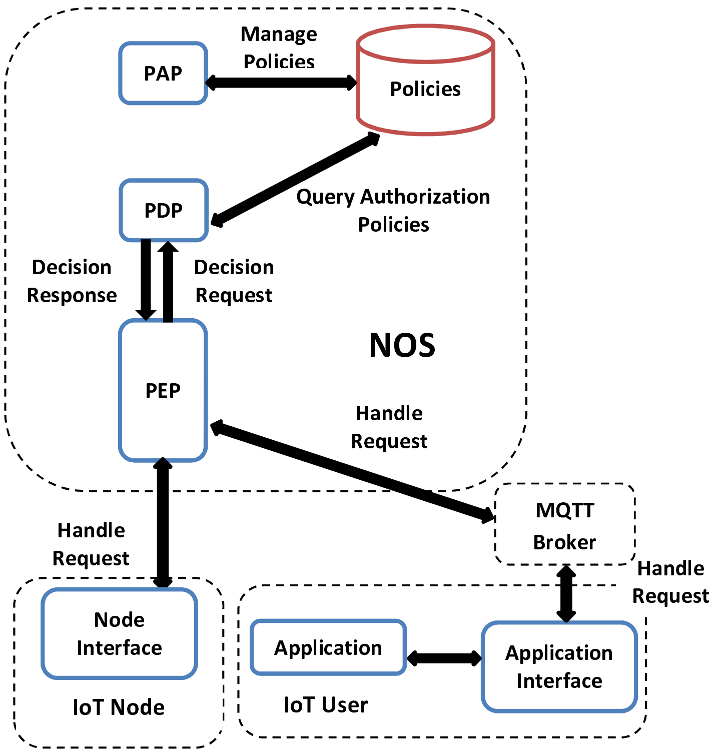


FIGURE 4.9: NOS enforcement framework

in the data representation, presented in Paragraph 3.3, and control rules can be defined according to the specific needs of the application domain. As widely acknowledged in the relevant literature, ABAC presents better scalability and flexibility than RBAC [178].

To ease interoperability and to enable the implementation of a policy enforcement system, a policy representation language has to be chosen. Given the large number of IoT domain applications, such a language has to be flexible enough to represent the analyzed contexts both in a general-purpose and in a customizable way. The policy language proposed hereby is specifically tailored to the management of enforcement, and is written in JSON syntax, being therefore suitable for integration with the database management system, used in the implementation, introduced in Paragraph 3.4 (i.e., *MongoDB*). It allows the expression of the whole set of policies for each entity involved (i.e., nodes and users). Each of them has specific attributes, as described in the following. According to the defined attributes, each entity can be allowed to perform different actions. It is worth remarking that the system allows the runtime change of policies, which can be dynamically loaded into the system through the aforementioned PAP.

4.4.2 Enforcement framework

The enforcement framework introduced above is in charge of ensuring that the system satisfies the QoP and DQ requirements of authorized users/nodes. Policies are applied to two types of entities:

- Data producers (IoT devices or nodes)

- Data consumers (users or applications).

In the following, some sample policy specifications will be investigated. Users and applications consume data and they must be registered. Nodes generate data and, as explained in detail in Paragraph 3.3.2, the system also accepts data generated by non registed nodes.

The user/application registration phase takes place through an exchange of credentials between the user/application and NOS (see Paragraph 3.3.2). In order to perform registration operations, a user/application must be authenticated with admin privileges. From the operational perspective, it is expected that entities consuming data (i.e., users/applications) will, in most cases, be registered by a system administrator.

When registering, an identifier is assigned by the system to each registered user/application, along with a function, conceived as a set of attributes used for filtering access to resources . Note that such attributes and the related access permissions are established by a system administrator, decoupled from NOS.

On the other hand, nodes can optionally self-register with NOS. Note that NOS also assigns an identifier to the registered nodes, which, in the registration phase, is specified as being the signature key, used for signing the data they send. Such credentials are eventually exchanged between the node and NOS through the proper source registration interface. For details about signature key use and definition, refer to Paragraphs 3.2.1 and 4.2.

Six fundamental actions, for which policies are specified, have been identified and formally described:

- *Node access control*
- *Node data transmission*
- *Node data processing*
- *User/application access control*
- *User/application service request*
- *Service provision*.

In line with the ABAC approach, policies are specified as a set of key-value pairs, each pair representing an attribute of the corresponding policy.

A policy is composed of three main building blocks:

- The first (*input*) defines the values that NOS expects to receive in input from the requesting entity (nodes or users/applications) and that are used for evaluating the policy for a specific action
- The second (*security*) defines the functions to be executed on the inputs provided to assess the policy
- Each function returns a value; such values are composed by means of the logic specified in the third block (*response*) to define whether the request shall be accepted or not.

The policies are represented in JSON format and are stored in the *Config* storage unit. The character @ is used to indicate the value taken by the corresponding field.

Node access control policy

Listing B.7 describes a sample version of the *NodeAccessControl* policy, which covers nodes wanting to send data to NOS. This policy is invoked by NOS before inserting the data into the *Raw Data* storage unit. The enforcement framework verifies whether the node is transmitting, along with the data, the node identifier and a signature key (specified as *node inputs* in Listing B.7).

The verification process is split into two branches:

- If the source is registered, NOS receives in input the node identifier and the signature key and can perform a *registrationCheck* (i.e., checking whether the identifier is known and valid) and the *signaturekeyCheck* (i.e, checking that the identifier and key are compliant for the requesting node). Conversely, if the source is unknown, the key is marked as *undefined* by the function *signaturekeyUndefinedMark* for the corresponding node
- In case of a non registered node, NOS keeps track of the source by assigning a pseudo-random identifier at the first communication exchange; such an identifier will be used in the following interactions. This approach allows NOS to verify whether the node is a new or a known one just by looking up its identifier.

If *registrationCheck* and/or *signaturekeyCheck* reveal that the credentials are not compliant with the requesting source, then the enforcement framework prevents such a node from interacting with NOS.

Once these checks have been passed and the node is allowed to send data to NOS, a pseudo-random session identifier is created and assigned by the *sessionAssignment* function. QoP and DQ are assessed on a per-session basis.

```
{ "NodeAccessControl": {
  "policy": "NodeAccessControl",
  "input": {
    "node": {
      "identifier": "@NodeID",
      "signaturekey": "@Key"
    }
  },
  "security": [ {
    "verificationRegistration": [{
      "registrationCheck": "@NodeID",
      "signaturekeyCheck": "@NodeID,@Key"
    }],
    "verificationUnknownSource": [{
      "signaturekeyUndefinedMark": "undefined",
      "identifierCheck": "@NodeID"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "@NodeID, timestamp"
    },
    {
      "signaturekeyCheck": false,
        "accessDenied": "@NodeID"
      }],
      "verificationUnknownSource": [{
        "sessionAssignment": "@NodeID, timestamp"
      }]
    }]
}}
```

LISTING 4.1: Node access control sample policy

Node data transmission policy

Once the node has completed the access control phase, then it can send data to NOS. Listing B.9 highlights the requested inputs for the corresponding policy, named *NodeDataTransmission*, which are: the session identifier, previously assigned by NOS to the node during the access control phase; the node identifier; the data itself and the data type.

Note that, at this stage, no security operations have yet been performed: if all the requested inputs are present (i.e., *requiredInformation* action is *true*), the data are stored in the *Raw Data* storage unit (i.e., *storeData* action is activated for the actual node). Data gets discarded only if the transmitting node fails to provide the required information to NOS (i.e., *discardData* action is undertaken).

```
{ "NodeDataTransmission": {
  "policy": "NodeDataTransmission",
  "input": {
    "message": {
      "session": "@Session",
      "identifier": "@NodeID",
      "data": "@d",
      "datatype": "@dt"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true,
      "storeData": "@NodeID,@d,@dt"
    },
    {
      "requiredInformation": false,
      "discardData": "@NodeID,@d,@dt"
    }]
}}
```

LISTING 4.2: Node data transmission sample policy

Node data processing policy

NOSs own processing modules that periodically fetch data from *Raw Data* or *Normalized Data* storage units and elaborate them, as detailed in Paragraph 3.3. The policy invoked at this step is called *NodeDataProcessing* and, as shown in Listing B.10, receives in input the same values of the *NodeDataTransmission* policy, but the action of data evaluation is enforced, before sending processed data to the publish/subscribe system.

For registered sources, NOS performs *decryptionData* and *decryptionDatatype* operations, thus decrypting the data and the corresponding data type using the key of the actual node. Hence, for both registered and non registered sources, *scoreAssessment* is executed and a score for each security and quality property is assigned to the data, using the algorithms defined in Paragraph 4.3.

```
{ "NodeDataProcessing": {
  "policy": "NodeDataProcessing",
  "input": {
    "message": {
    "session": "@Session",
    "identifier": "@NodeID",
    "data": "@d",
    "datatype": "@dt"
    }
  },
  "response": [{
    "evaluationRegisteredSource": [{
      "decryptionData": "@d,@NodeID",
      "decryptionDatatype": "@dt,@NodeID",
      "scoreAssessment": "@d,@NodeID"
    }],
    "evaluationNonRegisteredSource": [{
      "scoreAssessment": "@d,@NodeID"
    }]
  }]
}}
```

LISTING 4.3: Node data processing sample policy

User access control policy

Listing B.11 refers to the access request from a user/application who/which wants to receive data from NOS; such a request is sent to the MQTT broker, which performs an access request to the enforcement framework.

The corresponding policy, named *UserAccessControl*, is invoked before sending any data to the requesting entity. It verifies whether the user/application is registered, using for such a purpose the following parameters: username, user/application identifier, signature key and a function, previously specified during the registration phase.

The policy verification process includes the following cases:

- If the user/application is registered, NOS receives in input the user/application identifier, the function and the signature key and can perform *registrationCheck* (i.e., the identifier is known and valid for the specified function) and *signaturekeyCheck* (i.e, identifier, function and key are compliant for the requesting user/application)

- If the user/application is unknown, then the key is marked as *undefined* by the action *signaturekeyUndefinedMark* and the user/application is not authorized by the enforcement framework to access the system

- If a user/application tries to register with wrong credentials, for example with a function different from the one declared during the registration phase or with a different key, then the enforcement framework generates a negative response, alerts the user/application and does not allow any interaction with the IoT system

- Otherwise, in the case that the checks have been passed, the user/application is allowed to interact with NOS and a pseudo-random session identifier is assigned by the *sessionAssignment* function.

```
{ "UserAccessControl": {
  "policy": "UserAccessControl",
  "input": {
    "user": {
      "username": "@Username",
      "identifier": "@UserID",
      "signaturekey": "@Key",
      "function": "@Function"
    }
  },
  "security": [{
    "verificationRegistration": [{
      "registrationCheck": "@UserID,@Function",
      "signaturekeyCheck": "@UserID,@Key,@Function",
    }],
    "verificationUnknownUser": [{
      "signaturekeyUndefinedMark": "undefined"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "@UserID,@Function, timestamp"
    },
    {
      "signaturekeyCheck": false,
      "accessDenied": "@UserID,@Function"
    }],
    "verificationUnknownUser": [{
      "accessDenied": "@UserID,@Function"
    }]
  }]
}}
```

LISTING 4.4: User access control sample policy

User service request policy

Once the user/application has completed the access control phase and is authenticated, then it can receive data from NOS by activating the corresponding subscription to the MQTT broker. Listing B.12 highlights the requested inputs for the corresponding policy, named *ServiceRequest*, which are: (i) a session identifier, given by NOS to the user/application after the access control phase (computed randomly at each access, as for the nodes); (ii) the username and the identifier; (iii) the function; (iv) the requested service, along with the user preferences in terms of QoP and DQ.

For the service invocation, as discussed in Paragraph 3.3.2, a resource is treated as an object identified by a hierarchical name (e.g., a URI). A service is conceived as software able to fulfill a specific task making use of the available data. There is no direct interaction among users/applications and NOS resources, but a well-defined programming interface is needed through a software application. Resources can be accessed by users/applications only once they are published as object instances.

Note that at this stage no security operations are performed: the request is elaborated by the system if all the inputs are valid (i.e., *requiredInformation* action is *true*); if not, the request is discarded by the enforcement framework (i.e., *requiredInformation* action is *false*). The security and quality preferences are not mandatory: if they are omitted, the enforcement framework does not discard the request, but sets the corresponding constraints to the lowest admissible values.

```
{ "ServiceRequest": {
  "policy": "ServiceRequest",
  "input": {
    "message": {
      "session": "@Session",
      "username": "@Username",
      "identifier": "@UserID",
      "function": "@Function",
      "service": "@Service",
      "securityPreferences": "@confidentiality , @integrity ,
    @privacy , @authentication",
      "qualityPreferences": "@accuracy , @precision , @timeliness ,
    @completeness"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true ,
      "processRequest": "@Session ,@Username ,@UserID ,@Function ,
    @Service ,@confidentiality , @integrity , @privacy , @authentcation
    ,@accuracy , @precision , @timeliness , @completeness"
    },
    {
      "requiredInformation": false ,
      "discardRequest": "@Session ,@Username ,@UserID ,@Function ,
    @Service"
    }]
  }]
}}
```

LISTING 4.5: User service request sample policy

User service provision policy

Finally, *ServiceProvision* policy is activated after a data/service request, in order to verify the matching between the request itself and the requesting user/application, in terms of identifier and function, by performing *serviceAccessVerification* action (Listing 4.6).

Such a policy receives in input the same values of the *ServiceRequest* policy. Note that the parameters describing the requested data are sent encrypted by the requesting user/application. Therefore, NOS has to decrypt it (i.e., *decryptionRequest* action). From the identifier, NOS derives the signature key of the authenticated user/application and uses it to decrypt the message.

After the verification step, the *retrieveResults* action is performed. It retrieves the data corresponding to the requested service, for which the user/application is allowed to access. Before sending them back, the retrieved data are filtered on the basis of the QoP and DQ constraints. In case there is no matching among the parameters described (i.e., the user with the specified identifier and function is not allowed to access the requested service), the enforcement framework blocks the data/service provision process and sends an error message to the requesting entity.

```
{ "ServiceProvision": {
  "policy": "ServiceProvision",
  "input": {
    "message": {
      "session": "@Session",
      "username": "@Username",
      "identifier": "@UserID",
      "function": "@Function",
      "service": "@Service",
      "securityPreferences": "@confidentiality, @integrity,
    @privacy, @authentication",
      "qualityPreferences": "@accuracy, @precision, @timeliness,
    @completeness"
    }
  },
  "security": [{
      "decryptionRequest": "@Service,@UserID",
      "serviceAccessVerification": "@Service,@UserID,@Function"
  }],
  "response": [ {
      "serviceAccessVerification": true,
      "retrieveResults": "@Service,@UserID,@Function,
    @confidentiality, @integrity, @privacy, @authentcation,
    @accuracy, @precision, @timeliness, @completeness"
    },
    {
      "serviceAccessVerification": false,
      "accessDenied": "@Session,@Username,@UserID,@Function,
    @Service"
    }]
}}
```

LISTING 4.6: Service provision sample policy

A real application scenario exploiting the policies presented will be provided in Paragraph 5.5.

4.5 Secure publish&subscribe protocol

Besides providing a robust key management system, data assessment mechanisms, and policy enforcement, an efficient and secure communication mechanisms also represents a fundamental enabler for an effective IoT system. In such contexts, it is fundamental to define how the involved "things" could efficiently communicate and exchange information among themselves and with remote servers. One important challenge relates to the amount of generated data, which poses scalability issues. Furthermore, some of such data may represent sensitive or personally identifiable information. What emerges is that there are significant issues to be addressed in order to efficiently and securely manage IoT systems. Such problems are related to:

- The management of connections among the IoT system and the data sources (e.g., the devices which acquire information from the IoT environment), which could be affected by resource constraints in terms of energy and storage capacity

- The possibility, for the users, to control the distribution of their sensitive information through IoT connections as well as effective authentication and authorization mechanisms both for users and devices in order to prevent malicious access to resources.

As regards the first issue, several existing application-level protocols for IoT and Machine-To-Machine (M2M) [26] [4] systems have been designed. Such protocols are typically conceived to introduce little overhead and to minimize battery consumption, as well as to have better performance in the presence of many short messages. The most widely adopted communication protocols in such fields are MQTT (just introduced in Paragraph 3.3.2) and CoAP (Constrained Application Protocol) [167], which are based on TCP and UDP, respectively.

For the NOS middleware, MQTT has been adopted, due to its maturity, stability and the fact that, after the recent adoption by the OASIS Consortium as an official standard[1], it is likely to become the *de facto* standard for IoT.

Regarding the second issue, adequate mechanisms have been defined in Paragraph 4.4, in order to control the flow of information and to enforce proper policies implementing specific rules for the management of resources and for handling users' preferences.

At this stage, NOS functionality is extended with AUPS (AUthenticated Publish&Subscribe system), which represents a new secure MQTT mechanism further integrated with the policy enforcement framework, presented in Paragraph 4.4. In this way, the authentication and authorization of data sources via MQTT is guaranteed. In more detail, a secure publish/subscribe system extending MQTT by means of a key management framework and a policy enforcement one is provided. Therefore, the flow of information in MQTT-powered IoT systems can be efficiently controlled by means of flexible policies. The AUPS approach is published in [168].

4.5.1 MQTT protocol

MQTT is a lightweight event- and message-oriented protocol, which allows the devices to communicate asynchronously across constrained networks to reach remote systems, as happens in the typical IoT/M2M scenarios. It is also a broker-based messaging protocol, designed and developed for constrained devices and bandwidth-limited communications by IBM/Eurotech in 1999 [87].

MQTT is based on a publish/subscribe interaction paradigm, developed following an event-based architecture, in which publishers publish structured events to an event service, usually called a *broker*, and subscribers show their interest in a particular event through subscriptions. These subscriptions can be custom patterns over the structured events. Subscription notifications by publishers are sent to all the interested subscribers, in order to prevent the publishers from needing to synchronize with subscribers.

In particular, MQTT has been implemented for easily connecting the "things" to the web and supporting unreliable networks with low bandwidth and high latency. This protocol employs a client-server pattern in

[1]`https://www.oasis-open.org/committees/mqtt`

which the server part is represented by a central broker that acts as intermediary among the clients (i.e., the entities that produce and consume the messages). All the communications, which happen among server and clients via a publish/subscribe mechanism, are based on the topic concept. A topic is a mean for representing the resources (i.e., the information) exchanged within the system. Topics are used by clients for publishing messages and for subscribing to the updates from other clients.

Due to its simplicity and low overhead, MQTT is suitable for resource-constrained environments and has found application in several domains, including monitoring applications, applications with live feeds of real-time data (e.g., RSS feeds), dissemination of events related to advertisements, support for cooperative working where users/applications need to be informed about events of interest, support for ubiquitous computing etc.

MQTT specifications are freely accessible. Open source MQTT implementations are available for all major IoT development platforms, for the two major mobile platforms (i.e. *Android* and *iOS*), and for several programming languages (*Java, C, PHP, Python, Ruby, Javascript*).

It is worth remarking that MQTT has been chosen in this thesis, since it represents the *de facto* standard protocol used by a variety of IoT and M2M systems. However, MQTT natively provides a very simple security model. In particular, the current version of MQTT (3.1.1) does not natively support either mutual authentication mechanisms or techniques able to guarantee the integrity and the confidentiality of the transmitted information.

In fact, for authentication of clients by server, the current MQTT 3.1.1 protocol specification only allows the use of a username and, optionally, a password. Developers can define customized authentication mechanisms using AES and DES as cryptographic primitives. Nevertheless, this is not sufficient to guarantee mutual authentication among clients and servers; the integrity and the confidentiality of the transmitted information is also left open. In more detail, for encryption and transport-level security, the Transport Layer Security (TLS) standard is recommended, although this is not always appropriate for resource constrained devices. Some implementations also support the use of a Pre-Shared Key (PSK) with TLS for authentication as well as encryption. Such a solution is definitely not suitable for highly dynamic IoT environments, since this strategy would require frequent session re-negotiations to establish new cryptographic parameters (i.e., change authentication credentials) among NOSs and the registered sources/users/applications. Therefore, a more lightweight credential management solution should be introduced, also with respect to temporary keys with the aim of improving the system resilience towards malicious attacks. This is the aim of AUPS.

4.5.2 MQTT technical aspects

Some key features of MQTT are the following:

- It provides one-to-many message distribution and decoupling of information of sources and consumers
- It is agnostic about the content of the payload
- It is built over TCP/IP protocols

TABLE 4.2: MQTT operations set

Primitive	Description
publish	used by a publisher to disseminate an event
notify	used by a subscriber to receive an event notification (a topic update)
subscribe	used by a client to subscribe to a specific topic
unsubscribe	used by a client to unsubscribe from a specific topic

- It has a small transport overhead.

MQTT, as in general all the publish/subscribe models, consists of a small set of operations, including the primitives pointed out in Table 4.2.

As discussed above, all these operations are mediated by a broker, which is responsible for dispatching the events from the publishers to the interested subscribers. Both centralized and distributed architecture implementations are available. Obviously, the simplest approach is the centralized one: in this case, every communication (i.e., from publishers to broker or from broker to subscribers) takes place through a series of point to point messages. However, the broker could become a bottleneck. To prevent such a situation, the broker can be replaced with a network of brokers that cooperate to offer their services. In the solution proposed in the dissertation, for the sake of simplicity, consideration is restricted to the case of multiple NOSs connected to one broker. This can easily be replaced by a plurality of brokers in larger scenarios.

As far as MQTT topics are concerned, they present the following features:

- They are represented as UTF-8 strings used by the broker to filter messages for each connected client
- They consist of one or more topic levels separated by a forward slash, forming a logical tree structure (e.g., a topic for publishing the temperature information of a sensor with identifier *sensorId* could be `sensor/sensorId/temperature`)
- They are used by clients for publishing messages and for subscribing to the updates from other clients, thus avoiding a continuous polling between producers and consumers
- It is possible to subscribe to an exact topic or to multiple topics at once by using the wildcards, which are represented by the following symbols:
  - +, for a single-level wildcard (i.e., exactly one topic level)
  - #, for a multi-level wildcard (i.e., an arbitrary number of topic levels)

Table 4.3 shows the MQTT message format, consisting of three parts: a fixed and a variable header, and a payload. For further details, refer to [17].

TABLE 4.3: MQTT message format

Component	Information
fixed header	message type, Quality of Service (QoS) level, some flags, message length
variable header	it depends on the application domain (information type)
payload	the data value referred to the topic

TABLE 4.4: MQTT QoS

Component	Information
at most once	messages are delivered according to the best effort of TCP/IP networks, so message loss can occur
at least once	messages are assured to arrive, but duplicates may occur
exactly once	messages are assured to arrive exactly one time

Concerning reliability, MQTT is based on TCP, so it provides standard TCP delivery reliability.

Furthermore, three levels of QoS are supported, as summarized in Table 4.4.

MQTT also supports persistence of messages (depending on the specific use case) to be delivered to future clients that subscribe to a topic, and may be configured to send messages of specific topics when the subscriber connection is abruptly closed. Such a configuration is established by the IoT system administrator on the basis of the IoT application requirements, and, regarding the NOS architecture, it is specified in the *Config* storage unit.

4.5.3 AUPS - AUthenticated Publish&Subscribe

Given the background about MQTT technical aspects and motivations, it becomes possible to detail AUPS behavior and functionality.

Note that, NOS southbound interfaces are based on the HTTP protocol, which offers in-built authorization and authentication functionalities, thus generating no noticeable security issues. The problems arises with the northbound MQTT interfaces, as specified above. An application/user wanting to use data from NOS can access it through a subscription to the relevant topic(s), handled by the MQTT broker. As expected, the resources are accessible only on the basis of the policies defined within NOS enforcement framework. Therefore, the MQTT broker has to interact with the underlying PEP on NOS in order to accept or deny subscription requests.

In such a context, the AUPS solution aims to propose a new system for the enforcement management of users/applications authentication and authorization policies through integration with MQTT mechanisms. The entities involved and the flow of information are shown in Figure 4.10. Four relevant actors are present in the system:

- *Data source*, which communicates data to NOS using the HTTP protocol

- *NOS*, which processes the data according to the procedures described in Paragraph 3.3 and publishes them under the relevant topic(s) to the MQTT broker
- *MQTT broker*, which notifies the interested subscribers of the new incoming data
- *User (or, equivalently, service)*, which accesses IoT-generated data through subscriptions to the MQTT broker.

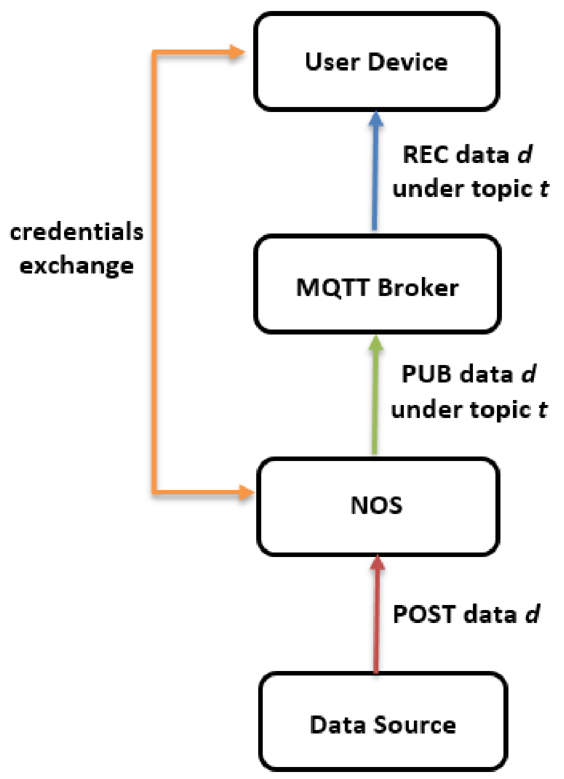


FIGURE 4.10: NOS high-level information flow

In particular, two transactions, strictly related to MQTT, have to be handled, for simplicity these are named *PUB* and *REC*: *PUB* represents the publication of new data to a topic by NOS; while *REC* represents the notification and reception of new published data to the subscribers.

Before detailing such operations, a block, named Key Topics Manager (KTM), has to be introduced. It is added as an external NOS component, which is able to interact with NOS by means of a secure HTTPS/SSL connection, in the same way as for the configurations contained in *Config* collection (i.e., managed by an external administrator). The KTM is in charge of managing temporary keys for topics access control. In particular, it handles a table structure with the following fields:

- *keyId*: the identifier of the corresponding key
- *keyT*: the actual key

- *val*: the expiration date of the key[2]
- *atb*: the attribute(s) owned by the users/applications allowed to access the resource. The key is associated with the correct users/applications on the basis of the attributes by the corresponding policies, expressed in PAP component and defined by the system administrator. In this way, the KTM is decoupled from the enforcement framework.

Note that the keys are not fixed, but they come with an expiration date, which is expressed in the field *val*. Moreover, the expiration times are not the same for all the generated keys. In this way, they are out-of-phase with each others, thus smoothing the system load. Since, as explained in Paragraph 4.4, the policies are global for all NOSs belonging to the IoT system, the key topics management must be global due to the attribute configurations. In this way, NOS synchronization in terms of active policies is also guaranteed. Moreover, as just introduced in Paragraph 4.4, the access control is based on ABAC; in fact, each user/application has to pass a registration phase before interacting with the IoT system, in which a set of attributes is assigned on the basis of the specific application domain (e.g., a manager and an employee of a financial company should have different attributes for accessing the resources of the company itself).

As regards operations, a *PUB* transaction is composed of the following steps, also represented in Figure 4.11. Note that, for the sake of simplicity, in the rest of the paragraph, the word NOS replaces references to the *NOS Core functionalities*, as shown in Figure 4.11. This is due to the fact that the enforcement framework belongs to NOS as a running module, but, at the same time, it is queried by proper *NOS Core functionalities* when needed.

1. NOS produces a new data item *d*, which has passed both the normalization and the analyzers phases (Paragraph 3.3)
2. The data *d* is assigned to a specific topic *t*
3. NOS queries the PEP in order to obtain the information useful for protecting the access to the resource represented by the data *d* relative to the topic *t*
4. The PEP queries the PDP in order to know which attributes *atb* a user/application has to own in order to access the resources represented by the topic *t*. The PDP returns to the PEP the information related to the attributes *atb* in accordance with the policies specified by PAP. Then, PEP can perform two actions:
   (a) If it owns a valid key for the access of users/applications with attributes *atb* to data published under topic *t*, then it sends *keyT*, along with the corresponding *keyId* and *val* to NOS
   (b) If it does not own a valid key for the access of users/applications with attributes *atb* to data published under topic *t*, it asks the KTM for a new valid key before sending to NOS *keyT*, *keyId* and *val*.

[2]This clearly requires synchronization among hosts.

Note that, once a PEP of a NOS has asked for a new valid key, it caches it locally. When the PEP finds out that a key is no longer valid, then a request to get a new one is triggered

5. NOS encrypts the data *d* with $keyT$, thus obtaining the encrypted data d_{enc} (Equation 4.13):

$$d_{enc} = enc(d, keyT, keyId), \tag{4.13}$$

   where any existing encryption mechanism $enc(\cdot)$ can be used; also *keyId* is used for the encryption operation in order to avoid the risk of duplicate information

6. NOS prepares the message for publication in the format specified by the following JSON syntax:

$$\begin{array}{l} \{ \\ \quad \text{“}keyId\text{”} : keyId, \\ \quad \text{“}data\text{”} : d_{enc} \\ \} \end{array} \tag{4.14}$$

   Note that the broker (or the brokers, in a totally distributed scenario) are in charge of guaranteeing the proper matching between encrypted data and assigned topic.

7. NOS MQTT client sends this information to the MQTT broker to be published under the specific topic *t* and to notify the interested subscribers

After the *PUB* transaction, the *REC* transaction takes place. Since the data *d* has been published, in encrypted form (i.e., d_{enc}), under the proper topic *t*, all the subscribers interested in the topic *t* have been notified about it. Supposing that a user device *u* receives a new notification (the case of an application is omitted, but analogous), the steps to be performed are the following, as also represented in Figure 4.12:

1. The user device *u* can perform two actions:
   (a) If it already owns a valid key for access to the resources specified by the topic *t*, then it can go to the final step
   (b) If it does not own a valid key for access to the resources specified by the topic *t*:
      i. *u* issues a request to NOS to access the information needed for decrypting the data d_{enc}
      ii. NOS queries the PEP, which in turn queries the PDP, in order to establish if the requesting user *u* with its attributes *atb*, is allowed to access to the resources of topic *t*; the response depends on the policies activated within PAP (and established on the basis of the attributes owned by the user as stated during the preliminary registration phase)
      iii. Then:

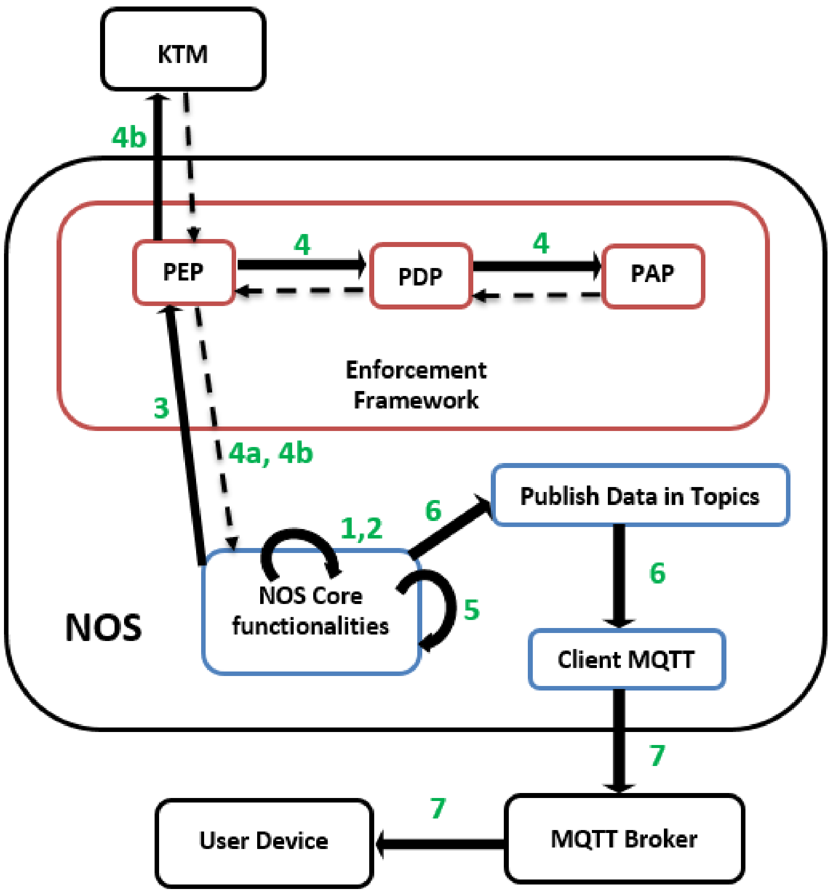


FIGURE 4.11: AUPS *PUB* transaction

A. If the response of the PEP is positive, then the PEP sends to NOS the information related to the key *keyT*, the identifier *keyId* and the validity *val*, which are then sent to u, encrypted with the key obtained by the user during the preliminary registration phase

B. If the response is negative, an error message is sent by PEP to NOS and, then, by NOS to u.[3]

2. Once in possession of the key *keyT* and the identifier *keyId*, the user (or service) u is able to decrypt the data d_{enc}. Note that a user does not have to request the key at each notification, since the key has a validity indicated by the timestamp *val*.

Summarizing, the adopted approach is able to effectively decouple aspects related to security from the usage of the MQTT publish/subscribe protocol; at the same time, network and computing resources are used effectively. Furthermore, the use of temporary keys and of user/application registration improves the system resilience to malicious attacks (e.g., man in the middle attacks, replay attacks, password discovery).

[3]Note that, if the PEP has not a valid key for the data belonging to topic t, it has to ask them to KTM, as described for *PUB* transaction. Moreover, the fields *keyT*, *keyId* and *val* are not sent in clear by NOS to u, but they are also encrypted with the key of the device u itself, established during the initial registration phase.

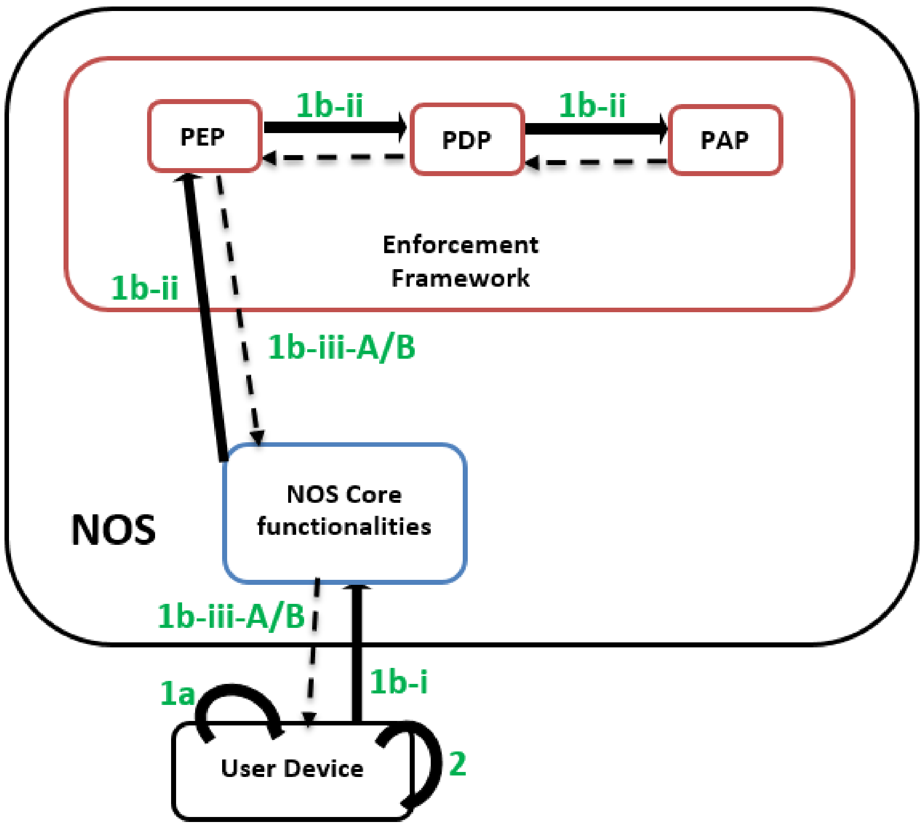


FIGURE 4.12: AUPS *REC* transaction

Paragraph 5.6 will present the evaluation of the overhead, latency and computational effort introduced by AUPS into NOS middleware.

4.6 Policy synchronization

Since, recently, information sharing has become a core requirement of the modern IoT applications, the new networking systems should support the management of data transmission across the boundaries of different domains, in order to integrate heterogeneous information to fulfill user and vendor needs, in terms of innovative services in real-time.

A crucial aspect still concerns the enforcement of the policies of the whole IoT system, which may require sychronization across multiple NOSs and heterogeneous application realms. Therefore, mechanisms should be integrated into NOS middleware, in order to control the flow of information and to enforce proper policies implementing specific rules for the management of resources, not limited to a single application scenario.

Such sychronization mechanisms should be flexible enough to support the wide range of technologies forming part of IoT infrastructures and the various application domains where users and devices could operate, as presented in Paragraph 4.4. Moreover, the aforementioned policies mainly regulate the access to resources, in order to deal with violation attempts. Note that policies mainly involve access to resources and are usually established by system administrators in accordance with the rules of each specific domain.

The definition of how information may be shared is a complex task and should be treated with a distributed approach because of the high number and the heterogeneity of entities (e.g., devices and users) and information involved in the IoT scenario.

In contrast, a centralized solution might not be sufficiently scalable [170]. In more detail, the main drawbacks of existing solutions are the followings: (i) they do not consider temporary changes during the activity of the system; (ii) they are not sufficiently flexible and scalable (i.e., they have not the capability of avoiding bottlenecks in the information flow; (iii) they do not adopt lightweight schemes for managing the various interactions.

In order to overcome such issues, NOS functionality is further extended with a distributed schema able to ensure the interoperability of policies in different application realms. Authorization and behavioral rules may be specified and synchronized among multiple networked NOSs, in order to determine conditions in which particular actions are permitted or must be executed by the different entities involved (i.e., NOSs themselves, data sources, users). The approach adopted is particularly relevant in certain scenarios, which require strict control over data and access to confidential resources, such as is found in healthcare applications, vehicular and military systems. The final contribution is presented in [184].

4.6.1 Policy synchronization algorithm

In order to obtain data from NOSs, users and applications employ the MQTT communication protocol mediated by a broker, as introduced in Paragraph 3.3. In particular, a user/application logs on to a service provided by the IoT system itself (e.g., an application running on user devices - smartphone, tablet, pc, etc.) and interacts through a proper GUI; as a consequence, a session is opened, during which the user/application can request the services provided by NOS on the basis of the accessible resources. The resources are accessible in accordance with the policies defined within NOS enforcement framework (see Paragraph 4.4). Moreover, the MQTT broker has to interact with the underlying PEP on NOS in order to establish which subscriptions to accept or deny, and enforce the current subscriptions themselves.

In the NOS system, policy distribution, update and synchronization tasks may be done by runtime configurations through the *Config* storage unit, which is in charge of dynamically updating the local PAP of each NOS. As a consequence, each NOS applies the policies currently specified in *Config* to the incoming data and to the user/application requests, via the MQTT broker (as shown in Figure 4.9).

Two main aspects have to be clarified: (i) which kinds of policies can be defined within NOS system and (ii) which entities decide how and when such policies have to be applied on data/requests. Firstly, as regards the set of policies, we distinguish them into two groups:

- P_{NOS}: policies related to NOS behavior, which includes, for example, the methods used for assessing the security and data quality properties (i.e., the behavior of the *Analyzers*), the format established for the normalized data (i.e., the behavior of the *Data Normalization* phase), the rate of data processing, etc

- P_{data_n}: policies concerning the access to the data themselves, which are strictly related to the topic assignment (i.e., which users or applications are allowed to access some topic on the basis of their attributes assigned during the NOS registration phase); such policies can be divided into *n* groups, where *n* is the number of organizations or companies in charge of managing the data provided by the sources. *n* may vary over the time, since organizations or companies may join or leave the IoT network. It is worth remarking that the scope of the proposed policy distribution and synchronization schema is not only tailored to a single scenario. Instead, it aims to be adopted by a multitude of companies, each one owning customized and shared policies. For these reasons we targeted our solution to more general organizations and we do not refer to user roles concerning a single application domain.

Therefore, multiple entities are involved in the policy definition. All of them are conceived as system administrators, but may be independent from each other (for example, they may belong to different companies or organizations). In more detail, we have:

- A_{NOS}, which represents the administrator of the set of policies P_{NOS}; such an administrator is unique and is responsible for the entire IoT system administration, since NOS behavior cannot be managed by parties involved in data provision (e.g., external companies could foster their business interests)
- A_{data_n}, which represents the *n* administrators of the resources; they define the policies to be applied to filter the access to the data processed and published by NOSs. Note that each A_{data_n} may belong to a diverse company or organization and is independent in terms of the resources provided and required policies. Also, hierarchy (e.g., subsets of policies) is permitted for the policies belonging to the same company or organization.

Once such distinctions among policies and administrators are clear, the main challenge is how to manage policy distribution to each NOS in an efficient way. The main issues to be faced regard the following aspects: (i) each NOS belonging to the IoT system has to be synchronized with each other in order to have the same behavior on the same data; (ii) policies may need to be updated over the time due to the provision of new data to NOSs or to changes in the company/organization resource disclosure.

The simplest solution would be the adoption of a central authority at the head of both A_{NOS} and A_{data_n}, able to intercept all the requests to add, update or remove policies and, then, propagate such changes to all NOSs, for example via a secure HTTPS or SSL channel. However, this may represent a bottleneck and does not ensure the real sychronization of all NOSs. In particular, a system of acknowledgments must be integrated in order to inform the central authority of the correct reception of the changes by all NOSs. Obviously, such a solution is not suitable for the wide IoT scenario, due to the huge amount of information and number of policies involved. Note that the introduction of a central authority would prejudice the main function of NOS, which is that of moving the data processing closer to the data source without delegating the activities to a central entity in charge of collecting and managing all the information transmitted within the IoT system.

Hence, a totally distributed approach is adopted, which exploits the MQTT connection protocol already integrated into NOS. The scope is to create different virtual channels in order to separate the management of the sets of policies P_{NOS} and P_{data_n} and to allow efficient transmission among NOSs and adminitrators. In particular, the following virtual channels are introduced:

- C_{data}, which is, in the previous version of NOS [193], the only MQTT channel used for publishing and notifying the processed data to interested users/applications

- C_{topic}, which aims to allow the administrators of the group A_{data_n} to add/remove/update the policies related to the information belonging to the topics under their authority. This channel is also based on MQTT

- C_{NOS}, which forms a sort of private channel among NOSs themselves and A_{NOS} used for security purposes, as clarified in the following. In fact, such a channel is used for checking the policy synchronization and is based on a secure HTTPS/SSL protocol.

Two aspects are worthy to remark: (i) C_{data} and C_{topic} coincides on a practical level, since both exploit the existing publish/subscribe mechanism, but they are separated from a conceptual point of view; (ii) all the interactions are mediated by the broker; in this case it is assumed that there is one broker to which a various number of networked NOSs are connected.

Figure 4.13 outlines the proposed schema including all the involved entities. Blue indicates the communications that happen within the virtual channel C_{data}; in fact the blue uni-directional arrows represent the route of the information processed by NOSs from their publication to the MQTT broker towards the final notification to the end users/applications. The access to such data is regulated by the policies applied to the topics associated with the information themselves, which are managed by the enforcement framework running on each NOS (see Paragraph 4.4).

How the policies are propagated to NOSs is handled by the C_{topic} virtual channel. In detail, all NOSs agree, in a preliminary phase (i.e., before NOSs deployment), with the A_{NOS} administrator on a particular topic *t* (e.g., *NOS/policy*); this is used for publishing the policies related to the adding/removing/updating operations, both by A_{NOS} itself or by the A_{data_n} administrators. Access to *t*, which may be structured in a proper hierarchy in order to obtain full expressiveness and flexibility, is restricted to NOSs, which are the only entities notified of this kind of information and able to know its content (which is transmitted in an encrypted form, by means of a proper algorithm agreed by NOSs and A_{NOS}). Therefore, users and applications are prevented from accessing the policies, despite the same "physical" channel (C_{data} correspond to C_{topic}) being used for the

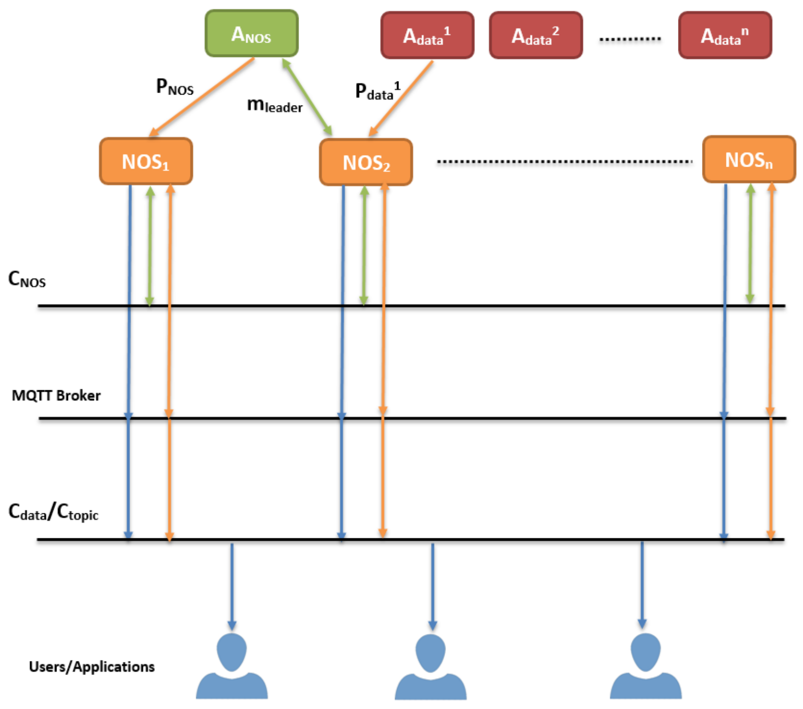


FIGURE 4.13: Policy sychronization management schema

notification of the data to which they are subscribed. In Figure 4.13, the orange arrows show the propagation of policies, which consists of the following steps:

1. A_{NOS}, for the P_{NOS} policies, and A_{data_n}, for the P_{data_n} policies, communicate to one NOS (chosen randomly at each time or on the basis of information about NOS locations), via a secure HTTPS/SSL channel, the existence of a new policy for a certain resource or the update of an existing one. Note that such administrators may also expose *RESTful* services to NOS for policy transmission

2. NOS stores or updates such a policy in the *Config* storage unit. As a consequence, the enforcement framework automatically starts to modify its behavior towards the specified resource accordingly. Note that this is an important feature of the proposed NOS modular architecture, which allows time and computational effort to be saved, since the system is able to be re-configured without re-starting or modifying the modules themselves

3. NOS has to publish the new policy or the update to the MQTT broker under the agreed topic t

4. All other NOSs are allowed to access the information under the topic t and, then, are able to store or update the policy.

From a practical point of view, all NOSs are subscribed to the topic t; therefore they are notified at each new incoming event related to t. In fact, the orange arrows are bi-directional, since a NOS may either publish an information to the MQTT broker or be notified about a new information.

Obviously, such data are not transmitted in clear over the network, but are ciphered by means of a proper encryption schema (e.g., RSA, PKI, etc.) shared by all NOSs and established, for example, before NOS deployment, as discussed above.

Summarizing, exploiting the MQTT publish/subscribe protocol potentialities, NOSs are able to manage and update the policies of heterogeneous companies or organizations in a distributed and lightweight way, without the need for a central coordinator, which may represent a single point of failure. Moreover, no additional modules have to be added to the previous NOS version, since the existing MQTT mechanism is used directly.

The last aspect to be considered is that of policy synchronization, because the IoT system, and, in particular, the A_{NOS} administrator, has to be able to know if all NOSs share the same policies at a certain interval of time. To this end, the C_{NOS} channel is introduced. Periodically, A_{NOS} selects a NOS as a leader. This choice can be made by means of one of the well-known algorithms available in literature for leader selection [106], which represent a well-investigated subject in distributed systems. In this thesis, the case in which A_{NOS} directly selects the leader (i.e., as a central server) is consider, and the case of NOS leader election is left as a future extension.

Examining the synchronization process, one finds the following steps, highlighted in Figure 4.13 with green arrows:

1. A_{NOS} elects a leader L_{NOS} among the networked NOS by sending a message m_{leader} only to the selected NOS via the secure HTTPS/SSL channel, just previously presented; the leader should change periodically in order to increase the robustness of the IoT system in case of link failure

2. When the leader L_{NOS} receives a policy notification, besides performing the publication to the MQTT broker, it sends an advertising message m_{id} to the other NOSs through the secure C_{NOS} channel, where *id* represents a progressive identifier chosen by L_{NOS} to identify the policy currently under the synchronization process:

$$\begin{aligned} &\{ \\ &\quad \text{``}NOS\text{''} : L_{NOS}, \\ &\quad \text{``}policyId\text{''} : id \\ &\} \end{aligned} \tag{4.15}$$

3. All NOSs must reply to such an advertisement with a simple acknowledgement message r_{id}, using the secure C_{NOS} channel:

$$\begin{aligned} &\{ \\ &\quad \text{``}NOS\text{''} : NOS_{id}, \\ &\quad \text{``}policyId\text{''} : id \\ &\} \end{aligned} \tag{4.16}$$

4. L_{NOS} verifies the reception of r_{id} from all NOSs; if one or more responses are missing, then L_{NOS} sends a report to A_{NOS}, which could take some actions (e.g., send other advertisements or do not consider the unsynchronized NOS for leader selection).

Note that some important parameters of the synchronization process have to be considered and customized depending on the number of NOSs and connected sources and on the data load. Such parameters include: (i) the rate of leader selection; (ii) the time waited by the leader for the responses by NOSs. These fall into the issue of determining the global state of a system in a certain interval time, which is also a well-investigated field in distributed environments [37].

The performance of the synchronization system will be investigated in Paragraph 5.7 in terms of execution time and computational load.

Chapter 5

Validation and Evaluation

5.1 Introduction

NOS functionalities were extensively presented in Chapter 4. They include the following features, which are validated during this chapter:

- Two key distribution and management systems originally conceived for WSN have been adapted for NOS to allow more secure communications among NOS and users/data sources; in Paragraph 5.3 such approaches are compared in terms of storage occupancy, overhead, delay, and robustness towards malicious attacks to reveal which is the most suitable for IoT applications
- A new algorithm for the assessment of QoP properties was introduced and formalized; in Paragraph 5.4, it is validated, along with DQ aspects, in order to demonstrate its capabilities fro performing an effective evaluation on data incoming to NOS
- A policy enforcement framework has been integrated into NOS architecture aiming to regulate and control the behavior of the IoT system, in response to actions and/or unexpected changes of the target environment; Paragraph 5.5 provides a real application scenario in which the behavior of the enforcement framework itself is clarified, also considering a variety of possible attacks
- In order to secure the information sharing, a lightweight authenticated publish/subscribe system, named AUPS, was added to NOS and further integrated with the aforementioned policy enforcement mechanism; the evaluation of performance metrics such as overhead, latency and computational effort is presented in Paragraph 5.6
- Policy across different application domains are efficiently synchronized by means of a proper system, whose performance is shown in Paragraph 5.7.

The forthcoming paragraphs are dedicated to the description of the outcomes of the experiments carried out in order to validate the proposed solutions in the field of security in IoT. Before presenting the measures and the use case examples, the experimental setup is detailed in Paragraph 5.2.

5.2 Experimental setup

NOS has been implemented as an IoT middleware, resulting in a real prototype, by means of *Node.JS* platform for NOS core functionalities, *MongoDB* for storage management, and *Mosquitto* for the publish/subscribe broker mechanism, as outlined in Paragraph 3.4.

In the experimental setup, NOS platform is deployed on a *Raspberry Pi*, which provides the computational and resources capabilities for running the NOS functionalities. To simulate the behavior in a real-world setting, it is connected to a number of open data feeds. Then, in order to verify the effectiveness of the proposed solution, simple use cases are developed and presented in the next paragraphs, on the basis of the usage of such IoT feeds.

In particular, these data are provided in real time from six sensors, measuring weather-relevant parameters and co-located within the meteorological station in the small town of Campodenno (Trentino, Italy); they can be accessed through the *Trentino Open Data* portal (`http://dati.trentino.it/dataset/raw-data-in-near-realtime-stazione-cmd001`). The measurements cover temperature, humidity, wind speed, energy consumption and air quality parameters.

In more detail, a laptop is used to emulate the behaviour of a set of nodes, basically reading data from the aforementioned feeds and sending them to the NOS, as if they came from six different nodes. Laptop and *Raspberry Pi* communicate via a WiFi network. A web service exposes them in JSON format, and NOS retrieves them through *HTTP GET* requests. Processed data are then transmitted to an MQTT broker.

The laptop also runs a simple visualization service, which fetches, according to user-defined constraints, data from NOS and displays them. The user can express constraints in terms of the required QoP and DQ levels, including aspects such as confidentiality, integrity, privacy, authentication, completeness, timeliness, and accuracy, as shown in the dashboard in Figure 5.1.

5.3 Key management system comparison

As regards the two key management approaches by Dini et al. and Di Pietro et al., presented in Paragraph 4.2, they have both been implemented in the NOS prototype, in order to make a performance comparison.

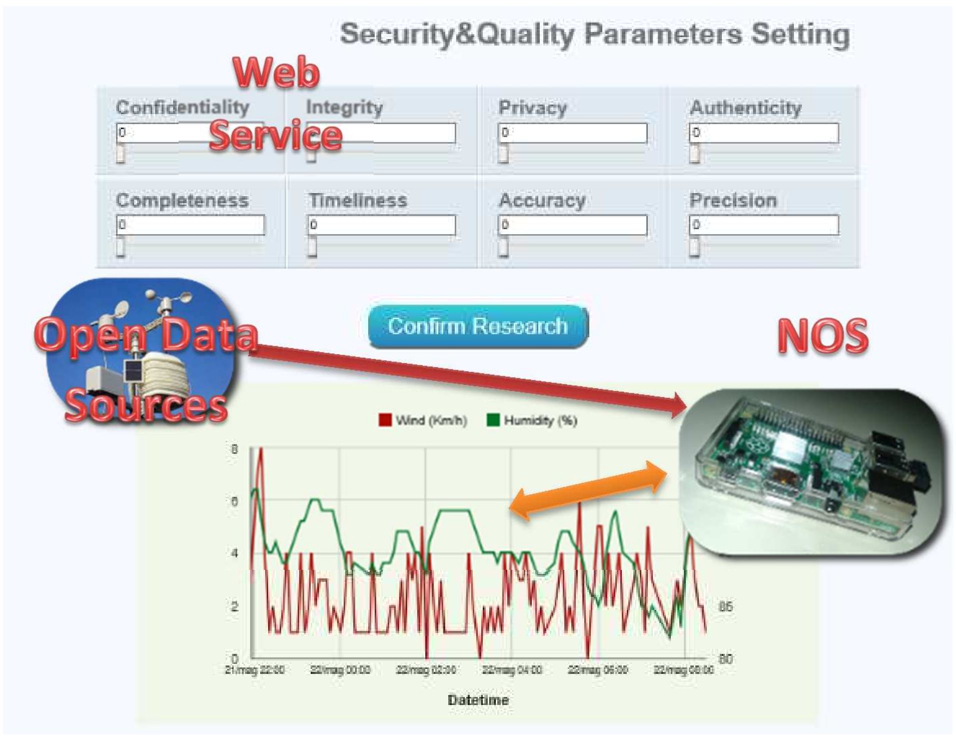
To this end, besides distinguishing the data sources as registered and non registered, two different scenarios are considered. The former includes two registered and four non registered sources; while the latter includes three registered sources and three non registered ones.

The overhead, the delay and the degree of robustness towards malicious attacks are analyzed in the following paragraphs.

5.3.1 Overhead

The overhead analysis takes into account three crucial metrics regarding:

- Execution time
- Storage capacity


Security&Quality Parameters Setting

Confidentiality	Integrity	Privacy	Authenticity
0	0	0	0

Completeness	Timeliness	Accuracy	Precision
0	0	0	0

FIGURE 5.1: User dashboard

- Processing effort required by the two algorithms for the management of the registered sources (i.e., key distribution, information storage, update messages).

As regards the average execution time *T(s, z)* of Dini et al. and Di Pietro et al., where s is the number of sources and z represents the number of parallelizable operations to be performed, refer to Equation 5.1:

$$T(s, z) = T_{seq} + \frac{T_c(z)}{s} + T_0(s) \tag{5.1}$$

Where:

- T_{seq} corresponds to the execution time of the non-parallelizable sections of the algorithm
- $\frac{T_c(z)}{s}$ is the execution time of the parallelizable sections of the algorithm (i.e., executable in parallel by all the sources)
- $T_0(s)$ represents the communication time spent among the sources for the information exchange.

For the first scenario, s is equal to two. In Dini et al. there are no parallelizable operations, therefore z is equal to zero. Hence:

- T_{seq} has been evaluated to be equal to 28 ms
- $\frac{T_c(0)}{2}$ is equal to zero, since no operation can be executed in parallel

- $T_0(2)$ has been evaluated to be equal to 4 ms.

The final execution time *T(2, 0)* is, then:

$$T(2,0) = 28 + 4 = 32ms \tag{5.2}$$

A similar result is obtained for the second scenario, with *s* equal to three. In general, the overhead depends on the number of hops required for the key propagation from NOS towards the most distant source.

Regarding the storage capacity required from the sources to execute the algorithm of Dini et al., the information stored in the connection and global key tables *CT* and *GT* are the following:

- Two bytes for the global key name
- Eight bytes for the global key value
- One byte for K_{incr} for each local key
- In the first scenario, K_{node} is equal to one byte, therefore the name of the local key requires two bytes of memory; while, in the second scenario, the name of the local key requires three bytes of memory (one byte for K_{incr} and two byte for K_{node})
- Eight bytes for the value of the local keys.

Summarizing, with two registered sources, 20 bytes of storage are required; while, with three registered sources, this increases to 21 bytes. Note that this metric is relevant, especially in case of constrained devices.

Concerning the size of the messages transmitted during the execution of the algorithm, each message for the initial local key generation is 10 bytes in length, for the first scenario, and 11 bytes in length for the second; while each message generated by the re-keying operation requires 18 and 19 bytes for the first and the second scenario, respectively. Note that, the execution of re-keying is expected to generate a response message, in order to verify the correct key replacement; such a packet has a size of one byte.

Finally, as regards the computational overhead, NOS performs the following operations:

- Two concatenations, in order to establish the name of the key (i.e., by concatenating K_{node} and K_{incr})
- Computation of the key value (which depends on the adopted encryption scheme).

Each source, upon receiving a new local key, has to perform:

- Two decryption operations, in order to know the encrypted content of the message containing the new key sent by NOS (i.e., K_{node} and K_{incr}).

Referring to the same overhead metrics in the algorithm of Di Pietro et al., the session key generation is a parallelizable operation, therefore:

- T_{seq} has been evaluated to be equal to 4 ms and 8 ms for the first and second scenario, respectively; this is the time required for the update of the source information in the local NOS storage unit, named *Sources* (see Paragraph 3.3.2)
- $\frac{T_c(1)}{2}$ has been evaluated to be equal to 3 ms for both the scenarios
- $T_0(2)$ is equal to zero, since no message exchange is performed.

The final execution time *T(2, 0)* and *T(3, 0)* are, then:

$$T(2,1) = 4 + 3 = 7ms \tag{5.3}$$

$$T(3,1) = 8 + 3 = 11ms \tag{5.4}$$

Concerning the storage required by Di Pietro et al., the two seeds (which can be arbitrarily long q bits) and the two keys of 8 bytes have to be stored at execution time (the actual key and the previous one, due to the time taken by the update phase, as discussed in Paragraph 4.2.2).

At the computational level, the required operation are:

- One decryption, in order to know the encrypted content of the initialization message sent by NOS
- Two hash functions, in order to generate the key value
- One XOR operation.

From such an analysis, it can be concluded that the algorithm of Di Pietro et al. presents better perfomance with respect to Dini et al., because:

- Some operations can be executed in parallel, thus reducing the execution time
- No message is exchanged among the sources, but the communications only take place with NOS (without intermediate sources), thus limiting the traffic in the IoT system
- The storage required by Dini et al. is influenced by the number of registered sources, therefore it may be more affected by scalability issues then Di Pietro et al.

5.3.2 Delay

The delay introduced by the algorithms of Dini et al. and Di Pietro et al. has been evaluated as the time required for the key generation, propagation and update.

Figure 5.2 shows the comparison between the two algorithms. In Dini et al., the key generation is more expensive (about 7 ms for the global keys, 10 ms for the local keys) than in Di Pietro et al. (about 3 ms), due to the more complex operations to be executed. However, such a delay, for Dini et al., will remain unchanged with respect to the increase of the number of sources, since the initial keys are assigned to the sources only once, during the registration, by NOS. Instead, for Di Pietro et al., the result will increase depending on the number of registered sources, since they are directly responsible for key generation.

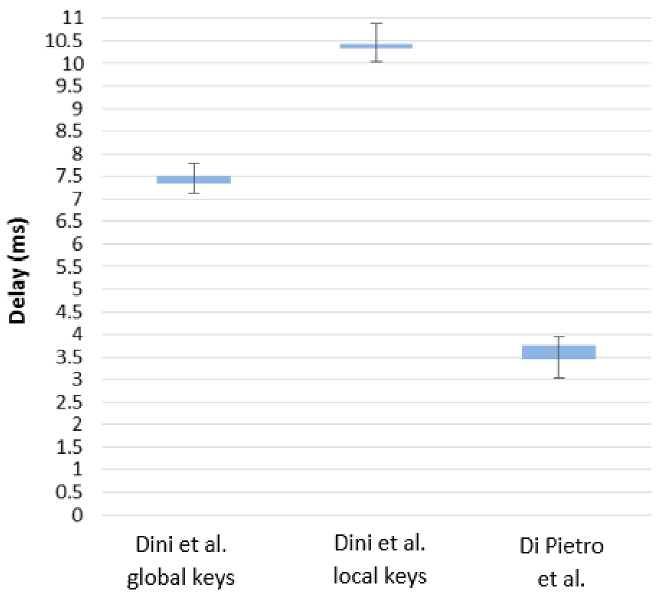


FIGURE 5.2: Delay of key generation of Dini et al. and Di Pietro et al.

As regards the key propagation, only the Dini et al. approach is considered, since Di Pietro et al. does not include any key communication among the data sources. The results are shown in Figures 5.3 and 5.4 in the two scenarios and take into account three different situations, respectively:

- The key propagation happens with a direct connection between the sources, by means of a shared global or local key
- The key propagation happens through adjacent nodes, also by means of a shared global or local key
- The last analysis concerns the case of unreachable sources (i.e., the key propagation does not succeed).

Finally, the delay in key replacement has been studied. In both the algorithms, the results depend on the number of registered sources (see Figure 5.5). Dini et al. approach presents a higher delay with respect to Di Pietro et al., due to the time required for key propagation; while in Di Pietro et al. the sources independently generate the new keys.

5.3.3 Robustness towards malicious attacks

It is worth remarking that, in Dini et al., NOS is responsible for the global and local key generation and for their replacement; while, in Di Pietro et al., the keys are independently generated by the sources after the initialization and distribution of the seeds, made by NOS during the registration phase.

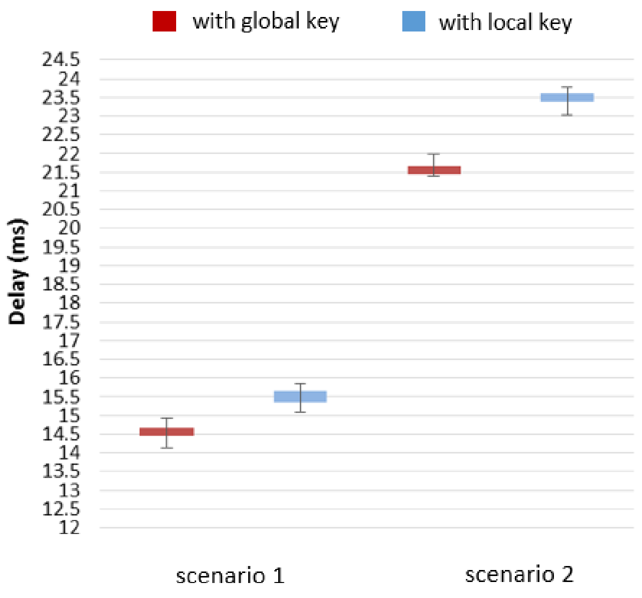


FIGURE 5.3: Delay of key propagation in Dini et al. with direct connections

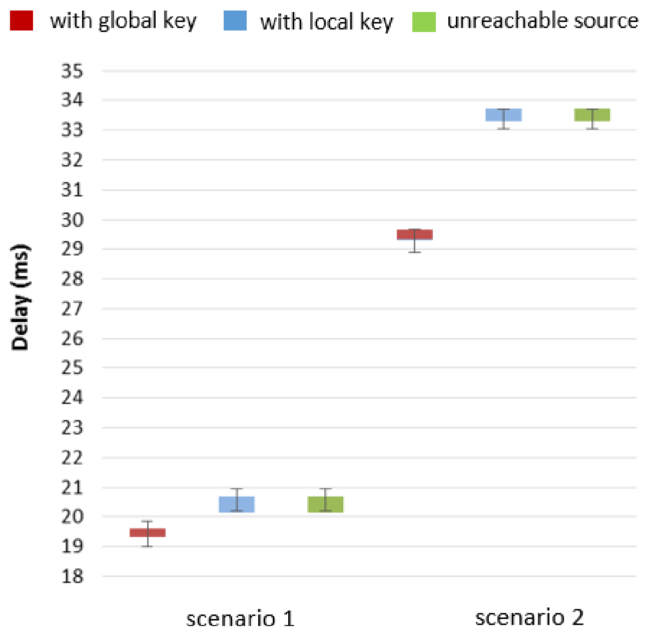


FIGURE 5.4: Delay of key propagation in Dini et al. through adjacent nodes

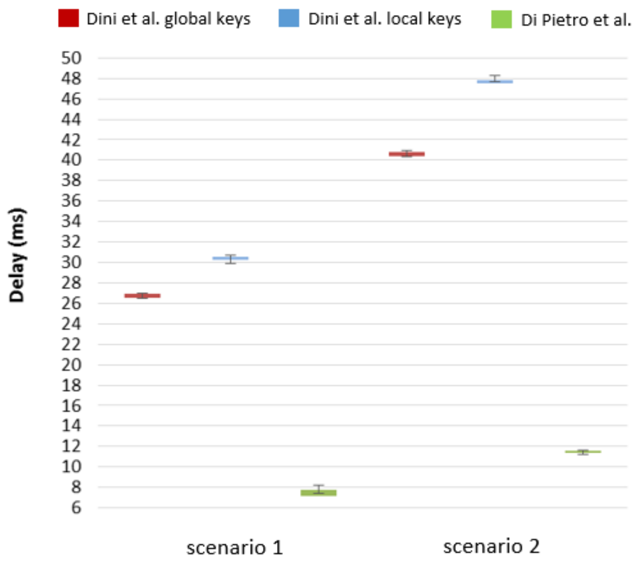


FIGURE 5.5: Delay of key replacement of Dini et al. and Di Pietro et al.

In order to compare the robustness of the two key management methods towards malicious attacks, it is important to distinguish between:

- External attacks, performed by entities acting outside the IoT secure network (e.g., non registered sources)
- Internal attacks, performed by the sources involved in the secure communications within the IoT infrastructure (e.g., registered sources).

In the remainder of this paragraph, such threats will be investigated.

External attacks

Examples of external attacks are man-in-the-middle, identity spoofing, Denial of Service (DoS) and so on. Concerning the algorithm of Di Pietro et al., the session keys are not exchanged among the registered sources and NOS; in this way, they are prevented from being discovered by an external node. Therefore, the only way to succeed with such a kind of attack would consist of: (i) discovering the seeds and (ii) computing the current session key. As regards the seeds, they are sent in clear by NOS to the sources during the registration phase. This aspect represents the main drawback of the Di Pietro et al. approach. In fact, if the seeds exchange is eavesdropped by a malicious entity, then the communications with the source may be effectively compromised. Computing the session key is a more complex task, since it continuously changes over time. Analyzing the traffic within the network, a malicious entity may recognize the frequency of re-keying, but it is not useful for obtaining the current session key.

As regards the Dini et al. approach, the algorithm of Eschenauer and Gligor [54] was adopted for the key initialization phase; this implies the use of a random key pre-generation scheme before any source can start to send or receive data to or from NOS. Then, during the registration phase, each incoming source receives a global key from NOS, from the large pool of keys generated by NOS after the aforementioned initialization phase. In this context, it should be pointed out that the value of a global key cannot easily be derived from the value of another global key, but if one of them is eavesdropped, then a part of the network may be compromised. In order to partially cover this issue, Dini et al. introduce the use of the local keys as a second level of security to the whole system; such a solution should reduce the possibility of compromising numerous sources as a consequence of the same attack. All these aspects make the approach of Dini et al. more robust than that of Di Pietro et al., which is only based on a single key level.

Furthermore, in Dini et al., an external source could eavesdrop a packet (i.e., a sort of packet sniffing attack) containing the frequency of re-keying and the name of the local key (without its value). However, in order to obtain the value of the local key, the attacker should know the encryption algorithm, used for ciphering the key itself, and the value of the global key. The only way to obtain such information is compromising NOS itself, but this case is not considered.

Note that, while a non registered source may be easily compromised (i.e., data integrity violation, confidentiality) and cannot interact in a secure way with the other sources, a registered one, with the adoption of the algorithms of Di Pietro et al. or Dini et al., should provide more reliable data to NOS, thus improving the robustness of the whole IoT system.

Several drawbacks are still open. In fact, both the approaches do not consider possible physical attacks on the data sources. In this case, in Di Pietro et al., a malicious node could access the seeds and the session number, thus allowing the generation of valid session keys; this action allows the attacker to decrypt all the data transmitted by the compromised sources; while, in Dini et al., an attacker could directly steal the global and local keys, thus hindering all the communications in a certain area.

Internal attacks

So far, attacks from entities acting outside the IoT network have been considered. The last analysis presented addresses the resilience of the two algorithms towards internal attacks carried out by a registered (i.e., internal) source. In their work, Dini et al. and Di Pietro et al. do not directly cover key revocation in response to an insider attack, and also in NOS implementation, the case of internal attack is not considered.

However, a preliminary analysis suggests viable solutions, which could be counted as a future extension for further evaluation. In general, if NOS recognizes misbehavior from a registered source, then it must be isolated from the system, by revoking its keys. Concerning the solution proposed by Di Pietro et al., the only way to prevent a source communicating in a secure manner with the IoT system may consist of revoking the two seeds. Obviously, it is an error-prone task, since it implies that NOS updates the seeds of all the other sources which share the same seeds of the misbehaving one (and then re-synchronize the generated keys). Another issue is that the data transmitted during the key update, and encrypted with the compromised key, may be freely accessible by malicious sources. Instead, in Dini et al. the isolation of a source by revoking the keys it uses (either local

TABLE 5.1: Validation of QoP and DQ assessment algorithms - Sources parameters

Sources	Authentication	Encryption technique
Source 1	IPSec	RSA
Source 2	16-bit password logon	AES
Source 3	none	MD5
Source 4	8-bit password logon	SHA-1
Source 5	none	none
Source 6	8-bit password logon	none

and global) is simpler, since NOS itself starts the key replacement procedure and could inform the involved sources to not share the new keys with the misbehaving ones.

Summarizing, the algorithm of Dini et al. presents better robustness against network traffic and internal attacks with respect to Di Pietro et al. Both the approaches are vulnerable to physical attacks to the sources. In general, the two solutions ensure a high level of confidentiality and integrity to the data transmitted by registered sources. Note that the overall resilience of the IoT system may vary depending on the number of sources and the robustness of the encryption schemes adopted by Dini et al. and Di Pietro et al. for ciphering the information, respectively.

5.4 Validation of QoP and DQ assessment algorithms

According to the algorithms presented in Paragraph 4.3, NOS processes incoming data by assessing their reliability in terms of QoP and DQ properties.

Since meteorological data are obtained from six different sources in real time, as previously specified (see Paragraph 5.2), it is assumed that each of them adopts different authentication and encryption methods for communicating with NOS (eventually agreed during the registration phase), as summarized in Table 5.1.

The evaluation of the data provided, both in terms of security and quality, is performed by NOS system following, for DQ assessment, the methods presented in Paragraph 4.3.1; while, for QoP assessment, the ones provided in Paragraph 4.3.2, returning a score for each metric.

For testing the effectiveness of the proposed mechanisms, the system was observed for a period of a week. Figures 5.6 (for QoP) and 5.7 (for DQ) show the day-by-day evaluation regarding the data provided by the six sources.

For security assessment, the outcomes are consistent with the expected robustness of the authentication and encryption techniques adopted by the monitored sources. Note that each score is initially set to the maximum value (i.e., 1), as specified in Paragraphs 4.3.1 and 4.3.2, and that a source may change its communication agreement with NOS, thus modifying the corresponding security assessment scores.

From the figures, it can be observed that NOS can interact with sources characterized, for example, by a good level of authentication, but, at the same time, by a low level of reliability in terms of confidentiality, integrity and privacy (e.g., source 6 in Figure 5.6) and vice versa (e.g., source 3 in Figure 5.6). This could happen if the data, acquired by a source, are corrupted during their transmission to NOS. Another case is that the data provided by a source may present good levels of completeness and timeliness, but poor accuracy and precision levels (e.g., source 4 in Figure 5.7).

The just presented evaluation confirms the potential of NOS approach in empowering the user with the ability to choose the requirements to be met by the data used by a given IoT-enabled service.

In order to better clarify the effectiveness of the new security evaluation algorithm, proposed in Paragraph 4.3.2, Figure 5.8 shows an analysis of the behaviour of source number 6 with respect to the integrity score in particular conditions. From Figure 5.6 it emerges that such a source obtains low values for data integrity (at day 7 the integrity score is equal to 0.2).

Now, the whole set of attack-countermeasure pairs, referring to the integrity of data of source number 6, are considered and remain constant for the whole observation period, except for one pair, which correspond to a man-in-the-middle-attack, able to modify the data content and, therefore, to violate the integrity of the information transmitted by the source itself.

As reported in Table 5.1, source number 6 does not adopt any encryption technique, therefore the man-in-the middle attack simulated within the network is successful. As a consequence, the integrity score reduces. However, if at day 2 the source decides to adopt AES for encrypting its data, sharing the proper keys with NOS, it is expected that the security algorithm changes the integrity evaluation accordingly. Figure 5.8 represents the integrity score assessment in the "normal" case (i.e., the source does not change its communication agreements with NOS) and in the "modified" case. Note that, at day 6, source number 6 revokes the use of the encryption scheme with NOS again sending the data in clear; as shown in Figure 5.8, the integrity score again starts to reduce.

These results represent an example of how NOS may interact with IoT data sources, classifying the received information accordingly. Moreover, it shows how NOS, although it does not directly tackle attacks able to compromise the IoT devices, recognizes the possible threats for each data source during a live attack and, consequently, estimates the corresponding levels of QoP. In this way, users can select the data with a deep level of awareness about the services offered by the IoT system.

5.5 Policy enforcement framework evaluation

The behavior of the enforcement framework has been evaluated in terms of memory occupancy, latency due to the execution of the enforcement operations, and, finally, with respect to the application of the policies themselves, even in case of violation attempts.

For such an analysis, the six data sources are associated with the QoP and DQ scores reported in Table 5.2, which were obtained by means of the algorithms presented in Paragraph 4.3.

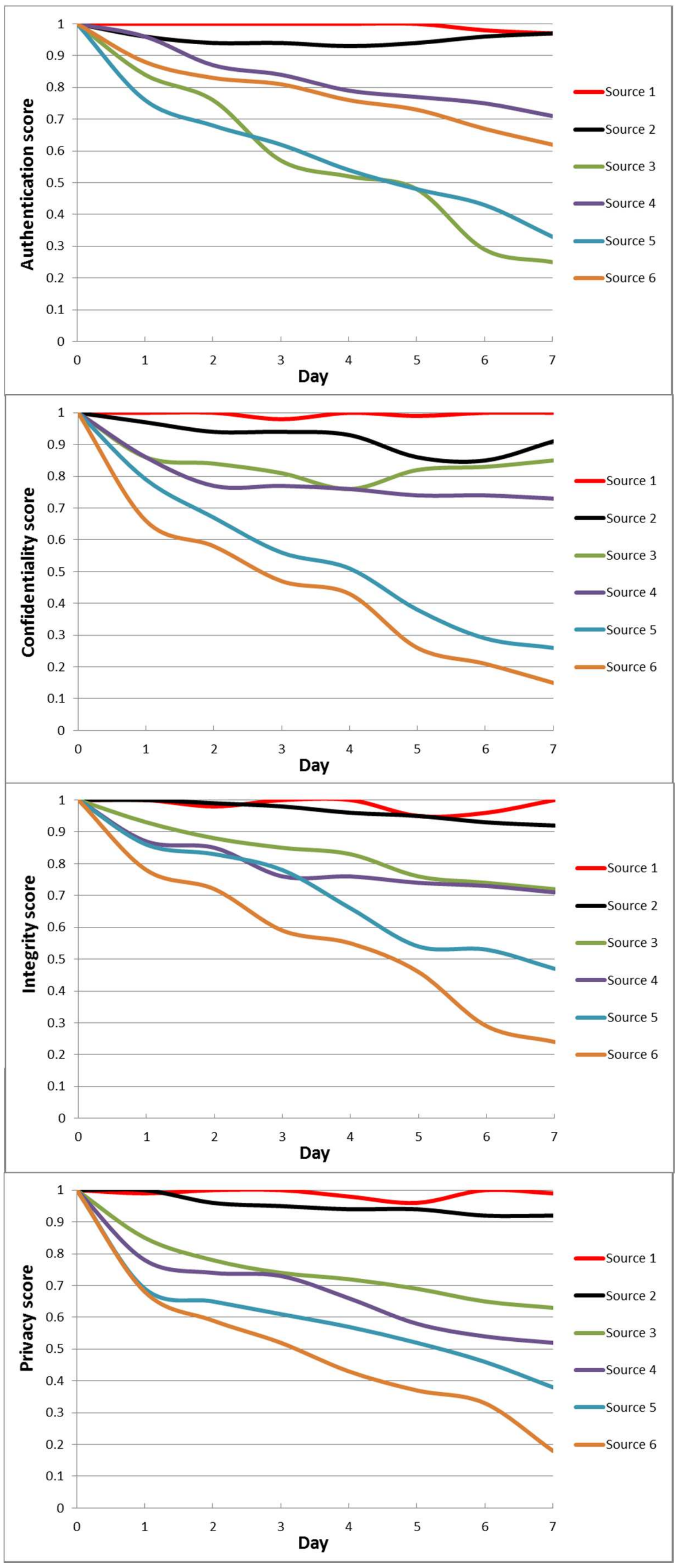


FIGURE 5.6: QoP score assessment

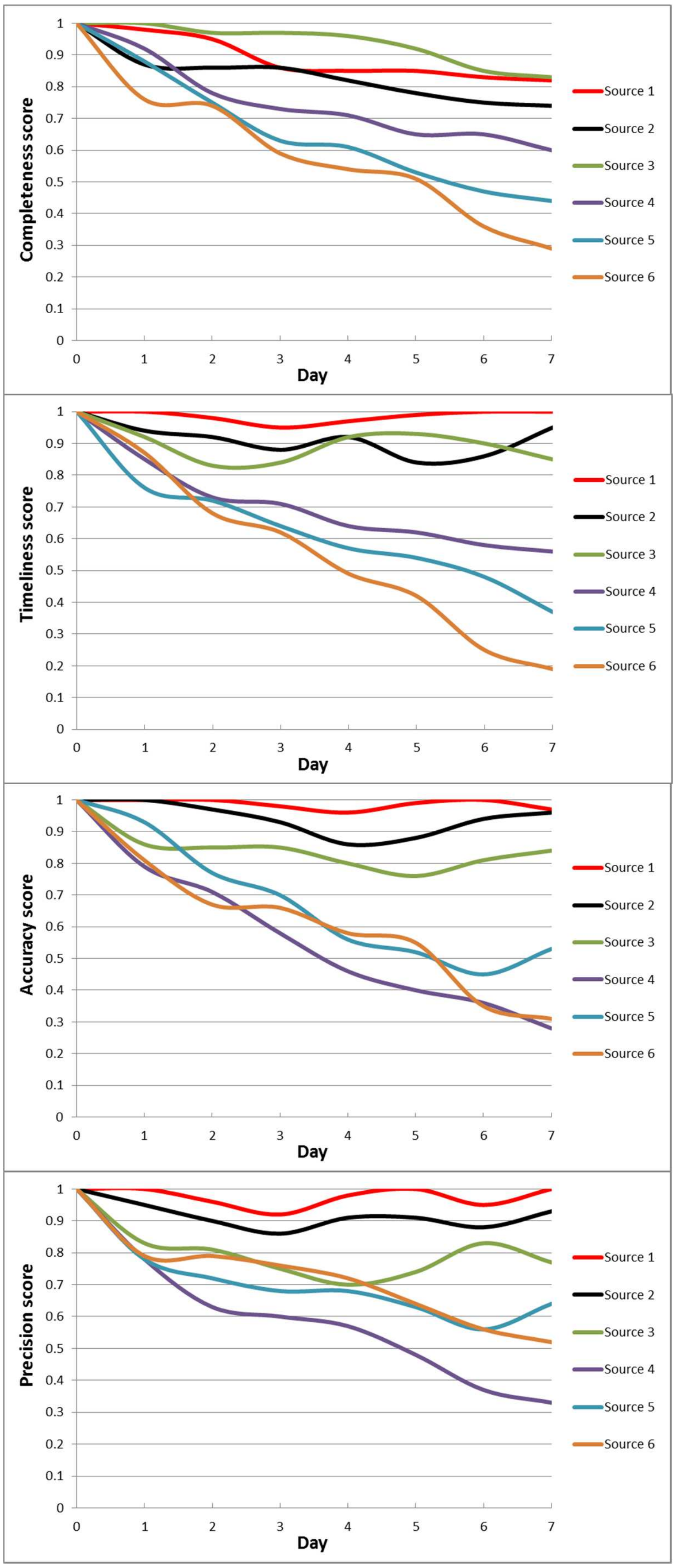


FIGURE 5.7: DQ score assessment

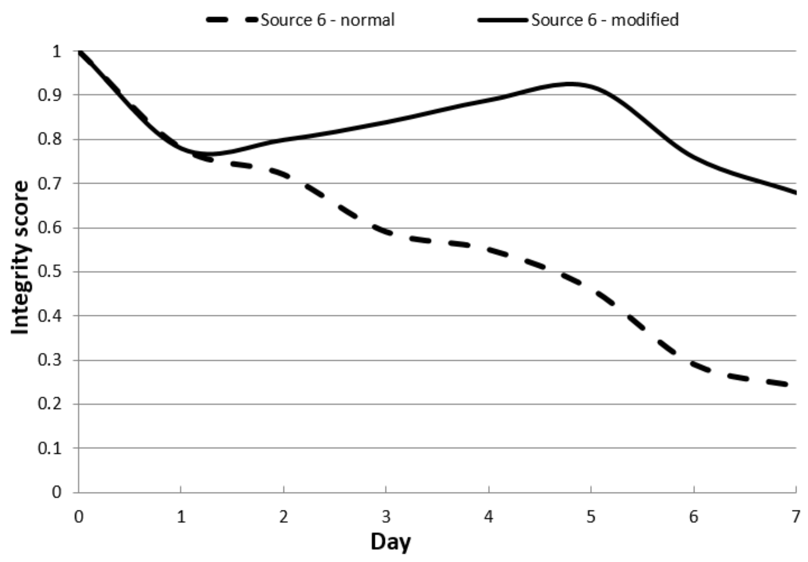


FIGURE 5.8: Attack-countermeasure analysis - source integrity

TABLE 5.2: Policy enforcement framework evaluation - sources parameters

Parameters	*Source 1*	*Source 2*	*Source 3*	*Source 4*	*Source 5*	*Source 6*
Authentication	0.8	0.4	0	0.2	0	0.2
Security schema score	1	0.6	0	0.2	0	0
Privacy schema score	1	0.6	0	0.2	0	0
Timeliness	0.9	0.8	0.6	0.3	0.9	0.7
Completeness	0.9	1	0.8	0.6	0.7	1
Accuracy	0.9	0.7	0.5	0.6	0.7	1
Precision	0.9	0.8	0.4	0.5	0.8	0.9

5.5.1 Storage

A first carried out analysis concerns the storage capacity required by the system for carrying out its operations. In this respect, it is worth recalling that NOSs do not support persistent storage of IoT-generated data. Rather, data are temporarily cached on NOS itself while being processed, before being submitted to the MQTT broker. Therefore NOS has to provide only temporary storage. When data are further pushed to or pulled from the server that handles the topic notification to subscribers, data can be safely flushed from NOS.

In this prototypical implementation, the in-memory capability of *MongoDB* for *Raw Data* and *Normalized Data* collections has been used; the *Config* and *Sources* databases are persistently stored on the local hard disk (see Paragraph 3.4).

Since NOS runs on a *Raspberry Pi*, the maximum storage capacity for IoT data with the actual technology corresponds to 1 gigabyte (i.e., the RAM provided by *Raspberry Pi* 2 and 3). In the presented implementation, a routine in charge of removing, from *Raw Data*, the data already normalized and, from *Normalized Data*, the data already published has been included.

The memory occupancy of NOS during operations has been measured, which resulted in an average slightly less than 7 megabyte. Such a value is only indicative, as the memory occupancy depends on a number of factors, notably:

- The frequency of data fetching from sources (in this example 10 packets per second)
- The frequency of execution of the routines for removing data from non-persistent collections (in this example every 5 minutes)
- The number of sources.

5.5.2 Latency

A further evaluation is performed in order to estimate the latency introduced by the policy enforcement framework into NOS architecture.

Figure 5.9 shows the results obtained from one run of one NOS prototype with the six data sources described above, over a period of one hour for two different sampling rates (i.e., how often NOS queries the data sources to fetch data): 10 and 20 packets per second, respectively.

The graph shows that the mean delay is almost constant over time. Furthermore, the latency introduced does not exceed 6.5 ms in the presented test case, which is a promising result in terms of the ability of this solution to deal with near real-time analysis of IoT data.

5.5.3 Policy behavior

As regards the enforcement actions undertaken by NOS, the following policies, formally described in Paragraph 4.4, are exemplified below:

- Node access control, for a registered source
- Node access control, for a non registered source

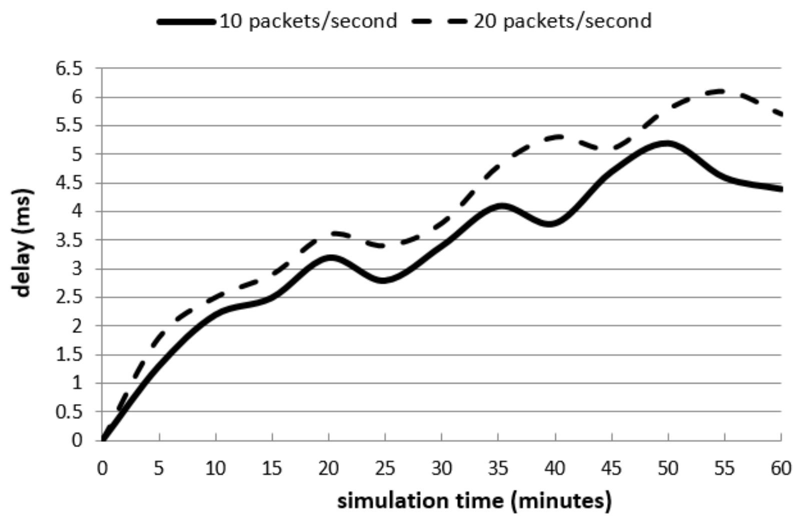


FIGURE 5.9: Latency introduced by the policy enforcement framework into NOS architecture

- Data transmission, for a registered source
- Data processing, for a registered source
- User access control, along with a violation attempt
- User service request, along with a violation attempt
- User service provision.

Before sending their data to NOS, the registered sources perform the access control operation. For example, in Listing 5.1 the policy activated for *Source 1* is represented: the request is valid if the signature key is compliant with that owned by NOS. The session identifier is supposed to be randomly generated by the node identifier and the actual timestamp is *2016-03-24 09:15:00* (as shown in Listings 5.2 and 5.3).

```
{ "NodeAccessControl": {
  "policy": "NodeAccessControl",
  "input": {
    "node": {
      "identifier": "1",
      "signaturekey": "*****"
    }
  },
  "security": [ {
    "verificationRegistration": [{
      "registrationCheck": "1",
      "signaturekeyCheck": "1,*****"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "1, 2016-03-24 09:15:00"
    }]
  }]
}}
```

LISTING 5.1: Node access control - registered source

In Listing 5.2 the same action is presented for the non registered *Source 3*; in this case the enforced action is the tracking of the node.

```
{ "NodeAccessControl": {
  "policy": "NodeAccessControl",
  "input": {
    "node": {
      "identifier": "3"
    }
  },
  "security": [ {
    "verificationUnknownSource": [{
      "signaturekeyUndefinedMark": "undefined",
      "identifierCheck": "3"
    }]
  }],
  "response": [ {
    "verificationUnknownSource": [{
      "sessionAssignment": "1, 2016-03-24 09:15:00"
    }]
  }]
}}
```

LISTING 5.2: Node access control - non-registered source

Now the registered *Source 1* is supposed to transmit data to NOS (Listing 5.3). Function *encr* specifies that the parameters of the message are encrypted. The only difference between this node and a non registered one is that, in the latter case, the parameters of the message would not be encrypted. The enforcement framework verifies whether the four requested values (i.e., session, identifier, data, datatype) are present in the message received by NOS. If this is not the case, NOS discards the message and, possibly, prevents other communication with the same source.

Note that a message sent from a node may include, besides that requested by the policy, also other data (such as a timestamp or a location, depending on the specific device): the policy only specifies conditions on the mandatory data.

```
{ "NodeDataTransmission": {
  "policy": "NodeDataTransmission",
  "input": {
    "message": {
      "session": "12345",
      "identifier": "1",
      "data": "encr(10.7)",
      "datatype": "encr(double wind speed)"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true,
      "storeData": "1,encr(10.7),encr(double wind speed)"
    }
        }]
  }]
}}
```

LISTING 5.3: Node data transmission

After validating such data, NOS can process them. Listing 5.4 shows the corresponding processing policy.

```
{ "NodeDataProcessing": {
  "policy": "NodeDataProcessing",
  "input": {
    "message": {
    "session": "12345",
    "identifier": "1",
    "data": "encr(10.7)",
    "datatype": "encr(double wind speed)"
    }
  },
  "response": [{
    "evaluationRegisteredSource": [{
      "decryptionData": "encr(10.7), 1",
      "decryptionDatatype": "encr(km/h wind speed), 1",
      "scoreAssessment": "10.7, 1"
    }]
  }]
}}
```

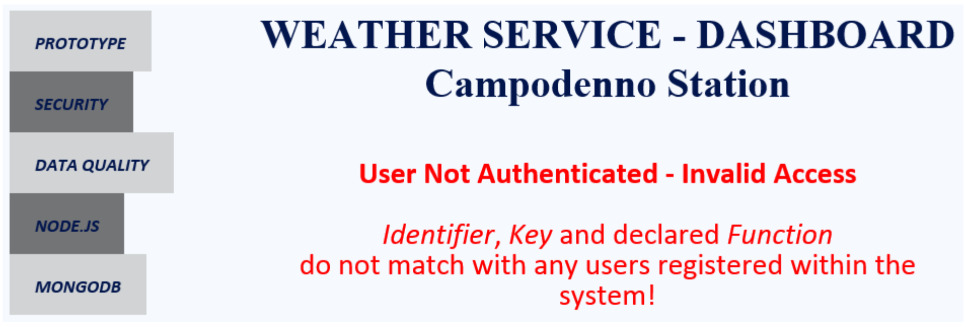
LISTING 5.4: Node data processing

Now, supposing that the user *Bob* had registered himself to the weather service exposed by NOS, with username *Bob* and *473* as identifier, he wants to be notified about the measurements acquired for some monitoring actions he has to do for his employer. Therefore, he registers himself with the role of *Monitor*. Firstly, he has to perform the access control: Listing 5.5 describes the activated policy. The generated session identifier is *13240*. If the credentials are not valid, it could be a violation attempt. In this case, the enforcement framework forces NOS to prevent user interactions, as illustrated in Figure 5.10.

```
{ "UserAccessControl": {
  "policy": "UserAccessControl",
  "input": {
    "user": {
      "username": "Bob",
      "identifier": "473",
      "signaturekey": "*****",
      "function": "Monitor"
    }
  },
  "security": [{
    "verificationRegistration": [{
      "registrationCheck": "473,Monitor",
      "signaturekeyCheck": "473,*****,Monitor"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "473,Monitor, 2016-03-24 09:20:00"
    }]
  }]
}}
```

LISTING 5.5: User access control

If the credentials are correctly verified, the user is allowed to make the desired service requests. Listing 5.6 shows an example of the corresponding



WEATHER SERVICE - DASHBOARD
Campodenno Station

User Not Authenticated - Invalid Access

Identifier, *Key* and declared *Function*
do not match with any users registered within the system!

FIGURE 5.10: Violation attempts of user access control policy

policy, invoked for requesting the service, named *Humidity and Wind Speed Real Time Measurements*, along with the security and quality constraints.

```
{ "ServiceRequest": {
  "policy": "ServiceRequest",
  "input": {
    "message": {
      "session": "13240",
      "username": "Bob",
      "identifier": "473",
      "function": "Monitor",
      "service": "Humidity and Wind Speed Real Time Measurements",
      "securityPreferences": "0.2, 0, 0.2, 0.1",
      "qualityPreferences": "0.4, 0.4, 0.4, 0.4"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true,
      "processRequest": "13240,Bob,473,Monitor,Humidity and Wind
    Speed Real Time Measurements, 0.2, 0, 0.2, 0.1, 0.4, 0.4, 0.4,
    0.4"
    }
  }]
}}
```

LISTING 5.6: User service request

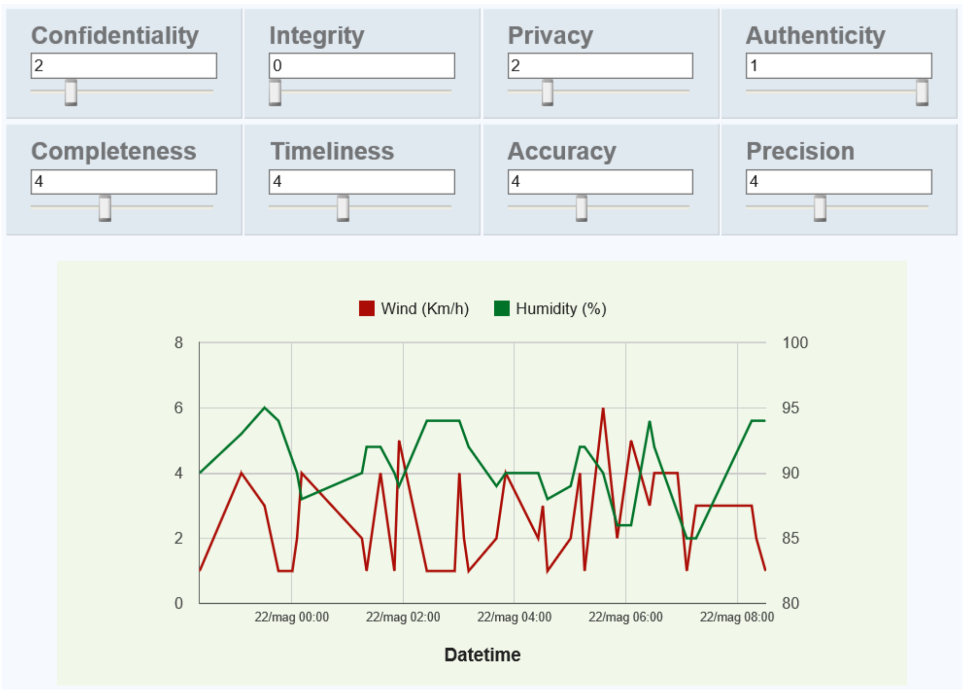
At this point, such a request has to be analyzed in order to establish if the user is entitled to receive the data corresponding to the service (Listing 5.7 represents a user which has access to the requested data).

```
{ "ServiceProvision": {
  "policy": "ServiceProvision",
  "input": {
    "message": {
      "session": "13240",
      "username": "Bob",
      "identifier": "473",
      "function": "Monitor",
      "service": "Humidity and Wind Speed Real Time Measurements",
      "securityPreferences": "0.2, 0, 0.2, 0.1",
      "qualityPreferences": "0.4, 0.4, 0.4, 0.4"
    }
  },
  "security": [{
    "decryptionRequest": "Humidity and Wind Speed Real Time
  Measurements,473",
    "serviceAccessVerification": "Humidity and Wind Speed Real
  Time Measurements,473,Monitor"
 }],
 "response": [ {
    "serviceAccessVerification": true,
    "retrieveResults": "Humidity and Wind Speed Real Time
  Measurements,473,Monitor, 0.2, 0, 0.2, 0.1, 0.4, 0.4, 0.4, 0.4"
   }]
}}
```

LISTING 5.7: Service provision

There are two possible outcomes:

- The user with the *Monitor* function is allowed by the sensor data administrator to access the requested measurements for the monitoring scope
- The user is not allowed to access these data for the declared scope.



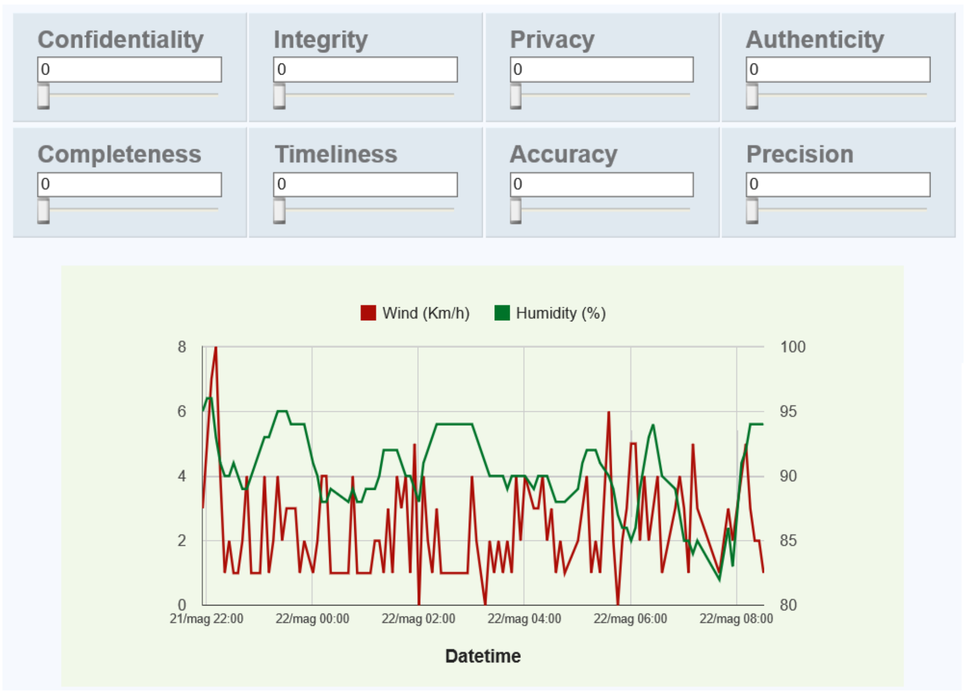
Confidentiality	Integrity	Privacy	Authenticity
2	0	2	1

Completeness	Timeliness	Accuracy	Precision
4	4	4	4

FIGURE 5.11: User data selection - QoP and DQ filters

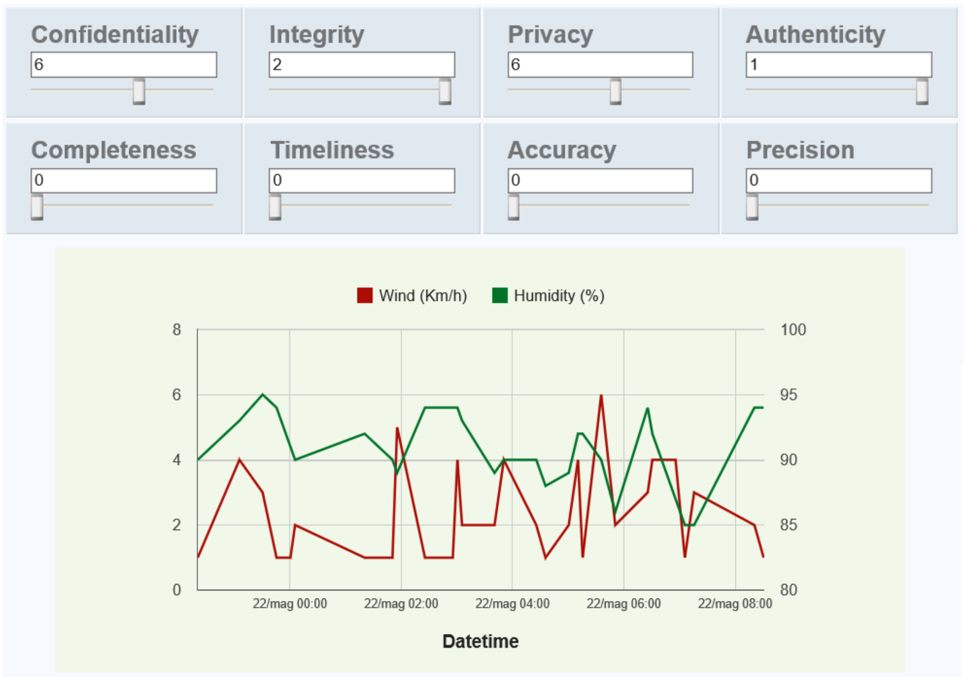
In the former case, the dashboard shown to the user is that represented in Figure 5.11; from here, the user can also change the QoP and DQ settings. If no security or quality constraint is specified, all data is considered valid and the resulting graphs for wind speed (in Km/h) and humidity (in %) look as in Figure 5.12.

In Figure 5.13 some QoP filters are applied; in particular, the system shows to the user only the data (i) provided by authenticated sources, (ii) for which the integrity is verified, (iii), for which the level of privacy and confidentiality is equal to or higher than 0.6. Obviously, in this case some



Confidentiality	Integrity	Privacy	Authenticity
0	0	0	0
Completeness	Timeliness	Accuracy	Precision
0	0	0	0

FIGURE 5.12: User data selection - no filters



Confidentiality	Integrity	Privacy	Authenticity
6	2	6	1
Completeness	Timeliness	Accuracy	Precision
0	0	0	0

FIGURE 5.13: User data selection - QoP filters

data are dropped, as they fail to meet the criteria specified by the user in terms of QoP and DQ of the data to use.

The graph in Figure 5.14 is obtained without any constraint specified on security, but considering valid only data scoring at least 0.6 in completeness and timeliness, and 0.8 in accuracy and precision.

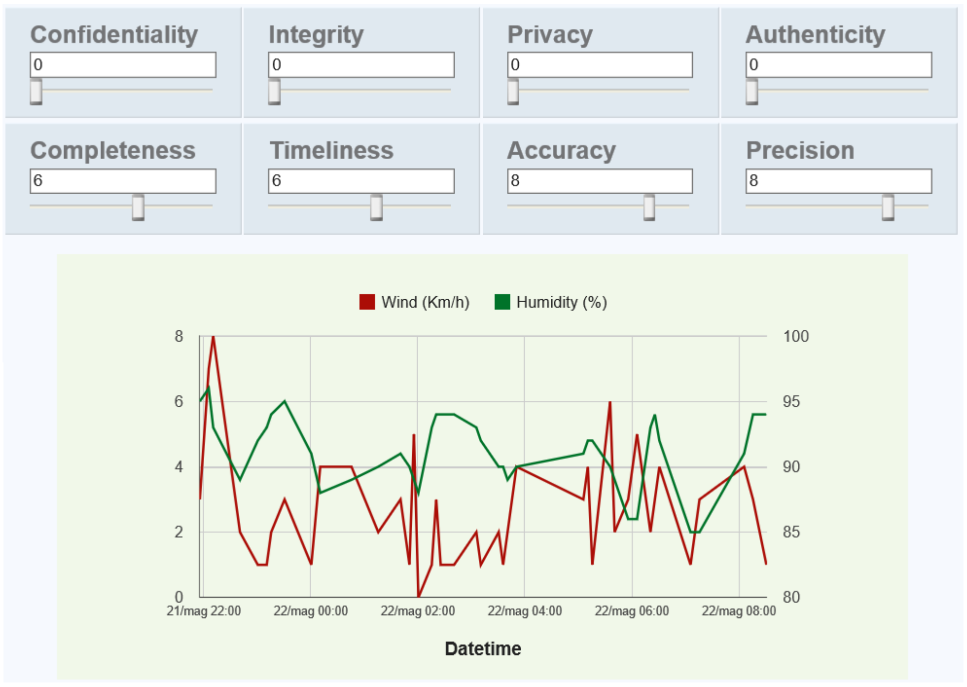
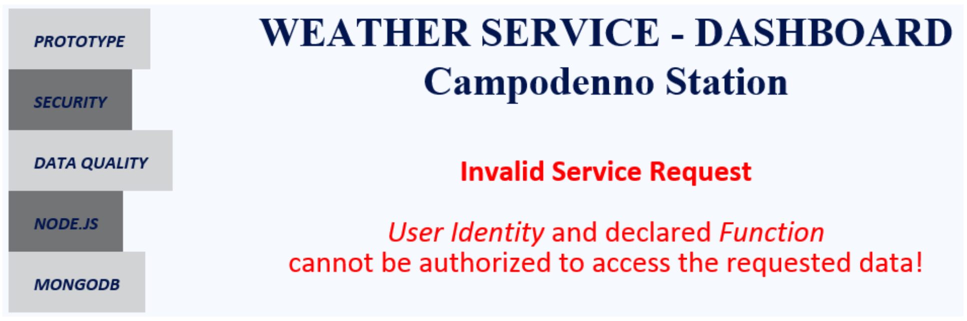


FIGURE 5.14: User data selection - DQ filters

In the latter case, there is a violation attempt, and the response of the system is the one shown in Figure 5.15.



WEATHER SERVICE - DASHBOARD
Campodenno Station

Invalid Service Request

User Identity and declared *Function*
cannot be authorized to access the requested data!

FIGURE 5.15: Violation attempt of user service request policy

One aspect that deserves some further clarification refers to the fact that, in this example, one single NOS has been considered, whereas the final aim is to deploy NOS middleware in a distributed environment. However, as stated in Paragraph 3.4, no NOS-to-NOS coordination is strictly required. In particular, each NOS is able to independently enforce the application of the defined policies. Therefore, it is possible to safely conclude that considering a single NOS scenario for validation purposes does not represent a limiting factor for the analysis just presented. With regards to scalability, further investigation is needed dealing with a larger testing environment. As a further step, Paragraph 5.7 shows an evaluation of performance about policy sychronization in the presence of more than one NOS (i.e., two NOSs). Another aspect which deserves more study is the behavior of the enforcement system in presence of multiple kinds of attack.

5.6 Publish&subscribe protocol performances

In this paragraph, the validation of AUPS (i.e., the secure publish&subscribe protocol described in Paragraph 4.5) is considered. In more detail, in the following paragraphs, a comparison of AUPS with existing mechanism, also based on MQTT, is presented, along with concrete performance results in terms of overhead, latency, and computing effort.

In the example used for exploring the performance, NOS fetches the data at two different rates: 10 packets per second and 20 packets per second. This frequency obviously influcences the memory occupancy as well as the computational effort. Other parameters that may affect the performance of the whole system are represented by the number of sources and of interacting users. As discussed above, six data sources are considered, whereas the number of users is fixed to two. The term "fixed" mean that, in a real application scenario, users may join (i.e., by means of NOS registration) or leave the IoT network at any time.

5.6.1 AUPS robustness evaluation

In order to further clarify the innovative contribution of the AUPS solution, it has been compared with existing approach integrated into *SecKit* [140], just briefly described in Paragraph 2.1.4. *SecKit* also aims to address the lack of security policy enforcement capabilities in existing MQTT implementations, but it follows a different approach.

SecKit extends the implementation of the open source *Mosquitto* MQTT broker with a security plugin, in a way that the PEP directly resides on the broker. Instead, in AUPS, the PEP and the other enforcement components reside on each NOS. Note that NOSs represent a real IoT middleware with data processing and source management capabilities; while in [140] the deployment of a large scale IoT middleware is deemed to be out of the scope.

Hence, [140] limits the presentation of the results to a restricted scenario, in which all the load of policy enforcement is moved to the MQTT broker, thus increasing its computational overhead. The major implication of porting the PEP to the broker, as *SecKit* does, is that the native lightweight efficiency of *Mosquitto* in terms of processing and memory capabilities may be compromised, in particular in a large-scale application setting, by a huge number of topics and adopted policies. For this reason, in the solution presented in this thesis, the security enforcement management is kept on NOSs, which have the necessary processing and storage resources.

Moreover, the usage of a uniform language for implementing PEP and PDP (in NOS case: javascript, using the *Node.JS* platform) ensures easier integration and maintenance than the mixed approach (PEP in *C* and PDP in *Java*) used in *SecKit*.

Furthermore, if the PEP resides on the MQTT broker, then the violation of the broker itself by malicious entities may compromise the whole network, breaking access control by allowing non authorized users/applications to access restricted information. In AUPS, NOS itself would need to be compromised, which is a more complex task and, in addition, the key management is controlled by another external entity, the KTM, which is decoupled with respect to NOSs, thus increasing the robustness of the entire system.

A further crucial difference between AUPS and *SecKit* is that NOS access control acts at the topic level, while, in [140], the authors propose to consider the delivery of individual messages and filter the access on the basis of the content of the payload. To do this, *SecKit* has to include in the model the concepts of context and situation, by means of which the PDP is able to assign different policies to the same information depending on the actual conditions. As a consequence, authorizations and obligations are specified using an *Event-Condition-Action (ECA)* structure and, at any time, before allowing the access to a resource, the system has to evaluate the actual events, which take place within the system. Such an approach may seriously affect the system performance; in fact, not only does an *ECA* hierarchy have to be defined for each new resource, but also the related events, to be executed in response to particular scenarios, need to be specified, thus requiring considerable computational effort. Therefore, in NOS, access control is performed at a coarser level of granularity, using ABAC (instead of *ECA*) and without requiring the access control operations at each notification event by introducing the concept of temporary keys, thus reducing the computational and storage overhead.

Another advantage of the adoption of KTM and temporary keys on NOSs is represented by robustness against violation attempts. In fact, the mechanism used by NOSs allows them to preserve the system from credential discovery and replay attacks, since the keys, used for decrypting the information, have a limited temporal scope.

Also as regards man in the middle attacks, even if a malicious entity listens to the communications between broker and user devices (or external application), it will not be able to intercept the clear content, since the encryption varies over time, without a fixed or predictable pattern.

If a malicious device obtains one or more *keyT* (e.g., by means of an exhaustion attack), then it may perform a DoS attack, for example, entering valid packets into the network, thus hindering the correctness of the spread information; this action is also mitigated since, in AUPS, when the key expires, the device will not own the correct credentials for authenticating itself to NOS, therefore it will not obtain new valid key and it will not continue to execute the attack. It is important to remark that the authors of *SecKit* do not specify possible countermeasures to address DoS attacks.

Finally, both the approaches, AUPS and *SecKit*, do not explicitly face physical attacks to the data sources or the devices used by the users for interacting with NOSs.

5.6.2 Overhead

The overhead in terms of storage capacity and computational effort is now analyzed. In the test-bed presented in Paragraph 5.2, NOS runs on a *Raspberry Pi* with 1 gigabyte of RAM and a variable physical memory, which depends on the capacity of the *microSD* card used. In the scenario under consideration, the *microSD* has 8 gigabyte of memory.

An important premise is that, as pointed out for the policy enforcement framework analysis (see Paragraph 5.5), NOS exploits the in-memory capability of *MongoDB* to flush the data from non persistent collections.

Hence, the memory occupancy was on average slightly less than 7.5 megabyte, where 8% is used for AUPS. In more detail, NOS has always to maintain the set of tuples *T* in the form *keyId, keyT, val, atb* for each combination of attributes associated with the topics and established by the policies themselves.

In this example, there are six topics (i.e., one for each kind of data provided by the sources). At this stage, the number of possible combinations of topics for access to resources depends on the granularity of the policies required by the system administrator. The system administrator may decide to associate a different attribute with access to each topic or, instead, group them, so that access is allowed to a subset of the data. Note that ABAC is adopted, as explained in Paragraph 4.4.

Two users are considered for the validation: the former with access to topics 1 (`campodenno/sensor1/temperature`) and 2 (`campodenno/sensor2/humidity`); the latter with only access to topic 2. Such policies may be replicated for each of the six kinds of data provided by the sources. As a consequence, the amount of memory required on NOS is strictly related to the structure and granularity of access permissions and, more generally, to the specific application domain.

In the case study presented, the physical memory needed is very low (the 8% of the memory occupancy, as mentioned), since few information sources are provided to the system by the meteorological sensors. Moreover, note that the dimension of each tuple *T* varies on the basis of the length of the fields *keyT* and *atb*. Hence, the application of the proposed approach in a wide real scenario has to take into account such aspects related to memory occupancy in the definition of attributes and access permissions.

As regards the computational overhead with respect to the original NOS MQTT version, it is worth remarking that, for each incoming data message, the following operations are executed (in brackets, the measured average execution time):

- Three database queries to obtain the access permissions to be associated with the new data (2.4 ms)
- One operation of encryption (0.5 ms)
- One HTTP call to the KTM, only in case of an expired key (3.2 ms).

When new data is notified to a user, then it is the user device that is in charge of executing the decryption operation; NOS receives a request for a new valid key, only when one expired on the user device. Such an activity requires:

- Three database queries to obtain the access permissions to be associated with the user (2.4 ms)
- One HTTP call to the KTM (3.2 ms).

Finally, an important remark has to be made about the KTM. Being an external entity, the KTM does not delegate the key and attribute management to NOS, thus saving memory and processing capacity. Moreover, KTM cannot be considered a bottleneck, besides it serves all NOSs, since its operations (i.e., the temporary key generation) are asynchronous with respect to NOSs' activity. In fact, the key expiration time is not supposed to be the same for all the generated keys, thus allowing a balanced load of HTTP calls, made by NOSs to the KTM during system operation. Obviously, a large scale evaluation should be done in order to better assess the KTM performance.

5.6.3 Latency

The next performance analysis of AUPS investigates the end-to-end latency introduced by the adoption of the proposed secure mechanism to the native MQTT transactions, executed in the previous version of NOS (i.e., with this secure extension disabled).

Latency is computed as the distribution of the elapsed time from the data reception at NOS until it is sent to the broker. It is evaluated for two different settings: with and without AUPS. For AUPS, the key expiration timeout is set to 5 minutes.

Figure 5.16 shows the distribution of the latency for one NOS and the six data sources, measured over a period of 24 hours and with two different data fetch rates: 10 packets per second and 20 packets per second.

In the original configuration (i.e., without AUPS) the average measured latency is approximately 3.5 ms and 4 ms for 10 and 20 packets per second, respectively. With the secure extension, the average time increases by 2 ms. What emerges is that the new operations add a stable increase to the delay, which remains under an acceptable threshold.

5.6.4 Computing effort

Using the same parameters as for the latency performance evaluation, Figure 5.17 shows the distribution of the CPU load on NOS with and without AUPS. In this case the computational effort is also stable during system operation and the results are very promising for encouraging the adoption of AUPS in a large-scale environment.

It is important to note that both latency and computing performances are affected by the number of users subscribed to the topics provided by NOS. In fact, such parameters influence the number of keys to be generated and, therefore, the overall resources required by AUPS. An evaluation concerning a large number of data/topics and users would allow to perform a proper analysis about the scalability of AUPS, in a more complex context, requiring the generation and update of many keys.

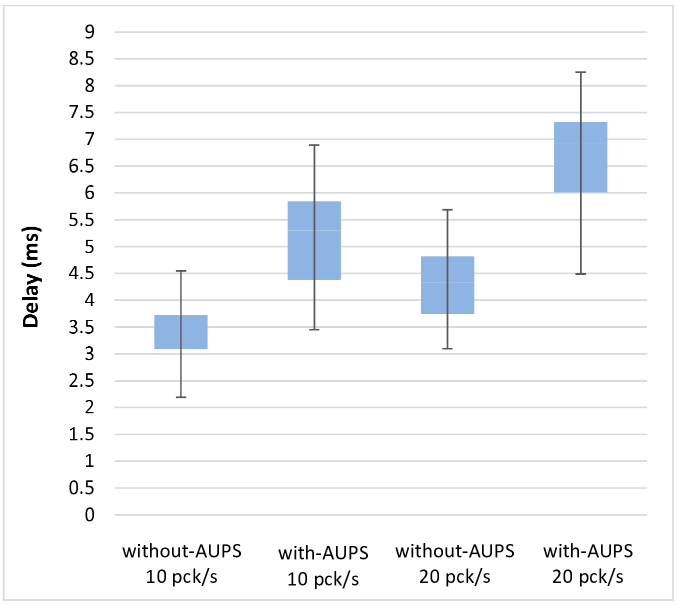


FIGURE 5.16: Latency - whiskers-box diagram for a system with and without AUPS with two different data fetch rate (10 and 20 pkt/s)

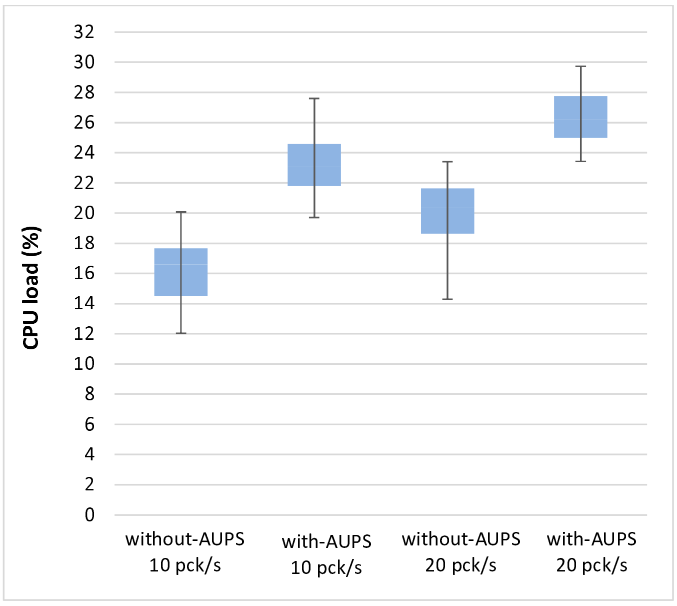


FIGURE 5.17: CPU load - whiskers-box diagram for a system with and without AUPS with two different data fetch rate (10 and 20 pkt/s)

5.7 Policy synchronization validation

The last analysis regards the execution time and computational load required by the sychronization system, presented in Paragraph 4.6. In this case, two NOSs are connected, as shown in the experimental setup in Figure 5.18.

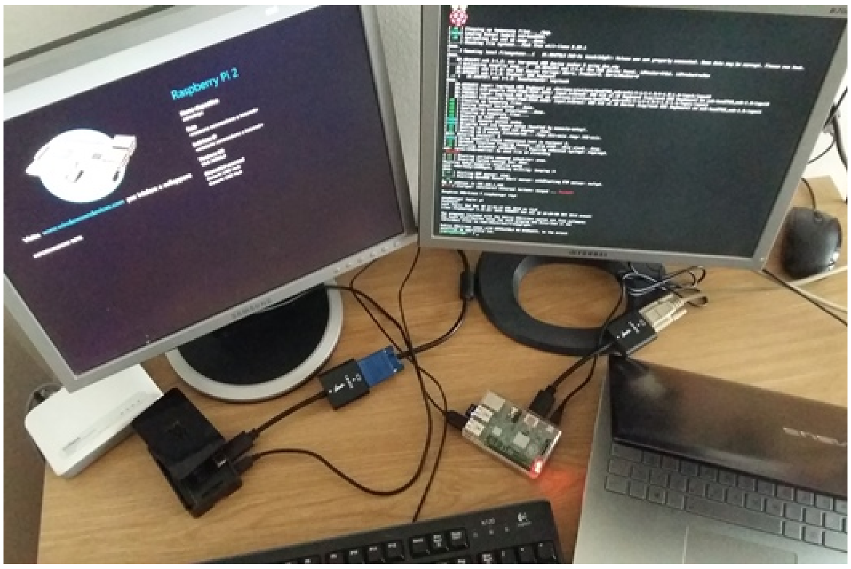


FIGURE 5.18: Experimental setup for policy synchronization validation

The meteorological information, sent as open data to NOS, are mapped to proper topics, such as *campodenno/sensor1/temperature*, *campodenno/sensor2/humidity*, and so on, as for AUPS validation, detailed in Paragraph 5.6. A set of policies is defined accordingly. The data rate is fixed, as before, to 10 and 20 packets per second.

In this case, the test-bed also simulates the behavior of an A_{NOS} administrator; while one A_{data_1} administrator is considered, since the data belong to a unique application contex. Finally, the rate of policy update varies in a random range from one to five minutes.

The execution time and CPU load, spent on each policy update and synchronization, were evaluated by means of a six-hours experiment. Figures 5.19 and 5.20 show, through a box-whisker plot, the results obtained from the two NOSs used for the test-bed. Both Figures 5.19 and 5.20 reveal promising outcomes, since both execution time and computational effort are low and stable over the time. The example presented is a very simple application involving only two NOSs in a context characterised by the analysis of real-time data, but it demonstrated the viability of the proposed schema in meeting its requirements.

Finally, it is worth remarking that other useful analysis of NOSs could be carried out, concerning how to make the techniques presented for data assessment, policy enforcement, authentication, and synchronization work effectively in situations in which the IoT data sources are power constrained or, for example, if part of the network is unavailable due to a malicious

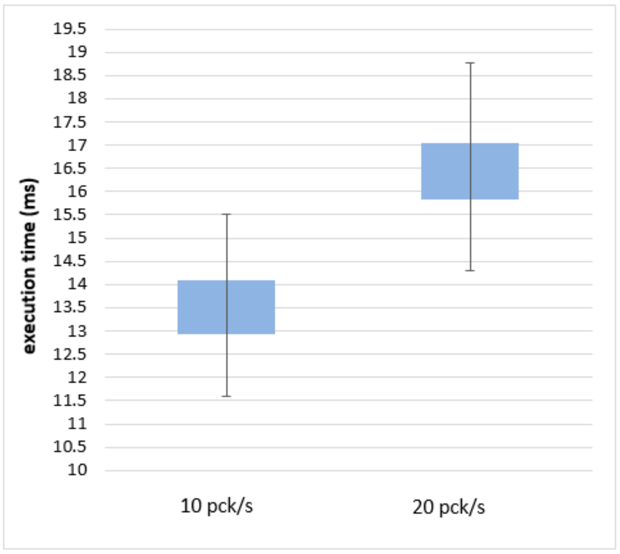


FIGURE 5.19: Latency - sychronization system with two different data fetch rate (10 and 20 pkt/s)

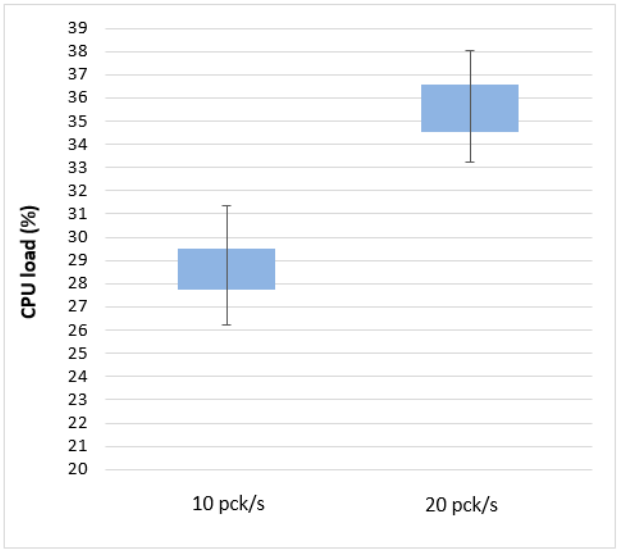


FIGURE 5.20: Computational load - sychronization system with two different data fetch rate (10 and 20 pkt/s)

attack. Clarifying such aspects would be a further advance in making NOS prototype more reliable and resilient in a large scale scenario.

Chapter 6

Conclusions

The lack of a comprehensive security solution represents a major threat to the growth and market take-up of IoT systems across a variety of vertical application domains. In order to fill this gap, the main goal of this PhD thesis concerned the design and development of a secure and distributed middleware platform, named NOS (NetwOrked Smart Object), targeted to IoT environments. The aims of NOS are many-fold, since they embrace the pivotal open issues in the definition of a scalable, interoperable, energy-saving and secure and data quality aware architecture for IoT applications.

From the analysis of the state of the art in the field of IoT security, many questions arose about how to improve the robustness towards malicious attacks of a typical IoT system, which comprises heterogeneous technologies and different communication protocols. This thesis addressed some of these questions. In particular:

- Authentication and confidentiality requirements have been met; on the one hand, by re-using traditional security mechanisms for key management, in order to ensure end-to-end integrity to the transmitted data; on the other hand, providing a source registration system, through which sources are allowed to agree, with the IoT platform, on an encryption mechanism for data exchange
- As far as access control is concerned, a scalable and distributed system for controlling the access to resources has been implemented, conceived both for users and “things”; entities and roles have been specified, in order to define proper authorization rules; moreover, a uniform data representation has been formalized along with a topic structure for efficient information sharing
- A complete enforcement framework as well as a policy language have been proposed
- A cross-domain secure middleware for the management of IoT data, provided by heterogeneous devices, has been defined.

Moreover, since the scope of IoT applications is the provision of services to the interested users, the processed and transmitted information have to be evaluated to make the users aware of their level of reliability, both in terms of security properties and data quality. With regards to such features, some important requirements have been identified. In particular, concerning security, the following QoP properties have been considered: confidentiality, integrity, privacy, and authentication; while, regarding DQ, the following have been analyzed: accuracy, precision, timeliness, and completeness.

The just presented QoP and DQ requirements have been included in the definition of a UML general conceptual model, representing the starting point of this work towards the development of the final NOS middleware. In more detail, the UML model aimed to include the entities involved in the IoT scenario along with their relationships. At this stage, the main goal was to obtain a model suitable for general-purpose contexts, which could potentially be adapted to various IoT domains and not only targeted at single case study. The main entities identified were:

- The nodes, which represent the sources of the information provided to the IoT platform
- The users, who make use of the services provided by the IoT platform
- The services, which act as a bridge from the raw data transmitted by the nodes and the processed data received by the users.

From this model, a high-level reference architecture of NOS was derived. In more detail, the following activities, performed by NOS on the data incoming from heterogeneous sources, have been specified:

- *Analysis*, which is responsible of the assessment of the data with respect to the defined QoP and DQ levels
- *Data Normalization*, which is in charge of putting the processed information in a uniform format, also adding the proper set of metadata related to the QoP and DQ evaluation, made after the analysis phase
- *Integration*, which may merge information obtained from different sources in order to provide more complex services.

The result obtained was a modular architecture, which exposes:

- Specific southbound interfaces, based on the HTTP protocol, towards the nodes, also allowing the registration of the nodes themselves to NOS
- The processing units described above
- Northbound interfaces, based on MQTT protocol, aimed at managing the sharing of information by means of a lightweight publish/subscribe mechanism.

NOS system has been implemented in a real prototype running on a *Raspberry Pi* and using the following components/technologies/libraries: (i) *Node.JS* platform for NOS modules; (ii) *MongoDB* for storage; (iii) *Mosquitto* for the MQTT broker.

As a further step, a variety of new modules have been developed for NOS, each one with different capabilities and addressing a specific security issue. In particular, the added functionality is:

- The adoption of two key management systems by Dini et al. and Di Pietro et al., for securing the communications among NOS and users/nodes with the distribution of well-defined keys; they also provide mechanisms for key replacement, which allow the improvement of the reliability of the system towards malicious attacks

- The definition of a new algorithm for QoP level assessment, able to perform an evaluation of the information with respect to data sources behavior
- The introduction of a policy enforcement framework, which provides a set of general-purpose rules aimed at regulating and controlling the actions performed by NOS and reacting towards possible violation attempts
- The integration of the enforcement mechanism with AUPS, which is able to effectively manage publications and subscriptions through MQTT interactions, securing the information sharing with users
- The development of a mechanism for allowing the propagation and synchronization of policies among multiple NOSs and/or belonging to different application domains.

All the proposed solutions were evaluated by means of a NOS prototype and real use cases. Relevant performance metrics were investigated, in order to clarify the effectiveness of NOS functionality in real conditions. Nevertheless, this thesis also leaves several open issues.

First of all, it would be useful to carry out an analysis of NOSs' functionality in presence of power constrained data sources and/or in situations including multiple live attacks to both NOSs and data sources. Furthermore, a wider and more complex case study would allow to better investigate the scalability of the proposed solutions.

The key management systems have shown good results, with some differences among the evaluated metrics. In fact, the outcomes of the experimental comparison between the two algorithms shows a lower overhead and key replacement delay for Di Pietro et al., but better robustness for Dini et al. A further improvement of these actual solutions relies on the adoption of a more robust key replacement and revocation schema, for example based on *DHKE*, which would allow a secure exchanging of cryptographic keys among entities that do not know each other (as is the case in lots of IoT applications).

With regards to the algorithm for the assessment of QoP property, it succeeded in representing correctly the trustworthiness of registered and non registered IoT data sources. Perhaps, it would be interesting to test this solution in a wider IoT scenario, consisting of a large number of heterogeneous nodes, providing data with various QoP as well as DQ features. Another point resides in the capability of NOS to directly counteract malicious activities, which still remains, at the moment, an open challenge.

In the actual prototype implementation, event reporting to the administrators is not considered, but, in a real application scenario there should be support for system monitoring (e.g., information regarding the service utilization, the storage, the malicious activities, and the NOS performance in general), including notifications and alerts to the system administrators. Given the distributed nature of NOS, a viable solution is represented by the popular *ELK (Elastic, Logstash, Kibana)* stack (`https://www.elastic.co/webinars/introduction-elk-stack`).

Such considerations are also strictly related to the policy enforcement framework, the purpose of which is just the tracking of normal and unexpected events happening during NOS operation. Note that its feasibility and performance have been proven by means of the development of a simple, yet real-world, use case. Since the framework supports, in a general-purpose way, security, privacy and data quality enforcement policies, it is re-usable across different domains.

A valuable extension would improve the defintion of *ad hoc* mechanisms in order to addresses the privacy issue more thoroughly.

The enforcement approach described above was enriched with AUPS, consisting of a set of methods for adding security to MQTT-based IoT systems. Its robustness was compared with existing mechanism, showing better performance for AUPS in terms of resilience towards malicious attacks, besides promising results in terms of overhead, latency, and computing effort. Future extensions include testing in a larger and more complex setting, possibly characterised by the presence of a plurality of networked brokers and NOSs, in order to better assess the KTM performance and the scalability of the whole system.

The same remarks also apply to the policy synchronization system, which should be tested in an IoT scenario, including more NOSs and data sources and different application domains (i.e., more A_{data_n} administrators) and, consequently, more complex policies (e.g., hierarchical structures).

Finally, the data processed by NOS may be exploited in a mobile application context. To this end, possible applications of NOS middleware include: energy management in a smart home/smart building scenario; monitoring of business processes and productive activities in real time; smart retail experiences services and, more in general, any application/service where decisions (either manual or automated) have to be taken based on IoT-generated data. This class of applications is expected to play a key role in the adoption of IoT technologies across a variety of vertical domains.

At the moment, ongoing work consists of the application and testing of NOS features inside the *University of Insubria* campus in Varese.

Appendix A

Application case study

An application example related to a university campus domain is presented in this appendix, in order to clarify the viability and the effectiveness of the conceptual model, presented in Paragraph 3.2.

In such a scenario, a student, a professor, an employee or a simple visitor may take advantage of an *ad hoc* advanced application, able to guide them through the campus, in order to interact with physical objects and obtain specific digital services in relation to their profile. A specific application running on the users' personal smartphones is in charge of providing the expected personalized content. A sketch of the scenario is shown in Figure A.1.

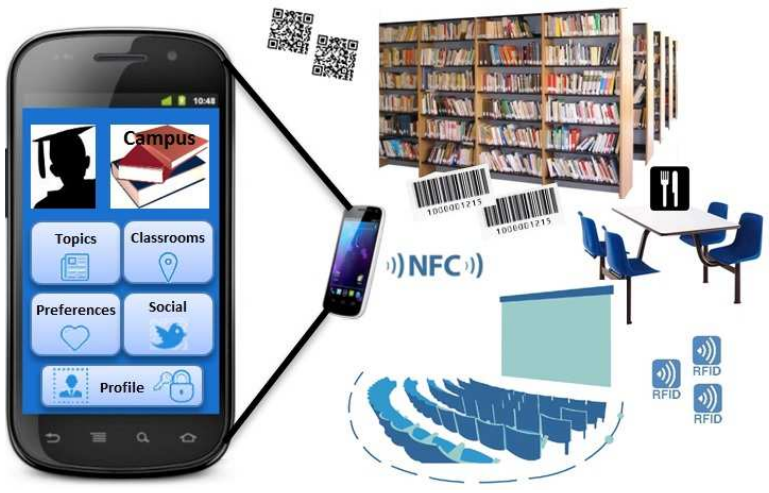


FIGURE A.1: IoT university campus scenario - application case study

Student, professor and employee represent instances of class *User*, and each of them owns a personal profile (*Profile*). Clearly, a student is likely to be interested in different services in comparison to the professor or employees. For example, a student may be interested in information related to the classrooms for studying, the classrooms where lessons or exams will take place, or the presence of the professor in a classroom, in order to establish whether she/he is early or late for a meeting or a lesson. An employee may instead want to monitor the state of classrooms' occupancy. Finally, a professor may want to be informed about his/her class schedules or conferences. Consequently, each user personal profile is created to reflect his/her role (*UserRole*) and function (*UserFunction*) within the IoT infrastructure. In particular, students', professors' and employees' profiles may be customized with information strictly related to their actions and needs within the university campus (e.g., curriculum, faculty membership, classroom preferences).

In order to exploit the services provided, each user has to be registered with the *IoTPlatform*. During the registration phase, *IoTPlatform* requires some personal information (e.g., first name, last name, age, role within the university) and also provides the credentials to access the system itself in the form of a *SignatureKey*. Information such as 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, for example, for statistical purposes. An important step is that each user must give the consent for the processing of personal information in order to let the system use his/her preferences to customize the provided services. Such personal information will be used according to the established privacy policies (e.g., information can be used for a specific purpose and under particular constraints). All these data belong to the user profile and are stored 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 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 statistical or monitoring purposes)
- *UserSignatureKey*: is the key chosen by the user during the registration phase; the user will use such a key for future interactions with the IoT system (i.e., login/logout operations)
- *UserActionKey*: is the 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*): indicates whether the user is regularly registered or not; if not, further interactions with the IoT system must be prevented

    - *Consent* (possible values *yes/no*): indicates whether the user gave consent to *IoTPlatform* to handle his/her personal data
    - *UpdatePreference* (possible values *not-specified/updated*): indicates whether 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*): indicates whether the user is permitted to request services of *IoTPlatform* (e.g., a user may be disabled for the misuse of the services)

- *PreferencesOnService*: are divided in:
    - *PreferencesOnSecurity*: specifies the security requirements and, possibly, their priority, which is defined by means of the *order* attribute, for example in the form of a decreasing scale from 1 to 4. The available properties are confidentiality, integrity, privacy, and authentication; for each of these, user can specify a score in the range [0:1]
    - *PreferencesOnQuality*: specifies the required quality requirements and, possibly their priority, which is defined by means of the *order* attribute, as for *PreferencesOnSecurity*. 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 confidentiality).

After registration, the user downloads the application interface on his/her smartphone, possibly equipped with technologies such as Bluetooth, ZigBee and/or NFC and connected to the Internet through WiFi or 3G. The service provider gives the necessary credentials to the smartphone, which aim to guarantee access to the service itself by means of a secure communication. Note that such credentials are instances of class *UserActionKey*. After the registration, the users can connect themselves to *IoTPlatform* through their smartphones or other personal devices (i.e., notebook, tablet) using the credentials exchanged and can then communicate through *IoTPlatform* with the nodes belonging to the network. The services are provided to the users taking into account both *Profile* and *PreferenceOnService*, which can be specified by the user through the downloaded application itself.

In this scenario, the installation of a number of nodes (i.e., locators) is assumed; they are conceived as fixed devices to track people's movement in real-time. Locators are able to detect and process the signals from the users' smartphones and, then, send data to the service provider or to other data management infrastructures. At the same time, the data collected by the system can be used for different purposes (i.e., performing analysis, making decisions about the management of the university itself). For example, by means of classroom occupancy information, the employees may plan an optimized distribution among courses. Another interesting analysis relates to the use of the personal car to reach the university by both students and other people, in order to design more efficient parking management; if the campus provides a cafeteria service, users might express some preferences in relation to personal tastes or intolerances, which can be taken into account in the menu definition.

Moreover, during his/her association with the university, the student or professor may receive further information according to his/her preferences (e.g., conferences on topics of interest). Therefore, the smartphones are also able to interact with the smart objects; for example, they can retrieve information from the tags placed on the shelves in the athenaeum library. In fact, a student/professor may be guided within the campus library, in relation to his/her topics of study or preferences, explicitly declared to the service provider. In this case, the books or the bookshelves should be equipped with tags, which may provide the user with information, such as the volume title, authors, year, a brief summary and so on. Moreover, the smartphone application may be connected with a social network, through which users may share their own experiences or, vice versa, the administrators of university social network page may communicate advice, university initiatives or other information. This scenario demonstrates the coexistence of system and human data.

In general, nodes are referred to as the sources of the data provided to *IoTPlatform*. A node is defined by the following properties and attributes:

- *Registered* (possible values *yes/no*): is an attribute which clarifies if the node is registered or not to *IoTPlatform*; note that the system accepts data both from registered and non registered sources, but it will manage the information provided 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*)
- *Source*: identifies the kind of source (e.g., sensor node, RFID, actuator)
- *CommunicationMode*: identifies the means of communication used in order to transmit the data (e.g., WiFi, 3G, Ehternet)
- *DataType*: represents the kinds 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 the case of non registered sources, the corresponding fields related to the encryption scheme and keys are marked as *undefined*
- *NodeSignatureKey*: is the key given by the node to *IoTPlatform* during the registration phase; the node will use such a key for the future interactions with the IoT system (i.e., authentication operations)

- *NodeActionKey*: is the 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, that relate to the function it plays 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* also begins to perform an analysis of such information, in terms of confidentiality, integrity, authentication, and privacy, taking 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 with the information provided by each source (both registered and non registered), which will allow the users to filter out the data they want to receive
- *DQmetadata*: as for security features, *IoTPlatform* also analyzes the data provided by each source in terms of completeness, accuracy, timeliness, and reputation, considering both historical feedbacks about the source behavior and the frequency of the information updates.

In this context, the definition of specific policies, according to the users' consesus, aims to guarantee the desired level of QoP and DQ. Such policies are stored in *IoTPlatform*, which provides access control, key management and enforcement capabilities. From the application scenario described, many issues arise with regards to security, such as: how to guarantee user privacy; how to ensure that information are kept confidential; how to manage authorizations and prevent unauthorized entities from accessing forbidden information. A practical example of policies addressing the emerged requirements is provided in Appendix B. Note that the case study presented concerns all the entities defined in the model in Paragraph 3.2, which interact among themselves in order to exchange useful information for everyday life and long-term decisions.

Appendix B

XML policy representation

A policy enforcement framework for NOS middleware was presented in Paragraph 4.4. Its components and interactions have been detailed, along with the specification of ABAC, as the adopted access control model, and JSON, as the policy language.

However, this result has been derived from another preliminary work, orthogonal to this thesis, in which all the policies, presented in Paragraph 4.4, for regulating the events of node/user/application access control, node data transmission, node data processing, user/application service request, service provision, were formalized by means of XML syntax.

In fact, policies were initially defined in XML and, later, translated into JSON format, to allow their integration into NOS. At first sight, during the design phase of the enforcement framework, XML was chosen due to its wide adoption in actual interoperable systems. But, for implementation purposes, JSON better fits to the adopted NOS technologies and libraries, mainly in terms of efficiency.

In order to provide a complete picture of the thesis work, in the remainder of this appendix, the policies, formalized in Paragraph 4.4 and exemplified in Paragraph 5.5 for the meteorological context, are hereby presented in relation to the case study described in Appendix A.

B.0.1 User registration

Firstly, a student registers himself to *IoTPlatform* in order to exploit the services provided, and chooses, for example, *Nick* as his username. The request for registration is an action performed by users and denoted by ”registration” in Listing B.1; it includes the following information as input: the chosen username, the chosen key (i.e., *UserSignatureKey*), the function for which he wants to register in order to exploit the provided services (e.g., Student), and the consent for handling his data.

```
<action type=='registration'>
  <input>
    <user>
      <username>Nick</username>
      <signaturekey>*****</signaturekey>
      <function>Student</function>
      <consent>yes</consent>
  </user>
  </input>
</action>
```

LISTING B.1: Student registration

IoTPlatform responds with a confirmation message (the action performed is denoted by ”registration response”) to the requesting user, in which a summary of the registration is included (i.e., it confirms the received username and function) and of the successful actions performed, which are: the assignment of an identifier and a session, the request for the consent for interacting with the IoT system (action ”consent acquisition”), the registration within *IoTPlatform* and enabling the request for services. Furthermore, *IoTPlatform* informs the user about the possibiliy of specifying his preferences about the QoP and the DQ levels of the information provided to him in the future interactions (at this time, the preferences are set to ”not specified”). Finally, *IoTPlatform* provides the user with the proper *UserActionKey* (see Listing B.2). In other words, the service provider gives the user smartphone the required credentials, which aim to guarantee access to the service itself by means of secure communications.

```
<action type=='registration response'>
    <user>
      <username>Nick</username>
      <identifier>34567</identifier>
      <session>11111</session>
      <function>Student</function>
      <actions consent='yes' registration='yes'
                 preferences='not specified'
                 servicerequest='enabled'>
        </actions>
      <actionkey>*****</actionkey>
  </user>
</action>
```

LISTING B.2: Student registration response

After the registration phase, the user may specify his preferences, which are elaborated by *IoTPlatform* in the way presented in Listing B.3 which is an example of student profile. In this example, the student requires a high level of confidentiality (*order* 1 and score equal or greater than 0.5), and gives less 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); while, as regards data quality, he requires a high level of completeness (*order* 1 and score equal or greater than 0.8) and then timeliness (*order* 2 and score equal or greater than 0.5), accuracy (*order* 3 and score equal or greater than 0.8), and precision (*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*. Summarizing, after the registration, the users can connect themselves to *IoTPlatform* through their personal devices using the exchanged credentials and can 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.

```
<users>
  <user>
    <username>Nick</username>
    <identifier>34567</identifier>
        <function>Student</function>
        <university>
          <faculty>Computer Science</faculty>
          <department>Science</department>
          <classroom annex>Annex A1
                </classroom annex>
          <classroom preferred>Classroom 4
                </classroom preferred>
          <travel>Bus</travel>
        </university>
        <cafeteria service>
          <intolerance>gluten</intolerance>
          <intolerance>milk products</intolerance>
        </cafeteria service>
        <actionkey>*****</actionkey>
        <signaturekey>*****</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>
    <precision order='4'>0.6
            </ precision>
       </preferenceonquality>
    </user>
</ users>
```

LISTING B.3: Student profile definition within *IoTPlatform*

For each registered user, *IoTPlatform* stores the information just described within the tags *<user>*; as a consequence, a block *<user>* is included for all the registered users in the parent tag *<users>*.

B.0.2 Node registration

If a sensor node, in charge of tracing the number of students who enter in/exit from a classroom, wants to register itself with *IoTPlatform*, it has to send a request, as presented in Listing B.4. It has to specify 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 future communication along with *NodeSignatureKey* owned by the node (e.g., in a pre-shared key distribution). As regards *NodeSignatureKey*, it may be based on a public-key algorithm, therefore the node sends to *IoTPlatform* its public *NodeSignatureKey*, while it preserves the private *NodeSignatureKey*. Such a system permits the verification of the node identity (see Listing B.7).

```
<action type=='registration'>
  <input>
    <node>
       <source>sensor node</source>
       <communicationmode>wifi</communicationmode>
       <datatype>integer</datatype>
       <encryptionscheme>RSA</encryptionscheme>
         <signaturekey>*****</signaturekey>
  </node>
  </input>
```

```
</action>
```

LISTING B.4: Sensor node registration

IoTPlatform replies to the node request by providing a message that contains the session, *NodeActionKey*, and the node identifier, established by *IoTPlatform* and encrypted with *NodeSignatureKey* (see Listing B.5).

```
<action type=='node registration response'>
    <node>
      <session>22222</session>
      <identifier>encrypt(12345, signaturekey)</identifier>
      <actionkey>*****</actionkey>
  </node>
</action>
```

LISTING B.5: Sensor node registration response provided by IoTPlatform

Within *IoTPlatform*, the registered and non registered nodes are stored as presented in Listing B.6, in which a non registered RFID is also included, besides the previous sensor node. As regards the non registered source, an identifier is also assigned and stored (see Listing B.7), since *IoTPlatform* is able to recognize a node that 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 non registered one.

```
<nodes>
  <node registered='yes'>
        <identifier>12345</identifier>
        <source>sensor node</source>
        <communicationmode>wifi</communicationmode>
        <datatype>integer</datatype>
        <encryptionscheme>RSA</encryptionscheme>
          <actionkey>*****</actionkey>
          <signaturekey>*****</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>
    <precision>0.6</precision >
        </qualitymetadata>
    </node>

    <node registered='no'>
        <identifier>12346</identifier>
        <source>RFID</source>
        <datatype>double</datatype>
      <encryptionscheme>undefined</encryptionscheme>
          <actionkey>undefined</actionkey>
          <signaturekey>undefined</signaturekey>
          <securitymetadata>
    <confidentiality>0</confidentiality>
    <integrity>0</integrity>
    <authentication>0</authentication>
    <privacy>0</privacy>
       </securitymetadata>
         <qualitymetadata>
    <completeness>0.5</completeness>
    <accuracy>0.4</accuracy>
    <timeliness>0.8</timeliness>
     <precision>0.4</precision>
       </qualitymetadata>
    </node>
</ nodes>
```

LISTING B.6: Node storage within *IoTPlatform*

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

B.0.3 Node access control

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 verification of *NodeSignatureKey* by means of the public-key algorithm previously introduced; in fact, the node sends its identifier encrypted along with its public *NodeSignatureKey*. As stated, 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 QoP and DQ requested by the users. If a source has already interacted with *IoTPlatform*, it will be recognized by the identifier previously assigned; if instead, it is still an unknown source, *IoTPlatform* assigns a new identifier in response (which is stored in *IoTPlatform*, as described in Listing B.6). Listings B.7 and B.8 expresses this behavior in XML syntax, considering the registered sensor node previously described. Note that the response includes the session for all kinds of sources. In such a way, *IoTPlatform* knows at each time the nodes currently connected to it, since in an IoT context the nodes continuosly join and leave the network.

It is worth remarking that, in the XML code presented here, the tag < *input*> identifies the information obtained, while the tags <*security*> and < *verification*> highlight the security tasks performed.

```
<action type='access control'>
  <input>
    <node>
      <identifier>'' ||12345 || encrypt(12345, signaturekey)</
    identifier>
      <signaturekey>*****</signaturekey>
    </node>
  </input>
  <security>
    <verification>
      <if registration='registered=='yes' '>
        <signaturekey>*****</signaturekey>
        <identifier>decrypt(12345, signaturekey)</identifier>
      <else/>
        <signaturekey>undefined</signaturekey>
        <identifier>12345</identifier>
      </if>
    </verification>
  </security>
</action>
```

LISTING B.7: Sensor node access control

```
<action type='access control response'>
    <node>
       <session>33333</session>
       <if registration='registered=='yes' '>
           <access>yes</access>
       <elseif registration='registered=='no' '>
           <access>yes</access>
        <else>
            <identifier>12346</identifier>
       </if>
  </node>
</action>
```

LISTING B.8: Sensor node access control response

B.0.4 Node data transmission

When the sensor node sends the data *d* relating to the classroom occupancy to *IoTPlatform*, it is stored in a repository as a "raw" data, waiting to be analyzed. The kind of action performed is defined as "transmission". *IoTPlatform* receives, as input from the node, its identifier, the data *d* and the type of data *dt* which is being transmitted (e.g., integer). If the source is registered, as it is for the sensor node, *d* and *dt* are encrypted with the proper *NodeActionKey*, while the identifier is encrypted with *NodeSignatureKey*. At this stage, no data verification has to be performed, since the data is not yet analyzed. Listing B.9 represents the XML syntax of the described behavior, where the tag *<message>* identifies the content of the transmitted packet.

```
<action type=='transmission'>
  <input>
    <message>
       <session>33333</session>
      <node>
      <identifier>12345 || encrypt(12345, signaturekey)</
    identifier>
      </node>
      <data>d || encrypt(d, actionkey)</data>
      <datatype>dt || encrypt(dt, actionkey)
          </datatype>
    </message>
  </input>
</action>
```

LISTING B.9: Sensor node data transmission

B.0.5 Data processing

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 *NodeActionKey* and the encryption algorithm is stored by *IoTPlatform* after the registration phase (Listing B.4). Then the data are also the object of the "score assessment" action and sent to specific modules for QoP and DQ level evaluation. Score assessment is also performed for the data received by non registered 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. The XML representation is shown in Listing B.10.

```
<action type=='data processing'>
  <input>
    <message>
       <session>33333</session>
      <node>
          <identifier>12345 || encrypt(12345, signaturekey)</
    identifier>
      </node>
      <data>d || encrypt(d, actionkey)</data>
      <datatype>dt || 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 B.10: Sensor node data processing

Summarizing, node interactions with *IoTPlatform* are represented in the sequence diagram in Figure B.1.

B.0.6 User access control

Once the users are registred with *IoTPlatform* and have an associated profile, then they can request the services provided by the IoT system, in compliance with the owned function, defined during the registration phase.

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 services provided and also to protect the exchanged information. The student accesses the IoT system through his username, identifier and *UserSignatureKey*. Moreover, the student registers himself for a session playing a specific function (in this case, a student), therefore the credentials have to match 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 credentials. For example, the student previously described accesses the IoT system as presented in Listing B.11. The first action performed is the evaluation of the existence of the

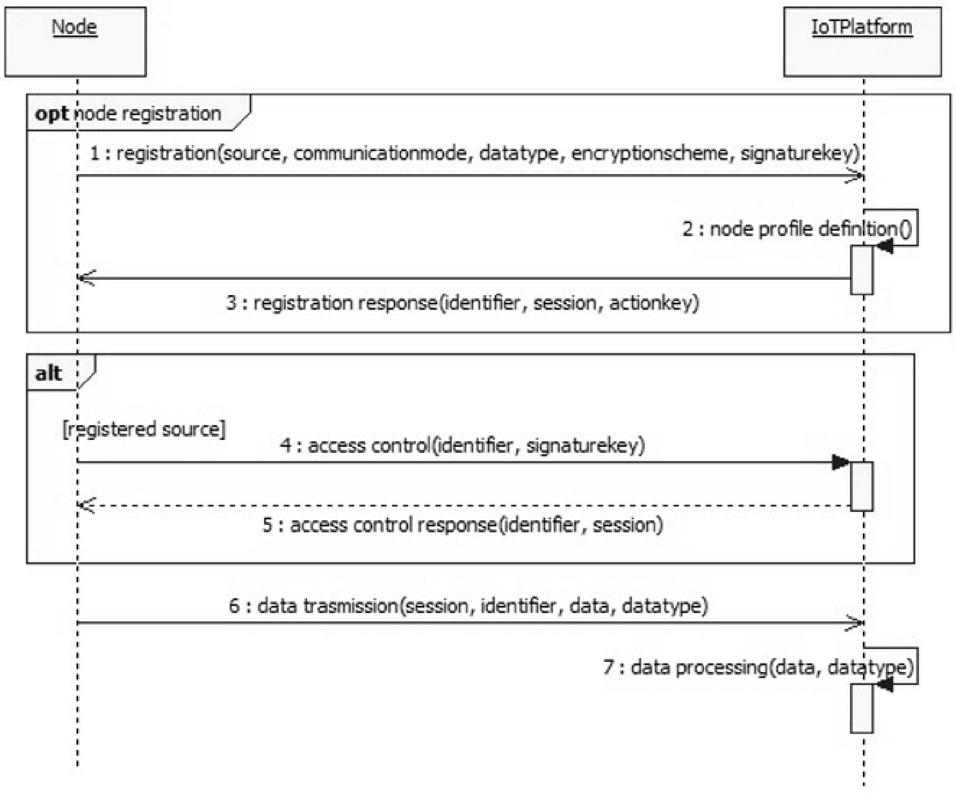

Node
IoTPlatform
opt node registration
1 : registration(source, communicationmode, datatype, encryptionscheme, signaturekey)
2 : node profile definition()
3 : registration response(identifier, session, actionkey)
alt
[registered source]
4 : access control(identifier, signaturekey)
5 : access control response(identifier, session)
6 : data trasmission(session, identifier, data, datatype)
7 : data processing(data, datatype)

FIGURE B.1: Node interactions with IoT platform

user with username Nick with the specified student function and identifier in the repository. Note that the service request and provision depend on the function currently performed by the user himself, since a user may be registered under different functions. In fact, for example, the university dean can be also a professor, but, in comparison to the professor function, the dean may access more information.

```
<action type=='access control'>
  <input>
    <user>
      <username>Nick</username>
      <identifier>encrypt(34567, signaturekey)</identifier>
      <signaturekey>*****</signaturekey>
      <function>Student</function>
  </user>
  </input>
  <security>
    <verification>
      <signaturekey>*****</signaturekey>
      <function>Student</function>
      <identifier>decrypt(12345, signaturekey)</identifier>
    </verification>
  </security>
</action>
```

LISTING B.11: Student access control

B.0.7 User service request

A student, who has registered, can ask for the services made available by *IoTPlatform*; for example, through the mobile application installed on his personal devices, he wants to be aware about the classroom occupancy, in order to find a place for studying. The request for 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 *ClassroomOccupancy*. Listing B.12 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 current session id, the activated functions, and the available permissions. Moreover, as regards the service invocation, the services are treated as object instances identified by a hierarchical name space (e.g. a URI). A service is conceived as a piece of 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.

```
<action type=='service request'>
  <input>
    <message>
      <session>44444</session>
      <user><username>Nick</username>
      <identifier>34567</identifier>
      <function>Student</function></user>
      <service>encrypt(ClassroomOccupancy, actionkey)</service>
    </message>
  </input>
</action>
```

LISTING B.12: Student service request

B.0.8 Service provision

User interactions with the IoT system are complex, since the data management must be customized depending on user functions and preferences in terms of security and data quality. Two functions are considered, for two kinds of users:

- A student, who wants to exploit the classroom occupancy service *ClassroomOccupancy* provided by *IoTPlatform* through his mobile device.

  The service request, previously encrypted with the proper *UserActionKey*, has to be decrypted, in order to verify that 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., the student): if the requesting user is not a student, then the request is discarded (as indicated by the tag *<discard>*); whereas, if the requesting user corresponds to a student, then *IoTPlatform* proceeds with the "service verification" action, in which the service *ClassroomOccupancy* request is decrypted with the proper *UserActionKey*. At this point, the system retrieves (through the function "getData") the data corresponding to the requested service *ClassroomOccupancy* (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 B.3). In Listing B.13, the XML code expresses this behavior defining a function named "dataComplianceForUser" with two parameters (the source of the data and the username); this function retrieves the QoP and DQ scores related to the source and compares them with the user preferences; as a result, only the compliant data are sent to the student (included in the tag *<result>*), the others are represented by an empty *<result>* tag. Note that the proposed language is independent from the implementation of the presented functions (e.g., "verification", "dataComplianceForUser").

```
<action type=='service provision'>
  <input>
    <message>
      <session>44444</session>
      <user><username>Nick</username>
      <identifier>34567</identifier>
      <function>Student</function></user>
      <service>encrypt(ClassroomOccupancy, actionkey)</
    service>
    </message>
  </input>
  <security>
  <if cond='user.function != 'Student' '>
      <discard/>
    <else/>
      <serviceverification>
  <service>decrypt(ClassroomOccupancy, actionkey)</service>
      </serviceverification>
    </if>
  </security>
    <results>
      <declare name='listOfData'
        function='getData' service='ClassroomOccupancy'/>
      <foreach item='data' items='listOfData'>
        <if  function='dataComplianceForUser'
        data='ClassroomOccupancy.data' user='Nick'>
          <result>data</result>
        <else/>
          <result></result>
        </if>
    </results>
</action>
```

LISTING B.13: Service provision - Student

- An employee with username *Bob*, who wants to analyze the student preferences and behaviors in terms of the times at which they occupy the classrooms for studying.

  With respect to the student, the employee can receive not only the data related to the occupancy of the classrooms at the requesting time (i.e., in real time), but all the data of a specific period of time (depending on the request). Listing B.14 describes the employee behavior; he is able to access all data without filtering, in order to perform further analysis (e.g., statistics). As regards the information disclosed, the employee should not be able to see data that could allow him to identify the students (classified as identifiable data). For statistical purposes, he does not need to know the specific identity of the students, in agreement with privacy law. Finally, note that, as for the case of the student, it is only possible to access data that are compliant with the function played.

```
<action type=='service provision'>
  <input>
    <message>
       <session>55555</session>
      <user><username>Bob</username>
      <identifier>56789</identifier>
      <function>Employee</function></user>
      <service>encrypt(ClassroomOccupancy, actionkey)</service>
    </message>
  </input>
  <security>
    <if cond='user.function != 'Employee' '>
      <discard/>
    <else/>
      <serviceverification>
       <service>decrypt(ClassroomOccupancy, +actionkey)</service>
      </serviceverification>
     </if>
  </security>
  <results>
    <declare name='listOfData' function='getData' service='
    ClassroomOccupancy'/>
      <foreach item='data' items='listOfData'>
        <result>data</result>
  </results>
</action>
```

LISTING B.14: Service provision - Employee

Summarizing, user interactions with *IoTPlatform* are represented in the sequence diagram in Figure B.2.

B.0.9 Service request violation

Since the goal of an enforcement mechanism is to force the system to be compliant with the policies defined, it then must initiate the proper countermeausers when an attempt to attack the system is detected (e.g., a user tries to request services or perform actions denied to him/her).

For example, if a user, registered with the function professor and username *Alice*, attempts to access the information related to the food intolerances of his/her students, the enforcement mechanism must prevent the disclosure of the requested data, which are only available, for instance, to the cafeteria service employees (presumably in anonymous form). Listing B.15 shows the *IoTPlatform* behavior in such a situation. Note that, with respect to Listing B.13, *IoTPlatform* does not proceed with the retrieval of the data corresponding to the requested service, since the couple username-function does not match 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 potential violations.

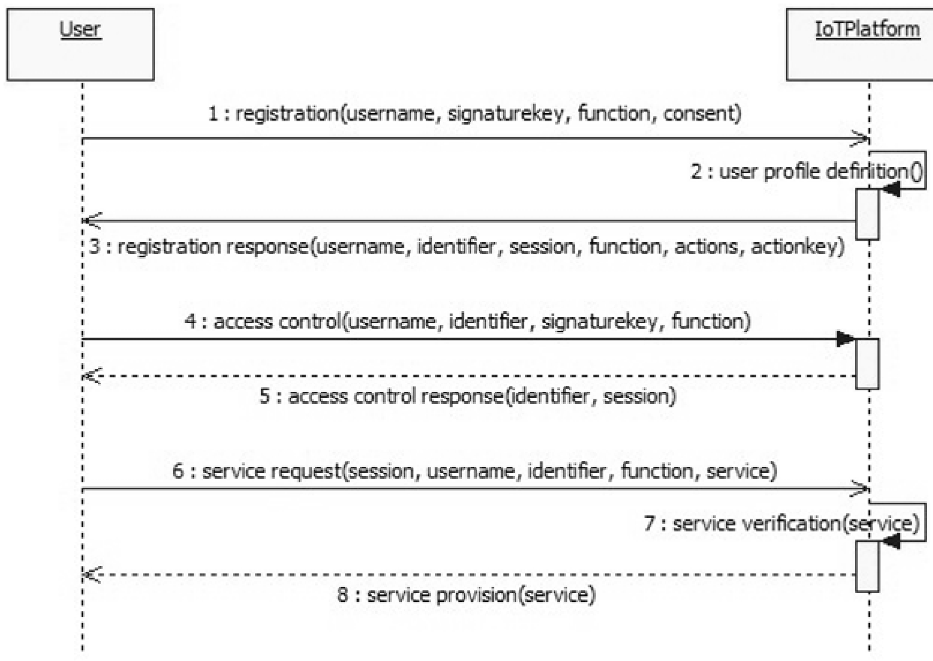

User
IoTPlatform
1 : registration(username, signaturekey, function, consent)
2 : user profile definition()
3 : registration response(username, identifier, session, function, actions, actionkey)
4 : access control(username, identifier, signaturekey, function)
5 : access control response(identifier, session)
6 : service request(session, username, identifier, function, service)
7 : service verification(service)
8 : service provision(service)

FIGURE B.2: User interactions with IoT platform

```
<action type=='service provision'>
  <input>
    <message>
      <session>666666</session>
      <user><username>Alice</username>
      <identifier>45678</identifier>
      <function>Professor</function></user>
      <service>encrypt(StudentFoodIntolerance, *****)</service>
    </message>
  </input>
  <security>
    <if cond='user.function != 'Professor' '>
      <discard/>
    <else/>
      <serviceverification>
        <service>decrypt(StudentFoodIntolerance, *****)</service>
      </serviceverification>
     </if>
  </security>
  <results>
        <result>Requested data are not available for a user
    registered as a professor</result>
  </results>
</action>
```

LISTING B.15: Service request violation

Bibliography

[1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S.B. Zdonik. "Aurora: a new model and architecture for data stream management". In: *VLDB Journal* 12.2 (2003), pp. 120–139.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. "The design of the borealis stream processing engine". In: *In CIDR*. 2005, pp. 277–289.

[3] M. R. Abdmeziem and D. Tandjaoui. "An end-to-end secure key management protocol for e-health applications". In: *Computers & Electrical Engineering* 44 (2015), pp. 184–197.

[4] L. A.Grieco, M. B. Alaya, T. Monteil, and K. Drira. "Architecting Information Centric ETSI-M2M systems". In: *IEEE PerCom*. 2014.

[5] M. Akgun and M. U. Caglayan. "Providing destructive privacy and scalability in RFID systems using PUFs". In: *Ad Hoc Networks* 32 (2015), pp. 32–42.

[6] I. Akyildiz, W.Su, Y.Sankarasubramaniam, and E. Cayirci. "A survey on sensor networks". In: *IEEE Communications Magazine* 40.8 (2002), pp. 102–114.

[7] A. Alcaide, E. Palomar, J. Montero-Castillo, and A. Ribagorda. "Anonymous authentication for privacy-preserving IoT target-driven applications". In: *Computers & Security* 37 (2013), pp. 111–123.

[8] M. Ali, M. ElTabakh, and C. Nita-Rotaru. *FT-RC4: A robust security mechanism for data stream systems*. Tech. rep. TR-05-024. Purdue University, 2005.

[9] J. An, X. Gui, W. Zhang, J. Jiang, and J. Yang. "Research on social relations cognitive model of mobile nodes in Internet of Things". In: *Journal of Network and Computer Applications* 36.2 (2013), pp. 799–810.

[10] J. Anderson and L. Rainie. "The Internet of Things Will Thrive by 2025, PewResearch Internet Project". In: *http://www.pewinternet.org/2014/05/14/internet-of-things/, May 2014* (2014).

[11] L. Atzori, A. Iera, and G. Morabito. "The Internet of Things: A Survey". In: *Computer Networks* 54.15 (Oct. 2010), pp. 2787–2805.

[12] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad. "Proposed embedded security framework for Internet of Things". In: *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.

[13] J.C.M. Baeten. “A brief history of process algebra”. In: *Theoretical Computer Science* 335.2-3 (2005), pp. 131–146.

[14] I.E. Bagci, S. Raza, T. Chung, U. Roedig, and T. Voigt. “Combined secure storage and communication for the Internet of Things”. In: *IEEE International Conference on Sensing, Communications and Networking*. New Orleans, USA, 2013, pp. 523–631.

[15] D. Ballou and H. Pazer. “Modeling data and process quality in multi-input, multi-output information systems”. In: *Management science* 31.2 (1985), pp. 150–162.

[16] S. Bandyopadhyay, M. Sengupta, S. Maiti, and 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.

[17] A. Banks and R. Gupta. “MQTT Version 3.1. 1”. In: *OASIS Standard* (2014).

[18] F. Bao and I. Chen. “Dynamic trust management for internet of things applications”. In: *International Workshop on Self-aware internet of things*. USA, San Jose, 2012, pp. 1–6.

[19] F. Bao and I. Chen. “Trust management for the Internet of Things and its application to service composition”. In: *13th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2012*. San Francisco, CA, United States, 2012, pp. 1–6.

[20] D. Barbagallo, C. Cappiello, A.Coen-Porisini, P. Colombo, M. Comerio, F. De Paoli, C. Francalanci, and S. Sicari. “Towards the Definition of a Framework for Service Development in the Agrofood Domain: A Conceptual Model”. In: *WEBIST 2012*. Porto, Portugal, 2012.

[21] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. “Methodologies for data quality assessment and improvement”. In: *ACM computing surveys* 41.3 (2009), p. 16.

[22] G. Bella, R. Giustolisi, and S. Riccobene. “Enforcing privacy in e-commerce by balancing anonymity and trust”. In: *Computers & Security* 30.8 (2011), pp. 705–718.

[23] A. Belokosztolszki, D. Eyers, P. Pietzuch, J. Bacon, and K. Moody. “Role-based access control for publish/subscribe middleware architectures”. In: *2nd international workshop on Distributed event-based systems*. 2003, pp. 1–8.

[24] C. Bertolissi and M. Fernández. “A metamodel of access control for distributed environments: Applications and properties”. In: *Information and Computation* 238 (2014), pp. 187–207.

[25] M. Bishop. *Computer Security: Art and Science*. Addison Wesley, 2003.

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

[27] M. Bovee, R. Srivastava, and B. Mak. “A conceptual framework and belief-function approach to assessing overall information quality”. In: *International Journal of Intelligent Systems* 18.1 (2003), pp. 51–74.

[28] D.F.C. Brewer and M.J. Nash. "The Chinese Wall security policy". In: *IEEE Symposium on Security and Privacy*. Oakland, CA, 1989, pp. 206–214.

[29] *BUTLER project*. http://www.iot-butler.eu.

[30] J. Cao, B. Carminati, E. Ferrari, and K. L. Tan. "CASTLE: Continuously Anonymizing Data Streams". In: *IEEE Transactions on Dependable and Secure Computing* 8.3 (2011), pp. 337–352.

[31] C. Cappiello and F. A. Schreiber. "Quality- and Energy-aware Data Compression by Aggregation in WSN Data Streams". In: *IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA, 2009, pp. 1–6.

[32] B. Carminati, E. Ferrari, and K. L. Tan. "A framework to enforce access control over data streams". In: *ACM Trans. Inform. Syst. Sec., TISSEC* 13.3 (2010), pp. 1–31.

[33] B. Carminati, E. Ferrari, and K. L. Tan. "Enforcing Access Control Over Data Streams". In: *12th ACM symposium on Access control models and technologies*. Sophia Antipolis, France, 2007, pp. 21–30.

[34] B. Carminati, E. Ferrari, and K. L. Tan. "Specifying access control policies on data streams". In: *Database System for Advanced Applications Conference*. Bangkok, Thailand, 2007, pp. 410–421.

[35] H. Chan and A. Perrig. "Security and privacy in sensor networks". In: *IEEE Communications Magazine* 36.10 (2003), pp. 103–105.

[36] H. Chan, V. Gligor, A. Perrig, and G. Muralidharan. "On the distribution and revocation of cryptographic keys in sensor networks". In: *IEEE Transactions on Dependable and Secure Computing* 2.3 (2005), pp. 233–247.

[37] K. M. Chandy and L. Lamport. "Distributed snapshots: determining global states of distributed systems". In: *ACM Transactions on Computer Systems* 3.1 (1985), pp. 63–75.

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

[39] T. Y. Chen. "Knowledge sharing in virtual enterprises via an ontology-based access control approach". In: *Computers in Industry* 59.5 (2008), pp. 502–519.

[40] A. Cherkaoui, L. Bossuet, L. Seitz, G. Selander, and R. Borgaonkar. "New paradigms for access control in constrained environments". In: *9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip*. Montpellier, 2014, pp. 1–4.

[41] *Cisco*. http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoE_Economy.pdf.

[42] D. Conzon, T. Bolognesi, P. Brizzi, A. Lotito, R. Tomasi, and M.A. Spirito. "The VIRTUS middleware: An XMPP based architecture for secure IoT communications". In: *21st International Conference on Computer Communications and Networks*. Munich, Germany, 2012, pp. 1–6.

[43] M. DellÁmico, G. Serme, M. S. Idrees, A. S. de Oliveira, and Y. Roudier. "HiPoLDS: A Hierarchical Security Policy Language for Distributed Systems". In: *Information Security Technical Report* 17.3 (2013), pp. 81–92.

[44] G. Dini and L. Lopriore. "Key propagation in wireless sensor networks". In: *Computers & Electrical Engineering* 41 (2015), pp. 426–433.

[45] G. Dini and P. Perazzo. "Integration of Privacy Protection Mechanisms in Location-Based Services". In: *International Conference on Advanced Information Networking and Applications Workshops*. 2013, pp. 717–724.

[46] G. Dini, F. Martinelli, I. Matteucci, A. Saracino, and D. Sgandurra. "Introducing Probabilities in Contract-Based Approaches for Mobile Application Security". In: *8th International Workshop on Data Privacy Management and Autonomous Spontaneous Security* (2014).

[47] M. C. Domingo. "An overview of the internet of underwater things". In: *Journal of Network and Computer Applications* 35.6 (2012), pp. 1879–1890.

[48] P. Dong, J. Guan, X. Xue, and H. Wang. "Attack-resistant trust management model based on beta function for distributed routing in Internet of Things". In: *China Communications* 9.4 (2012), pp. 89–98.

[49] W. Du, J. Deng, Y.S. Han, P. Varshney, J. Katz, and A. Khalili. "A pairwise key predistribution scheme for wireless sensor networks". In: *ACM Transactions on Information and System Security* 8.2 (2005), pp. 228–258.

[50] *EBBITS project*. http://www.ebbits-project.eu/.

[51] M. Elkhodr, S. Shanhrestani, and H. Cheung. "A review of mobile location privacy in the Internet of Things". In: *International Conference on ICT and Knowledge Engineering*. Bangkok, Thailand, 2012, pp. 266–272.

[52] Y. Elrakaiby, F. Cuppens, and N.Cuppens-Boulahia. "Formal enforcement and management of obligation policies". In: *Data & Knowledge Engineering* 71.1 (2012), pp. 127–147.

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

[54] L. Eschenauer and V. D. Gligor. "A Key-management Scheme for Distributed Sensor Networks". In: *9th ACM Conference on Computer and Communications Security*. 2002, pp. 41–47.

[55] J. P. Espada, V. G. Díaz, R. G. Crespo, O. S.Martínez, B.C. Pelayo G-Bustelo, and J. M. Cueva Lovelle. "Using extended web technologies to develop Bluetooth multi-platform mobile applications for interact with smart things". In: *Information Fusion* 21.0 (2015), pp. 30–41.

[56] *EU-Japan project*. http://www.eurojapan-ict.org/.

[57] *European FP7 IoT@Work project*. http://iot-at-work.eu.

[58] D. Evans and D.M. Eyers. “Efficient data tagging for managing privacy in the Internet of Things”. In: *IEEE Int. Conf. on Green Computing and Communications, GreenCom, Conf. on Internet of Things, iThings and Conf. on Cyber, Physical and Social Computing, CPSCom*. Besancon, France, 2012, pp. 244–248.

[59] H. Feng and W. Fu. “Study of recent development about privacy and security of the internet of things”. In: *International Conference on Web Information Systems and Mining*. Sanya, 2010, pp. 91–95.

[60] D. Ferraiolo, V. Atluria, and S. Gavrila. “The Policy Machine: A novel architecture and framework for access control policy specification and enforcement”. In: *Journal of Systems Architecture* 57.4 (2011), pp. 412–424.

[61] *FIRE EU-China project*. http://www.euchina-fire.eu/.

[62] *FIRE EU-Korea project*. http://eukorea-fire.eu/.

[63] *Forbes*. http://www.forbes.com/sites/louiscolumbus/2015/12/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2015.

[64] P. Fremantle, B. Aziz, J. Kopecky, and P. Scott. “Federated Identity and Access Management for the Internet of Things”. In: *IEEE International Workshop on Secure Internet of Things*. 2014, pp. 10–17.

[65] *Gartner*. http://www.gartner.com/newsroom/id/3114217.

[66] G.Colistra, V. Pilloni, and L. Atzori. “The problem of task allocation in the Internet of Things and the consensus-based approach”. In: *Computer Networks* 73.0 (2014), pp. 98–111.

[67] J.T. Geng and X.B. Xiong. “Research on mobile information access based on internet of things”. In: *Applied Mechanics and Materials* 539 (2014), pp. 460–463.

[68] T. Gomes, S. Pinto, A. Tavares, and J. Cabral. “Towards an FPGA-based edge device for the Internet of Things”. In: *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. 2015, pp. 1–4.

[69] A. Gòmez-Goiri, P. Orduna, J. Diego, and D. Lòpez-de-Ipina. “Otsopack: Lightweight semantic framework for interoperable ambient intelligence applications”. In: *Computers in Human Behavior* 30 (2014), pp. 460–467.

[70] F. Goncalves, J. Macedo, M. J. Nicolau, and A. Santos. “Security architecture for mobile e-health applications in medication control”. In: *21st International Conference on Software, Telecommunications and Computer Networks*. Primosten, 2013, pp. 1–8.

[71] T. Goovaerts, B. De Win, and W. Joosen. “Infrastructural Support for Enforcing and Managing Distributed Application-Level Policies”. In: *Electronic Notes in Theoretical Computer Science* 197.1 (2008), pp. 31–43.

[72] V. Goyal, O. Pandey, A. Sahai, and B. Waters. “Attribute-based Encryption for Fine-grained Access Control of Encrypted Data”. In: *13th ACM Conference on Computer and Communications Security*. Alexandria, Virginia, USA, 2006, pp. 89–98.

[73] T. M. Gronli, P. Pourghomi, and G. Ghinea. "Towards NFC payments using a lightweight architecture for the Web of Things". In: *Computing* (2014).

[74] G.Sharmam, S. Bala, and A. K. Verma. "Security Frameworks for Wireless Sensor Networks-Review". In: *2nd International Conference on Communication, Computing & Security*. 2012, pp. 978–987.

[75] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660.

[76] B. Guo, D. Zhang, Z. Wang, Z. Yu, and X. Zhou. "Opportunistic IoT: Exploring the Harmonious Interaction Between Human and the Internet of Things". In: *Journal of Network and Computer Applications* 36.6 (Nov. 2013), pp. 1531–1539.

[77] S. Gusmeroli, S. Piccionea, and D. Rotondi. "A capability-based security approach to manage access control in the Internet of Things". In: *Mathematical and Computer Modelling* 58.5-6 (2013), pp. 1189–1205.

[78] *HACMS project*. http://www.defenseone.com/technology.

[79] B. Halak, M. Zwolinski, and M. Mispan. "Overview of PUF-Based Hardware Security Solutions for the Internet of Things". In: (2016).

[80] M. A. Hammad, M. J. Franklin, W.G. Aref, and A. K. Elmagarmid. "Scheduling for shared window joins over data streams". In: *29th International Conference on Very Large Data Bases*. Berlin, Germany, 2003, pp. 297–308.

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

[82] Y. Hong. "A resource-oriented middleware framework for heterogeneous internet of things". In: *International Conference on Cloud Computing and Service Computing*. Shanghai, China, 2012, pp. 12–16.

[83] C. Hu, J. Zhang, and Q. Wen. "An identity-based personal location system with protected privacy in IoT". In: *4th IEEE International Conference on Broadband Network and Multimedia Technology*. Shenzhen, China, 2011, pp. 192–195.

[84] M. Hua-Dong. "Internet of Things: Objectives and Scientific Challenges". In: *Journal of Computer Science and Technology* 26.6 (2011), pp. 919–924.

[85] X. Huang, R. Fu, B. Chen, T. Zhang, and A.W. Roscoe. "User interactive Internet of things privacy preserved access control". In: *7th International Conference for Internet Technology and Secured Transactions*. London, United Kingdom, 2012, pp. 597–602.

[86] *HYDRA project*. http://www.hydramiddleware.eu/.

[87] *IBM and Eurotech "MQTT v3.1 protocol specification"*.

[88] *iCORE project*. http://www.iot-icore.eu.

[89] *IOT-EST project*. http://ict-iotest.eu/iotest/.

[90] M.A.M. Isa, N.N. Mohamed, H. Hashim S.F.S. Adnan, J. Manan, and R. Mahmod. “A lightweight and secure TFTP protocol for smart environment”. In: *IEEE Symposium on Computer Applications and Industrial Electronics*. Kota Kinabalu, Malaysia, 2012, pp. 302–306.

[91] A.J. Jara, V.P. Kafle, and A.F. Skarmeta. “Secure and scalable mobility management scheme for the Internet of Things integration in the future internet architecture”. In: *International Journal of Ad Hoc and Ubiquitous Computing* 13.3-4 (2013), pp. 228–242.

[92] Y. S. Jeong, J. D. Lee, J. B. Lee, J. J. Jung, and J. H. Park. “An efficient and secure m-IPS scheme of mobile devices for human-centric computing”. In: *Journal of Applied Mathematics* Special Issue (2014), pp. 1–8.

[93] J.Wang, S. Bin, Y. Yu, and X.X. Niu. “Distributed Trust Management Mechanism for the Internet of Things”. In: *Applied Mechanics and Materials* 347-350.4 (2013), pp. 2463–2467.

[94] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. “The eigentrust algorithm for reputation management in p2p networks”. In: *WWW’03*. New York, USA, 2003, pp. 640–651.

[95] K. Kang, Z. B. Pang, and C. Wang. “Security and privacy mechanism for health internet of things”. In: *Journal of China Universities of Posts and Telecommunications* 20 (2013), pp. 64–68.

[96] A. Kanuparthi, R. Karri, and S. Addepalli. “Hardware and embedded security in the context of internet of things”. In: *Proceedings of the 2013 ACM workshop on Security, privacy & dependability for cyber vehicles*. 2013, pp. 61–64.

[97] V. Kapsalis, D. Karelis, L. Hadellis, and G. Papadopoulos. “A context-aware access control framework for e-service provision”. In: *IEEE International Conference on Industrial Technology*. 2005, pp. 932–937.

[98] P. Kasinathan, G. Costamagna, H. Khaleel, C. Pastrone, and M.A. Spirito. “Demo: An IDS framework for internet of things empowered by 6LoWPAN”. In: Berlin, Germany, 2013, pp. 1337–1339.

[99] M. Kirsche and R. Klauck. “Unify to bridge gaps: Bringing XMPP into the Internet of Things”. In: *IEEE International Conference on Pervasive Computing and Communications Workshops*. 2012, pp. 455–458.

[100] A. Klein and W. Lehner. “Representing Data Quality in Sensor Data Streaming Environments”. In: *Data and Information Quality* 1.2 (Sept. 2009), 10:1–10:28.

[101] K. J. Knapp, R. F. Morris Jr., T. E. Marshall, and T. A. Byrd. “Information security policy: An organizational-level process model”. In: *Computers & Security* 28.7 (2009), pp. 493–508.

[102] T. Kothmayr, C. Schmitt, W. Hu, M.l Brunig, and G. Carle. “DTLS based security and two-way authentication for the Internet of Things”. In: *Ad Hoc Networks* 11.8 (2013), pp. 2710–2723.

[103] E. Kritzinger and S.H. Solms. “Cyber security for home users: A new way of protection through awareness enforcement”. In: *Computers & Security* 29.8 (2010), pp. 840–847.

[104] S. Kubler, K. Främling, and A. Buda. "A standardized approach to deal with firewall and mobility policies in the IoT". In: *Pervasive and Mobile Computing* (2014).

[105] R. Lacuesta, G. Palacios-Navarro, C. Cetina, L. Penalver, and J. Lloret. "Internet of things: where to be is to trust". In: *EURASIP Journal on Wireless Communications and Networking* 2012.1 (2012), pp. 1–16.

[106] L. Lamport. "Paxos made simple". In: *ACM Sigact News* 32.4 (2001), pp. 18–25.

[107] M. Langar, M. Mejri, and K. Adi. "Formal enforcement of security policies on concurrent systems". In: *Journal of Symbolic Computation* 46.9 (2011), pp. 997–1016.

[108] N. Lea, T. Austin, S. Hailes, and D. Kalra. "Expression of Security Policy in Medical Systems for Electronic Healthcare Records". In: *ICMISE* (2009).

[109] J. Y. Lee, W. C. Lin, and Y. H. Huang. "A lightweight authentication protocol for Internet of Things". In: *International Symposium on Next-Generation Electronics*. Kwei-Shan, 2014, pp. 1–2.

[110] J. Li, Y. Shvartzshnaider, J.-A Francisco, R.P. Martin, and K. Nagaraj D. Raychaudhuri. "Delivering internet-of-things services in MobilityFirst future internet architecture". In: *International Conference on the Internet of Things*. China, 2012, pp. 31–38.

[111] N. Li, N. Zhang, S. K. Das, and B. Thuraisingham. "Privacy preservation in wireless sensor networks: A state-of-the-art survey". In: *Ad Hoc Networks* 7.8 (2009), pp. 1501–1514.

[112] S. Li, P. Gong, Q. Yang, M. Li, J. Kong, and P. Li. "A secure handshake scheme for mobile-hierarchy city intelligent transportation system". In: *International Conference on Ubiquitous and Future Networks*. Da Nang, 2013, pp. 190–191.

[113] Z. Liang and W. Shi. "Enforcing cooperative resource sharing in untrusted p2p computing environments". In: *Mobile Network Applications* 10 (2005), pp. 251–258.

[114] W. Lindner and J. Meier. "User interactive Internet of things privacy preserved access control". In: *10th International Database Engineering and Applications Symposium*. Delhi, 2006, pp. 137–147.

[115] C. H. Liu, B. Yang, and T. Liu. "Efficient naming, addressing and profile services in Internet-of-Things sensory environments". In: *Ad Hoc Networks* 18.0 (2013), pp. 85–101.

[116] D. Liu and P. Ning. "Establishing pairwise keys in distributed sensor networks". In: *10th ACM conference on Computer and communications security*. Washington, DC, USA, 2003, pp. 52–61.

[117] T. Liu, Y.W. Guan, Y.Q. Yan, L. Liu, and Q. Deng. "A WSN-oriented key agreement protocol in internet of things". In: *3rd International Conference on Frontiers of Manufacturing Science and Measuring Technology*. LiJiang, China, 2012, pp. 1792–1795.

[118] Y. Liu, Z. Chen, F. Xia, X. Lv, and F. Bu. "A trust model based on service classification in mobile services". In: *IEEE/ACM International Conference on Green Computing and Communications, GreenCom, IEEE/ACM International Conference on Cyber, Physical and Social Computing, CPSCom*. Hangzhou, China, 2010, pp. 572–576.

[119] Y. Liu, Z. Chen, F. Xia, X. Lv, and F. Bu. "An integrated scheme based on service classification in pervasive mobile services". In: *International Journal of Communication Systems* 25.9 (2012), pp. 1178–1188.

[120] Y. B. Liu, X. H. Gong, and Y. F. Feng. "Trust system based on node behavior detection in Internet of Things". In: *Journal on Communications* 35.5 (2014), pp. 8–15.

[121] J. Ma, Y. Guo, J. Ma, J. Xiong, and T. Zhang. "A hierarchical access control scheme for perceptual layer of IoT". In: *Computer Research and Development* 50.6 (2013), pp. 1267–1275.

[122] R. Macfarlane, W. Buchanan, E. Ekonomou, O. Uthmani, L. Fan, and O. Lo. "Formal security policy implementations in network firewalls". In: *Computers & Security* 31.2 (2012), pp. 253–270.

[123] P. Mahalle, S. Babar, N.R. Prasad, and R. Prasad. "Identity management framework towards Internet of Things (IoT): Roadmap and key challenges". In: *Communications in Computer and Information Science* 89 (2010), pp. 430–439.

[124] P. N. Mahalle, P. A. Thakre, N. R. Prasad, and R. Prasad. "A Fuzzy Approach to Trust Based Access Control in Internet of Things". In: *3rd International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems*. NJ, Atlantic City, 2013, pp. 1–5.

[125] J. Mao and L. Wang. "Rapid identification authentication protocol for mobile nodes in internet of things with privacy protection". In: *Journal of Networks* 7.7 (2012), pp. 1099–1105.

[126] P. Martinez-Julia and A. F. Skarmeta. "Beyond the separation of identifier and locator: Building an identity-based overlay network architecture for the Future Internet". In: *Computer Networks* 57.10 (2013), pp. 2280–2300.

[127] I. Mashal, O.Alsaryrah, T. Chung, C. Yang, W. Kuo, and D. P. Agrawal. "Choices for interaction with things on Internet and underlying issues". In: *Ad Hoc Networks* 28.0 (2015), pp. 68–90.

[128] *Mckinsey*. http://www.mckinsey.com/industries/high-tech/our-insights/the-internet-of-things-sizing-up-the-opportunity.

[129] A. Metzger, C. Chi-Hung, Y. Engel, and A. Marconi. "Research challenges on online service quality prediction for proactive adaptation". In: *Workshop on European Software Services and Systems Research - Results and Challenges*. 2012, pp. 51–57.

[130] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. "Internet of Things: Vision, Applications and Research Challenges". In: *Ad Hoc Networks* 10.7 (Sept. 2012), pp. 1497–1516.

[131] *MobilityFirst-NP project*. http://mobilityfirst.winlab.rutgers.edu/.

[132] *MongoDB*. http://www.mongodb.org/.

[133] J. Montavont, D. Roth, and T.Noël. “Mobile {IPv6} in Internet of Things: Analysis, experimentations and optimizations”. In: *Ad Hoc Networks* 14.0 (2014), pp. 15–25.

[134] *Mosquitto, "An open source MQTT v3.1/v3.1.1 broker"*. http://mosquitto.org.

[135] *National Science Foundation project*. http://www.nsf.gov.

[136] *NDN-NP project*. http://named-data.net/.

[137] *NEBULA project*. http://nebula-fia.org/.

[138] R. Nehme, E. Rundesteiner, and E. Bertino. “A security punctuation framework for enforcing access control on streaming data”. In: *24th International Conference on Data Engineering*. Cancun, Mexico, 2008, pp. 406–415.

[139] R. Nehme, E. Rundesteiner, and E. Bertino. “Tagging stream data for rich real-time services”. In: *VLDB Endowment* 2.1 (2009), pp. 73–84.

[140] R. Neisse, G. Steri, and G. Baldini. “Enforcement of Security Policy Rules for the Internet of Things”. In: *IEEE WiMob*. Larnaca, Cyprus, 2014, pp. 120–127.

[141] K. T. Nguyen, M. Laurent, and N. Oualha. “Survey on secure communication protocols for the Internet of Things”. In: *Ad Hoc Networks* 32 (2015), pp. 17–31.

[142] Q. Ni, A. Trombetta, E. Bertino, and J. Lobo. “Privacy-aware role based access control”. In: *12th ACM Symposium on Access Control Models and Technologies*. New York, USA, 2007.

[143] H. Ning. “A security framework for the internet of things based on public key infrastructure”. In: *Advanced Materials Research* 671-674 (2013), pp. 3223–3226.

[144] M. Nitti, R. Girau, L. Atzori, A. Iera, and G. Morabito. “A Subjective Model for Trustworthiness Evaluation in the Social Internet of Things”. In: *IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications*. Australia, Sydney, 2012, pp. 18–23.

[145] B. Niu, X. Zhu, H. Chi, and H. Li. “Privacy and authentication protocol for mobile RFID systems”. In: *Wireless Personal Communications* 77.3 (2014), pp. 1713–1731.

[146] *Node.JS*. http://nodejs.org/.

[147] *oneM2M*. http://www.onem2m.org/.

[148] J. Padget and W.Vasconcelos. “Policy-carrying data: A step towards transparent data sharing”. In: *Procedia Computer Science* 52 (2015), pp. 59–66.

[149] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler. “Standardized Protocol Stack for the Internet of (Important) Things”. In: *IEEE Communications Surveys Tutorials* 15.3 (2013), pp. 1389–1406.

[150] S. Pallickara, G. Fox, and H. Gadgil. “On the Creation & Discovery of Topics in Distributed Publish/Subscribe systems”. In: *6th IEEE/ACM International Workshop on Grid Computing*. 2005, p. 8.

[151] S. Papadopoulos, Y. Yang, and D. Papadias. "Cads: continuous authentication on data streams". In: *33rd International Conference on Very Large Data Bases, address=Vienna, Austria, year = 2007, month=, pages = 135-146*.

[152] S. Papadopoulos, Y. Yang, and D. Papadias. "Continuous authentication on relational data streams". In: *VLDB Journal* 19.1 (2010), pp. 161–180.

[153] S. Papadopoulos, G. Cormode, A. Deligiannakis, and M. Garofalakis. "Lightweight authentication of linear algebraic queries on data streams". In: *ACM SIGMOD International Conference on Management of Data*. New York, USA, 2013, pp. 881–892.

[154] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. "Service-Oriented Computing: State of the Art and Research Challenges". In: *Computer* 40.11 (2007), pp. 38–45.

[155] J. A. Pavlich-Mariscal, S. A. Demurjian, and L. D. Michel. "A framework for security assurance of access control enforcement code". In: *Computers & Security* 29.7 (2010), pp. 770–784.

[156] *PeerTrack*. http://cs.adelaide.edu.au/peertrack/.

[157] L. Peng, W. Ru-chuan, S. Xiao-yu, and C. Long. "Privacy Protection Based on Key-changed Mutual Authentication Protocol in Internet of Things". In: *Communications in Computer and Information Science* 418 (2014), pp. 345–355.

[158] P. Perazzo and G. Dini. "A uniformity-based approach to location privacy". In: *Computer Communications* 64 (2015), pp. 21–32.

[159] *PERCI (PERvasiveServiCe Interaction)*. http://www.hcilab.org/projects/perci/index.htm.

[160] C. Perera, P.P. Jayaraman, A. Zaslavsky, D. Georgakopoulos, and P. Christen. "MOSDEN: An internet of things middleware for resource constrained mobile devices". In: *Annual Hawaii International Conference on System Sciences*. Washington, DC, USA, 2014, pp. 1053–1062.

[161] L. Pesonen. "A capability-based access control architecture for multi-domain publish/subscribe systems". In: *University of Cambridge, Computer Laboratory* UCAM-CL-TR-720 (2008).

[162] R. Di Pietro, L. Mancini, and S. Jajodia. "Providing secrecy in key management protocols for large wireless sensors networks". In: *Ad Hoc Networks* 1.4 (2003), pp. 455–468.

[163] G. Piro, G. Boggia, and L. A. Grieco. "A Standard Compliant Security Framework for IEEE 802.15.4 Networks". In: *IEEE World Forum on Internet of Things*. Seoul, South Korea, 2014, pp. 27–30.

[164] H. Pranata, R.I. Athauda, and G. Skinner. "Securing and governing access in ad-hoc networks of internet of things". In: *International Conference on Engineering and Applied Science*. Colombo, Sri Lanka, 2012, pp. 84–90.

[165] K. N. Prasetyo, Y. Purwanto, and D. Darlis. "An implementation of data encryption for Internet of Things using blowfish algorithm on FPGA". In: *IEEE 2nd International Conference on Information and Communication Technology (ICoICT)*. 2014, pp. 75–79.

[166] J. Rao, A. Sardinha, and N. Sadeh. "A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.1 (2009), pp. 40–56.

[167] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. "Lithe: Lightweight secure CoAP for the Internet of Things". In: *IEEE Sensors Journal* 13.10 (2013), pp. 3711–3720.

[168] A. Rizzardi, S. Sicari, D. Miorandi, and A. Coen-Porisini. "AUPS: An Open Source AUthenticated Publish/Subscribe system for the Internet of Things". In: *Information Systems* 62 (2016), pp. 29–41.

[169] A. Rizzardi, D. Miorandi, S. Sicari, C. Cappiello, and A. Coen-Porisini. "Networked Smart Objects: Moving Data Processing Closer to the Source". In: *2nd EAI International Conference on IoT as a Service*. Rome, Italy, 2015.

[170] R. Roman, J. Zhou, and J. Lopez. "On the features and challenges of security and privacy in distributed internet of things". In: *Computer Networks* 57.10 (2013), pp. 2266–2279.

[171] R. Roman, C. Alcaraz, J. Lopez, and N. Sklavos. "Key management systems for sensor networks in the context of the Internet of Things". In: *Computers & Electrical Engineering* 37.2 (2011), pp. 147–159.

[172] T. Roosta, S. Shieh, and S. Sastry. "Taxonomy of security attacks in sensor networks and countermeasures". In: *The first IEEE international conference on system integration and reliability improvements*. Vol. 25. 2006, p. 94.

[173] D. Rosario, Z. Zhao, A. Santos, T. Braun, and E. Cerqueira. "A beaconless Opportunistic Routing based on a cross-layer approach for efficient video dissemination in mobile multimedia IoT applications". In: *Computer Communications* 45.0 (2014), pp. 21–31.

[174] *Roseline project*. https://sites.google.com/site/roselineproject/.

[175] R. Safavi-Naini and H. Wang. "New Constructions for Multicast Rekeying Schemes Using Perfect Hash Families". In: *7th ACM Conference on Computer and Communications Security*. 2000, pp. 228–234.

[176] Y. Saied, A. Olivereau, D. Zeghlache, and M. Laurent. "Trust management system design for the Internet of Things: A context-aware and multi-service approach". In: *Computers & Security* 39 (2013), pp. 351–365.

[177] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. "Role-Based Access Control Models". In: *Computer* 29.2 (1996), pp. 38–47.

[178] S. R. Sandhu. "Role-based Access Control". In: vol. 46. Elsevier, 1998, pp. 237–286.

[179] A. A. Selcuk, E. Uzun, and M. R. Pariente. "A reputation-based trust management system for p2p networks". In: *CCGRID 2004*. Washington, DC, USA, 2004, pp. 251–258.

[180] J. Shi, Y. Li, and R. H. Deng. "A secure and efficient discovery service system in EPCglobal network". In: *Computers & Security* 31.8 (2012), pp. 870–885.

[181] S. Sicari, A. Rizzardi, C. Cappiello, and A. Coen-Porisini. "A NFP Model for Internet of Things Applications". In: *IEEE WiMob*. Larnaca, Cyprus, 2014, pp. 164–171.

[182] S. Sicari, C. Cappiello, F. De Pellegrini, D. Miorandi, and A. Coen-Porisini. "A Security-and Quality-aware System Architecture for Internet of Things". In: *Information Systems Frontiers* (2014), pp. 1–13.

[183] S. Sicari, A. Rizzardi, D. Miorandi, and A. Coen-Porisini. "Internet of Things: Security in the Keys". In: *12th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*. Malta, 2016, pp. 129–133.

[184] S. Sicari, A. Rizzardi, D. Miorandi, and A. Coen-Porisini. "Policy Enforcement and Synchronization across heterogeneous Internet of Things domains". In: *Technical Report* (2016).

[185] S. Sicari, A. Rizzardi, A. Coen-Porisini, L. A. Grieco, and T. Monteil. "Secure OM2M Service Platform". In: *IEEE International Conference on Autonomic Computing*. Grenoble, France, 2015, pp. 313–318.

[186] S. Sicari, A. Rizzardi, D. Miorandi, C.Cappiello, and A. Coen-Porisini. "Security Policy Enforcement for Networked Smart Objects". In: *Computer Networks* 108 (2016), pp. 133–147.

[187] S. Sicari, A. Rizzardi, L.A. Grieco, and A. Coen-Porisini. "Security, privacy and trust in Internet of Things: The road ahead". In: *Computer Networks* 76 (2015), pp. 146–164.

[188] S. Sicari, L.A. Grieco, A. Rizzardi, G. Boggia, and A. Coen-Porisini. "SETA: A SEcure sharing of TAsks in clustered wireless sensor networks". In: *IEEE WiMob*. Lyon, France, 2013.

[189] M. Simplício, P. Barreto, C. Margi, and T. Carvalho. "A survey on key management mechanisms for distributed wireless sensor networks". In: *Computer Networks* 54.15 (2010), pp. 2591–2612.

[190] J. Singh, J. Bacon, and D. Eyers. "Policy enforcement within emerging distributed, event-based systems". In: *8th ACM International Conference on Distributed Event-Based Systems*. 2014, pp. 246–255.

[191] J. Singh, D. Eyers, and J. Bacon. "Disclosure control in multi-domain publish/subscribe systems". In: *5th ACM international conference on Distributed event-based system*. 2011, pp. 159–170.

[192] J. Singh, L. Vargas, J. Bacon, and K. Moody. "Policy-based information sharing in publish/subscribe middleware". In: *IEEE Workshop on Policies for Distributed Systems and Networks*. 2008, pp. 137–144.

[193] S.Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, and A. Coen-Porisini. "A secure and quality-aware prototypical architecture for the Internet of Things". In: *Information Systems* 58 (2016), pp. 43–55.

[194] J. Su, D. Cao, B. Zhao, X. Wang, and I. You. "ePASS: An expressive attribute-based signature scheme with privacy and an unforgeability guarantee for the Internet of Things". In: *Future Generation Computer Systems* 33.0 (2014), pp. 11–18.

[195] G. Tesauro. *Practical issues in temporal difference learning*. Springer, 1992.

[196] G. D. Tormo, F. G. Marmol, and G. M. Perez. “Dynamic and flexible selection of a reputation mechanism for heterogeneous environments”. In: *Future Generation Computer Systems* (2014).

[197] M. Turkanović, B. Brumen, and M. Hölbl. “A novel user authentication and key agreement scheme for heterogeneous ad hoc wireless sensor networks, based on the Internet of Things notion”. In: *Ad Hoc Networks* 20 (2014), pp. 96–112.

[198] A. Ukil, S. Bandyopadhyay, and A. Pal. “IoT-Privacy: To be private or not to be private”. In: *IEEE INFOCOM*. Toronto, 2014, pp. 123–124.

[199] N. Ulltveit-Moe and V. Oleshchuk. “Decision-cache based XACML authorisation and anonymisation for XML documents”. In: *Computer Standards & Interfaces* 34.6 (2012), pp. 527–534.

[200] *Usable Trust in the Internet of Things*. http://www.utrustit.eu/.

[201] L. Veltri, S. Cirani, S. Busanelli, and G. Ferrari. “A novel batch-based group key management protocol applied to the Internet of Things”. In: *Ad Hoc Networks* 11.8 (2013), pp. 2724–2737.

[202] P. Waher. *Security in Internet of Things using Delegation of Trust to a Provisioning Server*. 2014.

[203] R. Wang and D. Strong. “Beyond accuracy: What data quality means to data consumers”. In: *Journal of management information systems* 12.4 (1996), pp. 5–33.

[204] X. Wang, J. Zhang, E.M. Schooler, and M. Ion. “Performance evaluation of Attribute-Based Encryption: Toward data privacy in the IoT”. In: *IEEE International Conference on Communications*. Sydney, NSW, 2014, pp. 725–730.

[205] Y. Wang and Q. Wen. “A privacy enhanced dns scheme for the internet of things”. In: *IET International Conference on Communication Technology and Application*. Beijing, China, 2011, pp. 699–702.

[206] Y. Wang, M. Qiao, H. Tang, and H. Pei. “Middleware development method for internet of things”. In: *Journal of Liaoning Technical University (Natural Science Edition)* 33.5 (2014), pp. 675–678.

[207] R. H. Weber. “Internet of Things - New security and privacy challenges”. In: *Computer Law & Security Review* 26.1 (2010), pp. 23–30.

[208] L. Wen-Mao, Y. Li-Hua, F. Bin-Xing, and Z. Hong-Li. “A Hierarchical Trust Model for the Internet of Things”. In: *Chinese Journal of Computers* 5 (2012), pp. 846–855.

[209] Z. Wu and L. Wang. “An innovative simulation environment for cross-domain policy enforcement”. In: *Simulation Modelling Practice and Theory* 19.7 (2011), pp. 1558–1583.

[210] Z. Q. Wu, Y. W. Zhou, and J. F. Ma. “A security transmission model for internet of things”. In: *Chinese Journal of Computers* 34.8 (2011), pp. 1351–1364.

[211] A. Wun and H.A. Jacobsen. “A policy management framework for content-based publish/subscribe middleware”. In: *Middleware*. Springer, 2007, pp. 368–388.

[212] *XIA-NP project*. http://www.cs.cmu.edu/ xia/.

[213] Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway. "A survey of key management schemes in wireless sensor networks". In: *Computer Communications* 30.11 (2007), pp. 2314–2341.

[214] L. Xiong and L. Liu. "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities". In: *IEEE Transactions on Knowledge and Data Engineering* 16 (2004), pp. 843–857.

[215] T. Yan and Q. Wen. "A secure mobile RFID architecture for the Internet of Things". In: *IEEE International Conference on Information Theory and Information Security*. Beijing, China, 2010, pp. 616–619.

[216] Z. Yan, P. Zhang, and A. V. Vasilakos. "A survey on trust management for Internet of Things". In: *Journal of Network and Computer Applications* 42.0 (2014), pp. 120–134.

[217] J. Yang and B. Fang. "Security model and key technologies for the Internet of things". In: *The Journal of China Universities of Posts and Telecommunications* 8.2 (2011), pp. 109–112.

[218] N. Ye, Y. Zhu, R. C. Wang, R. Malekian, and Q. M. Lin. "An efficient authentication and access control scheme for perception layer of internet of things". In: *Applied Mathematics and Information Sciences* 8.4 (2014), pp. 1617–1624.

[219] J. Yick, B. Mukherjee, and D. Ghosal. "Wireless sensor network survey". In: *Computer Networks* 52.12 (2008), pp. 2292–2330.

[220] B. Yu, M. P. Singh, and K. Sycara. "Developing trust in large-scale peer-to-peer systems". In: *First IEEE Symposium on Multi-Agent Security and Survivability*. 2004, pp. 1–10.

[221] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed. "Deploying and Managing Web Services: Issues, Solutions, and Directions". In: *VLDB Journal* 17.3 (May 2008), pp. 537–572.

[222] J. Zhang and V. Varadharajan. "Security and privacy in sensor networks". In: *Journal of Network and Computer Applications* 33.2 (2010), pp. 63–75.

[223] Y.L. Zhao. "Research on data security technology in internet of things". In: *2nd International Conference on Mechatronics and Control Engineering*. Dalian, China, 2013, pp. 1752–1755.

[224] W. Zhu, J. Yu, and T. Wang. "A security and privacy model for mobile RFID systems in the internet of things". In: *International Conference on Communication Technology Proceedings*. 2012, pp. 726–732.

[225] Y. Zhu, E. A. Rundensteiner, and G. T. Heineman. "Dynamic plan migration for continuous queries over data streams". In: *ACM SIGMOD International Conference on Management of Data*. Paris, France, 2004, pp. 431–442.