Security towards the Edge: Sticky Policy Enforcement for Networked Smart Objects

Sabrina Sicari*‡, Alessandra Rizzardi*, Daniele Miorandi§, Alberto Coen-Porisini*
*Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria,
via G. Mazzini 5 - 21100 Varese, Italy
§U-Hopper, via A. da Trento 8/2, 38122 Trento, Italy
‡Corresponding author
Email: {sabrina.sicari; alessandra.rizzardi; alberto.coenporisini}@uninsubria.it,
daniele.miorandi@u-hopper.com

***Abstract*—One of the hottest topics in the Internet of Things (IoT) domain relates to the ability of enabling computation and storage at the edges of the network. This is becoming a key feature in order to ensure the ability of managing in a scalable way service requests with low response times. This means being able to acquire, store, and process IoT-generated data closer to the data producers and data consumers. In this scenario, also security and privacy solutions must be applied in a capillary way at the edges of the network. In particular, a control on access to data generated by IoT devices is necessary for guaranteeing proper levels of security and privacy as well as for preventing violation attempts, while allowing data owners to monitor and control their information. In this paper, a sticky policy approach is proposed as a strategy for efficiently managing the access to IoT resources within an existing distributed middleware architecture. As demonstrated in the experimental evaluation, sticky policies represent a promising and efficient technique to increase the robustness (in a security perspective) of the IoT system.**

I. Introduction

Security and privacy, along with scalability and interoperability, are the main key issues potentially harming the widespread adoption of the Internet of Things (IoT) paradigm. In fact, IoT applications are typically characterised by the presence of heterogeneous entities interacting among themselves and acquiring data from the surrounding environment. A sort of global network is thus created, enabling services to end users across different application domains (e.g., smart monitoring, smart building, smart transportation, e-health context, and so on). Within such a network, a huge amount of data is sent using wireless communication technology, thereby presenting risks of violation of sensitive and/or personal information. Therefore proper mechanisms for authentication and access control must be put in place in order to protect private data by companies and service providers that want to foster the spreading of IoT-based platforms and services.

Proper solutions for security and privacy management can be modelled, exploiting various mechanisms [1], such as access-control policies enforcement, policy life cycle management, and so on. In this direction, an effective approach should be coupled with methods based upon edge computing concept [2]. Edge computing is an emerging paradigm aimed at improving the scalability and Quality of Services (QoS) for real time applications in IoT. If information and network resources are managed at the edges of the network, then also security and privacy policies should follow a similar approach. Moving data processing, storage, and security at the edge of the network brings several advantages [3], such as: (i) reducing network's latency and, thus, service response times; (ii) preventing unnecessary network resources' consumption by limiting the scope of data spreading and hence improve scalability; (iii) enhance service availability; (iv) increasing the robustness of the whole system thanks to the removal of single points of failures into the network infrastructure.

Besides the interest of companies and organizations, the new approaches should improve user experience, so that users can feel more in control of their confidential data, especially in the context of multiparty interactions, as typically happens in IoT settings. Little has been done so far to directly involve users (or entities acting on their behalf) in the explicit management and enforcement of security and privacy policies. By moving security and privacy enforcement closers to the end users it becomes easier to provide users with end-to-end control on how their data is accessed, processed and used.

Sticky policies fit very well this scenario. Being directly attached to the data, and flowing with the data themselves across one or more application domains, policies can be effectively enforced in a distributed manner, enabling access based on the preferences of the data owners.

This paper presents a security and privacy-aware IoT-

based platform, based upon the sticky policy paradigm, able to meet the following requirements:

- Allowing the users to set and manage specific access control policies on their own data before sharing them on the IoT network;
- Providing a middleware architecture that manages and enforces the sticky policies on the user behalf and that allows them to monitor and even control how their information is processed;
- Moving as much as possible the data and policy management at the edge of the IoT infrastructure, in order to enhance its performance (mainly in terms of delay) and resilience towards misbehaving nodes and users.

The remainder of the article is organized as follows: Section II describes and analyses the relevant state of the art; Section III describes the IoT middleware and the enforcement framework that represents the starting point of the proposed solution; Section IV details the new enforcement framework based on sticky policies; Section V presents the experimental validation that aims at demonstrating the feasibility of the proposed solution, before concluding the paper in Section VI.

II. State of the art

In order to better clarify the innovative contribution of the presented work, we hereby present a short overview of the related literature. Generally speaking, the level of maturity of solutions in the enforcement by means of sticky policies is still rather low, in particular when considering the IoT context. In fact, to the best of authors' knowledge, there are no available approaches specifically tailored to IoT constraints and requirements [4].

Companies operating in different application domains are actively searching for efficient security and privacy management tools, able to improve the robustness of their systems at different network levels. In particular, IoT platforms, which are intrinsically characterized by the presence of heterogeneous technologies communicating over the wireless medium, need to provide security and privacy guarantees. Moreover, data can flow through different realms and administrative domains. For example, data can be sold by a company to another one, thereby increasing the vulnerability with respect to different types of attack aimed at compromising the data transmitted over the network or at disclosing confidential information. Such weaknesses mainly involve end-devices, thus leading to conceive a solution able to fix the access control issue at the edges of the network.

Also national and international regulations are increasingly imposing baseline privacy requirements concerning information sharing within companies and across different organizations. As an example, it is worth citing the privacy principles of the Organization for Economic Co-operation and Development (OECD) [5] and the recently adopted EU General Data Protection Regulation (GDPR) [6]. However, it is a matter of fact that often companies processing and managing data are not fully aware of the legal constraints or are not aware of the level of consent that has been granted by data owner.

On the other hand, most individuals have limited understanding of security and privacy polices when applied to their confidential data. In addition, people have almost no control over the faith of their data, once they have been disclosed to a third party. It is in first place a matter of trust [7]: therefore solutions for identity and privacy management are going to play a key role in protecting identities and profiles. Good management practices could also help to detect malicious activities and support forensic analysis and monitoring. If people are not willing to be involved in the protection and management of their digital assets, trusted third parties could do this on their behalf and could provide people with easy-to-use tools to monitor and keep the situation under control.

Such considerations have been partially covered by some works in the area, as described below.

Some security solutions based on the use of sticky policies in cloud environments, such as [8] [9] [10] [11], have been proposed. Other application case studies for sticky policies are related to information exchanges among mobile devices [12] or digital ecosystems [13]. Such works present various limitations, mostly regarding how to keep users in control of the access to their personal (or even sensitive) data, as well as how to react against violation attempts. Approaches proposed in the literature are based upon the usage of robust encryption mechanisms, which are associated to sticky policies, along with the capability of updating and revoking them dynamically. Yet, the literature lacks a suitable and realistic approach for building a secure, customizable, and cross-domain solution. In this paper, we tried to cope with such issues in the context of IoT, by proposing an enforcement framework based on sticky policies within a scalable, cross-domain, security and data quality-aware IoT middleware, named NOS. As will be discussed in detail in in Section V, the proposed solution is able to effectively handle violation attempts and provides the users and data sources with a wide control over the management of their information.

Data and sticky policies are expected to be encrypted before transmission. In this direction, several mechanisms have been proposed, including: Public-Key Encryption (PKE), Identity-Based Encryption (IBE), Attribute-Based Encryption (ABE), and Proxy Re-Encryption (PRE) [14]. They differ from each other in terms of robustness and complexity. Such techniques can

be employed in order to enforce access control based on sticky policies, as described in [7] [15] [1] [14] [16] [17] [18]. All these solutions are not sufficient to enable a sticky policies framework coupled with an efficient key management system. Additionally, they do not define a hierarchy or a structured specification for sticky policies to be used in a corporate setting. Finally, an analysis of the overhead and delay of the proposed approaches is still missing. This work represents an attempt to fill said gaps.

Another important issue is about the semantic to be adopted for specifying sticky policies. No common standard exists on this aspect. Several languages may be used, such as JSON, XML, etc. In this work, we chose JSON because of its suitability for the existing NOS architecture and, mostly, because it allows the data model to dynamically evolve. Other works make use of XML, such as [19] [20], or the IBM EPAL (Enterprise Privacy Authorization Language), as in [21] [22].

A further requirement regards the access control model to be put in place. In this paper, we adopted ABAC, which allows regulating the access to resources in a more precise, flexible and fine-grained way, with respect to other solutions, based on RBAC, such as [23], or on simple privacy tags [24].

Finally, some relevant policy enforcement systems have been proposed, which do not make use of sticky policies. In Section III we referred to them as "traditional" approaches. The applicability to IoT scenarios has however received limited attention. Some works couple a standardized language, such as XML or XACML, with an extension of the baseline RBAC scheme to obtain a unitary framework [25] [26]. An *ad hoc* policy representation languages, named Hierarchical Policy Language for Distributed Systems (HiPoLDS), is defined in [27]. [28], [29] and [30] enforce access control policies by means of a proper framework named Policy Machine (PM) and of a semantic web framework, respectively. However, they do not refer to a distributed nature of the proposed solutions, which is a pivotal requirement in IoT applications.

III. System architecture

This work starts from an existing cross-domain middleware for IoT applications, called *NetwOrked Smart object (NOS)*, presented and described in [31]. NOSs are able to manage in a distributed manner the data acquired by heterogeneous sources and to enable relevant services to the end-users. In NOSs, the storage and the processing of the information are not performed by a central entity, but they are distributed among a network of smart nodes. The goal is minimizing latency, providing support for user/service mobility, and improving the resilience of the whole system. Figure 1 shows an overview of the system infrastructure, highlighting the proximity of the various network entities.

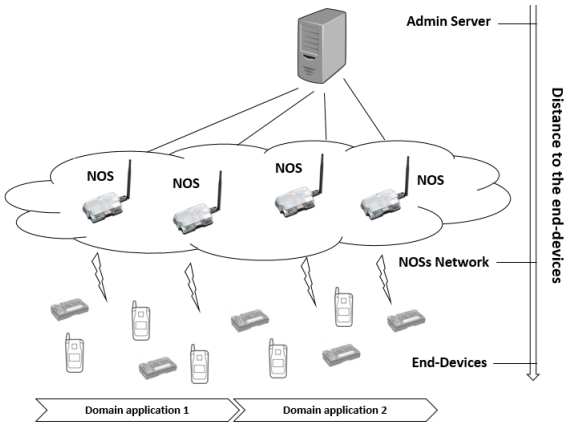

Fig. 1: Overall system infrastructure

NOSs are also able to evaluate, by means of proper algorithms [32], the security and data quality of the information transmitted, in order to satisfy user requirements, while providing a lightweight and secure information exchange process, based on an authenticated publish and subscribe mechanism [33]. In the next sections, the architectural components of NOSs will be detailed, as long as the actual enforcement framework, defined in [34]. First, we will discuss the limits of the actual policy enforcement approach and, then, we will present a more efficient solution, based on the usage of sticky policies.

A. Networked Smart Object Architecture

NOSs essentially include two main entities: (i) the nodes, conceived as heterogeneous devices (e.g., RFID, NFC, actuators, sensors etc.) which generate and process data for the IoT platform; (ii) the users, who interact with the IoT system through services making use of such IoT-generated data, typically accessing them by means of a mobile device (e.g., smartphone, tablet) connected to the Internet (e.g., through WiFi, 3G, or Bluetooth technologies).

Interfaces for the communications with the data sources (i.e., the nodes) and with the users have been defined. In the former case, HTTP protocol is adopted for collecting the data from the IoT devices and for allowing registering data sources. In fact, NOSs deal both with registered and non-registered sources. The registration is not mandatory, but it provides various advantages in terms of security: registered sources may specify an encryption scheme for their interactions with NOSs, thus increasing the level of protection of their communications. The information on the registered sources are recorded in the storage unit, named *Sources*. For

each incoming data unit, both from registered and non-registered sources, the following information are gathered: (i) the kind of data source, which describes the kind of node; (ii) the communication mode, that is, the way in which the data are collected (e.g., discrete or streaming communication); (iii) the data schema, which represents the type (e.g., number, text) and the format of the received data; (iv) the data itself; (v) the reception timestamp.

Since the received data are of different types and formats, NOSs initially put them in the *Raw Data* storage unit. Data in such a collection are periodically processed, in a batch way, by the *Data Normalization* and *Analyzers* units, in order to obtain a uniform representation and to add metadata on security aspects (i.e., level of confidentiality, integrity, privacy and robustness of the authentication mechanism) and data quality ones (i.e., level of accuracy, precision, timeliness and completeness). Data quality assessment is based on a set of rules stored in a proper format in another storage unit, named *Config*, and are detailed in [32]; this allows users who access the IoT data to filter directly by themselves the data processed by NOSs according to their personal preferences.

Communication between NOSs and users is based on the Message Queue Telemetry Transport (MQTT) protocol [35], which is used for disseminating the information to the interested users. Figure 2 summarizes the NOS's components just introduced.

A prototypical implementation of the NOSs specifications is openly accessible at https://bitbucket.org/alessandrarizzardi/nos.git.

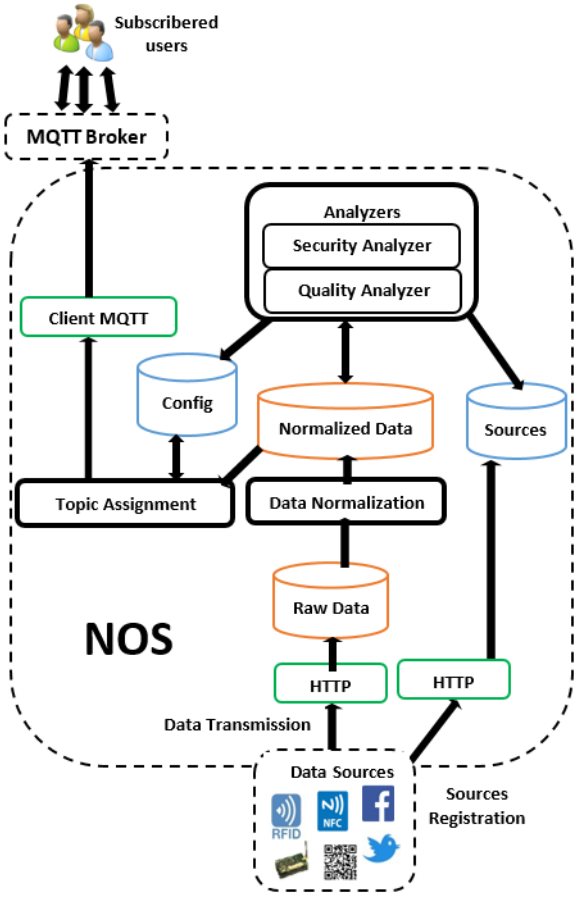

Fig. 2: NOS architecture

B. Networked Smart Object Enforcement System

NOSs' modules interact among themselves through *RESTful* interfaces. This enables NOSs' administrators to add new modules or modify the existing ones at runtime, as they work in a parallel and non-blocking manner. Moreover, the non-relational nature of *MongoDB* allows also the data model to dynamically evolve over the time. Such features allow NOSs to adopt an enforcement framework that acts as a sort of wrapper, responsible for properly managing the available resources and handling possible violation attempts without affecting the existing NOS functionality.

The integration of a policy enforcement framework with NOSs is the subject of [34]. Such framework foresees, for each NOS: (i) a *Policy Enforcement Point (PEP)*, which intercepts the access requests and queries the PDP about its acceptance; (ii) a *Policy Decision Point (PDP)*, which evaluates the access requests against the authorization policies and takes the authorization decisions; (iii) a *Policy Administration Point (PAP)*, which contains the full set of authorization policies established by the system administrators.

In our implementation, we express policies by means of a flexible interoperable specification language, based on *JSON* syntax. It is flexible enough to handle the features of IoT contextheterogeneous analyzed context both in a general-purpose and in a customizable way. The chosen access control model is the *Attribute Based Access Control (ABAC)* [36], because of its flexibility and effectiveness. In particular, Moreover, as attributes can be dynamically configured within NOSs, the system can effectively be adapted at run-time in order to accommodate specific application needs. Note that users have to register before interacting with NOSs; during the registration process, a set of attributes is assigned to them on the basis of their role in the specific application context, thus enabling them to access a given set of resources.

On this basis, a set of primitives, able to specify and enforce a large variety of attribute-based security and data quality policies has been identified, in particular, covering: node access control, node data transmission, node data processing, user access control, user service request, and service provision.

In [34] we demonstrated the technical feasibility of the proposed approach by means of the NOS's prototype in a real use case scenario, validating its robustness towards violation attempts, memory occupancy, and latency.

However, such an enforcement framework presents some limitations, more specifically in terms of data owners' monitoring capability. Moreover, since the approach followed by NOSs for data management is data-driven, then also policies should be expressed following a data-centric perspective. Therefore, security and privacy rules will be coupled with the corresponding IoT data and de-coupled from the connections among the involved entities. In other words, policies are directly attached to the data and not to the transactions, in order to cover the whole data lifecycle without requiring a central control server.

In fact, at this stage, the entire NOS IoT architecture is still conceived as a "traditional" one, i.e., based on the transmission of data to a system where access control is regulated in a centralized way. In this case, the owner of the data (i.e., a data source) has all the attributes, the encryption keys and the necessary credentials for identifying itself, thus determining access permission, and for ciphering the data to be sent to NOSs; each NOS that wants to decrypt such data has to own the related policies and credentials. Such policies/credentials must be shared and synchronized by all NOSs.

As said in Section sec:relatedwork, data owners and also data consumers are often partially or completely unaware of how their information is managed or what kind of information they receive from a particular service. This scenario is made even more complex by the presence of multiple actors, as typically happens in IoT environments. In this case, clearly, a centralized system in charge of handling the data along with the associated policies is no longer feasible. NOSs already meet the requirement of acting as a distributed middleware for data management as well as an enforcement system for regulating, in an efficient way, the access to the resources. In such a context, however, users should be more directly involved in the definition of the policies that specify how their data can be accessed and used. Also, the dissemination of the policies among different application realms (e.g., business companies) must be regulated in a more efficient, secure, and scalable way, possibly at the edges of the network. Delegating to NOSs the entire management of access control and policies' definition could make them even more vulnerable to malicious attacks (i.e., each NOS could become a single point of failure), which can hinder the reliability of the whole IoT system.

In this paper, we want to overcome such issues by providing NOSs with the ability of handling sticky policies, established by end-users and data sources and regulating how data can be accessed, by whom and for what. Following the sticky policies paradigm, policies are transmitted along with the associated data, as duly explained in Section IV-A. NOSs own no policies/credentials, while a trust authority is responsible for their management. The owner of the data sends them in an encrypted way along with the associated sticky policy; then each NOS can contact the trust authority in order to obtain the access permissions on the received data. In this way, no synchronization or policy sharing is required among the different NOSs. And, clearly, users and data sources have an in-depth control over the flow of their own information.

Figures 3 and 4 sketch the difference between the two approaches just described. In the next sections, the proposed solution will be detailed.

IV. Sticky Policy Enforcement Framework

A. Backgound on Sticky Policies

The sticky policy paradigm was first proposed by Karjoth, Schunter, and Waidner [18]. Sticky policies are able to regulate how data can be accessed and used. They are transmitted along the data they refer to throughout the entire data life cycle. Specifically, sticky policies allow to define the following aspects:

- The owner of the data;
- The data content, possibly encrypted;
- The scope of the data;
- Where and when data will be available;
- Specific obligations and restrictions.

In detail, the concept of sticky policy is to attach security and privacy policies to owners' data and drive access control decisions and policy enforcement. Sticky policies allow specifying access rule in an extremely fine-grained manner: in principle every data unit could have its own, unique, policy. Furthermore, as policies 'travel' with the data across the entire system, they could (again, in principle) provide protection over the entire data life cycle. Such an approach has been mainly introduced for security and privacy enforcement: when submitting data to a consumer, a user consents to the applicable policies selecting the proper preferences.

Such features are particularly interesting in some scenarios, as that of IoT, where user or business confidential information may flow across organizational boundaries [1]. For example, social networks may share some information with marketing companies; similarly, cloud applications may pass data, depending on a need, among different realms. Such situations represent well-known open issues in the field of security and privacy enforcement.

Starting from this premise, in this paper, we designed and developed a sticky policy-based enforcement framework tailored to the NOS middleware. As the previous

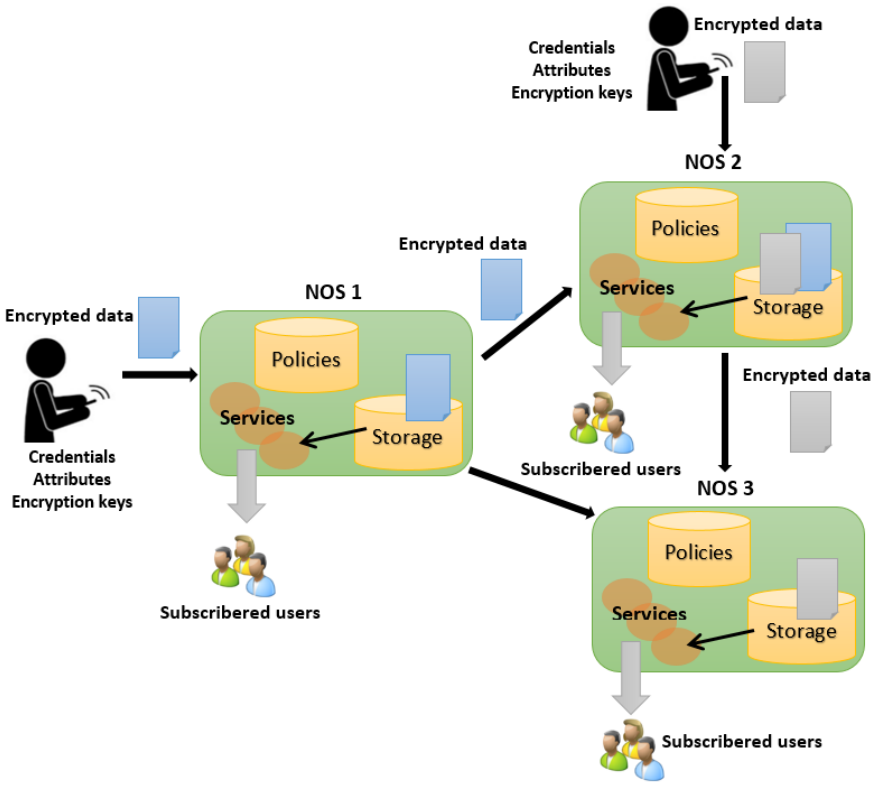

Fig. 3: The traditional approach adopted by NOS

enforcement system [34], it is integrated as a wrapper with respect to the NOSs' existing functionalities, but it works in a completely different manner, exploiting edge computing concepts, as explained in the next section.

B. Proposed solution

Data sources and IoT devices owned by the users can agree with NOSs on an encryption scheme and on encryption keys to cipher the data transmitted. This represents a first level of protection, covering basically the communication path. Together with data, also the relevant policy is sent, also properly encrypted. The policy specifies how NOSs should manage the data.

In our scheme, sticky policies are specified as described in Listing 1. A policy includes information on:

- The owner of the data (e.g., in the form of a unique identifier);
- One or more purposes for which the data can be used (e.g., statistical or analytical scope, sharing in social networks, private use within certain companies, and so on)
- A timestamp that points out the validity (i.e., the lifetime) of the data within the IoT system; once this time has elapsed, then the data must be discarded and no longer transmitted
- One or more constraints which represent the rules to be applied to data (e.g., with whom the data may be shared, if the data must be shared in an aggregated form or not, and so on).

```
{ "sticky policy": {
  "owner": "the owner of the data",
  "data": ''data content, possibly encrypted
    with the adopted encryption mechanism'',
  "policy": [{
    "scopes": [{
      "scope1": "allowed use for the data",
      "scope2": "allowed use for the data"
    }],
    "validity": ''expiration timestamp'',
    "constraints": [{
      "constraints1": "obligations and
    restrictions",
      "constraints2": "obligations and
    restrictions",
      "constraints3": "obligations and
    restrictions"
    }],
  }]
}}
```

Listing 1: JSON sticky policy schema

In order for the system to work, it is necessary to introduce a new component, i.e., the trust authority. The trust authority is responsible for:

- Defining and managing a dictionary of valid scopes and constraints, which can be required by the

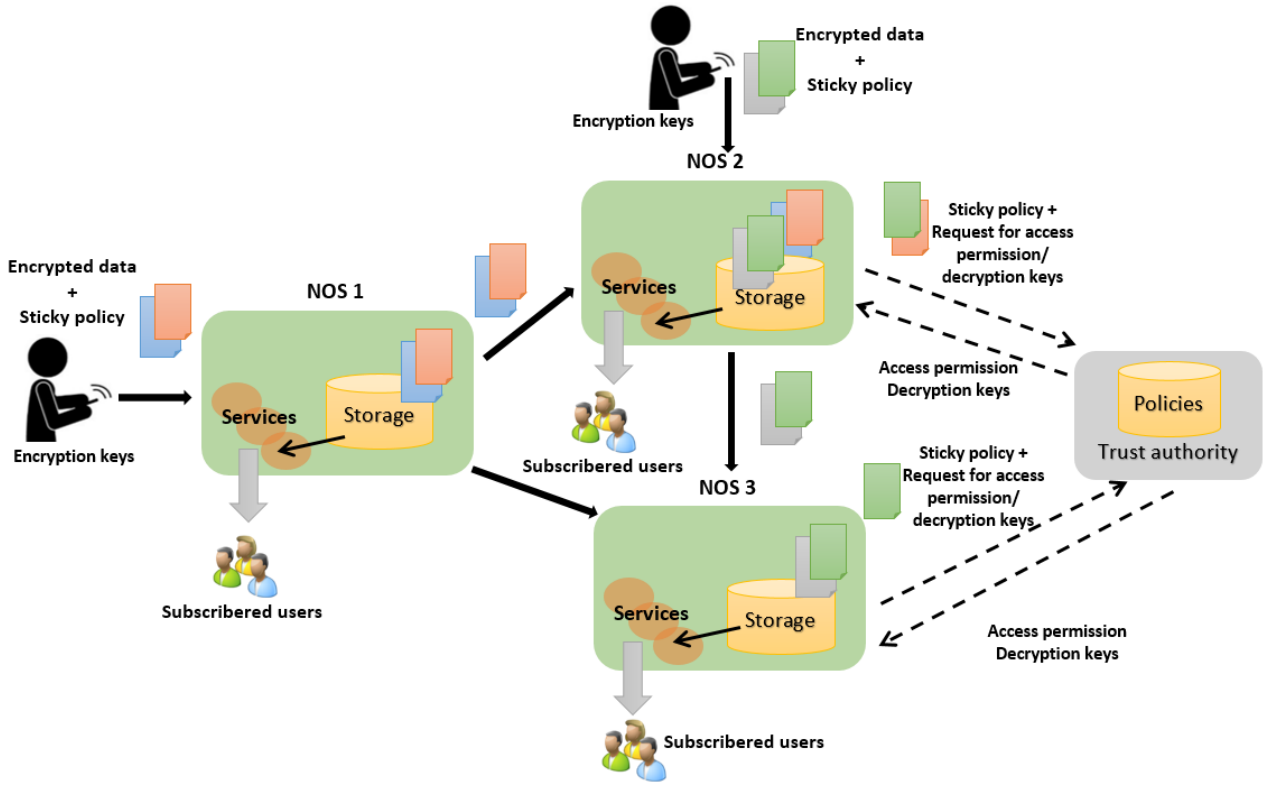

Fig. 4: The new sticky policy-based approach adopted by NOS

sources for the data transmitted within the IoT system. Such information depends on the specific application domain and may change over time.

- Providing NOS (upon reception of a query) with access decisions related to the current sticky policy as well as the decryption keys for accessing the data of interest. Note that, with respect to the previous solution presented in [34] and described in Section III-B, in this case only the PEP is located into NOSs; it represents a module in charge of intercepting the requests of access to resources, which are sent to the PDP that is located within the trust authority itself, as the PAP. As a consequence, the role of NOSs in the enforcement process is softened, and NOSs can be no longer considered single point of failure. By delegating some operations and controls to the trust authority, this also improves the overall efficiency of the NOSs middleware.

The interaction between a NOS and the trust authority works as follows:

1) When a NOS receives data from a source with the associated sticky policy, it first stores it in the *Raw Data* storage unit;
2) Once the data has been analyzed by the NOS following the procedure described in Section III-A, a topic is assigned to them;
3) Users/devices subscribed to such topic are notified of the new incoming message;
4) Access to topics is regulated on demand by NOSs through proper requests to the trust authority, by means of the associated sticky policy.

The subscription to certain topics is therefore regulated by the sticky policies associated to the single data units. More in detail, the sticky policies state the scope, the validity and the constraints to be applied to the current data; when a user/device is notified of a new published data, she has to demonstrate to own the correct attributes, compliant with the scope and the constraints contained in the sticky policy. If not, the received data cannot be decrypted. Therefore, also subscriptions will be accepted or revoked according to the current sticky policies. In this way, the owners of the data have a very powerful and fine-grained control over their information; they have not to contact NOSs or other control entities in order to modify the way in which their data are treated, since they can change directly their preferences by adapting the sticky policy sent along the dataMoreover, NOSs have not to synchronize themselves or share the same policies, because the trust authority is in charge of handling and, if required, updating the proper dictionary. Hence, the system is able to provide the following guarantees:

- Each source/user establishes its own policies on data, respecting the dictionary provided by the trust authority;
- Sources/users must trust the IoT platform (composed by NOSs and trust authority) that their policies are correctly stored, applied, and transmitted along with the associated data under specific topics;
- On the basis of the subscribers' requests, NOSs establish how to regulate the release of data under certain topics following the sticky policies' guidelines;
- In case data are transmitted towards different NOSs to be shared with further interested users/applications, sticky policies always travel with them;
- Policies and data are always transmitted in an encrypted manner; the encryption scheme is agreed between NOS and source/user case by case;
- NOSs themselves give the authorized subscribers the necessary credentials to access the information; if certain information is directly passed from an application domain to another one, then the latter must obtain the access permission by NOSs by presenting the sticky policy associated to the desired data.

What emerges is that all interactions are mediated by the NOSs, which have to be considered trusted by the sources/users. Therefore, no agreement on encryption schemes or policies formats has to be performed by the involved parties (i.e., data sources, users), but only among such parties and NOSs. Access to resources is regulated by the sticky policies attached to data, both in case of private and public networks.

Note that the original NOS data model, conceived in [34], had to be enhanced in order to accommodate the new information included by the sticky policies. As shown in Listing 2 and depicted in Figure 5, the new (normalized) data model includes, besides the pre-existing information, the associated sticky policy.

```
{ "normalized data": [{
   "data 1": {
    "timestamp": "the arrival time of data",
    "data": ''the data content, possibly
    encrypted '',
    ''datatype'': "the kind of data",
    "owner": "the owner of the data",
  "sticky policy": [{
    "scopes": [{
      "scope1": "allowed use for the data",
      "scope2": "allowed use for the data"
    }],
    "validity": ''expiration timestamp'',
    "constraints": [{
      "constraints1": "obligations and
    restrictions",
      "constraints2": "obligations and
    restrictions",
      "constraints3": "obligations and
    restrictions"
    }],
  }]
   },
   "data 2": {
     ...
   },
]}
}
```

Listing 2: JSON data schema

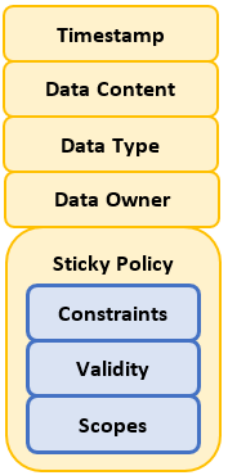

Fig. 5: Data model

NOSs should be equipped with a module able to parse the preferences expressed by the sticky policies themselves, in order to proper query the trust authority and obtain the access decision. To this end, a new module, named *Policy Parser*, is introduced. Figure 6 sketches the resulting interactions among components.

Obviously, the presence of a single trust authority for a potentially large IoT system may present scalability problems Therefore, multiple trust authorities may be set up. They may all be able to handle any type of policy (in this case they should be synchronized) or, alternatively, each trust authority may be assigned to a particular application domain or data set. In such a situation, NOSs should be aware of the relationships among the data provided by the sources and the trust authority/authorities that own the related policy set, in order to forward the queries to the right one. Hence, a sort of distributed trust authorities may be created over the NOSs middleware. The infrastructure thus defined aims at achieving a clever division of tasks so that the overall performances of the IoT system is enhanced, both in terms of resilience and efficiency.

Summarizing, the architectural components of the IoT system along with their interactions are depicted in the sequence diagram in Figure 7.

V. Experimental validation

We have developed a prototypical implementation of the proposed approach, in order to empirically measured

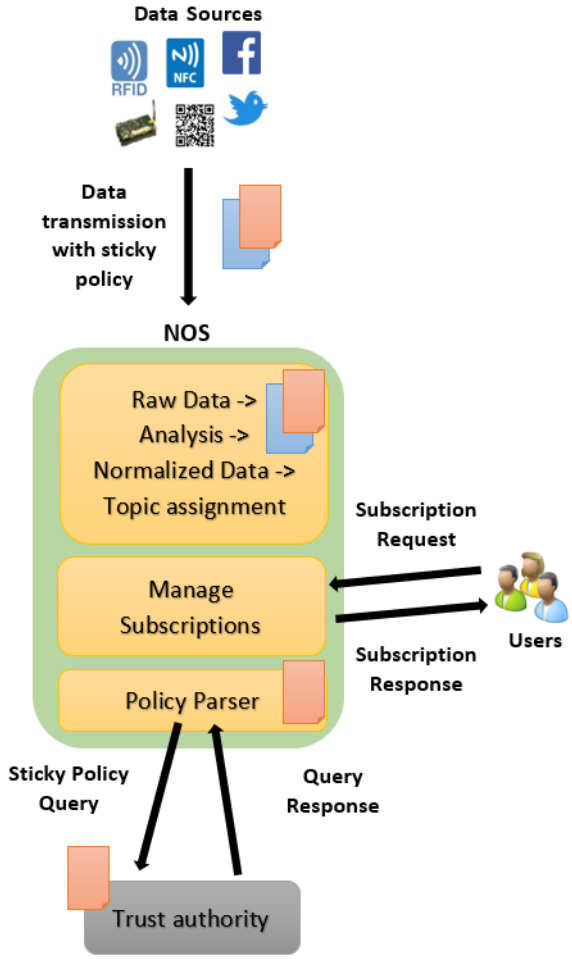

Fig. 6: Overview of components and interactions

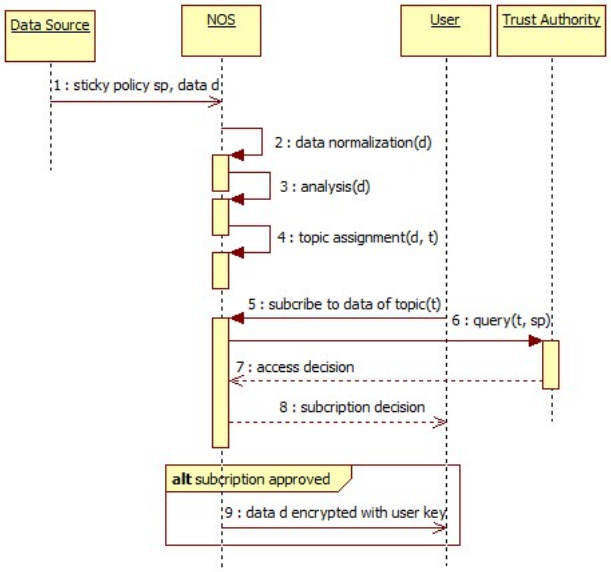

Fig. 7: Achitectural components and interactions

its performance and provide insight into its technical feasibility. The starting point is the prototype presented in [34], which however does not include support for sticky policies.

A. *Experimental settings*

NOSs are deployed on a Raspberry Pi. For validation purposes, data from real-world smart home testbed are used[1]. In particular, we used data from smart meter number 2 of *Home A*, which include, among the others, electricity consumption data of: kitchen lights, bedroom lights, duct heater HRV, and HRV furnace. Note that the home has a total of eight rooms and includes three full-time occupants. Measures are acquired by means of installed sensors that collect electricity data every minute for the entire home. To obtain more details about the deployment and data, please refer to [37].

The behaviour of a set of nodes that send such data to NOS, is emulated by means of a laptop, which uses WiFi network to communicate with the Raspberry Pi. The same WiFi connection is also used for the communications with the MQTT broker and with the trust authority module, implemented as a separate component, which interacts with NOSs depending on the sticky policy requests. The enforcement framework previously defined in [34] has been replaced with the new one, as described in Section IV.

In order to demonstrate the effectiveness of the proposed solution, a comparison with the previous approach in terms of storage, overhead and delay has been carried out.

Due to the nature of the information retrieved from the dataset, we can suppose an application case study where three kinds of users may be interested in the information provided by NOS: (i) the owner of the smart home, who wants to know the actual conditions of his home and to remotely control appliances (e.g., turn up the heating, turn off the light in case of is it turned on by mistake); (ii) the son of the owner, who also wants to receive some information from home (e.g., the presence of his parents or whether the lights are on), but, with respect to his parents, he has no right to give access control permissions to other people; (iii) an intruder who wants to get information about the habits of the home-dwellers for malicious purposes (e.g., get into the house when there is nobody). The owner of the house is registered to the NOS in charge of acquiring the data provided by the sensors placed into the home. Information are disclosed to the owner with “analysis” “control”, and “administrator” scopes

[1] http://traces.cs.umass.edu/index.php/Smart/Smart

and their sharing is restricted to users or devices which owns the right credentials to access the system and one of the following attributes/functions: owner, home identifier. Instead, his son is registered to the NOS with "analysis" and "control" scopes and attributes/functions: dweller, home identifier.

B. Storage overhead

NOS components have the following storage requirements:

- The data sources and the users have to store the credentials for ciphering the data to be transmitted to NOS. When sources/users transmit a data to NOS, they may also send the related sticky policy. This aspect causes, with respect to the "traditional" approach, an increase of traffic into the network, since not only the data is transmitted, but also the associated policy. We measured an average increase of 0.5 kilobytes for each transmitted data unit, considering the sticky policy format specified in Listing 1;
- NOSs have to store different kinds of information. Yet, it is worth remarking that NOSs do not support persistent storage of IoT data for *Raw Data* and *Normalized Data* collections. Indeed, incoming data are only temporarily cached on the NOSs' virtual memory while being processed before being submitted to the MQTT broker. Once data are further pushed to or pulled from the MQTT client (which handles the topics notification to subscribers), the data can be safely removed from NOSs. Instead, *Config* and *Sources* databases must be persistent because they contain information necessary for the correct operations of NOSs. In the previous approach [34], another collection was responsible for the policy storage, but, adopting the sticky policy based mechanism, NOSs have no longer to store all the policies managed by the IoT system. Therefore, the memory occupied on the hard disk decreases. However, the virtual memory occupancy of data for *Raw Data* and *Normalized Data* collections unavoidably increases because the data model has grown to store the sticky policies along with each data (see Listing 2 in Section IV-B). Since NOS runs on a Raspberry Pi, the maximum storage capacity with the actual technology corresponds to 1 gigabyte (i.e., the RAM provided by Raspberry Pi 2 and 3). With the "traditional" approach, we measured an average memory occupancy at runtime of 6.3 megabytes; whereas, with the sticky policy approach, it increased up to 10.2 megabytes on average. Note that these values are only indicative, since the memory occupancy depends on a number of factors, including: (i) the frequency of data fetching from sources; (ii) the frequency of execution of the routines for removing data from non-persistent collections (at the moment this task is executed every 5 minutes); (iii) the number of sources.
- The trust authority has to store the entire set of the valid scopes and constraints used for sticky policies' composition. The dimension of this storage obviously depends on the specific application domain. In the sample implementation this was negligible.

C. Delay

An important metric to consider is the additional delay introduced by the enforcement framework using sticky policies with respect to that proposed in [34]. The main difference between the two approaches resides in how policies are evaluated. In this work, policies are stored into NOSs and are transmitted along with data so that recipients can independently determine if they can access such an information or not; to obtain the access permission, the recipients can subscribe to certain topics and the subscription is accepted only if the request satisfies the requirements established by the sticky policies associated to the data. If a sticky policy is changed for a certain topic, the subscription may become not valid, thus requiring a new subscription. Access permissions are not locally evaluated by NOSs, but they are delegated to the trust authority; a query to the trust authority is required, which clearly requires time to be transmitted and processed. Instead, in the approach presented in [34], all the requests are executed within NOSs, thus increasing the computational load on NOSs themselves but presenting a lower delay. Moreover, in the case of update/addition/revocation of policies, NOSs had to be synchronized; conversely, with the sticky policy approach, the only task to be performed is the update of the dictionary of the trust authority. Note that, in case the dictionary becomes too large (maybe in presence of multiple application domains), as just said in Section IV-B, multiple trust authorities may be deployed.

Figure 8 shows a comparison of the distribution of the delays generated by the two approaches, measured with our prototypical implementation over a period of one hour. Data rate strictly depends on the fetching of data acquisition of the used data set, which is every minute. The considered time window concerns a week of measurements. The graph demonstrates that the new approach outperforms the traditional one, in particular in terms of minimum delay value. Such an evaluation has been performed by simulating several subscription/access requests from users with function of owner and dweller.

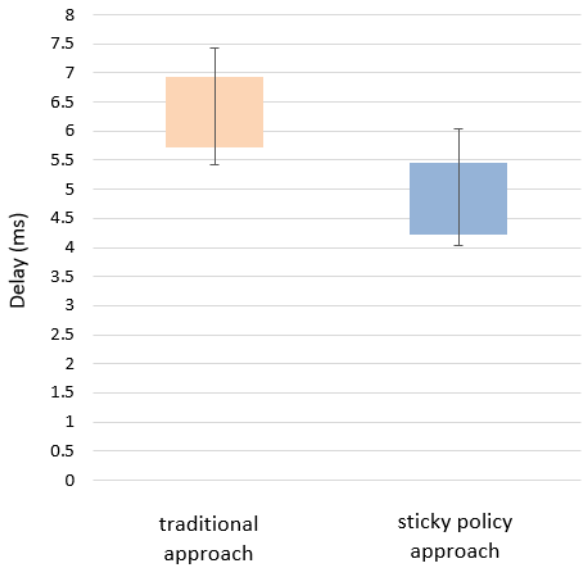

Fig. 8: Whiskers-box diagram of delay comparison: previous VS actual policy enforcement framework

D. System behaviour

In this section we examine the behaviour of the resulting system, in order to show that it is able to manage the desired policies and to counteract potential violation attempts. For more details about the functionality of the traditional approach we refer to [34].

The first step is the data transmission by one of the sensor, identified as "sensor BL-homeA", which is represented in Listing 3. The data is related to the bedroom lights; the scope of sticky policy is restricted to "analysis" and "control", and access permission is only allowed to users or devices which have "owner" or "dweller" as attributes, besides the correct home identifier. Note that the information is sent in an encrypted way.

```
{ "data transmission": [{
   "data content": {
    "timestamp": "31/12/2016 10:10:00",
    "data": "0.00465",
    "datatype": "kW",
    "owner": "sensor BL-homeA",
    "sticky policy": [{
      "scopes": [{
      "scope1": "analysis",
      "scope2": "control",
    }],
    "validity": "3 hours",
    "constraints": [{
      "constraints1": "access allowed for
     analysis, control scopes - owner",
      "constraints2": "access allowed analysis,
      control scopes - dweller"
      }],
   }
]}
```

```
}
```

Listing 3: Data transmission

We assume *Bob* to be registered as an "owner" to the monitoring service offered by the NOS installed into the smart home. Bob requests the subscription to the topic "bedroom lights", which allows access to the data related to the bedroom lights. The request of subscription is presented in Listing 4 and, as we can see, *Bob* declares his attributes to NOSs, in order to obtain the access.

```
{ "subscription request": [{
    "username": "Bob",
    "topic": "bedroom lights",
    "attributes": [{
    "function": "owner-homeA",
    "scope": "analysis, control, administrator"
  ]}
}
```

Listing 4: Subscription request

In this case, the policy is satisfied, therefore the subscription is accepted by the trust authority and, then, NOS sends a positive response to *Bob* with the access granted. *Bob* could also specify additional preferences on the received data, for example in terms of data quality (i.e., level of accuracy, precision, timeliness and completeness), as shown in Listing 5. Note that, by means of NOSs, users may declare that they will only get the data with a given level of security and quality. In the sample scenario, security levels strictly depend on the encryption mechanisms adopted within the smart home.

```
{ "data provision": [{
   "data content": {
    "timestamp": "31/12/2016 10:10:00",
    "data": "0.00465",
    "datatype": "kW",
    "owner": "sensor BL-homeA",
    "qualityLevels": "0.8, 0.8, 1, 1"
    "sticky policy": [{
      "scopes": [{
      "scope1": "analysis",
      "scope2": "control",
    }],
    "validity": "3 hours",
    "constraints": [{
      "constraints1": "access allowed for
     analysis, control scopes - owner",
      "constraints2": "access allowed analysis,
      control scopes - dweller"
      }],
   }
]}
}
```

Listing 5: Data provision

Once *Bob* receives the data of interest, it can send messages to the broker under a given topic, e.g., "bedroom lights-command", for remotely controlling some home appliances. For example, if all the occupants are out of home and he recognizes that bedroom lights are

on, he can turn off them from his office. In this case, the scope "control" is required by the sticky policy, therefore such a request, to be accepted by NOS, must be sent by a user with attribute "owner" or "dweller", and not, for example, "guest". Similarly, *Bob*'s son is prevented from creating and giving access permissions to other people, because he does not own the scope "administrator". The format of the request is similar to that of subscription in Listing 4 and it is always mediated by the trust authority, but must come from *Bob*.

Finally, we consider the presence of an intruder who eavesdrops the information that pass from the smart home to the owner *Bob* and his son. The system is robust because the intruder, in order to access the information, should know the credentials owned by the users/devices for ciphering the data along with the attached sticky policy. Such credentials are established a priori among users and the smart home system, therefore they cannot be derived from communications. Moreover, besides decrypting the information, the intruder should also be able to correctly satisfy the sticky policy. For such reasons, we can conclude that sticky policies add a further level of security to the transmitted data and, thus, increase the robustness of the whole IoT system.

VI. Conclusion

In this paper, we have presented an enforcement framework based on sticky policies, which has been integrated in the NOS distributed and cross-domain IoT middleware. With respect to the traditional approaches, the use of sticky policies provide users and data sources with an additional level of control over the disclosure of their information. As demonstrated in the paper, the use of sticky policies provides additional advantages in terms of computational and storage requirements, obtained by moving the management of policies to the trust authority. The feasibility and the performance of the proposed approach have been validated by means of a prototypical implementation and a set of tests performed using real-world data.

In the next future, we plan to focus on the deployment of this middleware and enforcement framework in a large-scale environment, in order to test its robustness and scalability in more realistic conditions and interesting scenarios. Furthermore, some open issues emerged, such as: the definition of a proper language for sticky policy representation; the design and development of a general-purpose interpreter for sticky policies within the IoT platform; the role of the trust authority into the access control model; the formalization of a threat model and the evaluation of the resilience of the system to various kinds of attack.

References

[1] S. Pearson and M. C. Mont, "Sticky policies: An approach for managing privacy across multiple parties," *Computer*, vol. 44, no. 9, pp. 60–68, 2011.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[3] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[4] S. Sicari, A. Rizzardi, D. Miorandi, and A. Coen-Porisini, "Sticky policy application for the future internet: A survey," *Technical Report*, 2017.

[5] "http://www.oecd.org/sti/ieconomy/oecdguidelinesontheprotectionofprivacyandtransborderflowsofpersonaldata.htm," *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*, 2016.

[6] "http://www.eugdpr.org/," *European GDPR*, 2017.

[7] M. C. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services," in *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*. IEEE, 2003, pp. 377–382.

[8] S. Trabelsi and J. Sendor, "Sticky policies for data control in the cloud," in *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, July 2012, pp. 75–80.

[9] C. Leng, H. Yu, J. Wang, and J. Huang, "Securing personal health records in the cloud by enforcing sticky policies," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 11, no. 4, pp. 2200–2208, 2013.

[10] S. Li, T. Zhang, J. Gao, and Y. Park, "A sticky policy framework for big data security," in *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*. IEEE, 2015, pp. 130–137.

[11] S. Pearson and A. Charlesworth, "Accountability as a way forward for privacy protection in the cloud," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 131–144.

[12] F. Di Cerbo, S. Trabelsi, T. Steingruber, G. Dodero, and M. Bezzi, "Sticky policies for mobile devices," in *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '13. New York, NY, USA: ACM, 2013, pp. 257–260.

[13] H. Koshutanski, M. Ion, and L. Telesca, "Distributed identity management model for digital ecosystems," in *The International Conference on Emerging Security Information, Systems, and Technologies (SECUREWARE 2007)*. IEEE, 2007, pp. 132–138.

[14] Q. Tang, "On using encryption techniques to enhance sticky policies enforcement," 2008.

[15] M. C. Mont, S. Pearson, and P. Bramhall, *Towards Accountable Management of Privacy and Identity Information*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 146–161.

[16] P. Ashley, C. Powers, and M. Schunter, "From privacy promises to privacy management: a new approach for enforcing privacy throughout an enterprise," in *Proceedings of the 2002 workshop on New security paradigms*. ACM, 2002, pp. 43–50.

[17] C. S. Powers, P. Ashley, and M. Schunter, "Privacy promises, access control, and privacy management. enforcing privacy throughout an enterprise by extending access control," in *Electronic Commerce, 2002. Proceedings. Third International Symposium on*, 2002, pp. 13–21.

[18] G. Karjoth, M. Schunter, and M. Waidner, "Privacy-enabled services for enterprises," in *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on*. IEEE, 2002, pp. 483–487.

[19] L. Bussard, G. Neven, and F.-S. Preiss, "Downstream usage control," in *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 22–29.

[20] D. Butin, M. Chicote, and D. Le Métayer, "Log design for accountability," in *Security and Privacy Workshops (SPW), 2013 IEEE*. IEEE, 2013, pp. 1–7.

[21] M. Backes, G. Karjoth, W. Bagga, and M. Schunter, "Efficient comparison of enterprise privacy policies," in *Proceedings of the 2004 ACM symposium on Applied computing*. ACM, 2004, pp. 375–382.

[22] A. I. Antón, E. Bertino, N. Li, and T. Yu, "A roadmap for comprehensive online privacy policy management," *Communications of the ACM*, vol. 50, no. 7, pp. 109–116, 2007.

[23] S. Yamada and E. Kamioka, "Access control for security and privacy in ubiquitous computing environments," *IEICE transactions on communications*, vol. 88, no. 3, pp. 846–856, 2005.

[24] X. Jiang and J. A. Landay, "Modeling privacy control in context-aware systems," *IEEE Pervasive computing*, vol. 1, no. 3, pp. 59–63, 2002.

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

[26] Z. Wu and L. Wang, "An innovative simulation environment for cross-domain policy enforcement," *Simulation Modelling Practice and Theory*, vol. 19, no. 7, pp. 1558–1583, August 2011.

[27] M. Dell'Amico, G. Serme, M. S. Idrees, A. S. De Oliveira, and Y. Roudier, "HiPoLDS: A Hierarchical Security Policy Language for Distributed Systems," *Information Security Technical Report*, vol. 17, no. 3, pp. 81–92, February 2013.

[28] D. Ferraiolo, V. Atluria, and S. Gavrila, "The Policy Machine: A novel architecture and framework for access control policy specification and enforcement," *Journal of Systems Architecture*, vol. 57, no. 4, pp. 412–424, April 2011.

[29] J. Rao, A. Sardinha, and N. Sadeh, "A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 1, pp. 40 – 56, 2009.

[30] ——, "A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 1, pp. 40–56, January 2009.

[31] 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*, Oct 2015.

[32] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, and A. Coen-Porisini, "A secure and quality-aware prototypical architecture for the internet of things," *Information Systems*, vol. 58, pp. 43–55, 2016.

[33] A. Rizzardi, S. Sicari, D. Miorandi, and A. Coen-Porisini, "AUPS: An Open Source AUthenticated Publish/Subscribe system for the Internet of Things," *Information Systems*, vol. 62, pp. 29 – 41, 2016.

[34] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, and A. Coen-Porisini, "Security policy enforcement for networked smart objects," *Computer Networks*, vol. 108, pp. 133 – 147, 2016.

[35] "IBM and Eurotech, MQTT v3.1 protocol specification," http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

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

[37] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht, "Smart*: An open data set and tools for enabling research in sustainable homes," *SustKDD, August*, vol. 111, p. 112, 2012.