Chapter 57
Applying Security to a Big Stream Cloud Architecture for the Internet of Things

Laura Belli
University of Parma, Italy

Simone Cirani
University of Parma, Italy

Luca Davoli
University of Parma, Italy

Gianluigi Ferrari
University of Parma, Italy

Lorenzo Melegari
University of Parma, Italy

Marco Picone
University of Parma, Italy

ABSTRACT

The Internet of Things (IoT) is expected to interconnect billions (around 50 by 2020) of heterogeneous sensor/actuator-equipped devices denoted as "Smart Objects" (SOs), characterized by constrained resources in terms of memory, processing, and communication reliability. Several IoT applications have real-time and low-latency requirements and must rely on architectures specifically designed to manage gigantic streams of information (in terms of number of data sources and transmission data rate). We refer to "Big Stream" as the paradigm which best fits the selected IoT scenario, in contrast to the traditional "Big Data" concept, which does not consider real-time constraints. Moreover, there are many security concerns related to IoT devices and to the Cloud. In this paper, we analyze security aspects in a novel Cloud architecture for Big Stream applications, which efficiently handles Big Stream data through a Graph-based platform and delivers processed data to consumers, with low latency. The authors detail each module defined in the system architecture, describing all refinements required to make the platform able to secure large data streams. An experimentation is also conducted in order to evaluate the performance of the proposed architecture when integrating security mechanisms.

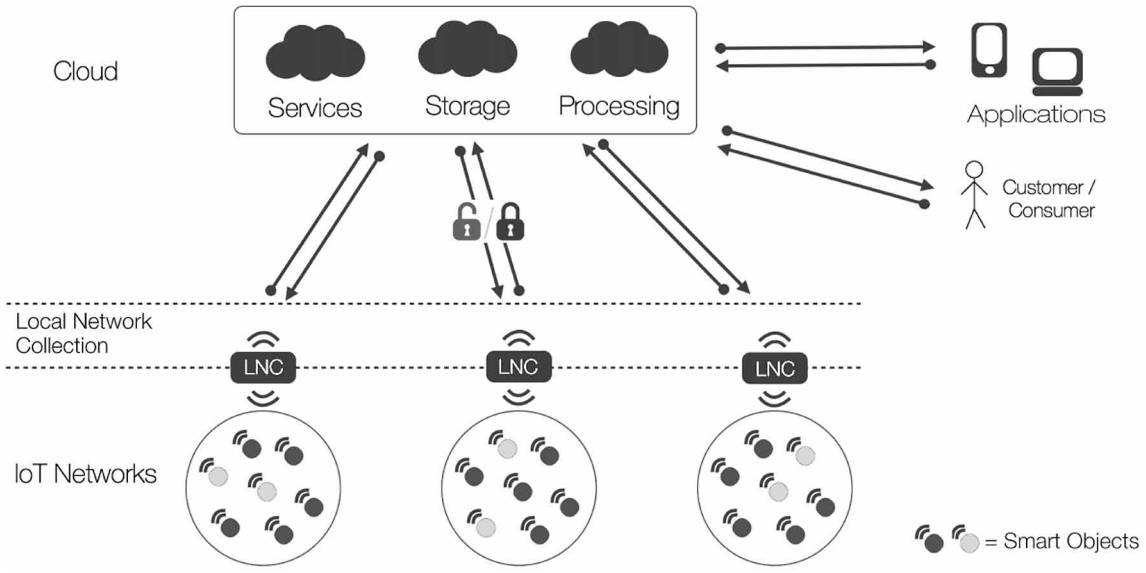
DOI: 10.4018/978-1-5225-9866-4.ch057

INTRODUCTION

In recent years, the forecast of a worldwide network of pervasively deployed heterogeneous networks is becoming a reality. The Internet of Things (IoT) involves billions of different devices, connected in an Internet-like structure, allowing new forms of interaction between things and people. The actors involved in IoT scenarios have extremely heterogeneous characteristics, in terms of processing and communication capabilities, energy supply and consumption, availability and mobility, spanning from Smart Objects (SOs) - i.e., constrained devices equipped with sensors or actuators, smartphones, wearables and other personal devices - to Internet hosts and the Cloud.

In order to allow heterogeneous nodes to efficiently communicate with each other and with existing Internet actors, shared and interoperable communication mechanisms and protocols are currently being defined and standardized. The most prominent driver for interoperability in the IoT is the adoption of the Internet Protocol (IP), namely IPv6. An IP-based IoT can extend and operate with all existing Internet nodes, without additional efforts. Standardization institutions, such as the Internet Engineering Task Force (IETF), and several research projects are contributing to the definition of new mechanisms to bring IP to SOs (e.g., 6LoWPAN (Kim, Kaspar, & Vasseur, 2012)). This is motivated by the need to adapt higher-layer protocols (e.g., application-layer protocols) to constrained environments. As a result, IoT networks are expected to generate huge amounts of data, which can be subsequently processed and used to build several useful services for final users. The Cloud has become the natural collection environment for sensed data retrieved by IoT nodes, due to its scalability, robustness, and cost-effectiveness. Figure 1 shows the hierarchy of different levels involved in data collection, processing and distribution in a typical IoT scenario.

Figure 1. Different actors and layers involved in IoT scenarios: sensed data are sent from IoT networks to the Cloud, which provides services to consumers. At an intermediate level, preliminary local operations, such as data collection, processing, and distribution, may be carried out



Sensed data are collected by SOs, deployed in IoT networks, and sent uplink to the Cloud which operates as collection entity and service provider. In some cases, intermediate Local Network Collectors (LNCs) can perform some preliminary tasks before sending data uplink, such as temporary data storage, data aggregation, and protocol translation. The described model is extremely general, and can be applied to different IoT scenarios. As an example, LNCs can be implemented by border routers or proxies.

Several relevant IoT application environments (e.g., industrial automation, transportation, and monitoring) require real-time performance guarantees or, at least, a predictable latency. Moreover, the performance requirements (e.g., in terms of data sources) may change even abruptly. On one hand, the large number of data sources represented by IoT nodes, generating a high rate of incoming data, and, on the other hand, the low-latency constraints call for innovative Cloud architectures able to handle efficiently such massive amounts of information.

A possible solution is given by Big Data approaches, developed in the last few years and become popular due to the evolution of online and social/crowd services, which are able to address the need to process extremely large amounts of heterogeneous data for various purposes. However, these techniques typically have an intrinsic inertia (as they are based on batch processing) and focus on the data itself, rather than providing real-time processing and dispatching (Zaslavsky, Perera, & Georgakopoulos, 2013; Leavitt, 2013). For this reason, Big Data approaches might not be the right solution to manage the dynamicity of IoT scenarios with real-time processing. In order to better fit these requirements, we propose to shift the Big Data paradigm to a new paradigm, denoted as "Big Stream."

While both Big Data and Big Stream paradigms deal with massive amounts of data, those two paradigms differ in the following aspects.

- **The Meaning of the Term "Big:"** In Big Data it refers to "volume of data," while in Big Stream it refers to "data generation rate."
- **Real-Time or Low-Latency Requirements of Different Consumers:** They are typically not taken in account by Big Data.
- **Nature of Data Sources:** Big Data deals with heterogeneous data sources in a wide range of different areas (e.g., health, economy, natural phenomena), not necessarily related to IoT. Instead, Big Stream data sources are strictly related to the IoT, where heterogeneous devices send small amounts of data generating, as a whole, a continuous and massive information stream.
- **Objective:** Big Data focuses on information management, storage and analysis, following the data-information-knowledge model (Aamodt & Nygard, 1995); Big Stream, instead, focuses on the management of data flows, being specifically designed to perform real-time and ad-hoc processing, in order to forward incoming data streams to consumers.

Big Stream-oriented systems should react efficiently to changes, providing smart resource allocation and, thus, implementing scalable and cost-effective Cloud services. This also affects the data selected as relevant, in different processing steps, for the final consumer. For instance, while for Big Data applications it is important to store all data in order to be able to perform any successive required computation, Big Stream applications might decide to perform data aggregation, filtering or pruning, in order to minimize the latency in conveying the final computation output to consumers, with no need for persistence. Eventually, as a generalization, a Big Data application might be a consumer of Big Stream data flows and perform storage operations.

For these reasons, in previous works (Belli, Cirani, Ferrari, Melegari, & Picone, 2014; Belli et al., 2015) we have designed and implemented a Graph Cloud architecture suited to applications with real-time and low-latency requirements (i.e., Big Stream applications) for IoT scenarios. The proposed architecture relies on the concepts of data listener and data-oriented graph processing, in order to implement a scalable, highly configurable, and dynamic chain of computation on incoming Big Streams, dispatching data with a push-based approach, thus providing the shortest delay between the generation of information and its consumption.

To fulfill the IoT vision, some issues must be addressed, such as security. Securing the IoT requires further research and entails both authentication and authorization mechanisms to address security and privacy problem, which is critical in several IoT applications.

As in our previous works (Belli et al., 2015), we assume that streams generated by nodes in the proposed Graph Architecture are "open" and potentially accessible by any interested subscriber. However, this assumption can not meet the security requirements of all consumers or developers accessing the architecture. Moreover, security management may become necessary in application scenarios that require to control or filter subscribers' access to one or more streams. For this reason, the architecture preliminary proposed in (Belli, Cirani, Ferrari, Melegari, & Picone, 2014; Belli et al., 2015) is now extended in order to take in account also security aspects. In particular, the extension introduces additional modules needed to make the proposed Graph architecture able to handle both secured and "open" data streams, focusing, in particular, on solutions able to decouple security roles and management purposes (e.g., OAuth protocol, an n-legged authorization protocol).

The rest of this work is organized as follows. In the following section an overview of related works is presented. Next, the new concepts and modules needed to "secure" data streams are detailed, analyzing how these entities increase platform security. We show some evaluations of the architecture, highlighting the differences between the unsecured platform and the secured one. Finally, we draw our conclusions and discuss future research directions.

RELATED WORKS

Internet of Things

The huge number of heterogeneous SOs deployed in an IoT system allows the development of ubiquitous sensing in most areas of modern living. The outcome of this trend is the generation of a huge amount of data that should be treated, aggregated, processed, transformed, stored, and delivered to the final users of the system, in an effective and efficient way, by means of traditional commodity services.

Several architectures for IoT scenarios have been proposed in the literature. The European Union (EU), under its 7th Framework Program (FP7, 2007-2013), has supported a significant number of IoT-related projects converging in the "IoT European Research Cluster" (IERC), and addressing relevant challenges, particularly from a Wireless Sensor Networks (WSNs) perspective. Among them, we recall "Internet of Things-Architecture" (IoT-A) and SENSEI. In the IoT-A project, developers focused their work on the design of an IoT architecture, aiming at connecting vertically closed applications, systems and architectures, in order to create integrated and open-interoperable environments and platforms. The main goal of the SENSEI project was to create a business-driven platform that addresses the scalabil-

ity problems for an amount of globally distributed devices in Wireless Sensor and Actuator (WS&A) networks, enabling an accurate and reliable interaction with the physical environment, and providing network and information management services.

"Connect All IP-based Smart Objects!" (CAL*IP*SO) was another EU FP7 project with relevant implications on the design of IoT platforms (Medagliani & al., 2014). The main goal of CAL*IP*SO was to enable IPv6-based connectivity of SOs to IoT networks with very low power consumption, thus providing long lifetime and high interoperability, also integrating radio duty cycling and data-centric mechanisms with IPv6.

Cloud Computing for IoT

Many researchers focus their efforts on the definition of Cloud Computing-based IoT architectures, where: the Cloud is the infrastructure for data and service management; and the final consumer/user drives the use of data and infrastructure to develop new IoT applications.

An interesting framework for IoT-based Cloud systems is OpenStack, an ubiquitous open-source Cloud Computing platform for public and private Clouds, aiming at delivering solutions for all types of Clouds by being massively scalable, feature-rich, and simple to implement. While OpenStack can be seen as a framework following a vendor-driven model, OpenNebula (Moreno-Vozmediano, Montero, & Llorente, 2012) represents an open-source Cloud platform focused on the user, aiming at delivering a flexible and simple feature-rich solution to build and manage virtualized data centers and enterprise Clouds.

An example of an open approach is given by the FI-WARE project, an open Cloud-based infrastructure for cost-effective creation and delivery of Internet applications and services. Its API specifications are public, royalty-free, and Open Cloud Computing Interface (OCCI)-compliant.

In the realm of Cloud-related IoT user-driven architectures, an innovative concept, known as Fog Computing, has been proposed as a novel and appropriate paradigm for a variety of IoT services and applications that require low-latency and location awareness (Bonomi, Milito, Natarajan & Zhu, 2014).

Fog Computing could be described as a highly virtualized platform that provides networking, processing, and storage services, acting on the edge of the constrained network, between endpoint devices and traditional Cloud Computing platforms. This can be integrated as an extension of the Cloud, providing support to the endpoints and services that could comply with low-latency and real-time consumer requirements.

Cloud Computing, as well as others paradigms, has to deal with different kinds of sources and data amounts; one of these data patterns is Big Data. Big Data architectures generally use traditional processing patterns with a pipeline approach (Hohpe & Woolf, 2003), where the downstream data flow goes, following a path, where data are sequentially handled, with tightly coupled predefined processing units. In this way, the Big Data paradigm can be defined as "process-oriented:" a central coordination point manages the execution of processing units in a given order, and each unit provides a specific output, which is created in order to be used only within the scope of its own process, without the possibility to be shared among different processes. This approach is representative of a deviation from traditional Service Oriented Architectures (SOAs), where primitive units are handled by external web-services, invoked by a coordinator process, rather than internal services (Isaacson, 2009), in order to accept, manage, and process requested actions onto the platform where web-services act.

Protocols and Communication Models for IoT

It is a common assumption that, in IoT, the most prominent driver to provide interoperability is IPv6. Referring to the IP stack, at the application layer developers find a variety of possible protocols applicable to different IoT scenarios, according to specific application requirements. Among many options, the following are relevant.

- HyperText Transfer Protocol (HTTP), is mainly used for the communication with the consumer's devices.
- Constrained Application Protocol (CoAP), defined in (Shelby, Hartke, Bormann, & Frank, 2014), is built on the top of User Datagram Protocol (UDP), follows a request/response paradigm, and is explicitly designed to work with a large number of constrained devices operating in Low power and Lossy Networks (LLNs).
- Extensible Messaging and Presence Protocol (XMPP), is based on XML, supports decentralization, security (e.g., TLS), and flexibility.
- MQ Telemetry Transport (MQTT) is a lightweight publish/subscribe protocol running on top of TCP/IP. It is an attractive choice when a small code footprint is required and when remote sensors and control devices have to communicate through low bandwidth and unreliable/intermittent channels. It is characterized by an optimized information distribution to one or more receivers, following a multicast approach. Being based on a publish/subscribe communication paradigm, it acts through a "message broker" element, responsible for dispatching messages to all topic-linked subscribers.
- Constrained Session Initiation Protocol (CoSIP), described in (Cirani, Picone, & Veltri, 2013; Cirani, Davoli Picone & Veltri, 2014), is a lightweight version of the Session Initiation Protocol (SIP) aiming at allowing constrained devices to instantiate communication sessions in a standard fashion, optionally including a negotiation phase of some parameters, which will be used for all subsequent communications.

Stream and Real-Time Management

The term Big Stream was first introduced in (Belli, Cirani, Ferrari, Melegari, & Picone, 2014). Since the concept is related to platforms abiding by low-latency and real-time requirements, it can be compared with other solutions relying on such constraints. Possible examples of such solutions are Apache Storm (Mera Pérez, Batko, Zezula, et al., 2014) and Apache S4 (Neumeyer, Robbins, Nair, & Kesari, 2010).

Apache Storm is a free and open-source distributed real-time computation system, oriented to reliably process unbounded streams of data. It can be integrated with different queueing and database technologies, providing mechanisms to define topologies in which nodes consume and process data streams in arbitrarily and complex ways. Apache S4 is a general purpose, near real-time, distributed, decentralized, scalable, event-driven, and modular platform that allows programmers to implement applications for data streams processing. Multiple application nodes can be deployed and interconnected on S4 clusters, creating more sophisticated systems.

The works described in (Marganiec et al., 2014; Tilly & Reiff-Marganiec, 2011) address the problem to process, procure and provide information related to the IoT scenario with almost zero latency. The authors consider, as a use case, a taxi fleet management system, which has to identify the most relevant

taxi in terms of availability and proximity to the customer's location. The core of the publish/subscribe architecture proposed in (Marganiec et al., 2014) is the Mediator, which encapsulates the processing of the incoming requests from the consumer side and the incoming events from the services side. Services are publishers (taxis in the proposed example) which are responsible to inform the Mediator if there is some change in the provided service (e.g., the taxi location or the number of current passengers). Thus, instead of pulling data at consumer's request time, the Mediator knows at any time the status of all services, being able to join user requests with the event stream coming from the taxis, using temporal join statements expressed through SQL-like expressions.

Security Issues in Cloud and IoT Publish/Subscribe Scenarios

In the literature, several methods and strategies to enable confidentiality in publish/subscribe IoT infrastructures are proposed. IoT systems have to avoid security threats, providing strong security foundations built on a holistic view of security for all IoT elements at all stages: from object identification to service provision; from data acquisition to stream processing. All security mechanisms must ensure: resilience to attacks; data authentication; access control; and client privacy.

In (Collina, Corazza, & Vanelli-Coralli, 2012), IoT systems are expected to bridge the physical and the "virtual" worlds, using a novel broker that supports protocols such as HTTP and MQTT, adhering to the REST paradigm and allowing developers to easily and responsively expose fundamental entities as REST resources. This broker does not address any security issues, claiming that possibles solutions could include: plain authentication; Virtual Private Networks (VPNs); Access Control Lists (ACLs); as well as OAuth, a new type of authorization which is used to grant access to personal data by third-party applications (Hardt, 2012).

In (Lagutin, Visala, Zahemszky, Burbridge, & Marias, 2010), the authors examine roles of different actors comprising an inter-domain publish/subscribe network, along with security requirements and minimal required trust associations between entities, introducing and analyzing an architecture that secures both data and control planes. The main security goals for a publish/subscribe architecture are: (i) integrity; (ii) scalability; (iii) availability, and (iv) prevention of underived traffic. Finally, in (Lagutin, Visala, Zahemszky, Burbridge, & Marias, 2010) different actors and security mechanisms are identified. The main mechanism is Packet Level Authentication (PLA) which, combined with cryptographic signatures and data identifiers tied to secured identifiers, creates a strong binding between data and traffic, thus preventing Denial of Service (DoS) attacks.

The work of (Wang, Carzaniga, Evans, & Wolf, 2002) treats security issues relying on the requirements of a particular application and on an external publish/subscribe infrastructure. General security needs of the applications include confidentiality, integrity, and availability. At the opposite the security concerns of the infrastructure focus on system integrity and availability. Security issues in publish/subscribe platforms rely on authentication, information integrity, subscription integrity, service integrity, user anonymity and information confidentiality, in addition to subscription confidentiality, publication confidentiality, and accountability.

In (Raiciu, & Rosenblum, 2006), the authors present a study of confidentiality in Content-Based Publish/Subscribe (CBPS) systems, defined as an interaction model storing the interests of subscribers in a content-based infrastructure, to guide routing of notifications to interested subjects. In agreement with the approach in (Wang, Carzaniga, Evans, & Wolf, 2002), confidentiality aspects are decoupled into two facets, namely notification and subscription, suggesting that: a confidential CBPS (C-CBPS) must

satisfy correctness and notification; whereas a subscription CBPS must satisfy unforgeability and security, and match isolation. A high level approach to obtain C-CBPS relying on notifications using simple blocks, that may be controlled and checked better than a whole encryption of those ones, is proposed.

In (Fremantle, Aziz, Scott, & Kopecky, 2014) the use of Federated Identity and Access Management (FIAM) in IoT is analyzed, following a consumer-oriented approach, where consumers own data collected by their devices, having a control over entities who access these data. Traditional security models, based on the concept of roles with a hierarchical structure, are not applicable for IoT scenarios (because of the billions of devices involved, the impossibility to adopt a centralized model of authentication, and the necessity to support mechanisms for delegation of authority). The authors propose OAuth2 (Hardt, 2012) as a possible solution to achieve Access Management with IoT devices which support the MQTT protocol. The overall system consists of: (i) a MQTT broker, (ii) an Authorization Server supporting OAuth2, (iii) a Web Authorization tool, and iv) a device.

The work of (Bacon et al., 2010) tackles the problem of application security in the Cloud, aiming at incorporating end-to-end security, so that Cloud providers not only can isolate their clients from each other, but can also isolate the data generated by multiple users which access a particular service provided by the Cloud. An approach called "application-level virtualization," which consists of (i) removing from applications all the details regarding security and flow control, (ii) placing the security management logic in the Cloud infrastructure, and (iii) allowing providers to permit only the interactions that the clients specify, is proposed.

BIG STREAM CLOUD ARCHITECTURE

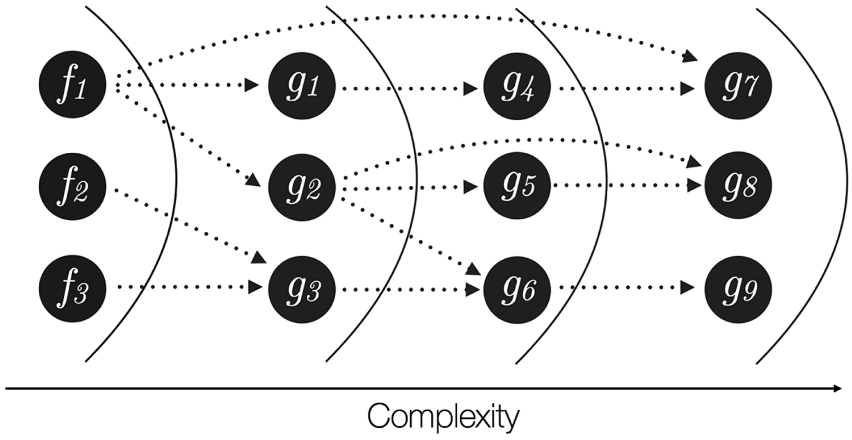
In (Belli, Cirani, Ferrari, Melegari, & Picone, 2014), a Graph-based architecture, explicitly designed to fit the Big Stream IoT scenarios, is proposed. In order to minimize the delay between the instant of raw data generation (e.g., at sensors) and the instant at which information is notified to the final consumer, the proposed platform adheres to the publish/subscribe model and is based on the concept of "listeners." More in detail, the fundamental components of the Graph-based Cloud architecture are the following.

- **Nodes:** processing units which process incoming data, not directly linked to a specific task. A Graph node can be, at the same time, a listener of one or more streams, coming from other nodes, and a publisher of a new stream, that could be of interest to some other nodes.
- **Edges:** flows of informations (streams) linking together various nodes: this allows complex processing operations.

In order to provide a set of common basic functionalities, nodes composing the Graph are divided into two groups.

- **Core Graph Nodes:** implement basic processing operations, provided by the architecture, and are deployed into inner layers of the Graph.
- **Application Graph Nodes:** represent the building blocks of outer layers, listening and accepting flows coming from nodes belonging to inner Graph layers. This type of nodes always receive already processed streams.

Figure 2. Basic processing nodes build the Core Graph Layer, the outer nodes have increasing complexity



Nodes in the Graph are organized in concentric layers, characterized by an increasing degree of complexity. This means that, as shown in Figure 2, data streams generated from nodes in a generic layer of the Graph, can be used by nodes in higher-degree layers, to perform more complex processing operations, generating new data streams which can be used in higher layers, and so on.

As a general rule, to avoid loops, nodes at inner Graph layers cannot be listeners of nodes of outer Graph layers. In other words, there can be no link from an outer Graph node to an inner Graph node, but only vice versa. Same-layer Graph nodes can be linked together if there is a need to do so. In particular, a Core Graph node can be a listener only for other nodes of the same layer and a source both for other Core or Application Graph nodes. In other words, we consider only acyclic graphs.

Moreover, the Cloud Graph architecture described in this paper has an optimal resource allocation: nodes with no listeners can be switched off and removed from the Graph. At the same time, taking advantage from the Cloud platform, nodes which have several listeners can be replicated, in order to maintain acceptable system performances.

In (Belli et al., 2015), a first implementation of the Big Stream architecture with open-source technologies has been presented. Three main modules concur in forming the entire system: (i) the Acquisition and Normalization modules; (ii) the Graph Framework module; (iii) the Application Register module.

As the architecture is based on a queue-communication paradigm, the implementation adopts an instance of RabbitMQ queue server as publish/subscribe engine.

The first Acquisition step handles raw data coming from external SOs through different application-layer protocols. The implementation proposed in (Belli et al., 2015) supports: (i) HTTP, through Nginx and PHP; (ii) CoAP, by means of the mjCoAP (Cirani, Picone, & Veltri, 2015) library; and (iii) MQTT, with the ActiveMQ library. The Normalization block comprises Java processes which work on incoming data from the Acquisition dedicated queues, and performing a first optimization, aiming at structuring data into a common and easily manageable JSON format.

The Graph Framework module represents the processing block, in which nodes (both Core and Application) are implemented with Java processes connected through queues. To ensure proper data propagation and avoid loops, layers in the Graph Framework have a dedicated RabbitMQ Exchange and are connected, through one-way links, with their own successor Exchange.

The data flow continues until it reaches the last layer's Exchange, responsible to manage notifications to external entities that are interested in final processed data (e.g., browsers, Data Warehouse systems, smart entities, external Cloud Graph processes).

The proposed architecture is managed by the Application Register module, a Java process that coordinates the interactions between Graph nodes and external services. The Application Register module has the fundamental responsibility to maintain all the information about the current status of the system. More precisely, the Application Register module performs the following operations:

- Attach new nodes or consumers, interested in some of the streams provided by the platform;
- Detach nodes from the Graph, when they are no longer interested in receiving flows, and, possibly, re-attach them;
- Handle nodes that are publishers of new streams;
- Maintain information regarding topics of data, in order to correctly generate the routing keys, and to compose data flows between nodes in different Graph layers.

The proposed architecture can be deployed on a single server. However, the exploitation on the Cloud is preferable, in order to better scale the system and manage the workload in presence of a huge number of data sources and processing nodes. Another important benefit is that the Cloud provides a common platform in which data streams can be shared among several actors (e.g., IoT data-sources, developers or consumers), in order to build useful services for Smart Cities, composing and aggregating different data streams.

Although the proposed architecture has not been deployed on the Cloud yet, in our vision this architecture is oriented to all developers interested in creating new useful services on top of IoT-generated data (e.g., Belli, Cirani, Davoli, Gorrieri, et al., 2015). For this reason, the platform will provide user-friendly tools to upload to the Cloud custom processing units (namely nodes) using currently available streams (already processed or not) as inputs, generating a new edge of the Graph which can be potentially employed from other developers. In this scenario, the paths are thus automatically generated on the basis of the needs and interests of each node.

Therefore, the proposed architecture can be applied to all scenarios in which scalability is required. We target IoT Big Stream applications because they represent an innovative field of research, which has not been fully explored.

ANALYSIS OF SECURITY ISSUES AND TECHNOLOGIES

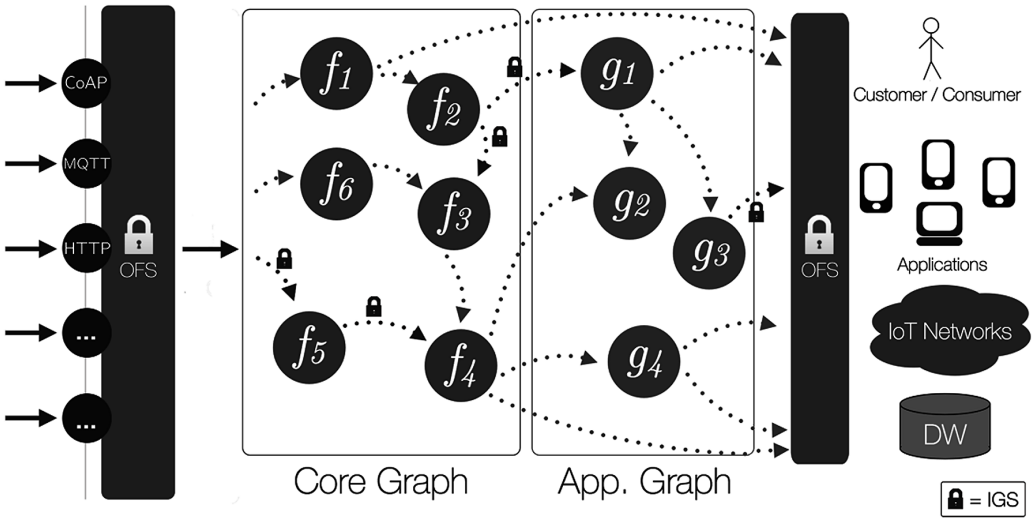
Main Security Modules

Addressing the security problem in the above Graph-based Cloud system entails a wide approach, owing to different needs of each specific component involved.

In Figure 3, the main building blocks listed above are shown. The main components, in correspondence to which security mechanisms are required, are explicitly indicated.

The enhanced Graph architecture provides security by means of the following two modules.

Figure 3. Main building blocks of the proposed listener-based Graph architecture. Nodes in the Graph are listeners, edges between nodes represent dynamic flows followed by information streams. Edges can be "open" or "secured"



- **Outdoor Front-end Security (OFS) Module:** Carries out security operations which could be applied a-priori, before receiving data from a generic external source, as well as before a final consumer can start interacting with the Graph-based Cloud platform.
- **In-Graph Security (IGS) Module:** Adopts security filters that could be applied inside the heart of the Graph-based Cloud platform, in order to make processing nodes able to control accesses to the streams generated by internal computational modules.

The OFS module is crucial for infrastructure safety: its role includes monitoring the access to the platform and authorizing the information flows coming from or directed to external entities. On one side, OFS must verify and authorize only desired external input data sources, allowing them to publish raw streams inside the IoT system. On the other hand OFS is required to secure outgoing streams, generated by different layers of the Graph platform itself, authorizing external consumers to use, if needed, the processed streams.

Consider, as example, the case of the company "C" that owns a set of particular sensors, and wishes to become an IoT stream source for the Graph-based Cloud platform. However, this company requires (i) to sell sensed data only to a specific subset of customers, in order to protect its commercial interests, and (ii) to reach a profit from these sales.

Therefore, the OFS module is strictly related to sensors and devices, at the input side, and to customers SOs, at the output stage, so that it becomes protocol-dependent and can be adapted to the specific technologies supported by the target devices.

The IGS module is not related to the OFS module, as it acts exclusively at the heart of the IoT Graph architecture coordinating and managing inner inter-node interactions. The IGS module must be implemented inside single processing nodes enabling them to define a set of rules which describe how entities may become listeners of a generated stream.

Referring to the Graph architecture, shown in Figure 3, edges in the Graph can be classified as follows:

- **"Open" Edges:** Data streams, generated by both Core or Application nodes in the Graph platform, that can be forwarded to all interested listeners without the need of isolation or access control.
- **"Secured" Edges:** Data streams that should comply with some specified rules or restrictions, which describe all possible consumers of generated data.

As an example, consider company "C," which provides its sensors as data sources and has notified the architecture that the streams produced by its sensors should be secured.

The integration of security modules in the IoT architecture entails modifications in the structure and into the modules of the first (not secured) architecture described in (Belli et al., 2015).

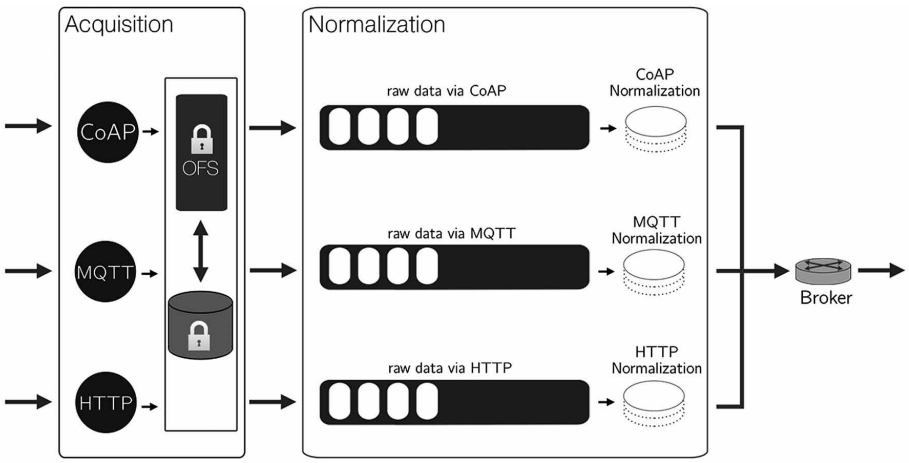
In the rest of this section, an analysis of each module composing the Graph architecture is presented, in order to explain how security mechanisms can be embedded and managed. In particular, we introduce the OFS module which supports the Acquisition and Normalization modules on authorization of external entities. Moreover, an enhanced version of the Application Register is described in order to underline the management of secure interaction with processing nodes. Finally, an overview inside Graph nodes, analyzing how security has been applied in processing stages, is presented.

Normalization after a Secure Stream Acquisition with OFS Module

The Acquisition and Normalization modules, shown in Figure 4, represent the entry point, for external sources (e.g., SOs deployed in different IoT networks), to the proposed architecture.

The purpose of the Acquisition block is to receive incoming raw data from heterogeneous sources, making them available to all subsequent functional blocks. Since raw data are generally application- and subject-dependent, the Normalization block has to "normalize" incoming data, generating a common adapted representation, suitable for further processing (e.g., suppression of unnecessary data, information addiction, format translation).

Figure 4. The OFS module manages security in the Acquisition and Normalization blocks, interacting with an authentication storage entity containing data sources identities



After the Normalization process, data are sent to the first Core layer. Within each Graph layer, streams are routed by dedicated components, denoted as brokers, which are layer-specific and, in the current implementation of the architecture, are represented by instances of RabbitMQ Exchanges.

As stated before, SOs can communicate using different protocols. For this reason, the Acquisition block has to include a set of different connectors, one for each supported protocol, in order to properly handle each protocol-specific incoming data stream.

As shown in Figure 4, these modules must cooperate with the OFS module, which has to be activated before an external source becomes able to operate with the Graph platform. At the Acquisition stage, in order for the proposed IoT platform to support both "open" and "secured" communications, protocol-specific security mechanisms have to be implemented at proper layers (e.g., at the network layer through IPSec, at the transport layer through TLS/DTLS, at the application layer through S/MIME or OAuth).

As stated before, the current implementation supports different application protocols at the Acquisition stage, namely: MQTT, HTTP, and CoAP. In order to secure all communications with these protocols, we need to introduce different protocol-specific policies. In the work of (Fremantle, Aziz, Scott, & Kopecky, 2014), an OAuth-based (Hardt, 2012) secure version of the MQTT protocol is proposed, showing that MQTT complies also with n-Legged OAuth protocol.

This means that the proposed IoT platform can provide a good way to authenticate external data providers, adopting open-source and well-known solutions. Therefore the OFS module can be secured by OAuth, being employed according to specific communication protocols supported by heterogeneous IoT SOs.

A suitable solution to provide authorization in IoT scenarios is IoT-OAS (Cirani, Picone, Gonizzi, Veltri, & Ferrari, 2015), which represents an authorization framework to secure HTTP/CoAP services. The IoT-OAS approach can be applied by invoking an external OAuth-based Authorization Service (OAS). This approach is meant to be flexible, highly configurable, and easy to integrate with existing services, guaranteeing: (i) lower processing load with respect to solutions with access control implemented in the SO; (ii) fine-grained (remote) customization of access policies; and (iii) scalability, without the need to operate directly on the device. Although the OFS module has not been implemented yet, provided references show that several options are available.

Following the previous example, company "C", to became a secured IoT data source, selects one of the supported protocols (HTTP, CoAP or MQTT) to send raw data stream in the secured version.

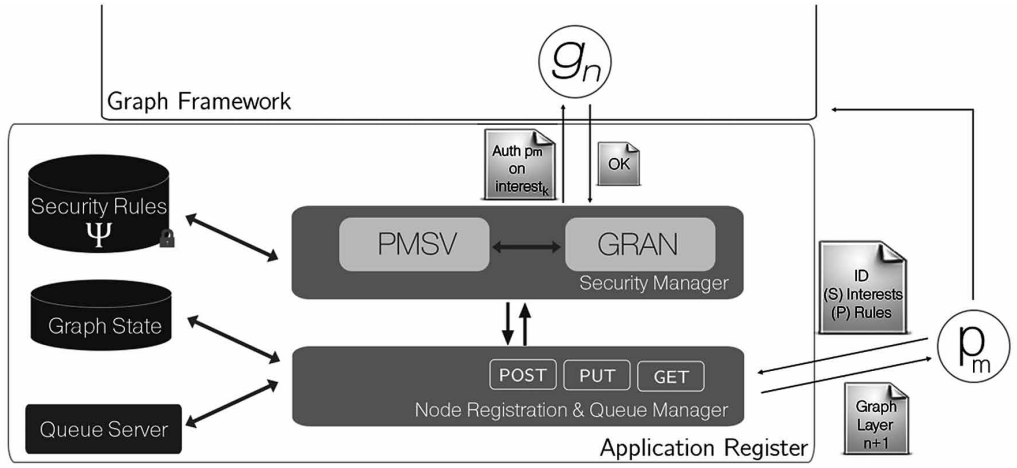
Enhancing Application Register with IGS Module

One of the main motivations to secure a system internally is the need to secure some of its operations, as well as to isolate some processing steps of the entire stream management. The security features should be coordinated by the Application Register module, which maintains and manages interactions between inner Graph nodes of the IoT platform using different communication protocols, as requested by the architecture itself.

In order to accomplish the operational functionalities listed previously, the Application Register module has two main components, as shown in Figure 5.

- The Graph State database, responsible to maintain all information about the current Graph status. Since this component is not critical from a performance viewpoint, it has been implemented through a simple relational SQL database.

Figure 5. The Application Register module structure with security elements. PMSV and GRAN modules interact with a storage entity to manage authorization in the Graph



- The Node Registration and Queue Manager (NRQM) module, which is responsible to manage communications with existing Graph nodes, as well as with external entities that ask to join the Big Stream architecture.

To add security features to these modules, the Application Register defines entities and modules specifically related to security management and coordination.

As shown in Figure 5, the Application Register is composed by the following additional modules:

- The **Policy Manager and Storage Validator (PMSV)**, responsible for managing and verifying authorization rules, interacts with a storage element, which persistently keeps updated authorization policies;
- The **Graph Rule Authorization Notifier (GRAN)** interacts with publisher Graph nodes and verifies if listener nodes are authorized to receive streams associated with specific topics;
- A **Persistent Storage Entity** (e.g., a non-relational database) maintains authorization rules specified by publisher Graph nodes.

While a processing node p_m asks to register to the IoT platform, requiring to join the Graph, after authentication (e.g., using a username/password pair, cryptographic certificates, ACLs, OAuth), there are two cases that require the use of security mechanisms and involve the defined modules:

- A registration request coming from a node that is willing to become a publisher node for a secured stream (e.g, an Application node created by developers of company "C");
- A registration request sent by a node which asks to be attached as a listener for some streams.

In the first case, when an external process $p_m$requests to register to the Graph architecture, in order to secure one or more of its own streams, it updates the PMSV module. After indicating its published topics, it specifies some policies and rules, to be stored, together with the assigned operative Graph

layer, into the persistent security storage, by PMSV itself. These rules will be checked in case of future subscription requests for the node.

In the second case, an external process p_m, upon issuing a request to attach to the Graph platform and to become a node, provides information related to its identity and also specifications about its interests, on which p_m asks to subscribe.

The Application Register, after having identified the Graph layer into which the new node could be placed, takes charge of these interests specifications, passing then to PMSV, that acts as follows.

- For each provided $interest_k$, the PMSV module interacts with the persistent storage entity, making a lookup for a matching between interest and stored publishing policies, and refining this lookup with layer matching:

$$Match = \left\{layer = x OR layer = \left(x+1\right)\right\} AND \left\{interest_k \in \Psi\right\}$$

where: x stands for identified listener Graph layer; $interest_k$ indicates each single specified interest, extracted from attaching request; Ψ represents the persistent storage element; and *Match* contains a list of publisher nodes that have to authorize subscriptions.

- If *Match*ontains some results (e.g., g_m node), these are forwarded to the GRAN module, which interacts with discovered publisher nodes, sending them the identity of the requesting listener node and asking them to allow or deny the subscription to the requested topics. This response is forwarded back to the GRAN, that analyzes it and, in compliance with the Application Register, authorizes or rejects the listener Graph node subscription.

In order to better explain the behavior of the Application Register module, in relation to the join operation of an external entity that asks to become a Graph listener, in the following Listing 1 we detail the interactions between this external entity and the Application Register, through a pseudocode representation.

The processing nodes in the Graph architecture can be, at same time, listeners as well as publishers, so that the previously detailed mechanisms could be applied together, without any constraint on the execution order. The flows shown in Figure 5 represent the interactions related to this mixed case. The rule on node authority, restricted to the same layer and to next one, decreases lookup times in rules matching execution.

Moreover, external SOs producers could also request a totally "secured" path, from source to final consumer. These constraints have a higher priority than policies defined by publisher Graph nodes, being forced by the stream generators. In this way, these external priority rules are stored into the persistent storage elements as well, and when a new Graph node registers to the proposed IoT platform, its Graph-related policies are left out, forcing these new nodes to comply with these external rules.

Securing Stream inside Graph Nodes

The Graph Framework is composed by several processing entities, which perform some kind of computation on incoming data, representing a single node in the Graph topology.

Listing 1. Pseudocode representation of the operations done by the Application Register, when an external process p_m asks to become a Graph listener in the Big Stream architecture

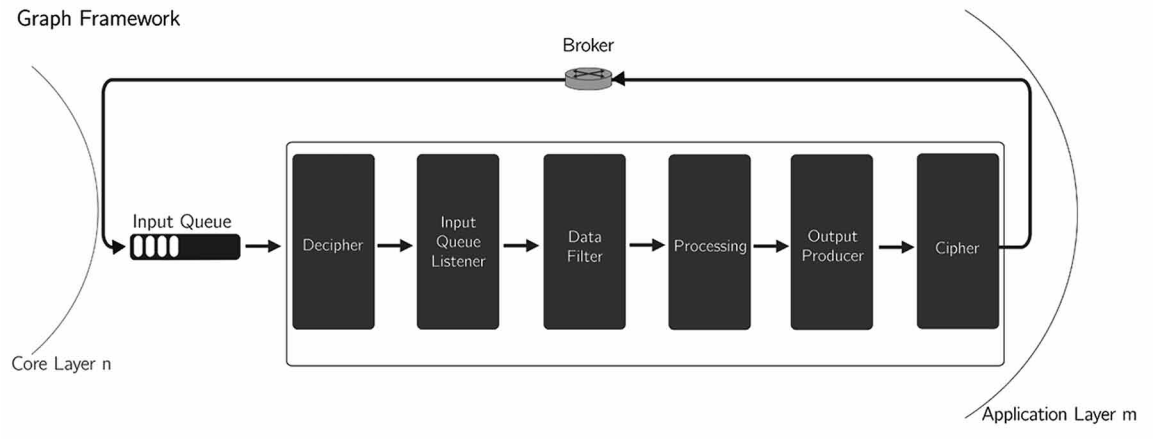
```
NodeIdentityCertificate = {
  NodeInformations = {....};
  INTERESTS = {interest_1, interest_2, ...., interest_N};
}

MAIN() {
  Pm = receive_join_request(NodeIdentityCertificate);
  authenticated = authenticate_node(Pm);
  if (authenticated is TRUE) {
    x = identify_graph_layer_for_node(Pm);
    foreach (interest interest_k in INTERESTS) {
      Match = send_request_to_PMSV(layer = (x OR x+1), interest = interest_k);
      if (Match is EMPTY) {
        REPLY_DENY(topic = interest_k, destination_node = Pm);
      }
      else {
        foreach (GraphNode Gm in Match) {
          allow = request_grant_to_owner_via_GRAN(topic = interest_k, owner = Gm, listener = Pm);
          if (allow is FALSE) {
            REPLY_DENY(topic = interest_k, destination_node = Pm);
          }
          else {
            REPLY_SUCCESS(topic = interest_k, destination_node = Pm);
          }
        }
      }
    }
  }
}

REPLY_DENY(topic, destination_node) {
  send_DENY_response_to_destination(required_topic = topic);
}

REPLY_SUCCESS(topic, destination_node) {
  send_SUCCESS_response_to_destination(required_topic = topic, security_parameters = {....});
}
```

The connection of multiple listeners across all processing units defines the routing path of data streams, from producers to consumers. All nodes in the Graph can be listeners for incoming data and output producers for other successor Graph nodes.

Since each Graph node is owner of the stream generated by its processing activity, it is reasonable to assume that it could decide to maintain its generated stream "open" and accessible to all its interested listeners, instead of applying securing policies, isolating its own information and defining a set of rules that restrict the amount of authorized listeners nodes. In this latter case, a "secured" stream should be created and encrypted using the algorithms selected by the owner. Each listener is thus required to decrypt incoming data before performing any processing. These encryption/decryption operations could be avoided if listeners adopt homomorphic encryption (Gentry, 2009), that allows to carry out computations on ciphertext, instead of on plaintext, generating an encrypted result that matches with one performed on the plaintext, without exposing the data to each of different steps chained together in the workflow. Homomorphic encryption allows to execute computation in the Encrypted Domain, providing end-to-end security, avoiding encryption/decryption hop-by-hop needs.

In Figure 6, the modules inside a Graph node are shown: the broker of the Core layer *n* forwards streams to interested Graph nodes, forwarding these data into the input queue of the single node. The output stream, generated by the processing of this node, will be sent to the same broker of the Core layer

Figure 6. Detail of the structure of a single Graph node: the decryption module is activated when the node is a listener of a secured stream, while the encryption module is activated if the node generates a secured stream



n, which is linked to the broker of the next Graph layer, that "spreads" generated streams to all interested nodes. Some of these modules will be activated only in specific situations. In particular, the illustrated node acts as listener of a "secured" data stream, so it has to decrypt an incoming message, activating the decryption module. Moreover, this Graph node acts also as a producer of a "secured" stream and, then, it has to encrypt its processed streams with the encryption module before forwarding it, thus hiding the stream from other unauthorized listener Graph nodes. Referring to the previously described example, this is the case in which a Graph node, that is already a listener of the secured stream owned by company "C," wants to secure the stream generated by its processing.

It is important to point out that each Graph node controls its generated flow with a visibility of only one step. This means that a listener of a "secured" flow can publish an "open" stream, and vice versa, thus producing "hybrid" path combinations, that across a flow from IoT source to final consumer produce a combination of "secured" and "open" steps.

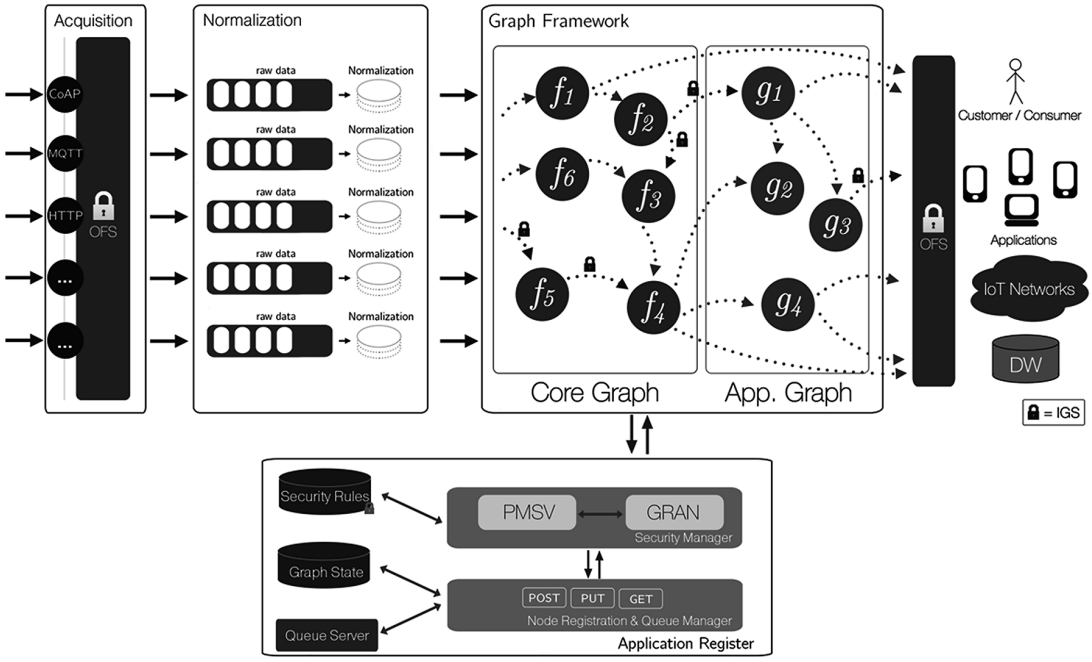
Referring to the example of company "C," which generates a "secured" stream with data coming from its sensors, an IoT developer can decide to create a new Graph node listening from both the secured stream of company "C" and the stream of another company "D." In the processing unit of the new Graph node, the developer can aggregate and transform input streams, generating new and different output streams, which can be published in an "open" mode, since the developer is the owner of this new produced stream.

According to the inner organization of the IoT architecture, there could exist a parallelism between actors enrolled in Graph Framework and OAuth roles, as illustrated in Table 1.

Table 1. Comparison between Graph Framework actors and OAuth roles

Graph Framework Actor	OAuth Role
Publisher Graph node, owner of the outgoing data stream	Resource Owner
Listener Graph node, willing to subscribe to interested topics	Consumer
Infrastructure routing element (Broker in a pub/sub paradigm)	Provider

Figure 7. Complete IoT Cloud architecture, including proposed security modules and showing different interactions, from incoming stage to final consumer notification



More precisely, OAuth roles could be detailed as follows.

- **Resource Owner:** The entity which owns the required resource and has to authorize an application to access it, according to authorization granted (e.g., read/write permission).
- **Consumer:** The entity that wants to access and use the required resource, operating in compliance with granted policies related to this resource.
- **Provider:** The entity that hosts the protected resource and verifies the identity of the Consumer that issues an access request to this resource.

As previously stated, each Graph node can apply cryptography to its streams, using encryption and decryption modules. The security mechanisms leave a few degrees of freedom to developers, who can adopt self-made solutions (e.g., using OAuth tokens) to secure streams, as well as rely on already secured protocols, thus adopting well-known and verified solutions. An overall view of the envisioned IoT architecture is shown in Figure 7, showing all component modules and their interactions.

EVALUATION OF THE PROPOSED ARCHITECTURE

Deployment Setup

In (Belli et al., 2015) a first implementation of the architecture, without security modules, has been presented, deploying it on a Virtual Machine and through the definition of a real use case, represented by

a Smart Parking scenario. The dataset used for the evaluation contains more than 600k parking events, related to the spots status (free/busy), but lacks geographical information (namely, geographic coordinates of the spots). Thus, in order to create a realistic scenario, parking spots have been located into 7 clusters.

Thus, in order to stress enough the proposed Big Stream platform, the evaluations have been conducted by varying the data generation rate in a proper range, forcing a specific frequency for incoming events, without taking into account real parking spots timestamps gathered from the dataset.

The evaluation involves a Java-based data generator, which: simulates events arrivals from IoT sensors networks; randomly selects an available protocol (HTTP, CoAP, MQTT); and periodically sends streams to the corresponding interface in the Acquisition Module. Once received, data are forwarded to the dedicated stage into the Normalization module, which enriches them with parking membership information, retrieving this association from an external SQL database, thus structuring the stream into a JSON schema compatible with the architecture. Once the Normalization module has completed its processing, it sends the structured data to the Graph Framework, that forwards the stream following paths based on routing keys, until final external listener is reached. The Graph considered in our experimental set-up is composed by 8 Core layers and 7 Application layers, within which different graph topologies (from 20 to 50 nodes) are built and evaluated.

Experimental Results

The proposed architecture has been evaluated by varying the incoming data stream generation rate between 10 msg/s and 100 msg/s. The first evaluation, which represents a benchmark for our performance analysis, has been made using the platform without security mechanisms. Then, we have introduced security mechanisms in the Graph Framework Module, in order to assess the impact of a security stage on the overall architecture.

The *first performance evaluation* has been conducted by measuring the delay (dimension: [ms]) between the instant at which normalized data are injected into Graph Framework and the instant at which the message reaches the end of its routes, becoming available for external consumers/customers. In order to consider only the effective overhead introduced by the architecture, and without taking into account implementation-specific contributions, performance results were obtained by subtracting the processing time of all Core and Application Nodes. Finally, these times have been normalized over the number of computational nodes, in order to obtain the per-node overhead introduced by the architecture, in a way that is independent of the implemented routing and topology configurations.

The *second performance evaluation* has been conducted adopting the same structure of the unsecured implementation, but introducing security mechanisms inside Graph nodes, through the adoption of symmetric encryption to encrypt/decrypt operations mentioned in the previous section. In order to guarantee a trade-off between security-level and reliability, we choose to use Advanced Encryption Standard (AES) which is a symmetric cryptosystem, in its 256-bit key version (Daemen & Rijmen, 2002). AES is a block cipher based on a substitution and permutation (SP) combination, working on 128-bits blocks. The chosen key size specifies the number of repetitions of transformation rounds that convert the input, applying several processing stages and depending on the encryption key. The strength of AES256 is derived by its key space, with 10^{77} possible 256-bit keys, which affects the time needed to made a successful brute-force attack to a system implementing this cipher.

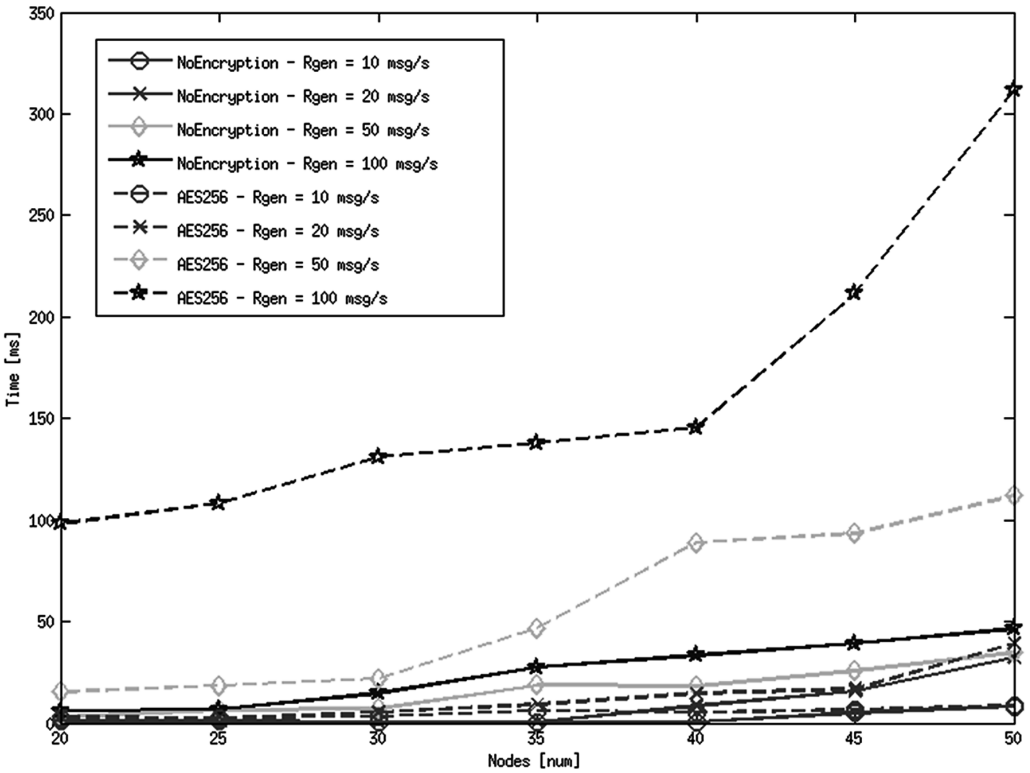
Moreover, in order to perform the second evaluation, we have implemented a new version of the processing Core and Application nodes, applying security at both input and output stages of the single Graph node, implementing the following behavior:

- If the processing node has received an AES 256-bit decryption key from the Application Register, it then uses this key to decrypt incoming messages, returning a plaintext useful for processing operations;
- If an AES 256-bit encryption key was provided by the Application Register to the Graph node, it then encrypt the processed stream before forwarding it to its proper Exchange, using this symmetric key as encryption secret.

This security model is applicable also to the previously described example, in which the company "C" would like to secure its paths into the Graph Framework. The second evaluation has been made providing encryption and decryption keys to all Graph nodes, in order to secure all the intermediate routes followed by streams owned by that company.

The results of the performance evaluations outlined above, carried out using different topologies obtained by varying the subset of nodes deployed, from 20 to 50, and the data generation rate R_{gen}, from 10 msg/s to 100 msg/s, are shown in Figure 8.

Figure 8. Average stream delay (dimension: [ms]) related to Graph Framework processing block, showing per-node time, in the case of unsecured communication as well as the case of adoption of symmetric encryption



The stream delay can be given using the following expression:

$$T_{processing_{freq}} = \frac{T_{out} - T_{in} - \sum_{k=1}^{N} GP_k}{N}$$

where: T_{out} is the instant (dimension: [ms]) at which parking data reach the last Application processing node; T_{in} indicates the instant (dimension: [ms]) in which normalized data comes to the first Core layer; and GP_k is the processing time (dimension: [ms]) of a Graph node $k \in \{1,\ldots,N\}$.

Moreover, in order to investigate the benefits and drawbacks of the adoption of other security solutions, different from symmetric encryption, we have implemented an asymmetric-cryptography version of the Graph processing nodes, adopting RSA (Rivest, Shamir, & Adleman, 1978) with 512-bit key, which represent a private-public key cryptosystem.

In Table 2, the performance results, in terms of stream delay, retrieved from a third evaluation scenario, obtained by replacing symmetric cryptosystem adoption with private/public RSA certificates provided to the Graph nodes by the Application Register module, are shown.

The obtained results shown in Table 2 highlight how the adoption of an asymmetric cryptosystem represents a bad choice for the Graph inter-node security. Asymmetric solutions might be adopted outside of the Graph nodes, when an external node is willing to become an operating entity of the Graph Framework, challenging an authentication transaction with its signed certificate, that allows to verify its identity by the Application Register. Therefore, in the joining phase the asymmetric solutions could also be motivated by the evidence that time consumption is not the main constraint of this step.

In order to better highlight this final analysis of the evaluation results, in Figure 9 a logarithmic-scaled version of previously carried results is shown, evaluating the logarithm of the stream delay as a function of the number of nodes in the Graph: (i) evaluation without encryption, (ii) evaluation with symmetric encryption (AES256), and (iii) evaluation with asymmetric encryption (RSA512), considering only, for comparison purpose, two values of data generation rate R_{gen}: 50 msg/s and 100 msg/sec.

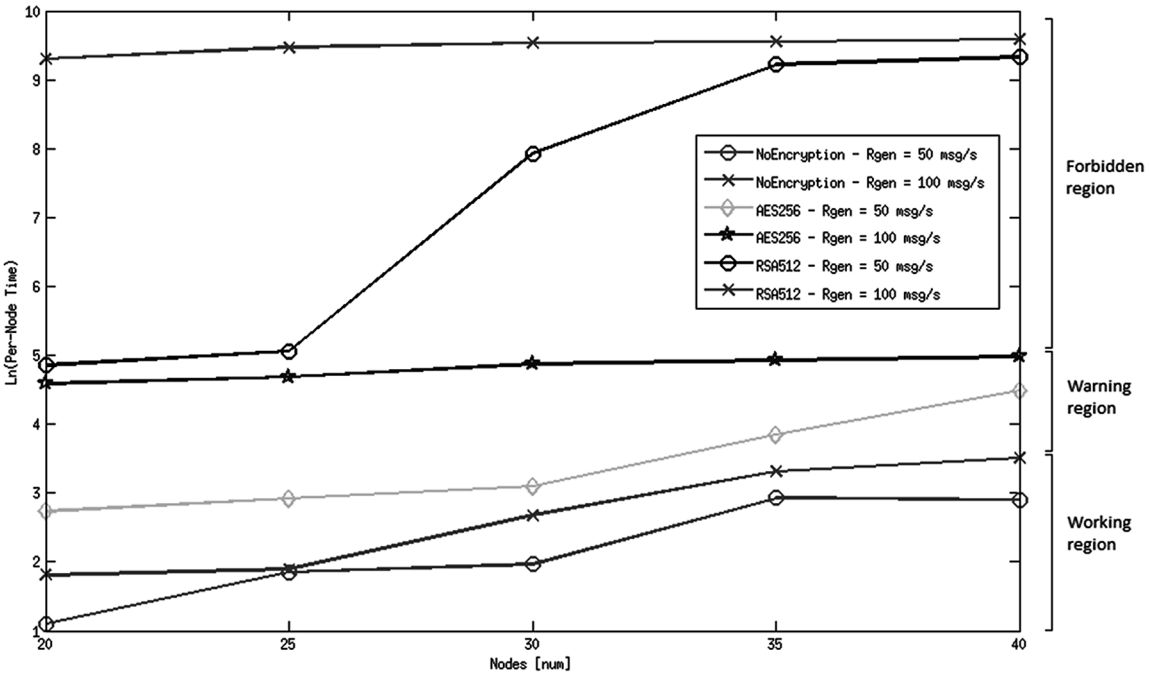
The obtained results allow us to highlight that it seems possible to identify 3 main performance regions, in which the proposed Big Stream platform could work:

- "Working" region, tentatively identified around the "NoEncryption" curves, on which the system has the benchmark results, and where processing do not introduce heavy delays;

Table 2. Average stream delay related to the adoption of asymmetric encryption solution (RSA) into the Graph Framework processing block

Number of Nodes	Stream Delay (R_{gen} = 50 msg/s)	Stream Delay (R_{gen} = 100 msg/s)
20	128.7453 ms	10890.7263 ms
25	156.909 ms	12962.2934 ms
30	2783.8599 ms	13841.4744 ms
35	10104.7272 ms	14048.8625 ms
40	11283.9916 ms	14515.0021 ms

Figure 9. Logarithmic representation of the stream delay as a function of the number of nodes of the Graph, evaluated both without security mechanisms, as well as with cryptographic features



- "Warning" region, limited around the "AES256" curves, in which delays introduced by security adoptions degrade performances in a little way, while maintaining good Quality of Service (QoS);
- "Forbidden" region, around the "RSA512" curves, in which the system incurs high delays, that may cause crashes and drop services, invalidating QoS and any Service Level Agreements (SLAs) signed with data stream producers and consumers.

CONCLUSION

In this paper, we have analyzed security issues in a Cloud architecture for the management of Big Stream applications in IoT scenarios. After defining characteristics of the Big Stream paradigm, we have described the basic concepts of a recently proposed Graph-based Cloud architecture for IoT. Then the implementation of each module of the platform (Acquisition module, Normalization module, Graph Framework, Application Register) is detailed taking particularly into account security issues. The OAuth protocol, adapted to the specific needs of each module, complies with proposed publish/subscribe platform and is expedient to carry out all security tasks. Unlike traditional authorization methods, based on role hierarchies, as a future work, we plan to implement the proposed security modules and evaluate how OAuth-based components, acting both at the boundaries (OFS module) and inside (IGS module) of the Graph-based Cloud platform, may impact the overall system performance, knowing that a solution based on symmetric cryptosystem is feasible, with little impact on the processing times.

REFERENCES

Aamodt, A., & Nygard, M. (1995). Different roles and mutual dependencies of data, information, and knowledge an AI perspective on their integration. *Data & Knowledge Engineering*, *16*(3), 191–222. doi:10.1016/0169-023X(95)00017-M

Bacon, J., Evans, D., Eyers, D. M., Migliavacca, M., Pietzuch, P., & Shand, B. (2010). Enforcing End-to-End Application Security in the Cloud. In Middleware 2010 (pp. 293-312). Springer Berlin Heidelberg. doi:10.1007/978-3-642-16955-7_15

Belli, L., Cirani, S., Davoli, L., Ferrari, G., Melegari, L., Montón, M., & Picone, M. (2015). A Scalable Big Stream Cloud Architecture for the Internet of Things. [IJSSOE]. *International Journal of Systems and Service-Oriented Engineering*, *5*(4), 26–53. doi:10.4018/IJSSOE.2015100102

Belli, L., Cirani, S., Davoli, L., Gorrieri, A., Mancin, M., Picone, M., & Ferrari, G. (2015, September). Design and Deployment of an IoT Application-Oriented Testbed. *IEEE Computer*, *48*(9), 32–40. doi:10.1109/MC.2015.253

Belli, L., Cirani, S., Davoli, L., Melegari, L., Mònton, M., & Picone, M. (2015). An Open-Source Cloud Architecture for Big Stream IoT Applications. In I. Podnar Žarko, K. Pripužić, & M. Serrano (Eds.), *Interoperability and Open-Source Solutions for the Internet of Things* (Vol. 9001, pp. 73–88). Springer International Publishing; doi:10.1007/978-3-319-16546-2_7

Belli, L., Cirani, S., Ferrari, G., Melegari, L., & Picone, M. (2014). A Graph-Based Cloud Architecture for Big Stream Real-Time Applications in the Internet of Things. In *Advances in Service-Oriented and Cloud Computing* (Vol. 508, pp. 91–105). Springer International Publishing; doi:10.1007/978-3-319-14886-1_10

Bonomi, F., Milito, R., Natarajan, P., & Zhu, J. (2014). Fog Computing: A Platform for Internet of Things and Analytics. In Big Data and Internet of Things: A Roadmap for Smart Environments (pp. 169-186). Springer International Publishing. doi:10.1007/978-3-319-05029-4_7

Cirani, S., Davoli, L., Picone, M., & Veltri, L. (2014, Jul). Performance Evaluation of a SIP-based Constrained Peer-to-Peer Overlay. In *2014 IEEE International Conference on High Performance Computing Simulation* (pp. 432-435). DOI: 10.1109/HPCSim.2014.6903717

Cirani, S., Picone, M., Gonizzi, P., Veltri, L., & Ferrari, G. (2015, February). IoT-OAS: An OAuth-based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sensors Journal*, *15*(2), 1224–1234. doi:10.1109/JSEN.2014.2361406

Cirani, S., Picone, M., & Veltri, L. (2013). CoSIP: A Constrained Session Initiation Protocol for the Internet of Things. In *Advances in Service-Oriented and Cloud Computing* (Vol. 393, pp. 13–24). Springer Berlin Heidelberg; doi:10.1007/978-3-642-45364-9_2

Cirani, S., Picone, M., & Veltri, L. (2015). mjCoAP: An Open-Source Lightweight Java CoAP Library for Internet of Things Applications. In Interoperability and Open-Source Solutions for the Internet of Things. LNCS (Vol. 9001). DOI: , Springer International Publishing Switzerland. doi:10.1007/978-3-319-16546-2_10

Collina, M., Corazza, G. E., & Vanelli-Coralli, A. (2012). Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. In 2012 IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (pp.36-41). 9-12 Sept. 2012. DOI: 10.1109/PIMRC.2012.6362813

Daemen, J., & Rijmen, V. (2002). *The design of Rijndael: AES - the Advanced Encryption Standard*. Springer-Verlag. doi:10.1007/978-3-662-04722-4

Fremantle, P., Aziz, B., Scott, P., & Kopecky, J. (2014, Sep). Federated Identity and Access Management for the Internet of Things. In *3rd International Workshop on the Secure IoT*. 10.1109/SIoT.2014.8

Gentry, C. (2009). A fully Homomorphic encryption scheme (Unpublished doctoral dissertation). Stanford University. Retrieved from http://crypto.stanford.edu/craig

Hardt, D. (2012). RFC 6749: The OAuth 2.0 Authorization Framework. Retrieved from http://tools.ietf.org/html/rfc6749

Hohpe, G., & Woolf, B. (2003). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing.

Isaacson, C. (2009). *Software Pipelines and SOA: Releasing the power of multi-core processing* (1st ed.). Addison-Wesley Professional.

Kim, E., Kaspar, D., & Vasseur, J. (2012, Apr). Design and application spaces for ipv6 over low-power wireless personal area networks (6LoWPANs) (No. 6568). RFC 6568 (Informational). IETF. Retrieved from http://www.ietf.org/rfc/rfc6568

Lagutin, D., Visala, K., Zahemszky, A., Burbridge, T., & Marias, G. F. Roles and security in a publish/subscribe network architecture. In *2010 IEEE Symposium on Computers and Communications* (pp. 68-74). 22-25 June 2010. DOI: 10.1109/ISCC.2010.5546746

Leavitt, N. (2013). Storage challenge: Where will all that Big Data go? *Computer*, *46*(9), 22–25. doi:10.1109/MC.2013.326

Marganiec, S. R., Tilly, M., & Janicke, H. (2014, Jun). Low-Latency Service Data Aggregation Using Policy Obligations. In *2014 IEEE International Conference on Web Services* (pp. 526-533). IEEE.

Medagliani, P., Leguay, J., Duda, A., Rousseau, F., Domingo, M., Dohler, M., . . . Dupont, O. (2014). Bringing IP to Low-power Smart Objects: the Smart Parking Case in the CALIPSO Project. Retrieved from http://www.ict-calipso.eu/

Mera Pérez, D., Batko, M., Zezula, P., et al. (2014). Towards fast multimedia feature extraction: Hadoop or storm.

Moreno-Vozmediano, R., Montero, R. S., & Llorente, I. M. (2012). IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, *45*(12), 65–72. http://doi.ieeecomputersociety.org/10.1109/MC.2012.76 doi:10.1109/MC.2012.76

Neumeyer, L., Robbins, B., Nair, A., & Kesari, A. (2010). S4: Distributed stream computing platform. In *2010 IEEE International Conference on Data Mining Workshops* (pp. 170–177). 10.1109/ICDMW.2010.172

Raiciu, C., & Rosenblum, D. S. (2006). *Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures* (pp. 1–11). Securecomm and Workshops; doi:10.1109/SECCOMW.2006.359552

Rivest, R., Shamir, A., & Adleman, L. (1978, February). A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Comm. ACM, 21 (2), 120-126. Retrieved from http://doi.acm.org/10.1145/359340.359342

Shelby, Z., Hartke, K., Bormann, C., & Frank, B. (2014). *RFC 7252: The Constrained Application Protocol (CoAP)*. Internet Engineering Task Force.

Tilly, M., & Reiff-Marganiec, S. (2011, March). Matching customer requests to service offerings in real-time. In *Proceedings of the 2011 ACM Symposium on Applied Computing* (pp. 456 -461). ACM. 10.1145/1982185.1982285

Wang, C., Carzaniga, A., Evans, D., & Wolf, A. L. Security issues and requirements for Internet-scale publish-subscribe systems. In 2002 Proceedings of the 35th Annual Hawaii International Conference on System Sciences (pp. 3940-3947). 7-10 Jan. 2002. DOI: 10.1109/HICSS.2002.994531

This work was previously published in the International Journal of Distributed Systems and Technologies (IJDST); 7(1) edited by Nik Bessis, pp. 37-58, copyright 2016 by IGI Publishing (an imprint of IGI Global).