IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios

Simone Cirani, Marco Picone, Pietro Gonizzi, Luca Veltri, and Gianluigi Ferrari, *Senior Member, IEEE*

***Abstract*— Open authorization (OAuth) is an open protocol, which allows secure authorization in a simple and standardized way from third-party applications accessing online services, based on the representational state transfer (REST) web architecture. OAuth has been designed to provide an authorization layer, typically on top of a secure transport layer such as HTTPS. The Internet of Things (IoTs) refers to the interconnection of billions of resource-constrained devices, denoted as smart objects, in an Internet-like structure. Smart objects have limited processing/memory capabilities and operate in challenging environments, such as low-power and lossy networks. IP has been foreseen as the standard communication protocol for smart object interoperability. The Internet engineering task force constrained RESTful environments working group has defined the constrained application protocol (CoAP) as a generic web protocol for RESTful-constrained environments, targeting machine-to-machine applications, which maps to HTTP for integration with the existing web. In this paper, we propose an architecture targeting HTTP/CoAP services to provide an authorization framework, which can be integrated by invoking an external oauth-based authorization service (OAS). The overall architecture is denoted as IoT-OAS. We also present an overview of significant IoT application scenarios. The IoT-OAS architecture is meant to be flexible, highly configurable, and easy to integrate with existing services. Among the advantages achieved by delegating the authorization functionality, IoT scenarios benefit by: 1) lower processing load with respect to solutions, where access control is implemented on the smart object; 2) fine-grained (remote) customization of access policies; and 3) scalability, without the need to operate directly on the device.**

***Index Terms*— Internet of Things, security, authorization, communication protocols.**

I. Introduction

THE evolution of online services, such as those enabled by social networks, has had a relevant impact on the amount of data and personal information disseminated on the Internet. Furthermore, it has determined the birth of applications that rely on the disseminated information in order to offer new services, such as aggregators. The information owned by online services is made available to third-party applications in the form of public Application Programming Interfaces (APIs), typically using HTTP [1] as communication protocol and relying on the REpresentational State Transfer (REST) architectural style. The possibility that someone else, besides the entity which generates the information and the service that is hosting it, can access this information has brought up concerns about the privacy of personal information, since the trust is no longer a pairwise relationship but possibly involves other parties, which may be unknown at the time of service subscription.

Open Authorization (OAuth) is an open protocol to allow secure authorization from third-party applications in a simple and standardized way [2]. The OAuth protocol provides an authorization layer for HTTP-based service APIs, typically on top of a secure transport layer, such as HTTP-over-TLS (i.e., HTTPS) [3]. OAuth defines three main roles in the above scenario:

- the *User* (*U*) is the entity which generates some sort of information;
- the *Service Provider* (*SP*) hosts the information generated by the users and makes it available through APIs;
- the *Service Consumer* (*SC*), also referred to as "client application," accesses the information stored by the *SP* for its own utilization.

In order to comply with the security and privacy requirements, *U* must issue an explicit agreement that some client application can access information on its/his/her behalf. This is achieved by granting the client an access token, containing *U*'s and *SC*'s identities, which must be exhibited in every request as an authorization proof. The OAuth 2.0 protocol is the evolution of the original OAuth protocol and aims at improving the client development simplicity by defining scenarios for authorizing web, mobile, and desktop applications [4]. While connecting to existing online services is a simple task for client application developers, implementing an OAuth-based authorization mechanism on the *SP*'s side is a more complicated, time-consuming, and, potentially, computationally intensive task. Moreover, it involves the registration of both users and client applications, and the permissions that *U*s grant to *SC* applications and integrating with authentication services.

The Internet of Things (IoT) refers to billions of interconnected devices, denoted as "Smart Objects" or "Smart Things." Smart Objects are typically equipped with sensors or actuators, a tiny microprocessor, a communication interface, and a power source. Smart Objects operate in challenging environments, such as lossy and low-power networks, have limited computational power, and are usually battery-powered, thus requiring a significant attention on keeping the energy consumption low. International organizations, such as the Internet Engineering

Manuscript received July 21, 2014; accepted September 22, 2014. Date of publication October 3, 2014; date of current version December 3, 2014. The work of S. Cirani, P. Gonizzi, L. Veltri, and G. Ferrari was supported by the European Community's Seventh Framework Program through the Project entitled Internetconnected Objects under Grant 288879, in part by the CALIPSO Project-Connect All IP-Based Smart Objects. The work of M. Picone was supported by Guglielmo S.r.L, Reggio Emilia, Italy. The associate editor coordinating the review of this paper and approving it for publication was Prof. Kiseon Kim.

The authors are with the Department of Information Engineering, University of Parma, Parma 43124, Italy (e-mail: simone.cirani@unipr.it; marco.picone@unipr.it; pietro.gonizzi@studenti.unipr.it; luca.veltri@unipr.it; gianluigi.ferrari@unipr.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSEN.2014.2361406

Task Force (IETF), the IPSO Alliance [5], and several research projects, such as the FP7 EU project CAL*IP*SO (Connect All IP-based Smart Objects!) [6], promote the use of the Internet Protocol (IP) as the standard for interoperability between Smart Objects. The protocol stack run by Smart Objects tries to match classical Internet hosts in order to make it feasible to create the so-called "extended Internet," i.e., the aggregation of the Internet with the IoT. The IETF CoRE Working Group has defined the Constrained Application Protocol (CoAP) [7], a generic web protocol for RESTful constrained environments, targeted to Machine-to-Machine (M2M) applications, which maps to HTTP for integration with the existing web.

Security in IoT scenarios is a crucial aspect that applies at different levels, ranging from technological to privacy and trust issues, especially in scenarios involving Smart Toys (used by children) or crowd/social behavior monitoring. This is related to the fact that Smart Objects might deal with personal or sensible data, which, if intercepted by unauthorized parties, may create ethical and privacy problems. While the use of the OAuth protocol has little impact, in terms of processing and scalability, on conventional Internet-based services, its adoption in IoT has to deal with the limitations and challenges of constrained devices. The limited computational power of Smart Objects may not be sufficient to perform the cryptographic primitives required for message authentication, integrity checks, and digital signatures, which may have a negative impact on energy consumption. Moreover, if the access permissions for the services provided by the Smart Object reside on the Smart Object itself, it could be extremely hard, if not impossible, to dynamically update them (e.g., in smart parking systems where Smart Objects may be embedded directly in the asphalt, such as Fastprk[1] by Worldsensing [8]) once they have been deployed.

In this paper, we present a novel architecture targeted to IoT scenarios for an external authorization service based on OAuth, denoted as *IoT-OAS*. The delegation of the authorization functionalities to an external service, which may be invoked by any subscribed host or thing, affects:

1) the time required to build new OAuth-protected online services, thus letting developers focus on service logic rather than on security and authorization issues;
2) the simplicity of the Smart Object, which does not need to implement any authorization logic but must only invoke securely the authorization service in order to decide whether to serve an incoming request or not;
3) the possibility to dynamically and remotely configure the access control policies that the SP is willing to enforce, especially in those scenarios where it is hardly possible to intervene directly on the Smart Object.

Experimental results are presented highlighting the existing trade-off between communication and processing costs.

The rest of this paper is organized as follows. In Section II, related works are presented. In Section III, the *IoT-OAS* architecture is presented. In Section IV, a few *IoT-OAS* application scenarios are presented. In Section V, an extensive experimental performance evaluation of the proposed solution is carried out. In Section VI, we discuss about open issues related to the proposed architecture and IoT scenarios. Finally, in Section VII we draw our conclusions.

[1] http://www.fastprk.com/

II. Related Work

In the rapidly evolving IoT scenario, security is an extremely timely issue. The heterogeneous and dynamic nature of the IoT brings up several questions related to security and privacy, which must be addressed properly by taking into account the specific characteristics of Smart Objects and the environments they operate in. Classical security algorithms and protocols, used by traditional Internet hosts, cannot simply be adopted by Smart Objects, due to their processing and communication constraints. An extensive overview of state-of-the-art security mechanisms in the IoT (including symmetric/asymmetric cryptographic algorithms, hashing functions, security protocols at network/transport/application layers), aiming at providing features such as confidentiality, integrity, and authentication, is provided in [9]. An architecture for solving the problem of securing IoT cyberentities (which include Smart Objects, traditional hosts, and mobile devices), denoted as "U2IoT," has been proposed in [10], with the goal of addressing the issues of expanding domains, dynamic activity cycles, and heterogeneous interactions. U2IoT takes into account security in interactions that occur in three different phases: preactive, active, and postactive. In particular, the active phase provides authentication and access control functionalities. Authorization is therefore being considered a major issue, since it is becoming increasingly evident that access to resources in a global-scale network, such as the IoT, must be controlled and restricted in order to avoid severe security breaches in deployed applications.

Several works have also addressed very specific issues in IoT. A lightweight multicast authentication scheme for small-scale IoT applications is proposed in [11]. In [12], the authors take into account user mobility (i.e., roaming) and propose CPAL, an authentication mechanism designed to provide a "linking function" that can be used to enable authorized parties to link user access information, while preserving user anonymity and privacy. The secure integration of Wireless Sensor Networks (WSNs) into IoT is discusses in [13]. The authors propose a security scheme, which allows secure communication with Internet hosts by providing end-to-end confidentiality, integrity, and authentication, based on a Public-Key Infrastructure (PKI). The proposed scheme also introduces a two-step (offline/online) signcryption mechanism, in order to minimize processing time.

Several authentication mechanisms have also been defined for other issues, such as network access, which are also relevant for IoT scenarios. The Protocol for Carrying Authentication for Network Access (PANA) [14] is an IETF standard defining a network-layer transport for network access authentication methods, which are typically provided by the Extensible Authentication Protocol (EAP) [15]. In particular, PANA carries EAP, which can carry various authentication methods. OpenPANA [16] is an open-source implementation of PANA.

The problem of service authorization has been extensively treated in literature. Several works have focused on how to implement different access control strategies. *Discretionary Access Control* (DAC) restricts the access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with certain access permissions can transfer that permission on to any other subject [17]. *Role-Based Access Control* (RBAC) relies on a policy that restricts access to resources to those entities which have been assigned a specific role [18]–[20]: RBAC requires that the roles are defined and assigned to users, and access permissions are set for resources. *Attribute-Based Access Control* (ABAC) restricts resource access to those entities which feature one or more specific attributes (e.g., age, geographic location, etc.) [21].

RBAC and ABAC are the most widespread approaches to restricting system access to authorized users. RBAC maps permissions to *roles* that a user has been assigned. On the other hand, ABAC maps permissions to *attributes* of the user. Typically, authorization mechanisms strongly depend on an authentication step that must have been previously taken, in order to identify users so that either their roles or their attributes can be verified and matched against the policies set for resource access.

In [22], the authors present a mechanism for fine-grained sub-delegation of access permissions for consumers of web applications, denoted as "DAuth." Applying access control mechanisms in constrained scenarios, such as wireless sensor networks, is a challenging task. A complex, context-aware access control system designed for a medical sensor networks scenario, which presents critical privacy and confidentiality issues, is described in [23].

The IETF ACE WG has also proposed "Delegated CoAP Authentication and Authorization Framework" (DCAF) [24]. The DCAF architecture introduces authorization servers, which are used to perform authentication and authorization, in order to unburden Smart Objects from storing a large amount of information by delegating such task to an external entity. While this solution is very similar to the one presented in this work, it focuses mainly on constrained environments, while the proposed one is intended to be generic and transparently integrated into IoT and Internet scenarios.

Although much work has been done with the goal of defining and integrating authorization mechanisms in several scenarios, the current paper, unlike others, focuses on the definition of a generic authorization service which can be integrated into both Internet and IoT scenarios. In particular, the proposed mechanism explicitly takes into account the hybrid nature of the extended Internet that will be deployed in the next years. Moreover, the proposed architecture aims at minimizing the effort required by service developers to secure their services by providing a standard, configurable, and highly interoperable authorization framework.

III. IoT-OAS Architecture

The OAuth-based Authorization Service Architecture (*IoT-OAS*) can be invoked by any subscribed host or Smart Object. It can be ideally thought of as a remotely triggered switch that filters incoming requests and decides whether to serve them or not. The design goal of the *IoT-OAS* architecture is to relieve Smart Objects from the burden of handling a large amount of authorization-related information and processing all incoming requests, even if unauthorized. By outsourcing these functionalities, Smart Objects can keep their application logic as simple as possible, thus meeting the requirements for keeping the memory footprint as low as possible, which is extremely important for constrained devices. From a broader perspective, entire IoT large-scale deployments can greatly benefit from the presence of *IoT-OAS* in terms of configurability: a single constrained node (or a group of constrained nodes as a whole) can have their access policies updated remotely and dynamically, without requiring any direct intervention, which is especially convenient for Smart Objects placed in hardly-reachable and/or unattended locations. OAuth allows third-party applications to get access to user-related information hosted on an online service. All issued requests must certify that the *SC* application has been granted permission by the user to access its personal information on its behalf, namely by adding an "access token," which relates the user's identity and the client application. For ease of presentation, the used acronyms are summarized in Table I.

TABLE I
USED MAIN ACRONYMS

U	*User* or Resource Owner
SP	*Service Provider*, which hosts users' resources
SC	*Service Consumer*, which accesses users' data stored by the *SP*
AS	*Authentication Service*, which is used by the *SP* to verify the identity of a user
RT	*Request Token*, a temporary ticket used by the *SC* to ask *U* to authorize access to its resources
AT	*Access Token*, used by the *SC* to perform authenticated requests
IoT-OAS	Delegated external authorization service, which can be invoked by Smart Objects to perform authorization checks to access protected resources

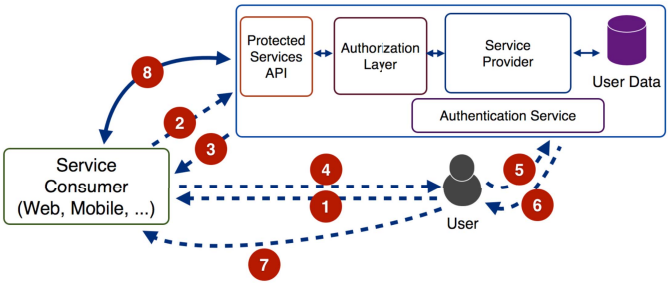

Fig. 1. Standard OAuth roles and operation flow.

Besides the three roles introduced in Section I (*U*, *SP*, and *SC*), OAuth adds an additional role: the Authentication Service (*AS*), which is invoked by the *SP* to verify the identity of a user in order to grant access tokens. The standard OAuth operation flow is shown in Fig.1. The procedure through which a *SC* can get a valid access token is the following:

1) *U* is willing to use the *SC*, either from a webpage, a mobile app, or a desktop application;

2) *SC* needs to access *U*'s personal information hosted on *SP*; *SC* asks the *SP* a *RT* carrying *SC*'s identity, which will be later exchanged for an *AT*;
3) *SP* verifies *SC*'s identity and returns a *RT*;
4) *SC* redirects *U* to the *SP*'s authentication service with the *RT*;
5) *U* contacts the *SP*'s *AS* presenting the *RT* and is asked to authenticate in order to prove its consent to grant access permissions to the *SC*;
6) the *RT* is exchanged for an *AT*, which relates *U* and *SC*;
7) the *SC* receives the *AT* through a redirection to a callback URL (i.e., authentication callback);
8) the *SC* can issue requests to *SP* including the *AT*, for services that require *U*'s permission (protected APIs).

The design goal of the *IoT-OAS* architecture proposed in this work is to enable *SP*s, either based on HTTP or CoAP, to easily integrate an authorization layer without requiring any implementation overhead, other than invoking an external service. Delegating the authorization logic to an external service requires a strong trust relationship between the *SP* and the *IoT-OAS*. Fig. 2 shows the operation flows for a) *AT* grant procedure and b) *SC*-to-*SP* interaction in the *IoT-OAS* architecture. A detailed description of these operation flows in the proposed *IoT-OAS* architecture is presented in the remainder of this section.

A. Granting Access Tokens

The operation flow to grant an *AT* to a *SC* is shown in Fig. 2(a). The procedure resulting in the grant of an *AT* to a *SC* is similar to that of the standard OAuth operation flow, yet it has the following relevant differences:

1) as in the standard operation flow, the procedure is initiated by *U*;
2) the *SC* regularly contacts a *SP* to receive a *RT*;
3) the *SP*, which does not implement any OAuth logic, contacts the *IoT-OAS* asking to issue a *RT* for the *SC* by performing a *generate_request_token* RPC request;
4) the *IoT-OAS* verifies the identity of the *SC* and issues a *RT*, which is returned to the *SP*;
5) the *SP* handles the *RT* back to the *SC*;
6) the *SC* redirects *U* to the *AS* with the received *RT*;
7) *U* contacts the *SP*'s *AS* presenting the *RT* and authenticates in order to prove its consent to grant access permissions to the *SC*;
8) the *AS* notifies the *SP* that the authentication is successful and presents the *RT* with *U*'s identity;
9) the *SP* asks the *IoT-OAS* to exchange the *RT* with an *AT* for *U* by issuing a *generate_access_token* RPC request;
10) the *IoT-OAS* generates the *AT* and returns it to the *SP*;
11) the *SP* handles the *AT* to the *SC* with an authentication callback.

The use of an external *IoT-OAS* is totally transparent to the *SC*, which has no knowledge of how the *SP* is implementing the OAuth protocol. This leads to full backward compatibility with standard OAuth client applications. On the *SP*'s side, all the OAuth logic is delegated to the *IoT-OAS*, with the only exception of the *AS*. However, it is not mandatory that the *AS* resides within the *SP*'s realm, as it might interface with third-party authentication services, such as OpenID [25]. The only information the *SP* must hold is the reference to users' identities in order to make it possible to setup access permission policies on a per-user basis.

B. Authorizing Requests

The interaction between *SP* and *IoT-OAS* when serving incoming requests is shown in Fig. 2(b). Since the presence of the *IoT-OAS* is totally transparent to the *SC*, the communication between the *SC* and the *SP* is a regular OAuth communication. The difference is, again, on *SP*'s side, which needs to contact the *IoT-OAS* to verify that the incoming requests received from the *SC* are authorized in order to decide whether to serve them or not.

The operation flow is the following:

1) the *SC* requests *U*'s information to the *SP* using the *AT* received after *U*'s authentication (as in standard OAuth consumer-to-provider communication);
2) the *SP*, which does not implement any OAuth logic, refers to the *IoT-OAS* to verify if the incoming request is authorized (in order to do so, the *SP* issues a *verify* RPC request);
3) the *IoT-OAS* verifies the *SC*'s request and informs the *SP* about *SC*'s authorization for the request by performing a lookup in the permission store;
4) the *SP* serves the *SC*'s request according to the *IoT-OAS*'s response.

C. SP-to-IoT-OAS Communication: Protocol Details

The *SP* interacts with the *IoT-OAS* with a simple communication protocol. The protocol comprises three Remote Procedure Calls (RPCs), which are detailed below. It is important to remark that delegating the authorization decision to an external service requires an extremely high trust level between the *SP* and the *IoT-OAS*. Moreover, all communications between them must be secured and mutually authenticated, so that the *SP* security level is at least as high as if the authorization service were implemented internally. To ensure that an appropriate security level is met, communications between the *SP* and the *IoT-OAS* must occur with a secure transport such as HTTP-over-TLS (HTTPS), CoAP-over-DTLS (CoAPs) [26], [27], or HTTP/CoAP over a secure host-to-host channel setup with IPSec [27]. Mutual authentication ensures that i) the *IoT-OAS* is verified and ii) the requests come from a verified *SP*, whose identity is, therefore, implicit. The three RPCs of the *SP*-to-*IoT-OAS* communication protocol are the following:

1) *generate_request_token():* this RPC is called by the *SP* to request the *IoT-OAS* to generate a request token for the given *SC*, to be later exchanged for an *AT*;
2) *generate_access_token(request_token, user_id):* this RPC is called by the RPC to request the *IoT-OAS* to exchange the given *RT* for a new *AT* related to the given user;
3) *verify(request, access_token):* this RPC is called by the *SP* to request the *IoT-OAS* to verify if the given *SC* request is authorized with the provided *AT* by performing a lookup into the permission store.

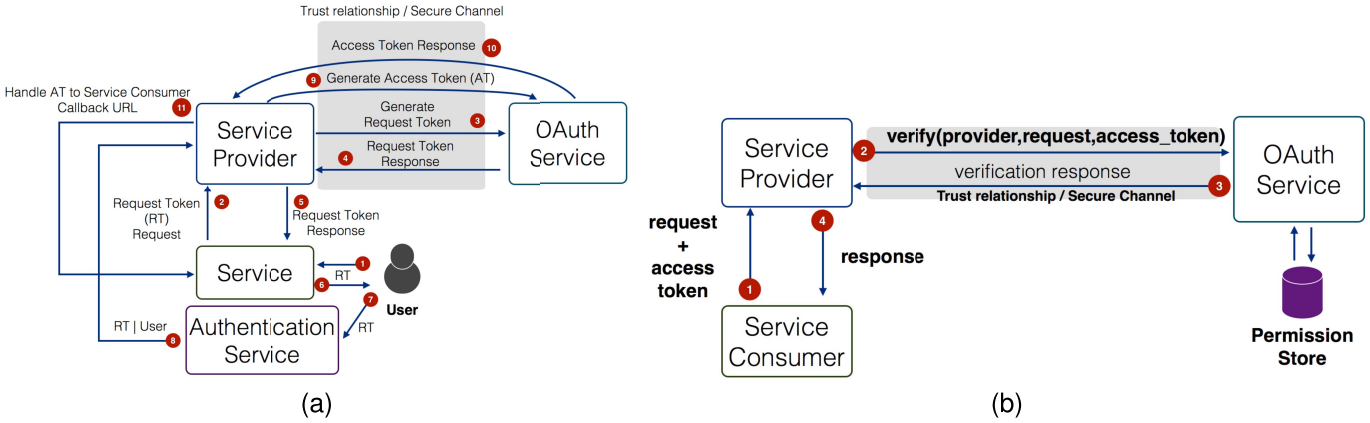

Fig. 2. IoT-OAS main procedures: (a) AT grant procedure and (b) SP integration with IoT-OAS for request authorization.

D. IoT-OAS Configuration

The *IoT-OAS* provides high customization to *SP*s by offering per-user and per-service access control policies. The *SP*s can remotely configure and manage the permissions that *SC*s are granted, which can be created, updated, revoked, and/or duplicated dynamically at any time. The *IoT-OAS* thus offers a dedicated *SP* access control configuration, denoted as "permission store." The permission store is a collection of the relations between *SC*s, users, and *SP* services and can ideally be seen as a lookup table.

The configuration of the permission store can occur through web interfaces or API calls provided by the *IoT-OAS*. The possibility to dynamically manage the permissions, rather than having them co-located with the *SP*, is an extremely valuable feature, especially if *SP*s are Smart Objects with the need to be deployed in hardly accessible locations and may be automatically and/or remotely reconfigured.

IV. Application Scenarios

In this section, we present four significant IoT application scenarios to properly illustrate the functionalities of the proposed *IoT-OAS* service architecture. In the following scenarios, we consider an external client (based on HTTP or CoAP according to the context) that is interested to access a remote service provided by a Network Broker (*NB*), which is a border network element that exposes services on behalf of constrained nodes residing in the internal network, or directly by a generic Smart Object *S* directly available in the network behind a network Gateway. To clarify the following description we assume that the OAuth credentials owned by the involved external client have been obtained through a prior configuration phase based on the *IoT-OAS* service described in Section III and that service discovery is performed through an application-specific procedure, which is not directly related with the approach presented in this work.

A. Network Broker Communication

In the first scenario, illustrated in Fig. 3(a), the client *C* (acting as a *SC*) discovers a $Service_A$ provided by the network broker *NB*. In order to serve external requests, *NB* can retrieve information from different Smart Objects in its network. In this case, in order to simplify the context, we assume that *NB* needs information only from the Smart Object S_2. The client, through a secure channel based on HTTPS or CoAPs, sends a request *R* for $Service_A$ including its OAuth credentials, denoted as *OAuth(C)*. The client's OAuth information is used by the service provider *NB* to properly validate the request and to verify the identity of *C* and that *C* has the right privileges to access the requested service. Since *NB* could be implemented using an embedded device or, generally, using a device with limited computational and storage capabilities (which, with high probability, does/should not implement a complex logic such as OAuth) delegates the verification of the incoming request to the *IoT-OAS*. Once *NB* receives *R* from *C*, it sends a verification request to *IoT-OAS* (always through a secure channel based on HTTPS or CoAPs, according to its implementation or capabilities) with the original incoming request and its credentials. The *IoT-OAS*, after verifying the validity of the submitted request (according to *NB*'s configuration) and *C*'s identity, replies communicating if *R* is authorized. If the feedback is positive and *C* is allowed to access requested service, *NB* internally contacts the Smart Object S_2 to retrieve the required information and sends back the response to *C*. If the response received from *IoT-OAS* is negative, *C* is not granted access and the *NB* responds immediately to *C* without any kind of interaction with the Smart Object.

B. Gateway-Based Communication

In the second scenario, illustrated in Fig. 3(b), an external client *C*, based on HTTPS or CoAPs communications, is interested in accessing a service directly provided by the Smart Object S_2 which does not manage HTTP or CoAP (due to computational or implementation constraints) and is behind a Gateway *G*. In particular, *G* has the role to translate the incoming requests from the external networks to available Smart Objects inside its own network. In this scenario, *C* sends a request *R* (including the client's OAuth credentials) to *G* for $Service_B$ provided by S_2. *R* is translated by *G* and forwarded to S_2 that, in order to validate the new request and the requestor's identity, sends through *G* a verification request

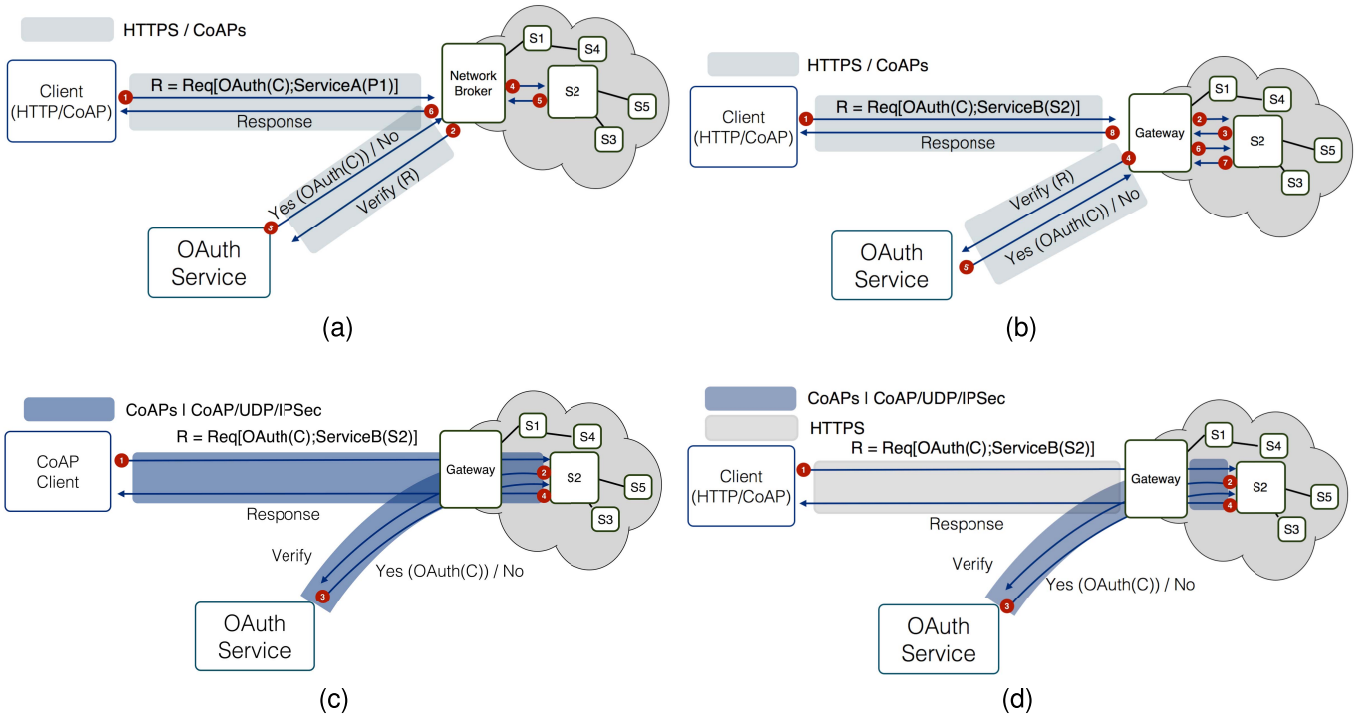

Fig. 3. Application Scenarios: (a) Client-to-Network Broker communication; (b) Gateway-enabled communication; (c) End-to-End CoAP Communication between the external client and the Smart Object; (d) Hybrid Gateway-based communication.

to the *IoT-OAS* using a secure communication protocol such as HTTPS or CoAPs. The verification message and the response (positive or negative) generated by S_2 are properly managed and translated by G to allow the communication among *IoT-OAS*, the Smart Object, and the client.

C. End-to-End CoAP Communication

Fig. 3(c) shows a different scenario where a Smart Object S_2 (i.e., reachable at an IPv6 address) in a sensor network provides directly a remote CoAP service $Service_B$. Since all involved entities can use the same protocol, in this case the network gateway acts only as a router without the need to translate incoming and outgoing messages between the external world and the sensor network. The CoAP client *CC* sends securely and directly to the Smart Object a request R containing its OAuth credentials and the reference for $Service_B$ provided by S_2. Since the Smart Object is usually a sensor or an embedded device with limited computational and storage capabilities (which, as previously described, does not implement a complex logic like OAuth), it delegates the verification of the incoming request to the OAuth Service. S_2 sends a verification request to *IoT-OAS* over CoAPs to check R. The *IoT-OAS* validates the request based on *CC*'s credentials and the type of requested service; it then informs the Smart Object S_2 about whether R can be served or not. S_2, according to the response of *IoT-OAS*, replies to the requesting client with the service outcome or, if *CC* is not allowed to access $Service_B$, with an error message.

D. Hybrid Gateway-Based Communication

The last scenario, shown in Fig. 3(d), is characterized by a hybrid approach where the external client uses an application protocol (such as HTTP) different from that used by Smart Objects (CoAP). Similarly to the case in Subsection IV-B, the gateway manages the communication between the external world and its network: in this case, it just translates incoming requests from HTTP to CoAP for S_2. Once a new request (with OAuth credentials and service reference) arrives to the Smart Object, it securely uses *IoT-OAS* to verify the validity of R, through CoAPs. The response (positive or negative, according to the *IoT-OAS* feedback) is translated by the gateway from CoAP to HTTP and forwarded to the client through a secure channel.

V. Experimental Results

In order to demonstrate the feasibility and performance of the proposed IoT-oriented authorization mechanism, we have conducted experimental tests to evaluate the energy consumption on Smart Objects. In fact, the security of the authorization process is guaranteed by the use of OAuth. We remark that the architecture scenarios in Fig. 3(a) and Fig. 3(b) are not critical from an energy consumption viewpoint, as they rely on communications between a gateway, which is a non-constrained node, and the external authorization service. For this reason, we have limited our experimental investigation to the scenarios in Fig. 3(c) and Fig. 3(d), which require that the Smart Object communicates with the authorization service: in this case, energy consumption is of concern.

A. Experimental Setup

Within the EU project CALIPSO [6], the *IoT-OAS* service has been implemented on a regular web server, based on open source technologies, equipped with HTTP/CoAP proxy capabilities in order to be easily integrated in all the application

TABLE II
EXPERIMENTAL SCENARIOS

		Authorization strategy	
		Smart Object implements OAuth	**Smart Object delegates to *IoT-OAS***
Security	**IP** (none)	*oauth-ip*	*oas-ip*
	IPSec (ESP)	*oauth-ipsec*	*oas-ipsec*

scenarios shown in Section IV. The *SP*-to-*IoT-OAS* communication has been implemented on a variety of devices, either regular hosts or Contiki OS-based constrained devices [28]. The choice of the Contiki OS has allowed to investigate the feasibility of the delegation approach to authorization in both simulated environments (using the Cooja simulator) and real testbeds, taking also into account the presence of duty-cycle. Moreover, the Contiki OS provides IPSec and DTLS implementations over 6LoWPAN [29], [30] and CoAP [31].

The conducted tests aim at evaluating the energy consumption of Contiki-based Smart Objects. The simulations have been performed using the Cooja simulator, in order to gather data for the activity of the CPU and radio interface. The used experimental platform is based on Zolertia Z1 nodes, with nominal 92 kB ROM (when compiling with 20-bit architecture support) and an 8 kB RAM. In practice, the compilation with the Z1 nodes has been performed with a 16-bit target architecture, which lowers the amount of available ROM to roughly 52 kB.

The experimental setup consists of a CoAP client node sending a request to a CoAP server. The CoAP server must then authorize the request and decide whether to serve it or not. This configuration is compatible with the application scenarios shown in Fig. 3(c) (denoted as *End-to-End CoAP Communication*) and Fig. 3(d) (denoted as *Hybrid Gateway-based Communication*). The tests involved four different scenarios, shown in Table II, classified depending on the type of authorization strategy adopted by the CoAP server (columns) and security level adopted at the network-layer (rows). As for security at the network layer, IPSec has been configured to work with Encapsulated Security Payload (ESP) only. ESP has been selected since it is necessary to encrypt the payload of the packets, in order to protect the access token, rather than authenticating the endpoints of communication through Authentication Header (AH). Regarding the implementation of the OAuth protocol, the verification procedure has been performed using the HMAC-SHA1 signature scheme [32]. Although OAuth provides also other signature schemes (PLAINTEXT and RSA-SHA1), HMAC-SHA1 has been selected as it is does not require a secure transport and a PKI for the management of public keys. As for the use of SHA-1, we point out that, in principle, different hash functions, such as SHA-2 and SHA-3 (also known as KECCAK [33]), might be used instead. However, this would be a violation of the OAuth specification, which would require SPs and SCs to be aware of these different signature schemes.

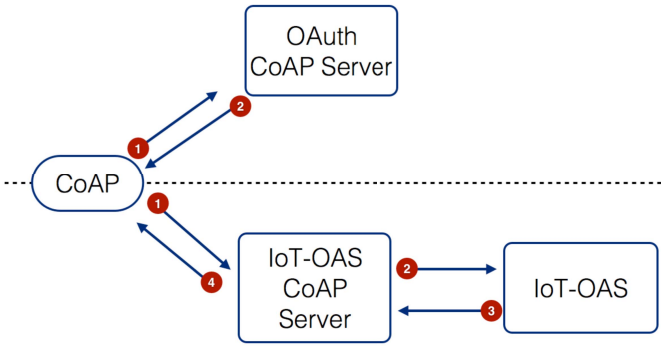

Fig. 4. Message Flow in the case OAuth is implemented in the CoAP server (top) and in case the CoAP server relies to the IoT-OAS server (bottom).

The message flow, which depends solely on the delegation to *IoT-OAS*, is shown in Fig. 4 considering two cases: OAuth is implemented in the CoAP server (top); the CoAP server relies on the *IoT-OAS* server (bottom).

B. Energy Consumption Evaluation

The energy consumption of the CoAP server has been evaluated using Powertrace [34], a tool for network-level power profiling for low-power wireless networks. Powertrace estimates the energy consumption of each device component, such as the radio chip and the microcontroller. It computes the amount of time a component is turned on (active mode) and off (sleep mode). In order to determine the energy consumption, we refer to the current consumption of each component indicated in the Z1 datasheet. In particular, the MSP430f2617 microcontroller consumes 0.515 mA in active mode and 0.5 μA in low-power mode (lpm), respectively. Similarly, the CC2420 radio transceiver consumes 17.4 mA in TX mode and 18.8 mA in RX mode. In order to obtain the total consumed energy for the Smart Object, the following conversion formula has been used:

$$E = \sum_{j \in \mathcal{M}} i_j \cdot v \cdot \Delta t_j \tag{1}$$

where $\mathcal{M}$ is the set of all operation modes of the Smart Object (active, lpm, TX, and RX); v is the nominal voltage of the Smart Object; and Δt_j is the time the Smart Object was in the j-th operation mode j.

In Fig. 5, the aggregate energy consumption (dimension [mJ]) for different hardware components in the four scenarios in Table II is shown. The energy consumption is broken down into figures related to CPU, radio transmission, and lpm activity. The obtained results provide interesting insights on the performance of the proposed delegation approach. In particular, it can be observed that if *IoT-OAS* is being used on top of IP, the amount of energy consumption related to processing is lower than if the OAuth logic is implemented directly on the Smart Object. However, the overall consumption is higher when relying on *IoT-OAS*, due to the contribution of radio transmission. The reason for this behavior is that the large size of application-level packets requires fragmentation and, thus, multiplies the number of transmitted

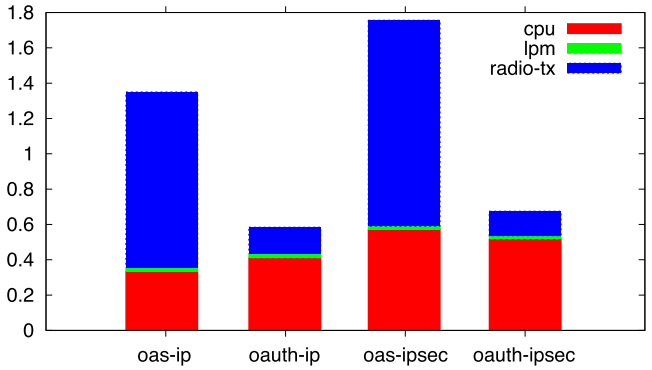

Fig. 5. Aggregate energy consumption (dimension [mJ]) for the four experimental scenarios.

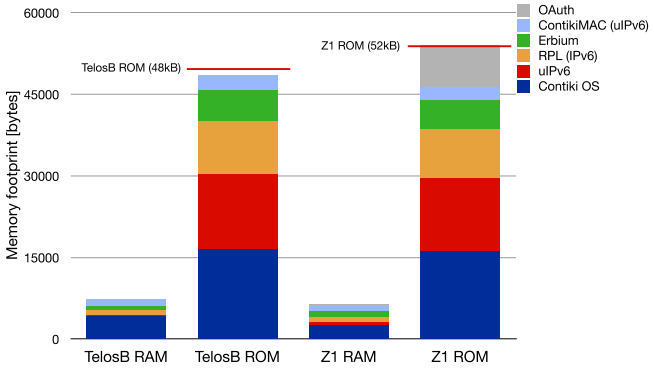

Fig. 6. Memory footprint (dimension [bytes]), with compiler optimization enabled, for different software modules in Contiki for TelosB (48 kB available) and Zolertia Z1 nodes (~52 kB available with 16-bit target compilation).

packets over IEEE 802.15.4 networks. If *IoT-OAS* is used, the number of transmitted packets doubles since the original packet must be relayed to the delegated service in order to perform the authorization checking procedure. When using IPSec, there is no gain, in terms of processing load, with respect to the local-OAuth approach. This happens because the number of decryption and encryption procedures also doubles and, since they rely on heavyweight asymmetric cryptographic primitives, also the processing load increases as well. However, the OAuth protocol specification states that, if using the HMAC-SHA1 signature scheme, a secure underlying transport is not mandatory.

C. Practical Considerations

Even though the energy consumption results presented in Subsection V-B may seem to discourage the adoption of a delegation approach for authorization grant, there are other considerations that strongly motivate the use of *IoT-OAS*.

First of all, some considerations on memory footprint should be made. In order for the OAuth software module to fit inside the ROM of a Smart Object, we had to turn off many features of the Contiki OS: i) RPL could not be used (all communication acts were single-hop); ii) Contiki MAC was replaced by NULL MAC; and iii) no radio duty cycling was active. In Fig. 6, the contributions of all the software modules to the available ROM are shown. The shown numbers are

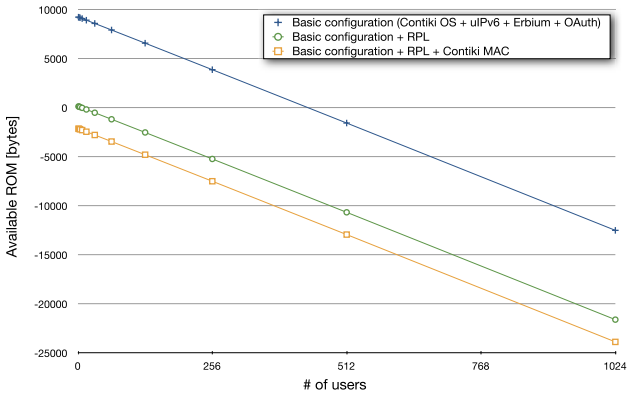

Fig. 7. Available ROM memory (dimension [bytes]) on a Zolertia Z1 (16-bit target compilation) when hosting a user database under different configurations of Contiki OS software modules.

best-case figures (e.g., no resources have been allocated for CoAP), but the Smart Objects are not operative in this case (any incoming request could not be served since no CoAP resource is hosted). Even in this best case, from a memory occupation viewpoint, from the results in Fig. 6 one can see that the amount of available ROM is not sufficient to host also the OAuth logic: therefore, some features must be excluded from the Smart Object to implement OAuth locally.

Moreover, in order to provide authorization, the Smart Object should also include a database of those clients that are authorized to perform requests for the resources managed by the constrained CoAP server, together with their access tokens. Of course, this is infeasible because of the memory shortage. On the other hand, an external authorization service might have a database large enough to fit all the information needed to manage any number of third-party clients. This is shown in Fig. 7, where one can see that the most lightweight configuration of software on a Zolertia Z1 node can at most fit a database of less than 500 users (assuming an average of roughly 20 bytes per user). From Fig. 7, it can also be observed that just integrating RPL decreases the maximum number of users stored in the database to 8.

Another important point that should be highlighted is that the delegation of the authorization procedures allows the owners of the Smart Objects to reconfigure, even with very fine-grained access policies, the authorization grants to external consumers without the need to re-program the Smart Objects, which is crucial when the Smart Objects are deployed in hardly reachable locations and their reconfiguration can be very complicated. As an external service can perform highly dynamic configuration, it could be very easy to grant or remove access permissions at any time with little, if any, need of human intervention, which can be a difficult and time-consuming operation. A mechanism that allows operators to change access policies remotely and possibly acting on multiple nodes at the same time can, therefore, be far more convenient by cutting down management costs and minimizing the time required for the re-configuration of Smart Objects.

One final remark, related to energy consumption, is the fact that the analysis in Subsection V-B is performed using the HMAC-SHA1 signature scheme. The OAuth protocol

specification considers also the use of the RSA-SHA1 signature scheme, which requires much higher processing load as it relies on asymmetric cryptographic primitives and introduces also the problem of public key management. The amount of processing load (and, thus energy consumption) considered in Fig. 5 is then underestimated.

In conclusion, the delegation approach can significantly improve and simplify the design and deployment of Smart Objects, allowing to focus on the fundamental functionalities that a constrained node should implement, rather than dealing with other aspects that could be easily outsourced with minimum impact on the development of the application. The use of *IoT-OAS* does not necessarily forbid or discourage an in-node implementation of authorization functionalities, which might be suited for smart objects with sufficient resources and capabilities. *IoT-OAS* can be applied effectively in several scenarios, with different network configurations and Smart Object capabilities. For instance, being aware of energy consumption issues, *IoT-OAS* can be used by network elements that do not present the same constraints of smart objects (e.g., LoWPAN border routers, LBRs, which can be connected to a fixed energy supply), as shown in Fig. 3(a) and Fig. 3(b). Such elements can apply access policies and filter incoming requests, in order to "shield" the constrained network, while allowing the Smart Objects to keep their implementation.

VI. Discussion: Advantages, Limitations, and Outlook

In this work, we have presented a novel general architecture for service authorization to be used in Internet, IoT, and hybrid scenarios. According to the architectural nature of this work and the experimental results presented in Section V, the following aspects need to be further discussed.

A. Security Issues

The use of Internet protocols and the very nature of Smart Objects raise security issues in the proposed *IoT-OAS* architecture. The interested reader is referred to [35] for an overview of security threats in IoT.

First, delegation to a third party service requires a strong trust on the third party. The trust relationship between a Smart Object and the *IoT-OAS* is actually a trust relationship between the owner of the Smart Object and the *IoT-OAS*. The establishment of such trust relationship falls beyond the scope of this paper, which describes the service architecture to which the Smart Objects subscribe. A strongly secure communication channel, such as a VPN tunnel, could be a suitable solution to provide secure and authenticated communications with the authorization service.

The use of standard Internet security protocols can have a negative impact on the performance of Smart Objects, as the communication overhead and computational cost might dramatically affect the energy consumption. In order to tackle these issues, the following actions may be taken:

- define constrained versions and implementations of the security protocols, in order to make the handshaking phase lighter and to minimize packet fragmentation;
- define a constrained version of the OAuth protocol, in a similar fashion to what is being carried out in the IETF CoRE Working Group with the definition of the CoAP protocol (with respect to HTTP);
- define new cryptographic suites for the security protocols by integrating lightweight cryptographic algorithms (such as TEA [36], SEA [37], and PRESENT [38]) and lightweight hash functions (such as DM-PRESENT [39] and SHA-3 [33]), which may be more suitable for constrained devices.

Other security aspects might be related to the following possible attacks that smart objects might undergo.

- Denial-of-Service (DoS) might occur if Smart Objects receive a large number of requests to serve. In this case, the use of the *IoT-OAS* would decrease the ability of the Smart Object to resist against this type of attacks. A solution could be to use the gateway to protect the Smart Object, possibly by offering caching capabilities.
- Man-in-the-Middle (MITM) attacks are possible if using a HTTP/CoAP proxy — which, by its nature, is a man-in-the-middle — thus destroying any form of end-to-end security. If the proxy is not trusted, it could access all communications among the endpoints and could possibly get access to the authorization information, thus gaining permission to spoof the requestor's identity.
- Physical threats are related to the impossibility to supervision constantly Smart Objects once they have been deployed in public/remote areas.

The considerations above are not strictly related to the *IoT-OAS* architecture, but are typically related to all IoT scenarios and are currently being investigated. However, even though they fall outside the scope of this paper, it is important to cite them as open issues. It is also important to remark that the *IoT-OAS* architecture is not a solution for security aspects in IoT, but, rather, is meant to provide an authorization layer which is easy to integrate and manage for Internet and IoT services.

B. Computational and Storage Overhead

The OAuth protocol has been designed to handle computation and storage overhead while providing a standard and simple authorization mechanism for resource access by third-party applications. The same considerations can be applied to our approach, as there is no computation and storage on the node, but all the information resides on the central authorization service. Typical examples of the scalability of OAuth-based services are online social networks, such as Twitter and Facebook, which deal with hundreds of millions of users, billions of requests, and several thousands of third-party applications which access the hosted resources. The above considerations are confirmed by the performance evaluation of the *IoT-OAS* architecture in Section V: i) the memory footprint of all software modules that should be run on the Smart Object leaves no space for the storage of user identities and access policies (as shown in Fig.7); ii) from a processing perspective, the Smart Object does benefit from a delegation approach,

as there is no computation to be performed besides sending a request to the *IoT-OAS*.

VII. Conclusion

In this paper, we have proposed a novel architecture to provide HTTP and CoAP service providers with an authorization layer to be able to disseminate their services without the need of implementing the OAuth logic, but, rather, by invoking an external OAuth-based authorization service, denoted as "*IoT-OAS*." The designed approach has been applied to significant IoT scenarios with multiple Smart Objects (or, more generally, constrained devices) characterized by limited computational power, operating in lossy and low-power networks, and usually battery-powered thus requiring extreme attention on energy consumption.

A performance evaluation has been performed by conducting simulations with Cooja targeting Contiki-based Zolertia Z1 nodes. The experimentation has shown that, from a purely energy consumption perspective, the delegation approach can increase the amount of energy consumed, due to the fragmentation of application-layer messages performed in order to fit in IEEE 802.15.4 packets. However, other issues, such as memory footprint and dynamic configuration capabilities, show that implementing the OAuth logic locally is infeasible with currently available Smart Objects, making the delegation approach provided by *IoT-OAS* preferable. Moreover, delegating the authorization logic to an external service leads to several additional benefits, such as: i) reducing the time required by service developers to implement OAuth-protected online services; ii) supporting legacy OAuth client applications seamlessly; iii) limiting the device complexity only to service logic, while still providing access control policies for its services. An extremely appealing advantage of externalizing the authorization logic is the possibility to dynamically and remotely configure fine-grained access control policies on a per-service and per-client basis, without the need of direct intervention on the deployed devices.

Security considerations have been also discussed, taking into account well-known IoT-related issues. As a future research direction, within the work of the FP7 EU project CALIPSO, the proposed architecture will be implemented and tested thoroughly in both simulation environments and real testbeds, in order to evaluate advanced performance metrics in constrained environments and in the presence of duty-cycle.

Acknowledgment

The work reflects only the authors views; the European Community is not liable for any use that may be made of the information contained herein.

References

[1] R. Fielding *et al.*, *Hypertext Transfer Protocol—HTTP/1.1*, RFC 2616, Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[2] E. Hammer-Lahav, *The OAuth 1.0 Protocol*, RFC 5849, Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5849.txt

[3] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, Internet Engineering Task Force, Aug. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[4] D. Hardt, *The OAuth 2.0 Authorization Framework*, RFC 6749, Internet Engineering Task Force, Oct. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6749.txt

[5] *IPSO Alliance*. [Online]. Available: http://www.ipso-alliance.org/, accessed Oct. 15, 2014.

[6] *Connect All IP-Based Smart Objects (CALIPSO)—FP7 EU Project*. [Online]. Available: http://www.ict-calipso.eu/, accessed Oct. 15, 2014.

[7] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252, Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7252.txt

[8] Worldsensing, Barcelona, Spain. [Online]. Available: http://www.worldsensing.com/

[9] S. Cirani, G. Ferrari, and L. Veltri, "Enforcing security mechanisms in the IP-based internet of things: An algorithmic overview," *Algorithms*, vol. 6, no. 2, pp. 197–226, 2013. [Online]. Available: http://www.mdpi.com/1999-4893/6/2/197

[10] H. Ning, H. Liu, and L. T. Yang, "Cyberentity security in the internet of things," *Computer*, vol. 46, no. 4, pp. 46–53, Apr. 2013.

[11] X. Yao, X. Han, X. Du, and X. Zhou, "A lightweight multicast authentication mechanism for small scale IoT applications," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3693–3701, Oct. 2013.

[12] C. Lai, H. Li, X. Liang, R. Lu, K. Zhang, and X. Shen, "CPAL: A conditional privacy-preserving authentication with access linkability for roaming service," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 46–57, Feb. 2014.

[13] F. Li and P. Xiong, "Practical secure communication for integrating wireless sensor networks into the internet of things," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3677–3684, Oct. 2013.

[14] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin, *Protocol for Carrying Authentication for Network Access (PANA)*, RFC 5191, Internet Engineering Task Force, May 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5191.txt

[15] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, *Extensible Authentication Protocol (EAP)*, RFC 3748, Internet Engineering Task Force, Jun. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3748.txt

[16] P. Moreno-Sanchez, R. Marin-Lopez, and F. Vidal-Meca, "An open source implementation of the protocol for carrying authentication for network access: OpenPANA," *IEEE Netw.*, vol. 28, no. 2, pp. 49–55, Mar. 2014.

[17] United States Department of Defense, "Department of Defense Trusted Computer System Evaluation Criteria," U.S. Dept. Defense, Tech. Rep. DoD 5200.28-STD, Dec. 1985. [Online]. Available: http://csrc.nist.gov/publications/history/dod85.pdf

[18] D. Ferraiolo and R. Kuhn, "Role-based access controls," in *Proc. 15th NIST-NCSC Nat. Comput. Security Conf.*, Baltimore, MD, USA, Oct. 1992, pp. 554–563.

[19] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, Aug. 2001. [Online]. Available: http://doi.acm.org/10.1145/501978.501980

[20] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[21] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, vol. 856, Jul. 2005, p. 2.

[22] J. Schiffman, X. Zhang, and S. Gibbs, "DAuth: Fine-grained authorization delegation for distributed web application consumers," in *Proc. IEEE Int. Symp. Policies Distrib. Syst. Netw. (POLICY)*, Jul. 2010, pp. 95–102.

[23] O. Garcia-Morchon and K. Wehrle, "Modular context-aware access control for medical sensor networks," in *Proc. 15th ACM Symp. Access Control Models Technol. (SACMAT)*, New York, NY, USA, 2010, pp. 129–138. [Online]. Available: http://doi.acm.org/10.1145/1809842.1809864

[24] S. Gerdes, O. Bergmann, and C. Bormann, "Delegated CoAP authentication and authorization framework (DCAF)," IETF Internet Draft, Tech. Rep. draft-gerdes-ace-dcaf-authorize-00, Jul. 2014. [Online]. Available: http://tools.ietf.org/html/draft-gerdes-ace-dcaf-authorize-00

[25] "OpenID authentication 2.0—Final," OpenID Foundation, Tech. Rep., Dec. 2007. [Online]. Available: http://openid.net/specs/openid-authentication-2_0.html

[26] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, RFC 6347 (Proposed Standard), Internet Engineering Task Force, Jan. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6347.txt
[27] S. Kent and R. Atkinson, *Security Architecture for the Internet Protocol*, RFC 2401 (Proposed Standard), Internet Engineering Task Force, Nov. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2401.txt
[28] *The Contiki Operating System*. [Online]. Available: http://www.contiki-os.org, accessed Oct. 15, 2014.
[29] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing communication in 6LoWPAN with compressed IPsec," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Barcelona, Spain, Jun. 2011, pp. 1–8.
[30] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN compressed DTLS for CoAP," in *Proc. 8th IEEE Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Hangzhou, China, May 2012, pp. 287–289.
[31] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for Contiki," in *Proc. Workshop Internet Things Technol. Architect. (IoTech)*, Valencia, Spain, Oct. 2011, pp. 855–860.
[32] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104 (Informational), Internet Engineering Task Force, Feb. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt
[33] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The Keccak SHA-3 submission," in *Submission to NIST (Round 3)*. National Institute of Standards and Technology (NIST), 2011. [Online]. Available: http://keccak.noekeon.org/Keccak-submission-3.pdf
[34] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level power profiling for low-power wireless networks," Swedish Institute. Comput. Sci., Kista, Sweden, Tech. Rep. T2011:05, Mar. 2011. [Online]. Available: http://soda.swedish-ict.se/4112/1/T2011_05.pdf
[35] O. Garcia-Morchon, S. Keoh, S. Kumar, R. Hummen, and R. Struik, "Security considerations in the IP-based internet of things," IETF Internet Draft, Tech. Rep., Mar. 2012. [Online]. Available: http://tools.ietf.org/id/draft-garcia-core-security-04
[36] D. Wheeler and R. Needham, "TEA, a tiny encryption algorithm," in *Fast Software Encryption* (Lecture Notes in Computer Science), vol. 1008, B. Preneel, Ed. Berlin, Germany: Springer-Verlag, 1995, pp. 363–366. [Online]. Available: http://dx.doi.org/10.1007/3-540-60590-8_29
[37] F.-X. Standaert, G. Piret, N. Gershenfeld, and J.-J. Quisquater, "SEA: A scalable encryption algorithm for small embedded applications," in *Smart Card Research and Advanced Applications* (Lecture Notes in Computer Science), vol. 3928, J. Domingo-Ferrer, J. Posegga, and D. Schreckling, Eds. Berlin, Germany: Springer-Verlag, 2006, pp. 222–236. [Online]. Available: http://dx.doi.org/10.1007/11733447_16
[38] A. Bogdanov *et al.*, "PRESENT: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), vol. 4727, P. Paillier and I. Verbauwhede, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 450–466. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74735-2_31
[39] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, and Y. Seurin, "Hash functions and RFID tags: Mind the gap," in *Proc. 10th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2008, pp. 283–299. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85053-3_18

Simone Cirani is currently a Post-Doctoral Research Associate with the Department of Information Engineering, University of Parma, Parma, Italy, where he received the Dr.Ing. (Laurea) (*cum laude*) degree in computer science, and the Ph.D. degree in information technologies from the Department of Information Engineering, in 2007 and 2011, respectively. His research interests are Internet of Things, peer-to-peer networks, network security, pervasive computing, and mobile application development.

Marco Picone is currently a Post-Doctoral Research Associate with the University of Parma, Parma, Italy, where he received the Laurea (*cum laude*) degree in computer engineering and the Ph.D. degree in information technologies in 2008 and 2012, respectively. In 2011, he was a Research Visitor with the Computer Laboratory, University of Cambridge, Cambridge, U.K. His research activity focuses on mobile and pervasive computing, location-based services, distributed systems, and Internet of Things.

Pietro Gonizzi received the master's diploma in telecommunications engineering from the University of Parma, Parma, Italy, in 2011, with a thesis entitled "Low-Complexity Redundant Distributed Data Storage in Wireless Sensor Networks: Design and Experimental Validation." He is currently pursuing the Ph.D. degree at the Wireless Ad-Hoc Sensor Network Laboratory, University of Parma. His research interests are efficient routing and low-power MAC techniques in wireless sensor networks and Internet of Things.

Luca Veltri received the Laurea degree in telecommunication engineering, and the Ph.D. degree in communication and computer science from the University of Rome La Sapienza, Rome, Italy, in 1994 and 1999, respectively. Since 2002, he has been an Assistant Professor with the University of Parma, Parma, Italy, where he currently teaches classes on telecommunication networks and network security. His main research fields are peer-to-peer systems, future Internet, and network security.

Gianluigi Ferrari (SM'12) is currently an Associate Professor of Telecommunications with the University of Parma, Parma, Italy, where he coordinates the Wireless Ad-Hoc and Sensor Networks Laboratory of the Department of Information Engineering. As of today, he has published extensively in the areas of wireless ad hoc and sensor networking, adaptive digital signal processing, and communication theory. He was a recipient of Paper/Technical Awards at IWWAN'06, EMERGING'10, BSN'11, ITST'11, SENSORNETS'12, Evo-COMNET'13, and BSN'14. He currently serves on the Editorial Boards of several international journals.