OAuth-IoT: an access control framework for the Internet of Things based on open standards

Savio Sciancalepore[1,2], Giuseppe Piro[1,2], Daniele Caldarola[1,2], Gennaro Boggia[1,2], and Giuseppe Bianchi[2,3]
[1] Department of Electrical and Information Engineering (DEI), Politecnico di Bari, Bari, Italy; e-mail: {name.surname}@poliba.it.
[2] CNIT, Consorzio Nazionale Interuniversitario per le Telecomunicazioni
[3] Department of Electronic Engineering, University of Rome Tor Vergata, Rome, Italy; e-mail: giuseppe.bianchi@uniroma2.it

***Abstract*—While the Internet of Things is breaking into the market, the controlled access to constrained resources still remains a blocking concern. Unfortunately, conventional solutions already accepted for both web and cloud applications cannot be directly used in this context. In fact, they generally require high computational and bandwidth capabilities (that are impossible to reach with constrained devices) and offer poor interoperability against standardized communication protocols for the Internet of Things. To solve this issue, this contribution presents a flexible authentication and authorization framework for the Internet of Things, namely *OAuth-IoT*. It leverages and properly harmonizes existing open-standards (including the OAuth 2.0 authorization framework, different token formats, and the protocol suite for the Internet of Things tailored by the Internet Engineering Task Force), while carefully taking into account the limited capabilities of constrained devices. Functionalities and benefits offered by OAuth-IoT are pragmatically shown by means of an experimental testbed, and further demonstrated with a very preliminary performance assessment.**

I. Introduction

The Internet of Things (IoT) is emerging as a concrete communication paradigm allowing smart objects to realize a capillary networking infrastructure connected to the Internet. With the widespread diffusion of advanced IoT applications and services, the definition of proper secure mechanisms is gaining momentum in the research and industrial communities [1]. Nevertheless, while many *turnkey* solutions already exist for both data confidentiality and key agreement [2][3], the discussion on authentication and authorization services is still open to novel and effective approaches.

The present contribution focuses on a scenario where an IoT network exposes restricted resources outside, third-party applications may be interested to reach these resources, and access requests must be authorized by resource owners.

In the current Internet, this problem was smartly solved by OAuth 2.0 [4]. The adoption of OAuth 2.0 in the IoT was preliminary investigated in [5], and a concrete proposal was discussed in [6]. In particular, [6] introduced a new centralized entity (namely IoT-OAS) that is in charge of validating access requests sent by a third-party application, on behalf of the server that exposes IoT resources. Thus, this additional component makes the proposal not completely standard compliant and reaches the integration of OAuth 2.0 and accepted communication protocols for the IoT by means of a customized scheme. Additional lightweight approaches to solve authentication and authorization in the IoT context are available in the literature, like [9]. More recently, Internet Engineering Task Force (IETF) Authentication and Authorization for Constrained Environments (ACE) working group is promoting the adoption of the OAuth 2.0 approach in scenarios where applications and resources are available on constrained devices, both connected to the same IoT domain [7][8]. All of these solutions, however, do no match the scenario considered in this contribution.

With reference to the target scenario, the design of an access control mechanism should consider that: (i) OAuth 2.0 cannot be directly used in the IoT context, because of its expensive computational and bandwidth requirements; (ii) the communication protocol for the IoT proposed by the IETF does not natively offer such kind of mechanisms; and (iii) no solutions addressing the interoperability between OAuth 2.0 and IETF protocol stack were already presented in the literature.

In line with these premises, this paper presents the OAuth-IoT framework, which offers resource access control mechanisms for IoT resources, by leveraging and properly harmonizing existing open-standards, while taking care of the limited capabilities of constrained devices. In OAuth-IoT, IoT resources are discovered and exposed through well-known mechanisms standardized by the IETF [10]. A gateway device stores details on available resources, provides an interface between the IoT network and the Internet, verifies authorization permissions of third-party applications interested to access the exposed resources, and implements a caching mechanism for efficiently serving multiple requests with limited freshness requirements. OAuth-IoT also integrates an Authorization Server that authenticates the resource owner and authorizes a third-party application to access to one of the aforementioned IoT resources. While the entire process is completely based on the OAuth 2.0 specification, authorization grants may be handled with a variety of tokens formats, as bearer, JSON Web Token (JWT) and Proof-of-Possession (PoP) tokens. In order to show the effectiveness and benefits of the proposed framework, also an experimental testbed has been developed and made publicly available at *http://telematics.poliba.it/oauth-iot*.

978-1-5386-1629-1/17/$31.00 ©2017 IEEE

The rest of the paper is structured as follows. Reference open standards are discussed in Section II. Section III describes components and functionalities of the proposed OAuth-IoT framework. Section IV highlights its features through real experiments. Finally, Section V draws conclusions and future works.

II. Reference open standards

A. The IETF protocol stack for the IoT

The IETF dedicated many efforts to the definition of standard protocols for IoT networks based on the IEEE 802.15.4 radio [11]. The resulting communication stack integrates many protocols, including Constrained Application Protocol (CoAP), Routing Protocol for Low Power and Lossy Networks (RPL), IPv6 over Low-power and Lossy Networks (6LoWPAN), and 6top.

The discovery of IoT resources was standardized in [10]. The procedure is initiated by the coordinator of an IoT system (e.g., the *sink node*), every time it recognizes that a new device joins the network. In this case, it sends to the aforementioned device a GET message to the well-known resource Uniform Resource Identifier (URI) $/.well-known/core$. Then, the device answers with a message containing information related to the resources it exposes, expressed according to the *CoRE Link Format* [10]. Reported information include 'resource type' (i.e., a string that semantically describes the resource, through an ontology), 'interface description' (i.e., the URI associated to the resource), and 'maximum size estimate' (i.e., the maximum size of the resource representation that can be returned to the GET request). They are stored within the so called *Resource Directory*.

Note that the *Resource Directory* provides a complete overview of resources exposed by an IoT network: any external user can retrieve these details, learn about the URI associated to the resources, and generate requests. Thus, the IETF protocol stack for the IoT does not provide any explicit access control mechanism.

B. OAuth 2.0

OAuth 2.0 is an authorization framework standardized in [4]. It was designed in order to allow a third-party application to access to restricted resources under the control of a user, without requiring the sharing of user's credentials. The architecture embraces four actors, that are a third-party application willing to access to protected resources (i.e, the *client*), the *Resource Owner* able to grant or deny access to a protected resource, the *Resource Server* that exposes protected resources, and the *Authorization Server* that controls the authorization process. These actors establishes an end-to-end Transport Layer Security (TLS) channel and interacts each other during the resource access procedure, as summarized in what follows: (1) the client contacts the Resource Owner of the resource; (2) the Resource Owner grants the access to the client by sending an *authorization code*; (3) the client delivers the received authorization code to the Authorization Server; (4) the Authorization Server verifies the authorization code and releases a token containing the details of the consent provided to the client (time limit, scope, and so on); (5) the client forwards the token to the Resource Server; (6) the Resource Server checks the validity of the received token and, in affirmative case, it provides the protected resource. The definition of the token structure is out of scope of the standard.

OAuth 2.0 cannot be directly used to control the access to resources available within an IoT network. Constrained devices, in fact, do not have enough computational and bandwidth capabilities to support the establishment of a TLS connection, to implement all the tasks envisaged for the Resource Server, as well as to exchange heavy messages (that also include tokens) within acceptable latencies.

In line with these premises, the IETF ACE working group, is promoting the adoption of a lightweight implementation of OAuth 2.0 in scenarios where both applications and resources are available within the same IoT network, evidently installed on constrained devices. In particular, OAuth 2.0 interactions are enabled by means of CoAP messages, carried through a Datagram Transport Layer Security (DTLS) channel [7]. Moreover, new token formats, more suitable for low-power short-range and lossy wireless links, are defined in [8].

These solutions, however, cannot be successfully applied in the scenario considered in this contribution, where the user interested to reach constrained resources is located outside the IoT network and would like to interact with OAuth 2.0 and IoT entities by using conventional communication protocols for the Internet.

C. IETF token formats

Token format is out of scope of OAuth 2.0, but some interesting proposals comes from the IETF. For instance, *Bearer Tokens* are defined in [12] as simple containers of information. The majority of OAuth 2.0 implementations adopts this solution, even if no embedded security mechanisms are provided. In this case, token confidentiality is delegated to the mandatory TLS protocol. The *JSON Web Token (JWT)* is a compact way for carrying the access grants [13]. It leverages the standard JSON format for storing and assess common claims (including time validity, issuer, owner and revocation time) and authentication fields. There is also the possibility to extend the token format with unregistered private claims, useful for specific purposes. Finally, the *Proof-of-Possession (PoP) tokens* have been defined in [14] and can be used in scenarios demanding additional security protection. In fact, the main idea behind PoP tokens is that a client must demonstrate possession of cryptographic keying material when making requests for accessing protected resources. Both symmetric and asymmetric cryptography can be used in this case.

Given the presence of different solutions under investigation, a flexible access control framework for the IoT should be designed in order to support all the possibilities emerging from the literature and standardization areas.

III. The proposed OAuth-IoT framework

The developed OAuth-IoT framework offers access control mechanisms for the IoT, by properly leveraging and harmonizing the widely used open standards described in the previous Section. The reference architecture depicted in Fig. 1 is considered. It embraces the following components:

1. Internet of Things network. It integrates many constrained devices able to sense the surrounding environment, acquire data (such as temperature, humidity, luminosity, and acceleration) and deliver them to a sink node, also called *network coordinator*, through a low-power and short-range wireless communication technology. The sink node is attached to the *Gateway*, that acts as a Resource Server and offers many functionalities listed below.
2. Client. In line with the OAuth 2.0 authorization framework, it is a third-party application willing to reach resources belonging to an IoT network, on behalf of the resource owner. It would access remote resources through OAuth 2.0 primitives.
3. Gateway. It is a key node of the proposed architecture, that implements the OAuth 2.0 Resource Server and an interface between OAuth 2.0 and the IETF protocol stack. Indeed, it offers security functionalities (e.g., establishment of TLS channel with the client, authentication, and access control), the tracking of available resources (through the resource discovery procedure), and other system functionalities (e.g., data caching and freshness controls).
4. Authorization Server. It manages authorization mechanisms, as detailed by the OAuth 2.0 authorization framework.

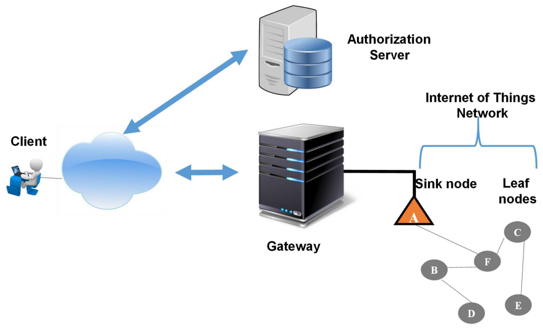


Fig. 1. Reference architecture.

A. Resource management

The gateway represents the key point of contact between OAuth 2.0 and the IoT ecosystem. In fact, it is in charge of storing information related to available resources, as well as exposing them to the rest of the architecture through OAuth 2.0 authorization primitives. To reach this goal, it hosts three different data structures, that are *Routing Table*, *Resource Directory*, and *Data Table*. They are populated thanks to the continuous interaction between the sink node of the IoT network and the Resource Server.

The *Routing Table* contains the identifier of all nodes currently active in the IoT network, along with information about the path that is used to communicate with each of them. According to the RPL protocol, the sink node periodically sends Destination Oriented Directed Acyclic Graph (DODAG) Information Object (DIO) messages within the IoT network, for discovering constrained devices. When a new node joins the network, it answers with a Destination Advertisement Object (DAO) message. In OAuth-IoT, the sink node delivers information stored within the aforementioned DAO message to the gateway, which will update the *Routing Table* accordingly.

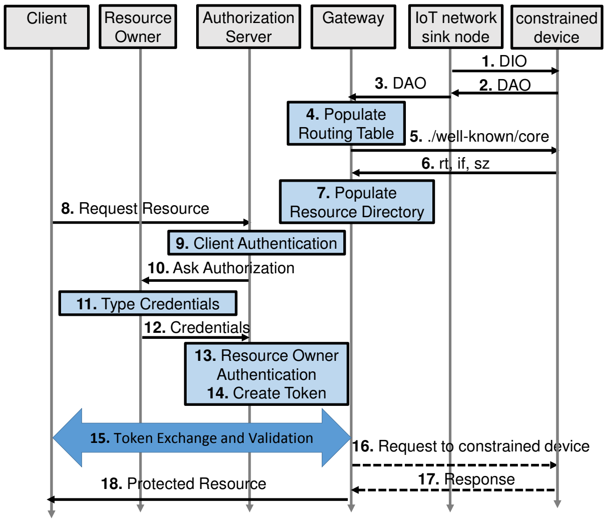


Fig. 2. Resource access procedure implemented in OAuth-IoT.

Once a new record is added to the *Routing Table*, the gateway initiates the resource discovery procedure by polling the identified constrained device at the path *./well-known/core*. The corresponding answer is properly processed for populating the *Resource Directory*.

When a new data is acquired from the IoT network, the gateway updates the *Data Table*. Specifically, each record of that table stores identification data about the resource (i.e., the Resource Type attribute previously received with the Core Link Format message), the IPv6 address of the node exposing the resource, the timestamp (i.e., the time instant at which the data has been generated), and the value of the data itself. In other words, the *Data Table* acts as a cache and can be used to generate answers to external requests, thus limiting the overall energy consumption within the IoT network.

B. Resource access procedure

Fig. 2 shows the sequence diagram related to the resource access procedure implemented in OAuth-IoT.

At the top-right of the figure, it is possible to observe the interaction between the gateway and the IoT network, needed to populate the *Routing Table* and the *Resource Directory*, already explained in the previous sub-section.

On the left-side of the figure there is the client, that would like to access to a resource available in the IoT network. To this end, it starts the authorization procedure based on the OAuth 2.0 work flow. At the beginning, it sends a request to the Authorization Server that, in turns, asks the authorization to the resource owner. The resource owner (i.e., the user that is using the application) types username and password, authorizing the access. Then, the Authorization Server generates the token and forwards it to the client. Then, the gateway processes the token (see Section III-C for more details) and generates the answer (according to the procedure described in Section III-D).

It is worth noting that the gateway can quickly response to the application if the requested resource is already in the database and meets the conditions requested by the client (i.e., freshness). If this is not the case, the HTTP request is mapped to a CoAP request and forwarded to the node that provides it. For this reason, the messages 16 and 17 in Fig. 2 are optional and they have been illustrated through dotted lines.

C. Details on tokens

OAuth-IoT natively supports any token format. To make concrete examples, we explain in more details the adoption of bearer, JWTs, and PoP tokens.

With bearer tokens, no cryptography technique is used to protect the content of the token. Thus, as shown in Fig. 3(a), the gateway processes immediately the token at the reception time, without any cryptography validation.

If JWTs are used, the token contains a *sign* field, used to validate the content. It can be generated both by using symmetric or asymmetric cryptography techniques. The asymmetric cryptography case is considered herein. Let PV_{as} and PB_{as} be the private and public keys of the Authorization Server, respectively. The Authorization Server signs the token T as in Eq. 1:

$$S_T = E(H(T), PV_{as}), \tag{1}$$

where $H(\cdot)$ is a generic hashing function and $E(\cdot)$ is a generic public-key encryption algorithm. As shown in Fig. 3(b), the client forwards the token to the gateway, which validates the sign by checking that $H(T) = E(S_T, PB_{as})$.

In the case of PoP tokens, a cryptographic value is delivered by the Authorization Server to the client and then used by the client to create a proof that demonstrates the possession of the secret. The Authorization Server also shares that secret with the gateway by means of different approaches (including *token introspection* or shared database) [14]. The symmetric PoP scheme is investigated in the sequel. As shown in Fig. 3(c), the Authorization Server generates a secret k every time a new client requests a token. Then, the token and the secret are delivered to the client. The token stores the confirmation claim: $CNF = H(K)$. The client must demonstrate that it is the real owner of the token. To this end, it uses the received secret for calculating a Message Authentication Code (MAC), M_T, of the token itself:

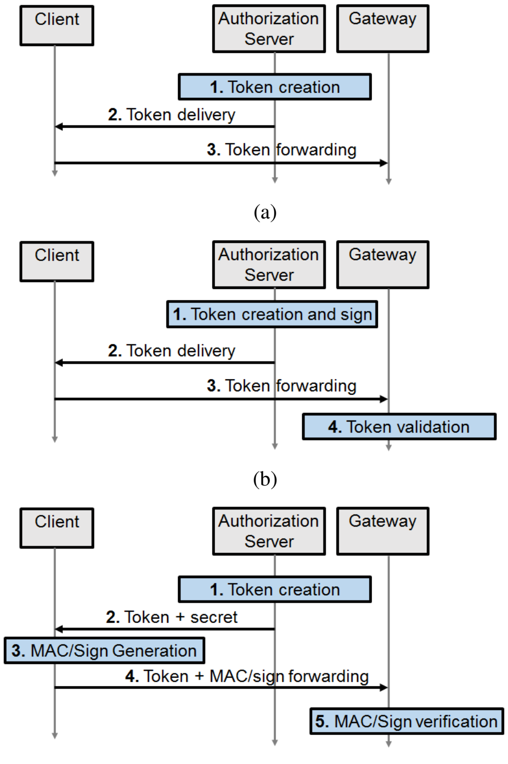
$$M_T = E(H(T), k), \tag{2}$$

Then, the client delivers to the gateway the token and the calculated MAC. To validate the token, the gateway retrieves the secret (by using one of the aforementioned approaches, as suggested in [14]) and verifies that both the confirmation claim stored in the token and the MAC sent by the client are correct.

D. Gateway's operations

The set of operations executed by the gateway are illustrated in Fig. 4.

When the gateway receives a new request, it extracts the token, the identifier of the constrained node exposing the resource, the requested resource, and a data freshness parameter. The token is firstly validated. If the token is not valid, the gateway returns the well-known *403 forbidden* error code and the request is rejected. Otherwise, the gateway checks the availability of the requested resource in the *Resource Directory*. In the case the resource is not available, the gateway returns the well-known *404 not found* error code and closes the session. If the resource is available, the gateway checks the presence of the requested data within the *Data Table*. If a record is found and the associated timestamp satisfies the freshness requirement reported in the request, the gateway immediately delivers to the client the cached value (i.e., without contacting the constrained node in the IoT network that exposes the requested resource). If not, the gateway generates a GET request using CoAP and sends it into the IoT network. Now, in the case the node is not reachable, the *503 service unavailable* error code is generated and the session is closed. On the contrary, the contacted constrained node answers with the requested data that will be processed and stored within the *Data Table* of the gateway (first) and pushed back to the client (then).



(a)

(b)

(c)

Fig. 3. Token exchange and validation, when (a) bearer, (b) JWT, and (c) PoP tokens are used.

IV. Testbed at work

An experimental testbed has been developed to practically demonstrate the main functionalities of the conceived OAuth-IoT framework. It is available at: **http://telematics.poliba.it/oauth-iot**.

The IoT network hosts three OpenMote[1] constrained devices

[1] The OpenMote-CC2538 is equipped with a 32-bit Cortex-M3 microcontroller, 32 kB of RAM memory, and 512 kB of Flash. It can be connected to the OpenBattery containing four different sensors, i.e. temperature, humidity, light, and acceleration.

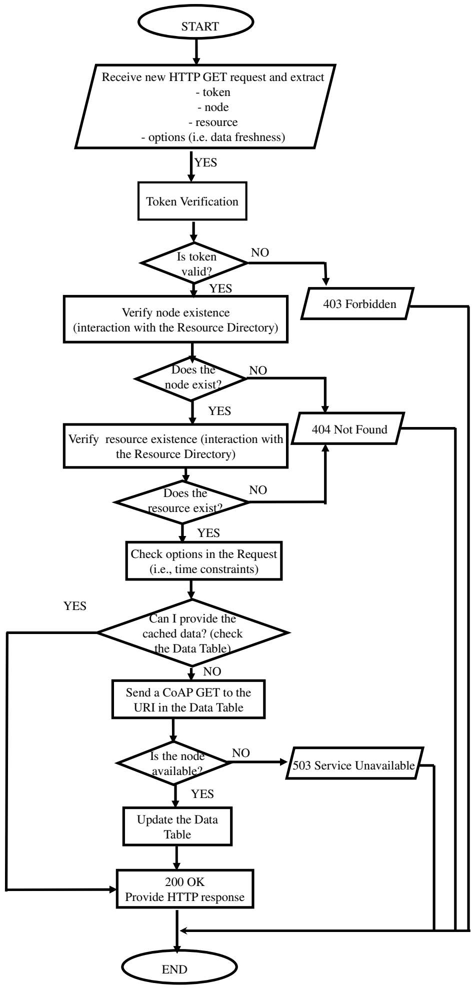

START
Receive new HTTP GET request and extract
- token
- node
- resource
- options (i.e. data freshness)
YES
Token Verification
Is token valid?
NO
YES
403 Forbidden
Verify node existence (interaction with the Resource Directory)
Does the node exist?
NO
YES
404 Not Found
Verify resource existence (interaction with the Resource Directory)
Does the resource exist?
NO
YES
Check options in the Request (i.e., time constraints)
YES
Can I provide the cached data? (check the Data Table)
NO
Send a CoAP GET to the URI in the Data Table
Is the node available?
NO
503 Service Unavailable
YES
Update the Data Table
200 OK
Provide HTTP response
END

Fig. 4. Flow diagram of the operations executed by the gateway when a new HTTP request arrives from the Internet.

[15]: one sink node and two leaf nodes exposing resources (i.e., temperature, humidity, light, and acceleration). They run the OpenWSN stack [16], which is an open source implementation of the IETF protocol suite described in Section II.

Both Authorization Server and gateway components are implemented on a Pandaboard ES platform. It is an embedded computer, integrating dual-core 1.2GHz ARM Cortex-A9 MPCore CPU, 384 MHz PowerVr SGX540 GPU, IVA3 multimedia hardware accelerator with a programmable DSP, 1GiB of DDR2 SDRAM, as well as SD Card slot, 10/100 Ethernet, Wi-Fi, Bluetooth interfaces, output video signal via DVI and HDMI interfaces, and two USB ports. Authorization Server and Resource Server (RS) processes are implemented by using the open-source python framework Djangoand the open-source database PostgreSQL. Without loss of generality, the testbed includes the symmetric PoP architecture to create, manage and validate tokens. The Advanced Encryption Standard (AES) encryption algorithm and the Secure Hash Algorithm (SHA) hashing algorithm, used for the management of PoP tokens, are implemented through the open-source PyCrypto library. Django interacts with PostgreSQL to verify if requested resources are available and if a valid copy is already cached. In the case an interaction with the IoT network is needed, Django and OpenWSN interact each other to generate CoAP requests and handle responses from constrained devices.

Resources can be requested by using a web-based application running over TLS, directly connected with a web server running on the gateway. It has been realized by using the open-source library JQueryand it is able to manage multiple AJAX calls.

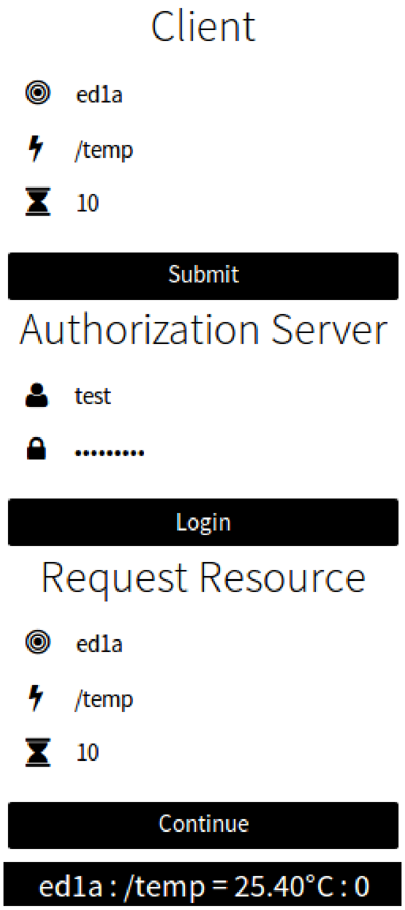
Fig. 5 shows the GUI of the OAuth-IoT testbed with the three tasks controlled by the client. The client can select the constrained node (i.e., **ed1a** and **ed8a**), the resource (i.e., **/light** for the brightness, **/temp** for the temperature, **/humid** for the humidity, and **/accel** for the acceleration), and the desired freshness (i.e., 10 seconds). The request is then processed by the Authorization Server, that initiates the authentication procedure. A recognized user has **username = test** and **password = oauth-iot**. Finally, the client can proceed to effectively send the request to the gateway. The answer is reported at the bottom of the web-page.

To provide a further insight, a preliminary performance analysis was carried out for evaluating the impact of the OAuth-IoT work flow on the amount of time needed to successfully reach the requested resource. Two freshness requirements were considered: hard and limited. In the first case, the client selects a very little freshness value and its requests are always forwarded to the IoT resource. In the second case, instead, the client selects an higher freshness value, thus allowing the gateway to answer with data already available in the cache. In addition, the comparison with respect to the simple scenario that does not consider any access control mechanism (i.e., the client directly contacts the gateway) is provided. Fig. 6 shows the Empirical Cumulative Distribution Function (ECDF) of the measured latencies, achieved by considering 10 different measurements of the times necessary to perform all computations and communications tasks, while neglecting times necessary to type input parameters. Reported results clearly demonstrate that security functionalities introduced by OAuth-IoT increase the amount of time needed to successfully reach the requested resource. In particular, the token management process introduces an average increment of latencies equal to 1.1s. However, in the case the client selects a limited freshness value, the caching of data brings to a reduction of the latencies up to 85%.

As a final consideration, it is important to note that the overall system response time is strictly influenced by the configuration of the IoT network. In the developed testbed, the IoT network uses a simplified TSCH schedule made up of three slots lasting $10ms$, as described in [17]. Different choices, that may depend on the network load and the service to be provided, can modify these performances indexes.

V. Conclusions and future works

This paper presents the OAuth-IoT framework. It provides access control functionalities for resources exposed by an IoT domain, by leveraging and properly harmonizing existing and widely accepted open-standards. The key element of the OAuth-IoT framework is the Gateway, which handles the following functionalities: (i) it collects information produced



Client

◎ ed1a

ϟ /temp

⧗ 10

Submit

Authorization Server

👤 test

🔒 ••••••••

Login

Request Resource

◎ ed1a

ϟ /temp

⧗ 10

Continue

ed1a : /temp = 25.40°C : 0

Fig. 5. GUI of the OAuth-IoT testbed.

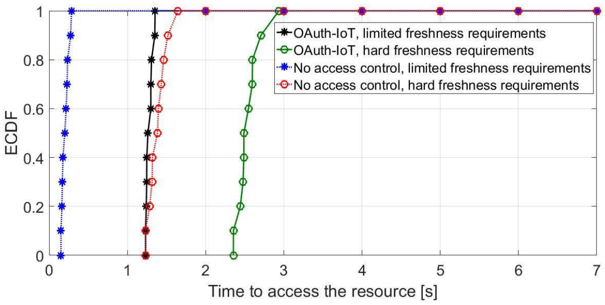


Fig. 6. Measured latencies.

by constrained devices through lightweight protocols recently standardized in the IETF context, (ii) it controls access requests originated by third-party applications through the well-known OAuth 2.0 authorization framework, (iii) it supports a variety of token formats, for properly handling applications' authentication and authorization, and (iv) it caches retrieved data for opportunistically serving future requests with limited freshness requirements. Components and functionalities of OAuth-IoT have been carefully described, and an experimental testbed have been developed. Moreover, a preliminary performance assessment has been carried out for demonstrating how OAuth-IoT is able to integrate the OAuth 2.0 authorization framework, with token formats advised by IETF, to manage access control in the concrete IoT scenarios, without any burden on constrained devices. Future directions of our research include the evaluation of different scenarios in which one or more owners of the resource are not online or note of them can be not identifiable with the client, and the integration of advanced access control schemes, such as the Attribute-Based Access Control paradigm. More complex scenarios, involving the interoperability between multiple IoT platforms, will be taken into account, too.

Acknowledgements

This work was framed in the context of the project SymbIoTe, which receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 688156.

References

[1] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1294–1312, 2015.

[2] "IEEE Recommended Practice for Transport of Key Management Protocol (KMP) Datagrams," *IEEE Std 802.15.9-2016*, Aug 2016.

[3] S. L. Keoh, S. Kumar, and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," *IEEE IoT J.*, vol. 1, no. 3, 2014.

[4] D. Hardt, "The OAuth 2.0 Authorization Framework," IETF, RFC 6749, Oct. 2012.

[5] S. Emerson, Y. K. Choi, D. Y. Hwang, K. S. Kim, and K. H. Kim, "An OAuth based authentication mechanism for IoT networks," in *Int. Conf. on Inform. and Commun. Techn. Convergence (ICTC)*, Oct. 2015, pp. 1072–1074.

[6] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari, "IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios," *IEEE Sensors J.*, vol. 15, no. 2, pp. 1224–1234, Feb. 2015.

[7] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, "Authorization for the Internet of Things using OAuth 2.0 draft-ietf-ace-oauth-authz-06," IETF, Internet Draft, Mar. 2017.

[8] M. Jones, H. Tschofenig, and S. Erdtman, "CBOR Web Token (CWT) draft-ietf-ace-cbor-web-token-03," IETF, Internet Draft, Mar. 2017.

[9] J. L. Hernndez-Ramos, M. P. Pawlowski, A. J. Jara, A. F. Skarmeta, and L. Ladid, "Toward a Lightweight Authentication and Authorization Framework for Smart Objects," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 4, pp. 690–702, Apr. 2015.

[10] *Constrained RESTful Environments (CoRE) Link Format*, Internet Engineering Task Force Std. RFC6690, 2012.

[11] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized Protocol Stack for the Internet of (Important) Things," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[12] M. Jones and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage," IETF, RFC 4950, May 2012.

[13] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 5719, May. 2015.

[14] P. Hunt, J. Richer, W. Mills, P. Mishra, and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture draft-ietf-oauth-pop-architecture-08.txt," IETF, Internet Draft, Jul. 2016.

[15] X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister, *OpenMote: Open-Source Prototyping Platform for the Industrial IoT.* Springer International Publishing, 2015, pp. 211–222.

[16] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. D. Glaser, and K. S. J. Pister, "OpenWSN: a Standards-Based Low-Power Wireless Development Environment," *Trans. on Emerg. Telecom. Technol.*, vol. 23, no. 5, pp. 480–493, 2012.

[17] X. Villajosana and K. Pister, "Minimal 6TiSCH Configuration, draft-ietf-6tisch-minimal-21," IETF, Internet Draft, Feb. 2017.