Annals of Telecommunications (2018) 73:415–427
https://doi.org/10.1007/s12243-018-0641-8

Cloud-based federated identity for the Internet of Things

Paul Fremantle[1] · **Benjamin Aziz**[1]

Received: 27 March 2017 / Accepted: 29 April 2018 / Published online: 29 May 2018

Abstract
The Internet of Things (IoT) has significant security and privacy risks. Recent attacks have shown that not only are many IoT devices at risk of exploit, but those devices can be successfully used to attack wider systems and cause economic damage. Currently, most devices connect to a cloud service that is provided by the manufacturer of the device, offering no choice to move to more secure systems. We outline a proposed model for IoT that allows the identity of users and devices to be federated. Users and devices are issued with secure, random, anonymised identities that are not shared with third parties. We demonstrate how devices can be connected to third-party applications without inherently de-anonymising them. Sensor data and actuator commands are federated through APIs to cloud services. All access to device data and commands is based on explicit consent from users. Each user's data is handled by a personal cloud instance providing improved security and isolation, as well as providing a trusted intermediary for both devices and cloud services. We demonstrate this model is workable with a prototype system that implements the major features of the model. We present experiment results including performance, energy usage, capacity and cost metrics from the prototype. We compare this work with other related work and outline areas for discussion and future work.

Keywords Cloud computing · Distributed authentication and authorisation · Federated identity management · IoT

1 Introduction

Internet of Things (IoT) devices are proliferating throughout the world, bringing with them a significant threat to privacy and security. There are multiple concerns with IoT devices. One of the key issues in the IoT space is the concern that when a device is purchased, it is tied to a specific web system or *cloud service*, which is owned and managed by a *cloud service provider* (CSP).

In traditional Internet systems, users can migrate away from insecure services, because of interoperability and choice. For example, the instant messaging market has seen the emergence of a number of more secure systems, including WhatsApp, Signal and Threema. In the IoT, the devices are often hard-coded to work with only specific Internet systems, using private protocols and specific servers.

✉ Paul Fremantle
paul.fremantle@port.ac.uk

Benjamin Aziz
benjamin.aziz@port.ac.uk

[1] School of Computing, University of Portsmouth, Portsmouth, UK

The result is that users are reliant on particular CSPs. These may be hacked, or may go out of business rendering devices inoperable. A common attack on CSPs has been to steal credentials and publish users' passwords.

Hacked devices not only can be used to steal personal data but also can be used to launch attacks on other systems. In late 2016, large parts of the US East Coast Internet were disabled by a distributed denial of service attack (DDoS), launched from a *botnet* running across approximately 100,000 IoT devices including IP-enabled CCTV cameras [1].

In addition, there are other security and privacy concerns. Small, inexpensive and/or low-power devices do not properly support encryption allowing communications to be stolen. Few devices use well-defined identity models, meaning that spoofing attacks are possible. Data may be validly shared by multiple cloud services but then aggregated to de-anonymise user information and infringe on privacy. We have previously identified a number of security and privacy concerns with IoT which are documented in [2].

We outline a new model, together with a prototype and experimental results that aims to address these issues. We call this model *OAuthing*.

1.1 Aims and objectives

Based on the privacy and security concerns identified above, the following research questions were identified:

- **RQ1**: Is there a model and architecture for IoT that effectively federates identity and consent management?
- **RQ2**: Can we provide pseudonymous access to IoT systems such that identity is not implicitly shared even in secure flows?
- **RQ3**: Does this approach provide improved privacy and security for IoT networks?
- **RQ4**: Can this approach be implemented without adding an infeasible burden to the performance, cost and energy usage of an IoT network?

The aim of this work is to answer these four research questions.

1.2 Contributions

The contributions of this work in addressing these issues include the following:

- An architecture and system model that allows the federation of multiple parties: the manufacturer, the identity provider, the device identity management, and cloud services and applications that require access to IoT data. This federation and decoupling encourages choice of provider as well as reducing the data available in any given attack.
- Clearly defined device and user registration processes, based on the OAuth2 protocol, that have been extended to support IoT devices and be effective in device scenarios.
- A model for providing anonymous identities for users, reducing the chance that leaked data can be tied to users.
- An architecture that provides each user with a separate cloud instance to handle sharing device data, and an approach for dynamically provisioning these cloud instances.
- A demonstration of the workability of the model through the creation of a working prototype, together with a test harness.
- Experimental results on the including energy usage, performance, cost and capacity metrics of the prototype.

1.3 Assumptions and scope

The Internet of Things encompasses a vast array of environments, from small devices that connect over low-power protocols like Bluetooth LE, to industrial monitoring devices. While we believe that this work could be extended to cover a significant portion of this space, we have chosen to restrict the effort to cover:

- Devices with a direct TCP/IP connection to the Internet
- Devices that can be owned or managed by a single person
- Systems where the privacy of the data is relevant to the owner of a device.

For example, this work does not yet address devices where the device connects via bridging through an app on a mobile phone or via a device gateway. However, the app on the phone or the local gateway could be secured using this model.

1.4 Outline of the paper

The remainder of this paper continues with the following organisation. In Section 2, we propose a model of the different parties involved and an architecture that supports decoupling of different actors in the environment. In Section 3, we outline a prototype middleware system that we have created to implement this model, alongside prototype devices and cloud services that demonstrate end-to-end flows. In Section 4, we outline the experimental results, together with the test harness that was built to gather data. We discuss the related work and compare it to our results in Section 5. In Section 6, we analyse the outcomes of the experimental results and the overall model. We also outline a set of further work identified during this research.

2 Model

In the following section, we outline a model—called *OAuthing*—that aims to federate identity and consent management for IoT systems. We utilise the UML2 [3] graphical modelling notation to model the participants and flows of this approach.

2.1 Participants

Figure 1 shows the current situation for many IoT systems, where there is no federation and the device talks to a single service that manages identity, stores data, provides a user web interface etc.

By comparison, the federated model we present in Fig. 2 allows different federated parties to provide different services that work together.

The participants of the OAuthing model are as follows:

- **The user identity provider (UIdP)**: this is an existing login system where users present their credentials

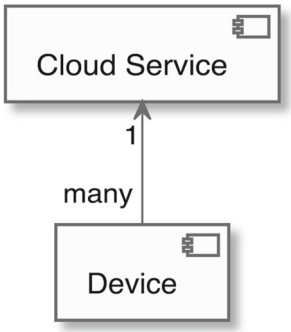


Fig. 1 Existing approach for IoT device cloud

(e.g. Google, Facebook, Github, Twitter or any other federated login).

- **The user**: A user may own one or more devices. A user must have at least one identity with a UIdP.
- **The device identity provider (DIdP)**: this is an *Identity Broker* that first authenticates a user with a UIdP using existing federated identity protocols including OAuth2, OpenID Connect (OIDC) or SAML2. Once the identity is validated, it then creates a secure random anonymous identity which is used in all further processing.
- **Personal cloud middleware (PCM)**: this is an isolated broker that shares data between devices and third-party applications (TPAs) on behalf of the user. The PCM talks to the devices and the TPAs. Within the remit of a single OAuthing instance, there is one PCM per user. We utilise a cloud environment to dynamically launch PCM instances on behalf of users as needed.
- **Intelligent gateway (IG)**: the IG interfaces with the DIdP to validate identities and access authorisation policies and to the cloud infrastructure to instantiate new PCMs. Devices and CSs connect to the IG, and it routes requests to each user's PCM.
- **Third-party application (TPA)**: a device is an IoT device if and only if it shares or receives data and commands with an Internet service. Users control which TPAs can access their sensor data or control their actuators by explicitly consenting to authorise a TPA. Any third party can provide a TPA. If no TPA is authorised by the user, then a device's data is neither shared nor stored.
- **The device**: the device consists of one or more sensors and actuators together with a controller. Devices with sensors will **publish** sensor *data*. CSs may **subscribe** to this data if and only if users consent to it. Devices with actuators will subscribe to *commands*. CSs may publish commands to devices if and only if users consent to it.
- **The manufacturer**: the manufacturer is the logical organisation that creates and markets the device, irrespective of whether they actually outsource any part of the physical manufacturing to a third party. In this model, the manufacturer configures each device with a single DIdP.

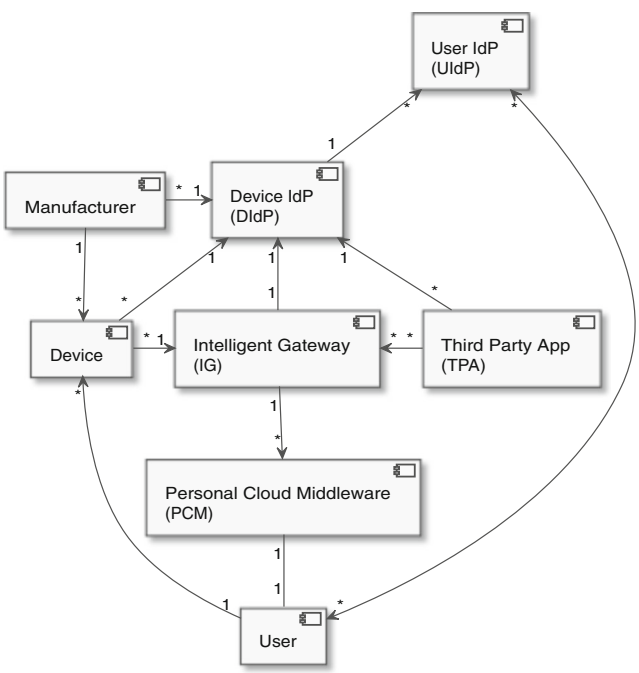


Fig. 2 Proposed model

The OAuth2 protocol [4] is a widely used federated authentication and authorisation protocol. This model utilises the OAuth2 model as a basis for the identity and ownership of devices. While the system can support other protocols such as SAML and OpenID Connect for user login, the device is constrained to use only the OAuth2 protocol, as both those protocols impose too great an impact for a constrained device [5]. One concern with IoT is that hardware devices can be compromised and secrets read from them. It is therefore important that each device has its own credentials. We map each device to be a unique OAuth2 Client, and we use the OAuth2 Client ID as a secure device ID that is only ever shared with the DIdP. We define *ownership* of a device by the user authorising the issuance a security token to the device giving it permission to act on the user's behalf.

Figure 3 shows the UML sequence diagram of a runtime interaction between an owned device and a cloud service.

2.2 Lifecycle

We modelled the lifecycle of a device. The UML lifecycle diagram is shown in Fig. 4.

Once the device is initially flashed, it is connected to a manufacturing server. The manufacturer then uses the DCR

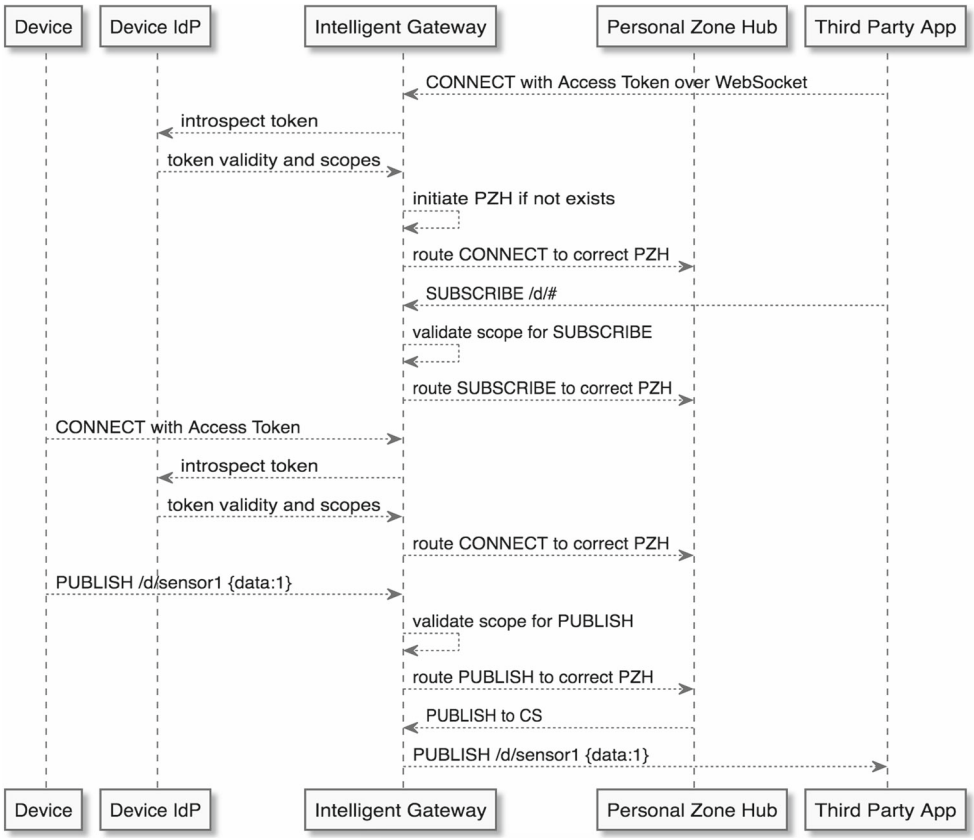


Fig. 3 Device publishing data to CS

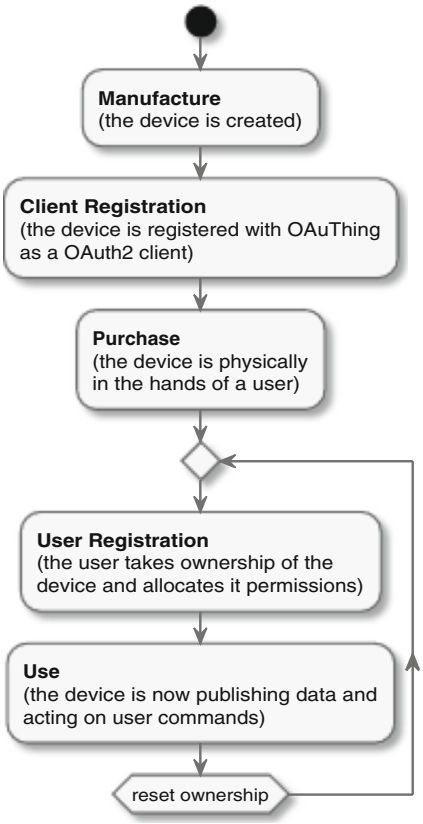


Fig. 4 Lifecycle of a device

API into the DIdP to request a client ID and secret. These are configured into the device by the manufacturing server. At the same time, the DIdP returns a unique user registration URL (URU) that is printed onto the device (usually as a QR code) by the manufacturer.

When the user buys the device, they scan the QR code or otherwise access the URU. This directs the user to the DIdP, which presents a choice of UIpDs to the user. Once the user is authorised with their existing UIpD, they are asked in turn to authorise the device. The resulting OAuth2 refresh token is then stored on the device and represents the logical ownership of the device.

This flow is documented in the UML sequence diagram in Fig. 5.

2.3 Personal cloud middleware

A key part of the model is the concept of a personal hub: where each user's data is routed to its' own hub, protecting the data from multi-tenant attacks. Each hub is run in its own virtualised cloud environment. When a request comes in from a device or CS, we use the anonymous identity associated with the bearer token to route the request to an instance that is specific to that user. If there is no cloud server available, the routing system makes a call to the cloud management system to instantiate a new PCM "on-demand"

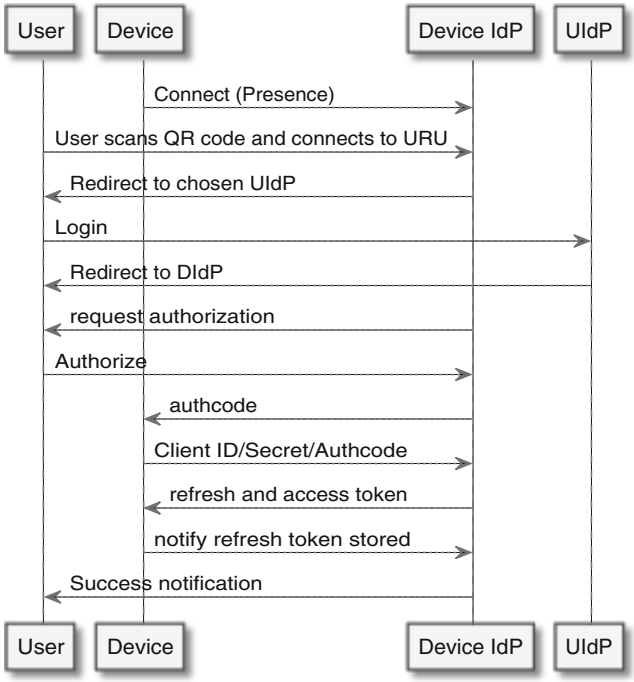


Fig. 5 User registration sequence diagram

and then waits until the instance is running before routing the request to the PCM. In the model, the PCM supports routing and distribution of data and commands, as well as summarisation and filtering of data. These capabilities have an important role in protecting users privacy: firstly, the runtime does not inherently share data such as IP addresses or MAC addresses that can be used to identify devices or users. Secondly, by filtering or summarising data, the PCM can avoid many fingerprinting attacks on devices [6]. The PCM can also provide protocol mapping and device shadow capabilities, meaning that it is simpler for TPAs to connect to devices.

2.4 Scopes and APIs

The DIdP implements consent-based authorisation policies called *scopes*, as defined in the OAuth2 specification. Each scope controls access to a set of APIs. These APIs may be implemented in multiple protocols. Users may consent to a third party to have access to a specific scope, which is captured in a token. The model-defined APIs are shown in Fig. 6, together with the provider of the API and the associated scope.

One of the outcomes of defining scopes as part of this model is that there is a clean mapping between the different roles in the system and the associated scopes, which is shown in Fig. 7.

2.5 Information visibility

The model allows us to analyse which identifiers and data each participant has access to. A key aim of this model is to ensure that each party has a restricted view of a user's actions and therefore can only breach privacy in a limited way or with the help of a third party.

In Fig. 8, we identify each participant and show what access they have to credentials, identities and data.

The manufacturer only knows the original device identity (e.g. MAC address) and the client ID that was issued at manufacturing time. Unless the user chooses to share information with the manufacturer, then the manufacturer does not know the owner of the device and do not see any device data.

Fig. 6 APIs

API	Client	Provider	Scope	Description
Dyn Client Registration	Manu-facturer	DIdP	dcr	This allows the creation of OAuth2 clients.
OAuth2 Token API	Device CS	DIdP	NA	This API is defined by the OAuth2 specification.
Introspection	IG	DIdP	intro	IG to DIdP to introspect bearer tokens
Publish Data	Device	IG	pd	Publish Data from a device sensor
Subscribe to Data	CS	IG	sd	CS subscribing to Data from a sensor
Publish Commands	CS	IG	pc	CS sending commands to an actuator
Subscribe to Commands	Device	IG	sc	Actuator subscribing to commands from a CS

Fig. 7 Mapping of scopes to participants

Role	Scopes	Description of roles and scopes
UIdP	N/A	This IdP is the primary source of identity to the Device IdP and does not have any OAuth2 scope permissions
DIdP	openid ***(or UIdP Specific)***	The Device IdP is the "source" of scopes to the other roles. It requires access to the third-party IdPs, which may define their own scopes.
Manu-facturer	dcr	Dynamic Client Registration (DCR): allows caller to create new ClientIDs using the DCR API
Intelligent Gateway	intro	Introspection: allows the IG to ask the DIdP for the pseudonym and scopes for a given Bearer Token
TPA	Rd, Pc	Read/Subscribe to Data (Rd) and Publish Commands (Rc) The TPA may be allowed one or other or both
Device	Pd, Rc	Publish Data (Pd). Read/Subscribe to Commands (Rc).

The DIdP knows the original user profile as provided by the UIdP. This is only used to ensure that two tokens issued to the same user will result in messages being routed to the same PCM. The DIdP is aware of the device presence (because the device must regularly refresh its tokens, and because the IG must introspect those tokens). However, the DIdP does not see any of the device data or commands. The DIdP is aware of which TPAs a user is subscribed to, but does not know which devices are interacting with which TPAs. In the event the DIdP is compromised, the attackers cannot steal passwords or any user data—only the link between the UIdP user identity and the anonymous identity.

The IG is given an anonymous random ID for the user, through an *Introspection API* on the DIdP, which is only available to IGs. The IG does not store any data. If it is compromised, only data flowing through the system can be stolen, and this cannot inherently be tied to a user or any device identifier. Both the IG and the DIdP would need to be compromised to tie that data to a specific user.

The device and any TPAs are not aware of the user's identity—they only see a refresh or bearer token and do not have the authorisation to call the introspection API. Therefore, a TPA cannot deduce the user's identity from

Fig. 8 Information visibility matrix

	UIdP	DIdP	Mfr	Dev	IG	TPA
User Profile	✔	✔				
HW ID			✔	✔		
Secure Device ID		✔	✔	✔		
Device Secret		✔		✔		
Pseudonym		✔			✔	
Token		✔		✔	✔	
Device Data				✔	✔	✔

the OAuthing system. However, a TPA may be aware of the user's identity through out-of-band means. For example, if the user decides to share their IoT data publicly on a webpage, this webpage identifies them. Even in this case, the TPA does not know the device ID, only the data that is shared. Alternatively, the data itself may contain personally identifiable traits: this may be mitigated by sharing a summary or filtered data. Finally, the device data itself is only visible to the device, the PCM and authorised TPAs.

3 Implementation

In order to validate the model, we built a prototype of the system.

The following sections outline the components that were implemented to demonstrate the system.

3.1 Protocol mapping

The model has been designed to work independently of specific protocols. However, in order to implement a prototype, we needed to make specific choices on protocols. We also provide a mapping of part of the OAuth2 protocol into MQTT [7] (an IoT-optimised protocol), allowing the device to utilise a single protocol for identity, authorisation and data. This simplifies the device coding and reduces the memory footprint of OAuthing support.

3.2 The OAuthing DIdP

The prototype DIdP is called the OAuthing DIdP.

We implemented the OAuthing DIdP as a set of containers running in the Docker[1] container system, allowing it to be efficiently deployed and tested in a cloud environment.

One of the key aims of our model is to preserve privacy. In our implementation, we do not store any identifiable aspects of the user. Each UIdP provides the OAuthing IdP with an identifier. Rather than directly storing these identifiers, we create a new secure random anonymous identity and store this against the hash of the UIdPs unique ID.

A key difference between this and previous prototypes is that we implemented the MQTT APIs using an *embedded broker pattern*. In this model, the MQTT broker is part of the OAuthing IdP. The major benefit of this approach is that the broker can implement a more secure model, especially when emulating request/response flows over the asynchronous MQTT protocol.

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

3.3 IGNITE

The prototype of the IG is called IGNITE (Intelligent Gateway for Network IoT Environments).[2] IGNITE runs in a Docker cloud container environment and has access to control this Docker environment. When a device or TPA initiates an MQTT connection with the `CONNECT` packet, IGNITE first validates the bearer token by calling the DIdP's introspection API. This either returns a random anonymous ID together with a set of scopes or informs IGNITE that the token is invalid. IGNITE is then responsible for launching a new cloud instance to act as the PCM on behalf of the user, or routing the request to the existing PCM.

3.4 Personal cloud middleware

The PCM was implemented using the open source RSMB MQTT broker.[3] This broker has a very low memory overhead and enabled us to run a significant number of PCM containers on standard hardware. We have not yet implemented summarisation and filtering on the PCM, which will potentially enlarge the memory footprint, but we did not yet optimise the Docker runtime of the PCMs or the underlying operating system, and therefore, we are confident that this can be offset.

3.5 Device hardware

We aimed to build the simplest possible device to provide a baseline evaluation of whether the model was implementable on very small footprint devices. We chose the commonly available ESP8266 platform for our reference device. This chip provides an embedded 32-bit processor, Wi-Fi connectivity and a number of digital inputs and outputs for less than US$2.50 each (at the time of writing). Currently, the ESP8266 supports TLS without full certificate authority chains. Instead, it uses fingerprints of SSL certificates to validate them.

3.6 Sample third-party application

In order to demonstrate this system, we also created a simple web-based cloud service. This first connects to the OAuthing DIdP using a standard OAuth2 HTTP flow to request access to IGNITE. The user logs in using the same UIdP that they registered their device with. After the user authorises the Sample TPA to access IGNITE, the sample app is loaded. This uses MQTT over WebSockets to

[2] Please note that there is another IoT framework with a similar name. These systems are unrelated and the system described here predates the other system: https://www.iot-ignite.com/

[3] https://github.com/eclipse/mosquitto.rsmb

communicate and presents a simple UI, which allows users to interact with the device.

4 Evaluation

In this section, we present the results of testing both the device and the cloud systems involved in the OAuthing prototype. Note that there was no optimisation of the systems towards better performance: the system is a prototype and we did not perform profiling or optimisations. We therefore consider the results of these tests to be an upper bound on the extra costs of using OAuthing.

4.1 Test methodology

The test methodology was influenced by academic papers both on middleware performance testing [8, 9] and commercial testing [10, 11].

We identified four key measures for evaluating the prototype.

- **Transactions per second**
  The total number of transactions per second (TPS) that a system can handle is the most common measurement of distributed systems.
- **Latency**
  The latency of a system measures how quickly the system reacts. We measured the time taken from the start of a publication process on the publisher to the time that the message is received at the subscriber.
- **Device memory**
  Devices often have constrained memory, and therefore, a key measure we identified was to understand the memory requirements of our firmware and therefore the remaining memory available to device designers to implement their own logic.
- **Power consumption and energy usage**
  A key aspect for IoT devices is power consumption. Many IoT devices (including the prototype system) run off of a battery with a limited capacity and therefore rely on using low power. We therefore created a test harness to accurately measure power usage and energy consumed by our sample device. Further details of the power measurement system are described below.

In all the tests described below, we ensured that the systems were warmed up and running and that all the results were repeatable across multiple tests. In addition, we created a *baseline* system for comparison. The baseline system does not use the OAuthing backend, but instead uses the commonly used Mosquitto MQTT server. All the tests were carried out using standard cloud server instances with fixed sizes from the Digital Ocean cloud provider.

In order to simulate multiple clients for the latency and TPS tests, we created a test harness designed to run across multiple cloud servers. The test harness consists of multiple test load drivers (TLDs). Each TLD can simulate one or more clients, emulating the network behaviour of the IoT device. The TLDs then report the performance and latency data to a test manager. The test manager is running on a separate instance, and this service collates the results. The TLDs implemented two different workloads:

- **One second client**
  The *one second client* emulates a device that sends one message per second. This is designed to test the system under moderate load.
- **Stress client**
  The *stress client* emulates a device that sends messages continuously. This is designed to test the system under heavy load.

Both tests time the latency between sending the message from the device and receiving it in the TPA (or vice-versa).

4.2 Results

One clear result is that there is a working prototype of the model that demonstrates all aspects of this decoupled approach, including support on a low-cost device and a cloud implementation of the server-side components.[4]

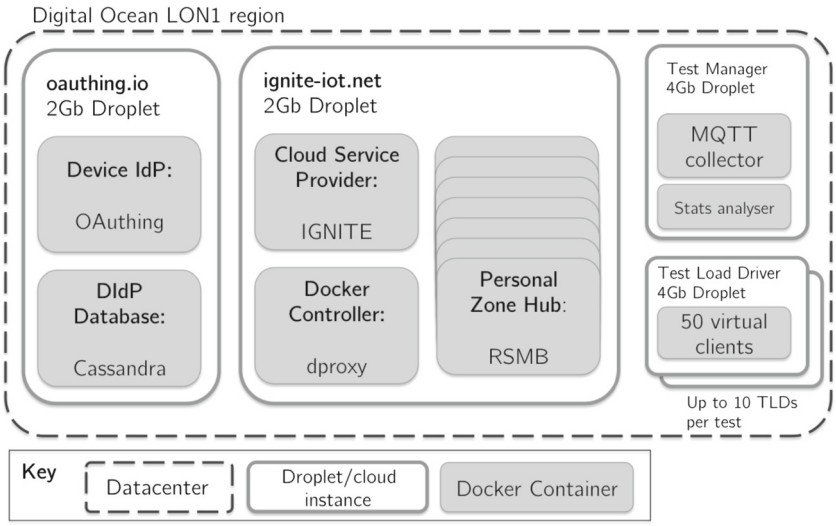
Figure 9 shows a simplified diagram of the test environment, which is running in a public cloud environment.

Figure 10 shows the latency comparison of OAuthing compared to the baseline system (Mosquitto) using the *one second client*. Figure 11 shows the mean and 95% and 99% percentiles for the IGNITE latency responses in the same test. The graph demonstrates that OAuthing shows consistently low latencies across all workloads. The additional latency added to message interactions compared to Mosquitto was around 1 ms. The percentiles show that 99% of requests had latency of less than 11 ms even when the system was loaded with 400 test clients, and 95% of requests had latency less than 6 ms. Given that average round trip ping times over the Internet are in the 20–80-ms range, these results demonstrate that the overhead of the proposed approach is insignificant to users.

The next data point collected was the maximum number of PCMs that could be run on a single cloud server. The tests were run on a server with 2 GB of memory and no swap configured, costing US$20/month. This environment was able support at least 400 PCM instances, with the server running out of memory beyond 415 containers. Simply

[4] A short video showing the registration and data-sharing process is available at http://freo.me/oauthing-video

Fig. 9 Test environment



adding swap will increase this number at the cost of some latency, but we have not yet evaluated this balance.

Figure 12 shows the average connect time for three different scenarios. The fastest is the Mosquitto broker, with an average connect time of 24.5 ms. The slowest is the OAuthing when the user has not previously connected. In this scenario, the system needs to introspect the token and then wait until the new container is launched and ready. This takes on average 1294 ms (1.3 s). While this is comparatively large, it happens rarely in the system and in practice devices take between 2 and 10 s to connect to local Wi-Fi networks. This could be ameliorated by pre-loading unused containers and associating them with users at connect time.

The third scenario is the connect time when there is already a user container running. The average time here was 35.9 ms. The extra latency compared with Mosquitto (35.9 vs 24.5 ms) is well within acceptable ranges.

Figure 13 shows the performance of the OAuthing system under stress. This shows that the server was handling more than 4500 TPS at all levels and the average latency rose to 83.3 ms when the system had 400 concurrent clients. This test demonstrates that the system as deployed in the test environment can support each user owning 600 devices each interacting once a minute, even when the system is fully loaded with 400 concurrent PCM containers. The latency line shows that as new clients are added the latency increases in direct proportion, demonstrating fair allocation of resources.

Figure 14 shows the performance of the OAuthing IdP while issuing new Client IDs during manufacturing. This is the least well-performing part of the system because we chose to use a secure hashing algorithm (PBKDF2 [12]) to ensure that our password database is resistant to dictionary attacks. The result is that adding a client incurs considerable CPU time.

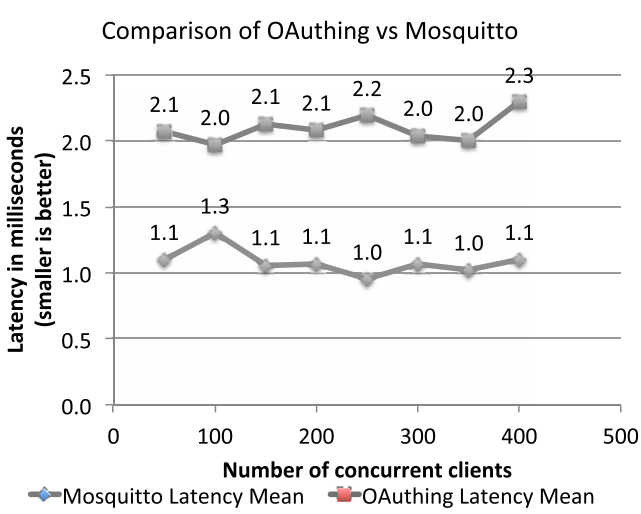


Fig. 10 One second client comparing OAuthing vs Mosquitto

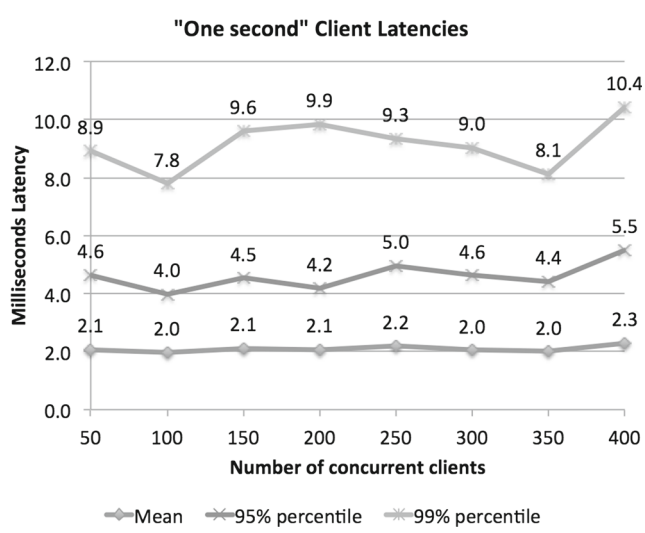


Fig. 11 One second client OAuthing percentiles

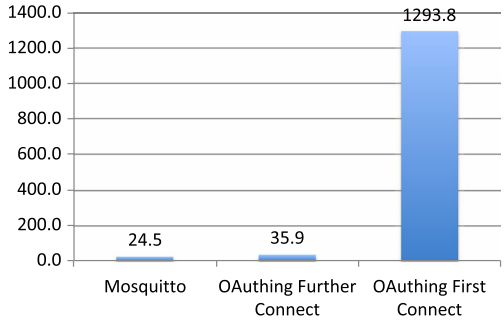


Fig. 12 Device connect time

We also performed performance tests measuring the latency and throughput of the DIdP under introspection, when the gateway asks the DIdP for the anonymous identity and authorisation policies. This is presented in Fig. 15. The results show that the introspection service can successfully scale to support many IGs. Given that CSs and devices only connect intermittently (due to the persistent TCP session model of MQTT), even a single DIdP server could handle significant numbers of devices and third-party clients.

4.3 Device memory usage

The Arduino development environment that was used to create the device firmware provides statistics on the program and variable memory usage. Figure 16 shows the program and variable memory usage of the ESP8266 when loaded with the OAuthing sample device loader code. The graph captures the usage of different components of the loader. Each column includes the previous column. The largest component is the base C libraries needed by the Arduino system, which take up 40% of variable memory and 24% of program memory. The next largest aspect is the TLS support which incrementally takes 5.5% of the program memory and 7.7% of the variable memory. Overall, the loader leaves over 38 k of variable memory and over 700 k of program memory for the developers of any device sensor and actuator logic, which is sufficient to build complex device applications and support a variety of sensors and actuators. We did not perform any code optimisation.

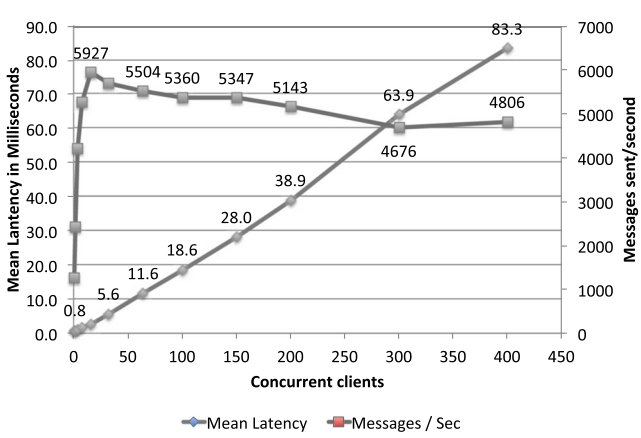


Fig. 13 Stress client OAuthing performance

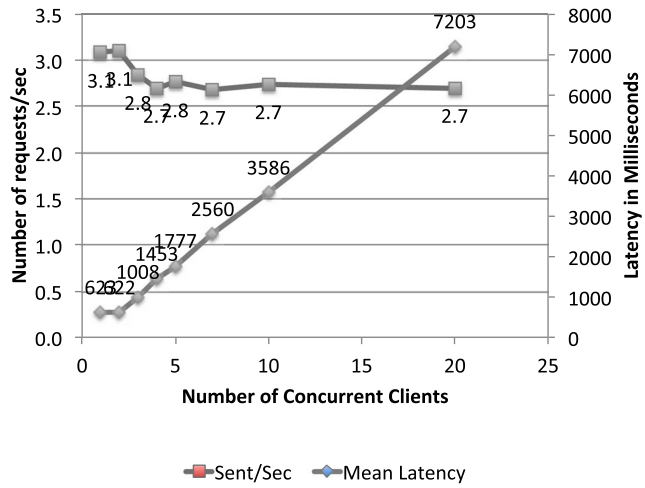


Fig. 14 Dynamic client registration latency and throughput

4.4 Energy and power measurement

In order to test the additional power burden of using the OAuthing approach, we created a power measurement harness to evaluate the power usage of the system. Traditional power measurement systems are not optimised for measuring sub-watt power usage and therefore, we could not use an off-the-shelf system. Figure 17 shows a logical diagram of the power management test system that we created.

The created system measures milliwatt power usage with better than 1% accuracy.

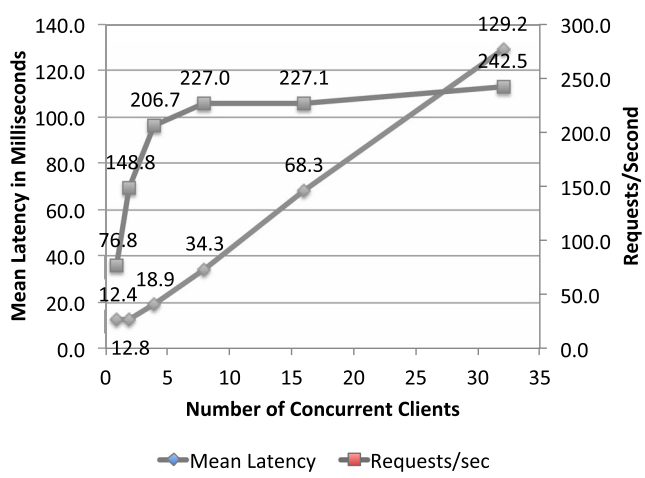
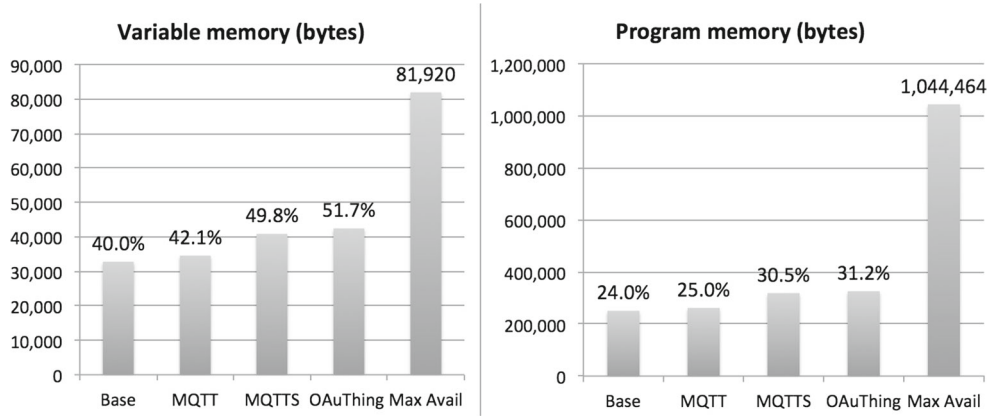


Fig. 15 Throughput and latency of the introspection API on the DIdP

Fig. 16 ESP8266 memory usage



We measured two different scenarios, comparing the OAuthing device to the same device configured to talk to Mosquitto without using the OAuthing model

- The first scenario was measuring the total energy usage from initial power-on until the first message is sent to the server. This measures the bootstrap power phase, especially capturing the overhead of the refresh flow and the credential based MQTT CONNECT message. This is measured in milliwatt-hours. We ran each test 20 times.
- The second scenario was the on-going power usage over the next 15 min after startup. To ensure both systems were comparable, we waited until the device was fully warmed up and then took 900 s of samples (approximately 280,000 samples).

Figure 18 shows the time-to-first-message results. The data shows that the OAuthing device took 0.68 s (5.7%) longer to send its first message, using 7.5% more energy (0.195 mWH) than when connecting to Mosquitto.

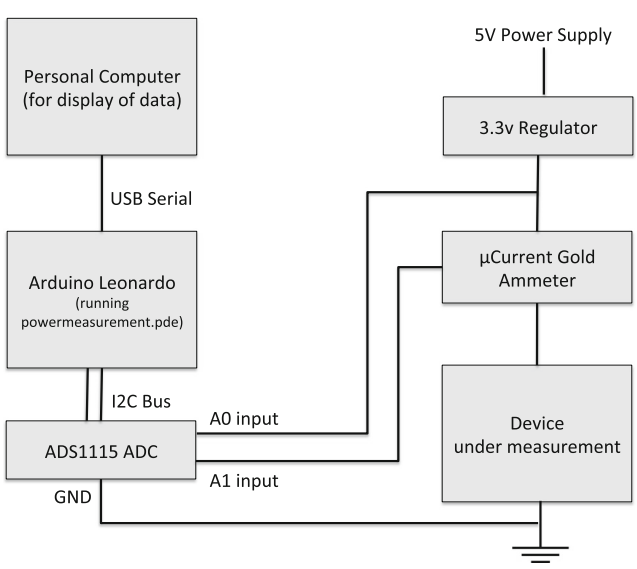


Fig. 17 Power management test system

Figure 19 shows the on-going power requirement is 26 mW higher for the OAuthing device, using 11.5% more power than the device connecting to Mosquitto. This much extra power usage is slightly unexpected and requires further investigation. The ESP8266 offers three different low-power modes, none of which were implemented, and therefore, we expect the magnitude of this power usage to be significantly reduced in a production device.

5 Related work

In [5], we previously looked at using federated identities for IoT, especially mapping OAuth2 tokens to work with the MQTT protocol. In a follow up work [13], we demonstrated the use of the *dynamic client registration* (DCR) API to support each device having a unique OAuth2 client identifier. In each case, identifiers needed to be manually added to the device, which is unrealistic in manufacturing processes. Existing public IoT middleware such as IBM Watson IoT and AWS IoT also have this concern. Compared to these previous works, this current model adds a clear automated device and user registration process. It also adds anonymous identities and personal cloud middleware.

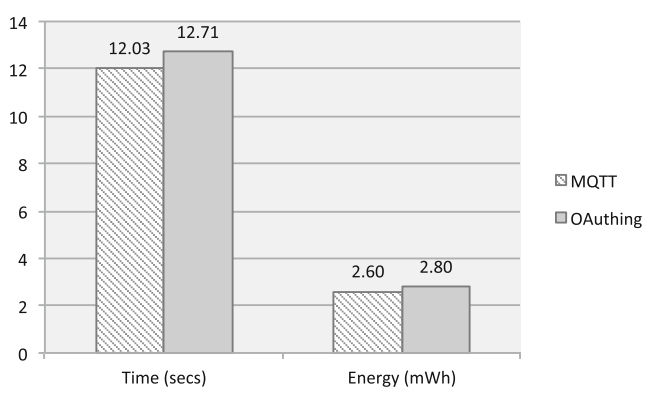


Fig. 18 Time and energy taken to bootstrap

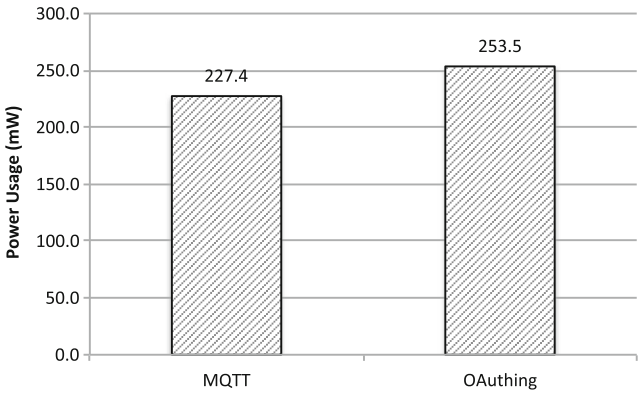


Fig. 19 Power usage

IOT-OAS [14] addresses the use of OAuth2 with the CoAP protocol. The mapping of the OAuth2 Token API to support IoT devices using the CoAP protocol is being formalised in [15] and is described in [16]. In [17], there is a demonstration of the OAuth1 protocol with MQTT, favouring OAuth1 over OAuth2 for IoT devices. The reasons for choosing the older OAuth protocol are obviated by the mapping of the refresh flow which OAuthing offers. In [18] and in [19], there are platforms that support OAuth2 for IoT devices that communicate via HTTP and WebSockets. None of these works address automated registration processes, and none provide the privacy controls of anonymous identifiers or isolated personal cloud instances.

In [20], a capability-based access system is described that allows anonymous identities to be used. Bernabe et al. [21] provides an architecture reference model for an approach that supports anonymous identities. Neither of these systems separates the provision of anonymous identities from the data-sharing middleware.

The concept of a personal zone hub (PZH) is described in the Webinos [22] system: this is similar to our PCM. However, in Webinos, users must instantiate the PZH themselves, and there is no analysis of the cost per user. We extend the PZH concept to support a dynamic instantiation of PCMs as containers and provide a cost model. Webinos does not address secure federated device identities and does not provide a registration process.

6 Discussion and conclusions

The OAuthing model provides significant improvements over existing systems, providing much stronger guarantees of privacy. Data and identity are not shared without consent, and data can be shared anonymously. Device and user registration are automated, and the PCM model can prevent fingerprinting and sharing of IP and MAC addresses, as well as user and device identity.

A key concern around the PCM model is that the cost per user might be too high. The prototype demonstrates that PCMs can be automatically deployed on behalf of the user with acceptable times. The experimental results demonstrate that a US$20/month cloud server can support 400 users, resulting in a cost per user of just $0.05 per month. Further optimisation could reduce this cost.

The OAuthing model and prototype demonstrate that devices can be connected to TPAs without inherently leaking the user's identity to either system. User's may choose to provide TPAs with their identity, but that becomes a positive consent of the user rather than the default. In addition, users can bring pre-existing identities to the system rather than being required to create new credentials, which reduces the chances of password theft and gives users a choice of identity provider.

We therefore assert that we have answered the research questions as follows:

- **RQ1**: The OAuthing model and architecture provides an approach whereby IoT networks can be decoupled. In particular, the identity and consent management can be decoupled from both the manufacturer and the data sharing middleware, providing users with a choice of identity provider and of which systems they consent to share data with.
- **RQ2**: The OAuthing model and prototype demonstrate that secure data sharing can happen between systems, including consent, without sharing identity as a pre-requisite.
- **RQ3**: We assert that the OAuthing model provides significant improvements in privacy and security for IoT networks over existing approaches, as discussed above.
- **RQ4**: The additional performance, latency, cost and energy usage of this model have been demonstrated to be feasible. The overheads in terms of latency are within the norms of Internet latency, and the small overhead in energy consumption is acceptable. All the data was collected on a non-optimised prototype and therefore provides an upper bound to the overheads.

There remain a number of unexplored aspects of this model. We expect to add tests that evaluate the performance of the OAuthing DIdP. In addition, we plan to extend the system to support multiple co-existing DIdPs. We have discussed this model with two significant device manufacturers. One potential concern is that some device manufacturers' business models are based around collecting user data and therefore, this system is unattractive precisely because it improves users privacy. However, there are a number of areas where this approach offers significant benefits, for example in medical devices, where the manufacturer does not wish to access data for regulatory

reasons. In addition, it is possible to start with the OAuthing model and progressively lessen certain privacy controls to provide a system that still shares data but provides stronger guarantees.

We have identified a number of future items of research around this, including developing a full threat model for the system; supporting devices that communicate via gateways (e.g. Bluetooth devices talking to a phone or hub); and demonstrating clustering and high-availability for the OAuthing DIdP and IGNITE. We have also identified that further de-centralisation maybe possible by utilising distributed ledger technologies such as Blockchains with the DIdP to reduce the requirement to have a central IdP for devices.

References

1. Wei J (2016) DDoS on Internet of Things–a big alarm for the future
2. Fremantle P, Scott P (2017) PeerJ Comput Sci 3:e114
3. Rumbaugh J, Jacobson I, Booch G (2004) Unified modeling language reference manual. The Pearson Higher Education
4. Hardt D (2012) The OAuth 2.0 authorization framework. Tech. rep. IETF
5. Fremantle P, Aziz B, Scott P, Kopecký J (2014) In: 3rd International workshop on the secure IoT
6. Kohno T, Broido A, Claffy KC (2005) Remote physical device fingerprinting. IEEE Trans Depend Secur Comput 2(2):93
7. Locke D (2010) MQ Telemetry Transport (MQTT) V3.1 protocol specification. Tech. rep., IBM Corporation
8. Liu Y, Gorton I, Liu A, Jiang N, Chen S (2002) In: Proceedings of the fortieth international conference on tools pacific: objects for internet, mobile and embedded applications. Australian Computer Society, Inc., pp 123–130
9. Jayaram K, Eugster P, Jayalath C (2013) Parametric content-based publish/subscribe. ACM Trans Comput Syst (TOCS) 31(2):4
10. AdroitLogic. Esb performance. http://esbperformance.org/ (2016). Accessed 20 March 2017
11. Council TPP Tpc. http://www.tpc.org/ (2017). Accessed 27 March 2017
12. Kaliski B (2000) RFC 2898; PKCS# 5: Password-Based Cryptography Specification Version 2.0
13. Fremantle P, Kopecký J, Aziz B (2015) Web API management meets the Internet of Things. Springer International Publishing, Cham, pp 367–375. https://doi.org/10.1007/978-3-319-25639-9_49
14. Cirani S, Picone M, Gonizzi P, Veltri L, Ferrari G (2015) Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios. IEEE Sensors J 15(2):1224
15. IETF Authentication and authorization for constrained environments (ACE)—documents https://datatracker.ietf.org/wg/ace/documents/ (2016). Accessed 30 Aug 2016
16. Tschofenig H (2016) Datenschutz und Datensicherheit-DuD 40 (4):222
17. Niruntasukrat A, Issariyapat C, Pongpaibool P, Meesublak K, Aiumsupucgul P, Panya A (2016) In: 2016 IEEE International conference on communications workshops (ICC). IEEE, pp 290–295
18. Raggett D (2015) COMPOSE: An open source cloud-based scalable IoT services platform. ERCIM News 101:30
19. Emerson S, Choi YK, Hwang DY, Kim KS, Kim KH (2015) In: 2015 International Conference on information and communication technology convergence (ICTC). IEEE, pp 1072–1074
20. Rotondi D, Seccia C, Piccione S (2011) 1st IoT International forum. Berlin
21. Bernabe JB, Hernández JL, Moreno MV, Gomez AFS (2014) In: International Conference on ubiquitous computing and ambient intelligence. Springer, pp 408–415
22. Desruelle H, Lyle J, Isenberg S, Gielen F (2012) In: Proceedings of the 2012 ACM conference on ubiquitous computing. ACM, pp 733–736