symmetry

Article

Security-Oriented Architecture for Managing IoT Deployments

André Zúquete [1,*], Hélder Gomes [2], João Amaral [3] and Carlos Oliveira [3]

1 DETI/IEETA, University of Aveiro, 3810-193 Aveiro, Portugal
2 ESTGA/IEETA, University of Aveiro, 3810-193 Aveiro, Portugal; helder.gomes@ua.pt
3 IEETA, University of Aveiro, 3810-193 Aveiro, Portugal; joaop.amaral@ua.pt (J.A.); carlosmbo@ua.pt (C.O.)
* Correspondence: andre.zuquete@ua.pt

Received: 16 September 2019; Accepted: 15 October 2019; Published: 19 October 2019

Abstract: Assuring security and privacy is one of the key issues affecting the Internet of Things (IoT), mostly due to its distributed nature. Therefore, for the IoT to thrive, this problem needs to be tackled and solved. This paper describes a security-oriented architecture for managing IoT deployments. Our main goal was to deal with a fine-grained control in the access to IoT data and devices, to prevent devices from being manipulated by attackers and to avoid information leaking from IoT devices to unauthorized recipients. The access control is split: the management of authentication and access control policies is centered on special components (Authentication, Authorization, and Accounting Controllers), which can be distributed or centralized, and the actual enforcement of access control decisions happens on the entities that stay in the path to the IoT devices (Gateways and Device Drivers). The authentication in the entire system uses asymmetric cryptography and pre-distributed unique identifiers derived from public keys; no Public Key Infrastructure (PKI) is used. A Kerberos-like ticket-based approach is used to establish secure sessions.

Keywords: Internet of Things; authentication; authorization; access control; secure communication; privacy; pseudonymity

1. Introduction

We are currently experiencing the next evolutionary step of the Internet, widely known as the Internet of Things (IoT), where everyday objects are equipped with identifying, sensing, acting, networking and processing capabilities [1,2]. In the last decade, this new paradigm has been in the spotlight as a research area of interest for academics and as a profitable market with a wide variety of appliances for companies and startups, ultimately creating new business opportunities [3].

However, the feverish race of companies to develop new IoT products has led to the creation of a security apocalypse resulting from the lack of consideration and application of security schemes not only during their design phase but also after, once they are already sold and operating online ([4], Cap. 1).

Nonetheless, the whole IoT ecosystem is, by nature, easily exploitable. This is due to its inherent heterogeneity, with support for many different protocols and technologies and its large coverage area of connected devices. Additionally, most of these devices are meant to operate continuously and unattended, resulting in them being easily accessible and propitious to physical attacks. Furthermore, communications are mainly done using wireless schemes, thus facilitating eavesdropping attempts, and IoT devices are usually restricted in computational power, thus hindering the adoption of already existing security schemes in classic Internet applications [5,6]. Therefore, for the IoT to thrive, it is necessary to develop and adopt security and privacy schemes oriented specifically for the IoT.

1.1. Motivation and Goals

In many IoT architectures, IoT devices can have a direct interaction with the Internet, which they use to send data to target recipients (e.g., Cloud repositories) or to receive data from authorized sources (e.g., update services) [7]. However, security and privacy can be at stake when this freedom of interaction with the Internet is available. First, because IoT devices may send private data to unwanted recipients using cover channels [8,9]. In addition, given the actual and foreseen IoT device connection rate [10], their Internet-wide accessibility makes them valuable targets for zero-day attacks and widely available bases for launching DDoS attacks [9–12]. Thus, one of our goals was to provide an architecture where (1) IoT devices are not free to get in contact with arbitrary Internet targets and (2) IoT devices are not directly reachable from any Internet host, not even from within the same local area network.

The communication with IoT devices, in many IoT architectures, is based on REST APIs and OAuth-based authentication and authorization [13]. In both cases, they need to use TLS [14] (usually indirectly, through HTTPS [15]) as secure transport protocols [16]. However, this also implies using a PKI (Public Key Infrastructure) for managing the public key certificates required to authenticate the TLS server side, which is a managing burden. Furthermore, the mutual authentication with TLS or HTTPS and OAuth is performed at different protocol levels (secure transport level for the server side, with TLS or HTTPS, and application level for OAuth), which creates confusion. Thus, another of our goals was to design a PKI-free architecture where it could be easy and fast to (1) set up key material required for mutual authentications and (2) perform an integrated, mutual authentication between an IoT device and an application willing to interact with it.

One of the problems with IoT architectures is how to define and enforce a fine-grained access control of client applications to IoT devices, to extract information from them or to control their actions. Adding this burden to the IoT devices is not an easy task, mainly from the configuration point of view. OAuth could do the trick but, as explained before, it requires the use of HTTPS as a secure transport. Thus, another of our goals was to design a scalable and easy-to-manage architecture for defining and enforcing access control policies, both for configuring IoT deployments and for interacting with the IoT devices.

1.2. Contribution

In this paper, we propose a new security-oriented architecture for IoT deployments. Our solution was designed with scalability concerns in mind while abstracting the underlying communication technology of the devices and the applications it may employ. As such, it is possible to run multiple applications from different companies or manufacturers without any interference among them.

The architecture establishes a clear separation from the entities providing the IoT-related services and those that authorize the access to the former after an authentication step. This separation allows a more coherent definition and deployment of a security policy implemented on a reduced set of agents, possibly much less than the number of IoT devices they supervise.

Since the IoT devices encompass not only sensors but also actuators, this implies that their compromise by attackers may have extra consequences that are transposed into the real world, possibly resulting in physical damage or threaten human lives. As such, the access control is performed at the level of the IoT device, and not simply at the level of the generated and transmitted data.

The clients of the system never interact directly with the devices themselves, but rather through properly configured Gateways that relay the requests to their destination and mediate the interaction between clients and devices. This mediation also prevents devices from delivering data to destinations other than authorized clients, which is desirable for promoting the privacy of the data they produce.

Additionally, since the devices operate without supervision, it is also important to be aware that any security credentials saved in the device itself are virtually vulnerable and prone to being accessed and extracted. Therefore, the definition of these credentials is also of major concern and must be done in such a way that its knowledge by an attacker does not compromise the whole IoT infrastructure.

In our solution, this is achieved by using a public key pair for most of the components of the IoT architecture. Furthermore, no PKI is used, since the knowledge of the relevant public keys can be established a priori (just as in SSH [17]) with a pre-distribution of their values by the relevant entities. Due to the resource constraints that devices might have, the use of asymmetric cryptography, which is computationally expensive, has been kept to a minimum, applied only when necessary. The remaining cryptographic operations are performed employing symmetric cryptography techniques.

Our architecture uses the concept of tickets, introduced by Kerberos [18], to enforce access control and perform key distribution for sessions. However, our tickets are based on asymmetric cryptography, while Kerberos uses only symmetric; we have many ticket issuers per IoT deployment, while Kerberos centralizes this task; and our tickets identify their owners with pseudonyms, while Kerberos does not.

2. Architecture

The architecture of the system includes the following components (see Figure 1): IoT devices, Device Drivers (DD), Device Hosts (DH), Gateways, Authentication, Authorization and Accounting Controllers (A3C), DH Managers (DHMs) and Clients.

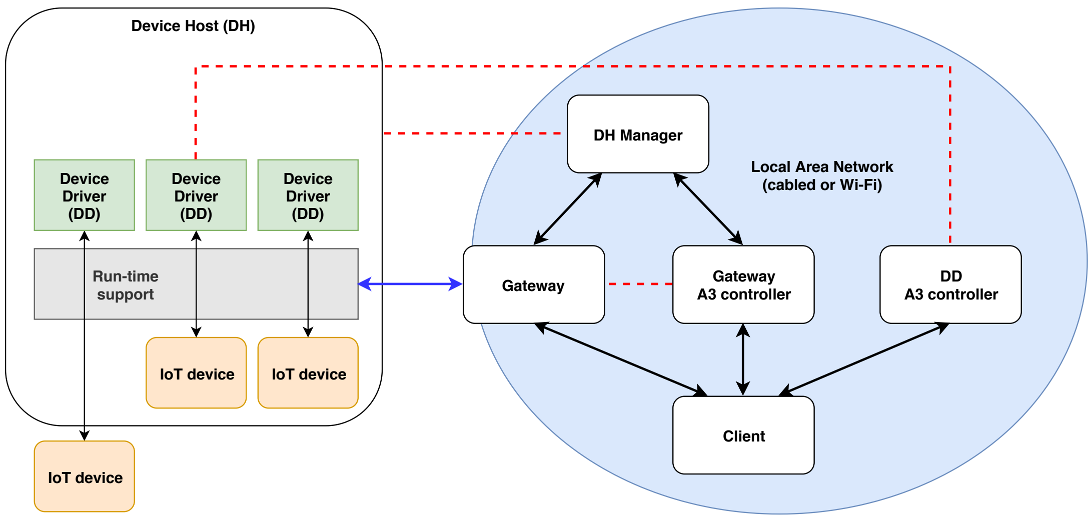


Figure 1. Architecture of our IoT system. Rounded rectangles represent computing systems or other hardware devices; rectangles represent software modules; solid arrows represent direct communication flows; dashed lines represent logical associations. The communication between Clients, Gateways, DHMs and A3Cs is supported by a cabled or wireless local area network (e.g., Wi-Fi). The communication between a Gateway and a DH uses a cabled or wireless link (e.g., serial line or Bluetooth).

2.1. IoT Devices and Their Device Drivers (DDs)

IoT devices, or the "things", are pure sensors or actuators. They can sense the environment or to act on that same environment.

IoT devices have no notion of security, nor can they have the power to establish network communications; for that, they rely on a host (Device Host, DH), which runs for them a dedicated driver (Device Driver, DD). They connect to the DH and, indirectly, to the DD, through a local hardware link or bus (e.g., I2C) and a run-time support system.

Each IoT device has a controller (AAA Controller, A3C), which defines its access control policy relatively to Clients. DDs enforce access control decisions taken by their A3C. Those decisions, which can include detailed API grant/deny permissions, are given by tickets, which are produced by the A3C and given to the Client for setting up a session with the DD.

DDs have low-level access to IoT devices and high-level access to communication endpoints (message queues) maintained by a Gateway and defined by Clients. Those communication endpoints are the only possible destination of the data that DDs produce, either sensors' readings or action logs. DDs can deal with the security of the data they send or receive to and from Clients, e.g., they may implement data encryption and decryption. This enables end-to-end, Client-DD communications to be protected from malicious Gateways or network eavesdroppers.

2.2. Device Host (DH)

A DH is a (low-power) computational system that controls the operation of one or more devices. It can establish a communication link with a Gateway, to enable a multi-hop, end-to-end communication with other elements, namely Clients. The communication link would preferably use low-power, wireless transmissions (e.g., Bluetooth) but cabled links are possible as well (e.g., power-line communication).

A DH contains a primordial housekeeping application responsible for managing DDs, namely to load and run their code. It is also responsible to manage the communication link with a Gateway.

Each DH is bound to a Manager (DHM), which is an entity that has the power to manage it remotely through the primordial housekeeping application. Furthermore, it defines which Gateways can connect to the DH by providing them the proper configuration, which includes a ticket.

2.3. Gateway

The main function of a Gateway is to provide an indirect IP access of several different applications to IoT devices (through their DH and DD). For this, Gateways must be able to connect to an IP local area network (e.g., through Wi-Fi). This makes Gateways accessible to a wide variety of computing devices, such as smartphones, tablets, laptops or desktop computers.

A Gateway is the pivotal agent between Clients and IoT devices. For this purpose, Gateways run a set of services that enable Clients to find IoT devices, interact with devices and receive the data they provide.

Each Gateway runs a messaging service for connecting Clients and DDs. By default, each DD has a message queue, which is mainly used to implement request/response interactions. For data flows from DDs to specific Clients, the latter may create new, specific queues for that purpose.

Each Gateway uses an authentication mechanism to implement a first-order barrier to filter-out unwanted clients. For this purpose, it is bound to an A3C, which gives authorized clients a ticket for accessing the Gateway. The reason for using an extra authentication and authorization indirection through an A3C to gain access to a Gateway is to facilitate the management of many Gateways with a similar access control policy (pretty much as access networks use a backend AAA service for tackling requests originating from many Network Access Servers). Note, however, that a Gateway can implement its own A3C, if deemed necessary.

A Gateway can be discovered by using a broadcast discovery request in the IP local network where it is connected to (see Section 3.3). Clients would use the discovered Gateways to interact with the DDs they are connected to. DHMs would use the discovered Gateways to manage the DHs they can connect to.

2.4. AAA Controller (A3C)

An A3C is an entity that defines the access control policy for an IoT device (represented by its DD) or a Gateway. The access control is defined relatively to Clients, which are the entities willing to explore the IoT devices, or to other DHMs, when the later attempt to configure Gateways.

The main purpose of a Gateway A3C is to enforce a high-level access control over Clients regarding the access to the IoT infrastructure entry points (which are the Gateways). In fact, only Clients that are authorized to access a particular Gateway may then get access to one or more DDs running in the DHs that are connected to that Gateway. Clients without such authorization cannot interact with a Gateway in any way, not even to gather information relative to the IoT infrastructure connected to that Gateway.

The main purpose of a DD A3C is to control which Clients can access a given DD, regardless of the DH where it runs and the Gateway that gives access to that DH.

An A3C waits for access requests for the components it supervises, authenticates those requests, and uses some security policy for taking an access decision. A successful decision will trigger the issuing of a ticket. Furthermore, in the case of DD's, the security policy must yield decisions for each function of the API implemented by the DD. Such decisions should also be packed in the ticket.

A ticket is a cryptographically protected token that can only be produced by an A3C and interpreted by the target system that will enforce the access decisions taken by the issuer (see Section 4).

2.5. DH Manager (DHM)

A DHM is a component with a two-fold purpose: enable Gateways and DHs to link to each other and manage DHs. A DH management includes all its internal setup and configuration, which consists on the installation and removal of DDs and the update of its run-time support. This management, however, is not performed using a direct physical link, but instead remotely through a Gateway. Thus, a DHM is a special Gateway client, which it uses to connect a DH to the network and to manage it.

For enabling the coupling of Gateways and DHs, DHMs must first provide Gateways with the configuration parameters they need to establish a link with the DHs. With this configuration, Gateways and DHs can link to each other whenever possible (e.g., when close enough for a Bluetooth link). As previously referred, this configuration includes a ticket that the Gateway must use upon connecting to the DH to prove that it was properly mandated by the DH's DHM.

A DHM can provision many Gateways to connect to a single DH, this way enabling some degree of flexibility to the final network interconnection graph. Naturally, DHM can only configure the Gateways that authorize that operation, and, as previously referred that authorization is given by the Gateways' A3Cs.

2.6. Client

A Client is an application that makes use of an IoT device. A Client uses a Gateway (and, transparently, a DH) to reach the IoT device's DD. However, prior to doing so, a Client must acquire two tickets: one to get access to the Gateway, and another one to get access to the DD. The first ticket is provided by the Gateway A3C, while the second is provided by the DD A3C (see Section 4).

Clients are the entities responsible for any permanent storage of data originated in IoT devices. Such storage and posterior analysis are beyond our scope.

2.7. Case Study

Imagine a large factory area, with many different machines. Each machine requiring monitoring or remote manipulation is equipped with one or more DHs. The factory has a local network encompassing many Gateways, desktop computers and mobile devices. Desktop computers are used to run all the applications that control the exploitation of IoT devices (Clients, DHMs, and A3Cs), while mobile devices can use Web interfaces to interact with all those applications.

In this scenario, DHMs could be applications provided by the manufacturers of the DHs used, thus independent of the actual machines where DHs are deployed. These DHMs would be responsible for the installation of DDs, their upgrade, etc.

Regarding A3Cs, we could have one per internal department responsible for the operation of a subset of the machines, thus each machine would be controlled by an A3C of a particular department. The department responsible for a machine could then authorize particular Clients to access the DDs of the machine's DHs. Clients would be applications especially developed to control a set of machines, and only selected users could access each Client. A Client could use the same key pair for all its users (e.g., a key pair per role), or use a key pair per user.

Many kinds of Clients can be imagined for this example: a Client for monitoring energy consumption in the whole factory or a Client for a detailed operation and monitoring of a particular machine. Each Client can be bound to a set of DDs that it should work with, regardless of DHs running them and the Gateways linked to the DDs' DH.

3. Identification and Naming

3.1. Universal Unique Identifiers (UUIDs)

Except for the IoT devices, all the components of the architecture have an asymmetric key pair, and its public part is their base unique identifier. However, since public keys can be rather long bit arrays, we adopted a shorter version for implementing the UUID: a 128-bit MD5 digest of the public key value. These UUIDs, much shorter than public keys, are more suitable for manual configurations than the corresponding keys, and the probability of involuntary collision with 128-bit values is null in practice.

MD5 is very fast [19], which is an advantage for DHs and Gateways with low computational power. Despite being presently banned from cryptographic operations requiring collision resistance, MD5 can be used for UUID generation because finding a collision for a given public key is not only extremely hard [20] but is also of no use for an attacker, because it still needs to have the corresponding key pair to prove the UUID ownership.

3.2. Resources' Names

The names of resources are formed by a set of attributes, following a tag+value tuple format. The set of attributes of each resource is subdivided in two subsets: machine-to-machine (M2M) attributes and machine-to-human (M2H) attributes.

M2M attributes are relevant for the autonomous operation of the entire system. They include the identification of the resource (UUID or public key), the identification of its A3C (usually an UUID), communication addresses, API, etc.

M2H attributes, on the contrary, are meant to help humans to get contextual information about a resource (purpose, location, person responsible, etc.).

The set of all the attributes assigned to a resource may be defined by different providers, and each of them signs the ones they provide. Furthermore, it is possible to bind the attributes given by one provider to the ones given by another provider, as exemplified in the next section.

Clients and DHMs are clients of the system; they do not provide any resource or service to others. Therefore, their name is just their public key, which is defined by themselves.

3.2.1. IoT Devices

The name of an IoT device is provided by two components: its DD and the A3C of its DD. The attributes provided by each of them are signed with their private key ($K^{-}_{DD}$and K^{-}_{A3C}, respectively).

A DD defines several operational attributes of its device. Some attributes may be given by the device itself (e.g., manufacturer, model) and are optional. Other attributes are related with the DD itself. These may include optional values (e.g., driver manufacturer, build version, release date, etc.) and a set of mandatory M2M attributes required by the architecture, which include the following: the DD's public key K^{+}_{DD}; the UUID of its A3C; and the DD's API.

An A3C mainly defines M2H attributes for an IoT device. For instance, we can have dozens of temperature sensors in a building and it is easier for a human to recognize them by their location than by an UUID. We believe that such M2H attributes are easier to manage on A3Cs, which potentially have control over many devices and, thus, provide an aggregated view and management interfaces over all their names. Returning to the example, one can move one sensor to a new location without having to reconfigure it; we only must update its A3C.

IoT devices' names are maintained by a name service running on the Gateway that provides access to them (through a DH). Upon connecting to a DH, the Gateway fetches from all its DDs the attributes about themselves and their devices, and then fetches the attributes from their A3C. The relationship between both sets of attributes is assured by the A3C UUID that is part of the (signed) attributes provided by the DD.

The IoT devices' names on their Gateway name service can be updated by their providers, DD and A3C, by providing an ownership proof: a signature with K^{-}_{DD} or K^{-}_{A3C}, respectively. These names are also transient; they disappear if the Gateway is turned off or upon a DH disconnection. Thus, the naming system is well adapted to respond promptly to topology changes.

3.2.2. Gateways

The name of a Gateway is defined by itself and by its A3C. The Gateway defines mostly M2M attributes, while its A3C mainly defines M2H attributes (e.g., location). Among the M2M attributes of a Gateway, the following are mandatory: the Gateway's public key K^{+}_{G}; the UUID of its A3C; and the identifiers of several resources (or services): DH configuration service, name service and messaging service. The first can be used by DHMs to configure the Gateway to connect their DH. The second allows Clients and A3C to find the names of connected DHs or IoT devices. The third handles all the communication between Clients and A3Cs and the connected DHs and their DDs.

3.2.3. A3Cs

The name of an A3C is solely defined by itself. Again, it can have M2M and M2H attributes. Among the M2M attributes of an A3C the following are mandatory: the A3C's public key K^{+}_{A3C} and the identifier of several resources (or services): ticket issuing service and name service (for providing tickets and attributes relatively to the components it controls).

3.3. Resource Discovery

There are several resource-discovery protocols:

- Gateway discovery protocol. It is used by Clients and DHMs to find the Gateways that exist in the local area network. Clients need to use it prior to access target IoT devices; DHMs need to use it to configure the connection of Gateways to DHs and to manage their DHs.
- A3C discovery protocol. This protocol is used by Clients and DHMs to find the A3Cs they need to contact to get a ticket and a session key to access some other component (Gateway or DD). It is also used by Gateways to find the A3Cs that must provide DD's attributes to the Gateway name service.
- DH discovery protocol. It is used by DHMs to find out the Gateways connected to their DHs. This protocol involves DHMs and Gateways.
- IoT devices' discovery protocol. It is used by Clients, to find and get access to IoT devices. This protocol involves Clients and Gateways, since Gateways are the components that run a name service with the attributes of the IoT devices they give access to.

The first two discovery protocols use broadcast queries in the local area network. Such queries can look for a particular resource or for all available resources of some kind. In the second case, they can have many responses, one for available resource (Gateway or A3C). CoAP [21] and UPnP are two alternatives for implementing both protocols. Anyway, regardless of the exact protocol used to implement them, they will be as follows for an initiator I and a resource R:

$$\begin{aligned} &\mathrm{I} \rightarrow \star : \text{Optional set of resource attributes} \\ &\mathrm{I} \leftarrow \mathrm{R} : \text{Resource name} \end{aligned}$$

The second message may occur more than once when several resources' names match the attributes in the request. Each response yields the resource endpoint identifier: TCP/IP address, Uniform Resource Identifier (URI), etc.

The last two discovery protocols use a name service provided by Gateways. This name service is itself a resource that can be discovered using the Gateway discovery protocol.

4. Access Control

In our system, the access control to every component is supported by tickets and sessions.

Tickets are unforgeable, cryptographically protected data sets that are issued by A3Cs and analyzed by the components they target to: a DD (on behalf of an IoT device), a DH or a Gateway. They are presented in the beginning of a session to determine (1) if the session can be established and (2) the access rights of the session initiator relatively to the services provided by the target of interest. A ticket is owned by the entity that requested it, but the owner cannot change it.

Tickets are mainly used by Clients. First, a Client needs to have a ticket for establishing a session with a Gateway, both to access its services and, through them, to access a DH and its DDs. Second, a Client needs to have a ticket for accessing a DD.

Tickets are also used by DHMs, in their interaction with Gateways for setting up their connection with the DHs they manage, and in Gateways, for proving their authorization to connect to a given DH.

Tickets are formed by three parts: secret, public and signature. The secret part contains a secret (master) session key, encrypted with the public key of the ticket target. If the ticket is accepted by its target to initiate a session, the session key will be used thereafter to protect that session (for data encryption or integrity control). Master session keys are never used directly to protect the sessions' traffic flows; those are protected with session keys derived from the master during the session setup (see Section 4).

The public part contains data that does not require confidentiality, namely the identifier of the target, a pseudonym of the ticket owner, a set of rights over the target and a ticket validity date.

The signature part contains a signature of the issuing A3C, computed with its private key $\mathrm{K}^{-}_{\mathrm{A3C}}$, over the two previous parts. It makes a ticket suitable for validation and acceptance by the targets that recognize the signer as their A3C.

For privacy sake, tickets' targets never know the real identity of a session requester. In fact, all they need to assert is if the requester is authorized to setup the session and, afterwards, which functions of the target API can be called within that session. However, for logging purposes and a posteriori forensic analysis, the ticket contains a pseudonym of its owner, generated by the ticket issuer. Such pseudonym can be used to associate operations executed within a session to a particular entity, with the help of the ticket issuer.

Ticket Fetching and Session Setup Protocols

The ticket fetching protocol is a two-message protocol that allows an entity to get a ticket for a target using the A3C of the later (see Figure 2). The protocol assumes that all requests are genuine, therefore it does not validate if the request is not a replay or if it comes from the claimed requester. This is not a problem, though, since the reply is of no use for rogue requesters.

The session creation protocol (see Figure 2) is a three-way handshake protocol that uses a ticket and two random values, R_1 and R_2. At the end of the protocol both extremes know that the session was established because they both know, and know that the other knows, a session key K' derived from the ticket master session key K. Its last message is fundamental to ensure that targets do not create a session for an attacker using a stolen ticket. Key K' is the one that should be used to protect interactions within the session.

This session key setup protocol does not provide Perfect Forward Secrecy (PFS), since K' is reveled if K is known. However, if PFS is a requirement, the protocol can be modified to agree on a K' using a Diffie-Hellman algorithm with ephemeral private keys and K can be used to authenticate the peers.

These protocols are described assuming an RSA-style public key encryption. If using elliptic curve cryptography, which usually relies on Diffie-Hellman key agreements to implement public key encryptions, the tickets and protocol messages would be slightly different, but the main concepts would remain.

C	Generates a random R_1
C $\rightarrow$ A3C	K^+_C, UUID_T, $\{R_1\}_{\mathrm{K}^+_\mathrm{A3C}}$
A3C	Checks if UUID_C can access UUID_T, recovers R_1 with $\mathrm{K}^-_\mathrm{A3C}$, generates a random R_2, computes $K = R_1 \oplus R_2$ and generates $\mathrm{T}_\mathrm{A3C}\,[\mathrm{C}\rightarrow\mathrm{T}, K]$ with K^+_T
C $\leftarrow$ A3C	$\{R_2\}_{\mathrm{K}^+_\mathrm{C}}$, $\mathrm{T}_\mathrm{A3C}\,[\mathrm{C}\rightarrow\mathrm{T}, K]$, $\mathrm{K}^+_\mathrm{A3C}$
C	Recovers R_2 with K^-_C, computes $K = R_1 \oplus R_2$

C $\rightarrow$ T	$\mathrm{T}_\mathrm{A3C}\,[\mathrm{C}\rightarrow\mathrm{T}, K]$, $\mathrm{K}^+_\mathrm{A3C}$, R_1
T	Checks if $\mathrm{K}^+_\mathrm{A3C}$ matches its $\mathrm{UUID}_\mathrm{A3C}$, validates ticket with $\mathrm{K}^+_\mathrm{A3C}$, recovers K from the ticket secret part with K^-_T, generates a random R_2 and computes $K' = \mathrm{digest}(K, R_1, R_2)$
C $\leftarrow$ T	R_2, $\{R_1\}_{K'}$
C $\rightarrow$ T	$\{R_2\}_{K'}$

Figure 2. Ticket issuing (**top**) and session setup (**bottom**) protocols for a Client C and a target T. $\mathrm{T}_\mathrm{A3C}\,[\mathrm{C}\rightarrow\mathrm{T}, K]$ means a ticket issued by A3C for allowing C to access T with session key K; $\{x\}_y$ means x encrypted with key y; K^-_Z and K^+_Z are the private and public keys of Z's asymmetric key pair, respectively.

In the description above, we assumed that an A3C makes a grant/deny ticket issuing decision based on the Client public key (or UUID). However, the A3C may require further information about the Client in such a decision-making process. In that case, it should be able to correlate the Client public key with extra Client identification attributes, which may be provided by centralized or distributed identity managers (e.g. based on blockchains, as in [22]). This topic, however, is out of the scope of our architecture.

5. Network Bootstrap and Usage

When the system initiates, its configuration is the following:

- DHs have their key pair (K^-_DH, K^+_DH) and the $\mathrm{UUID}_\mathrm{A3C}$ of their A3C. Each DH is connected to a set of IoT devices, but no DDs are installed; and they are disconnected from Gateways;
- DHMs have their key pair ($\mathrm{K}^-_\mathrm{DHM}$, $\mathrm{K}^+_\mathrm{DHM}$) and the operational parameters that enable Gateways to connect to the DHs they manage. DHMs also have the code of the DDs that should run on their DHs;
- A3Cs have their key pair ($\mathrm{K}^-_\mathrm{A3C}$, $\mathrm{K}^+_\mathrm{A3C}$);
- Gateways have their key pair (K^-_G, K^+_G) and the $\mathrm{UUID}_\mathrm{A3C}$ of their A3C;
- Clients have their key pair (K^-_C, K^+_C);
- DHMs, A3Cs and Gateways are connected to the same local area network.

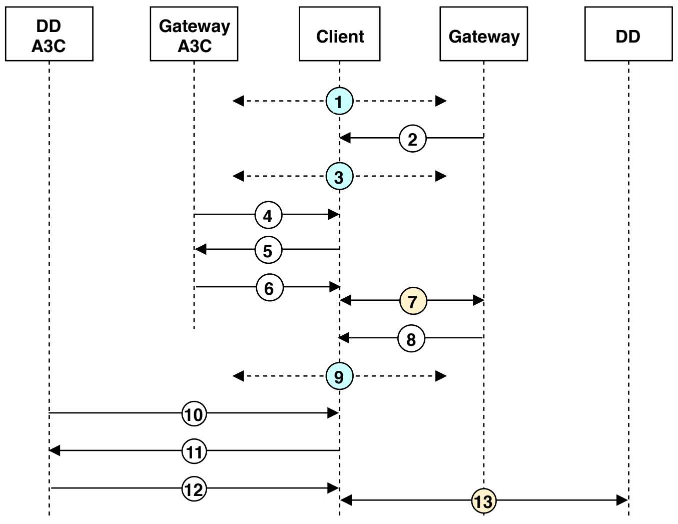
The A3Cs may not have from the start the identifiers of the authorized ticket requesters. In fact, they may include an interface for allowing humans to add those identifiers when using them. However, for the rest of this text we will assume that those identifiers are already known by A3Cs.

Starting from this initial configuration, the network will automatically take 3 steps for establishing an operational configuration. First, DHM will search for existing Gateways, will get an access token from their A3C, will setup sessions to Gateways and will upload to them the DHs connection configuration. Within this configuration goes a ticket that enables the Gateway to prove, to a DH, that it was authorized by its DHM to establish a connection with it. Second, Gateways and DHs establish a link with each other and set up a session. DHs only establish sessions with Gateways authorized by their DHMs. Third, each DHM finds out which Gateway is connected to one of its DHs, checks the DDs installed in each of its DHs and installs the missing DDs. In this step, DHMs can be alerted by Gateways upon their connection to DHs.

Client-DD Session Establishment

A Client needs to establish two sessions prior to being able to interact with an IoT device (through its DD): first, a session between the Client and the Gateway that provides access to the DH were the device exists; and second, a session between the Client and the DD that controls the device. Furthermore, a Client may have to discover several components in the process, namely the Gateway, the Gateway's A3C and the DD's A3C.

Therefore, if a Client C wants to find and access a given IoT device's DD, the complete protocol goes as shown in Figure 3. This protocol has several broadcast discoveries (steps 1, 2, 3, 4, 9 and 10) that can be omitted if the resource endpoint is already known. Steps 5 and 6 can also be omitted if C already has a ticket for G. Therefore, in the best case the establishment of a session between C and a (not used before) DD takes only 5 steps (7, 8, 11, 12 and 13).



1: Gateway discovery

2: K^{+}_{G}, $UUID_{G\ A3C}$

3: Gateway A3C discovery

4: $K^{+}_{G\ A3C}$

5: Request ticket for G

6: $T_{G\ A3C}\left[C \rightarrow G, K_{C,G}\right]$

7: Create session with $T_{G\ A3C}\left[C \rightarrow G, K_{C,G}\right]$, discover IoT device's DD

8: DD attributes, including K^{+}_{DD} and $UUID_{DD\ A3C}$

9: DD A3C discovery

10: $K^{+}_{DD\ A3C}$

11: Request ticket for DD

12: $T_{DD\ A3C}\left[C \rightarrow DD, K_{C,DD}\right]$

13: Create session with $T_{DD\ A3C}\left[C \rightarrow DD, K_{C,DD}\right]$

Figure 3. Complete protocol for establishing a Client-DD session. Steps 1, 3 and 9 are broadcast discovery requests. Steps 7 and 13 encompass several messages (cf. Section 4); in step 13 messages flow through the Gateway.

6. Security Analysis

One goal of our architecture was to prevent IoT devices to send data to an uncontrolled set of recipients. That could be used to conduct orchestrated attacks against selected victims or to leak personal information related with the environment where the device is installed. However, this is not possible. The devices' DDs can only communicate with the exterior of their DH by using the DH's run-time support, which provides a messaging interface for sending or receiving messages from message queues managed by the Gateway actually connected to the DH. Thus, a DD cannot send data to arbitrary Internet locations, only to selected message queues created and managed by Gateways on behalf of Clients.

Another goal was to be able to tolerate untrustworthy components in the architecture, such as malicious Gateways, Clients and DDs.

Malicious Gateways could try to connect to DHs they are not authorized to interact with, in order to corrupt their system, install or remove DDs, interact with DDs running in a particular DH or leak or tamper the data exchanged with them by Clients. However, this is not possible. Gateways can only connect to the DHs they are authorized to (by their DHM), and only DHMs can manage the configuration of the DHs they are responsible for. Finally, only Clients authorized by the A3C of each individual DD can interact with them, and the data they exchange can be encrypted and authenticated end-to-end by DDs and their Clients.

Malicious Clients cannot interact with the IoT infrastructure at will. In fact, they do not have the power to configure Gateways, they need an authorization ticket from a Gateway's A3C to interact with the Gateway, and they need an authorization ticket from a DD A3C to interact with the DD.

Malicious DDs cannot leak arbitrary data to the Internet and cannot receive commands from unauthorized Clients, as we saw. Furthermore, they are limited in the actions they can perform on the DH they run, and they do not have any clue about the Clients they serve (because tickets carry a pseudonym of their owners), therefore they cannot engage in DoS attacks against selected Clients nor can they leak any useful identity information related with Clients.

The trust anchors of the whole system rely on the quality of the cryptographic algorithms used and on the correct behavior of DHs, their DHMs and A3Cs.

DHs have access to the IoT devices; therefore, they need to be trusted to deal properly with them. Furthermore, DHs need to deploy a secure run-time environment to prevent DDs from interfering with each other, to prevent DDs from accessing IoT devices they are not meant to use, and to prevent the keys used by themselves and by the DDs they run to be leaked.

DHMs have the responsibility to bootstrap and configure the whole infrastructure, therefore their correctness is critical. They manage and authorize the connection of Gateways to DHs and they configure DH through Gateways, including the installation of DDs.

Finally, A3Cs are responsible for enforcing authorization policies regarding the access to Gateways and DDs. The authorization policy is enforced by the provision of tickets that enable a mutual authentication between a subject (ticket owner) and a target, allow a proper session key setup between the subject and the target, and can even contain a detailed target API authorization policy.

In our architecture, we do not need public key certificates or PKIs. A certificate is a trustworthy way to bind a public key to a named entity when names are independent from keys (for instance, when names are meaningful to people). In our case, we do not need to use names that are meaningful to people, since we are dealing with M2M interactions, therefore our names (UUIDs) can be generated from public keys. If the generation function does not allow the discovery of many public keys for a single name (UUID), our system can easily, and trustworthy, check whether or not a given public key belongs to a UUID.

7. Related Work

There are significant architectural differences between our proposal and other proposed/existent IoT architectures. The most significant difference is the focus on security, which has only recently been given more relevance when designing such systems. We will focus our comparison with other proposals on five topics: authentication, authorization and key distribution, access control, name services, standard proposals and frameworks.

7.1. Authentication, Authorization and Key Distribution

Our system uses tickets, semantically similar to the ones of Kerberos [18], to enable target components to assess the authorization of the requester to perform a given action. The authentication of the requester is performed by ticket issuers. Furthermore, our tickets allow interacting components to initiate a session protected by a shared, symmetric session key.

This approach is not common in the IoT universe. In [23], the authors analyzed the use of OAuth 2.0 [13] and an external OAuth-based Authorization Server for issuing authorization assertions in CoAP and HTTPS interactions. Several other works analyzed the use of OAuth for IoT, but their approach is different. In [24], they use OAuth to provide user access to an entire IoT network, and not to manage authorizations among several components of the IoT network, as we do. In [25], they refine the previous approach and an Authorization Server, which is contacted by a client application in the first place, asks Resource Owners about authorizations to issue OAuth credentials. However, a Resource Owner is not similar to our A3Cs; it is just an entity that asks a human for access credentials (username and password) that are then given to the Authorization Server.

The OAuth 2.0 model is not similar to our tickets. OAuth uses authorization grants, which can be used to get access tokens for accessing protected resources. Access tokens need to be confidential, which means they need to be exchanged over an encrypted channel (usually HTTPS) and their owners do not need to prove their ownership. On the contrary, our tickets do not need to be exchanged over encrypted channels and their use requires a proof of ownership (see protocol on Section 4). This way, we do not impose the setup of a confidential communication for ensuring a protected exchange of authorization tokens, while we allow peers to derive a session key from tickets for securing their posterior interaction if deemed necessary. Furthermore, in OAuth 2.0 the authentication of many relevant stakeholders (grant and access token issuers, resource provider) is performed by HTTPS using X.509 public key certificates, while our session setup protocol performs a lighter, certificate-free mutual authentication. In other words, OAuth 2.0 cannot work without HTTPS, while we can.

In [26], the author proposed a solution for end-to-end security under CoAP communication. Their framework comprises three elements: the IoT Application (corresponds to our Client), an IoT Broker and the IoT devices. It uses attribute-based encryption to encrypt AES message keys. Before communication, it implies the existence is an issuing phase, where the attributes of each entity are defined and certificates are issued to each entity with their attributes, and an attribute exchange phase, where the sender requests the attributes of the receivers to select the set of attributes to use. The entities satisfying the same given set of attributes are then able to communicate securely. However, it requires a complete trust on a central Certification Authority, which we do not.

In [27], the unique Device ID of sensors is used to generate a key pair to establish mutual authentication between devices and services. It uses asymmetric key encryption to share a symmetric session key for message transfer and a master key repository to generate, store and distribute the nodes' key pairs and the symmetric session keys. However, it requires a complete trust on that repository, which we do not.

7.2. Access Control

XACML [28] is an access control architecture that is very often used to implement access control policies in IoT system (e.g. [29,30]). In XACML, access control policies are interpreted by a PDP (Policy Decision Point), which provides a verdict for a PEP (Policy Enforcement Point) that stands in the way between the service requester and the service provider. By default, for each request the PEP must collect a verdict from the PDP. In our case, an A3C acts as a PDP that provides a verdict once per issued ticket (and posterior related session), and the ticket consumers (DDs and Gateways) use that verdict to derive several other decisions regarding each access within the session. In [29], on the contrary, the authors use an assertion to get access to a device (and the assertion issuing depends on a macroscopic positive verdict), but are devices themselves that use local policies and parameters to authorize each request. We believe that our approach scales better, since we can centralize policy definitions on A3C components, rather than spreading them throughout all DDs. Furthermore, identity-based access control decisions can be carried away in A3Cs, which are the entities that know the real identity of ticket requesters, while in [29] the identity of requesters may have to be revealed to devices.

Databox [31] is an architecture that uses a networked device (databox) to mediate the access to individuals' personal data to third party applications (data processors). Databoxes aggregate personal data from many sources (mobile devices, domestic meters, TVs, etc.). Compared to our system, databox is a bundle of all our components except DHM's and A3Cs.

7.3. Name Services

The names of resources in our system are distributed among all components and, apart from a few M2M mandatory attributes, they have a high degree of freedom as there is no unique naming schema (as in [32], with LDAP [33]). Since there is no central directory system to describe all resources, there is no central point of failure regarding resource discovery. Alternative approaches, such as using a Distributed Hash Table for storing resource names (as in [34]) creates another layer of complexity with no benefit for our target deployments.

7.4. Standard Proposals

There are several standard proposals for implementing IoT deployments, but the ways they tackle security is usually not organized in a coherent proposal.

OneM2M [35] describes many security mechanisms that can be used in an IoT infrastructure, but does not provide rules to explore those mechanisms. Furthermore, in OneM2M there is no notion of security control such as the one we can implement with our A3C components, which is very convenient for centralizing the definition and enforcement of access control policies.

FIWARE [36] focuses more on the security of data produced by IoT devices and stored in data silos, such as cloud providers. Such problem is complementary to ours, since we care, first, about which applications gather IoT data in the first place, and not with what happens with it next. Furthermore, we are concerned with the security of other kinds of IoT devices, such as actuators, which, in the case of FIWARE, is handled by unspecified Generic Enablers.

7.5. Frameworks

In [37] the authors surveyed the eight major IoT Frameworks, including AWS IoT, Azure and SmartThings, for their security features.

AWS supports mutual authentication of IoT devices, which can be done using X.509 certificates (the most used), AWS IAM users, groups and roles, and AWS Cognito identities. The authorization is policy-based, with the rules mapped to each certificate. It allows the owners of devices to define their own rules. Regarding, secure communications, all traffic is encrypted over TLS.

Azure IoT, authentication and secure communication is based on TLS, with every entity owning a X.509 certificate. Authorizations and access control is policy-based and uses the Azure Active Directory.

Finally, Smart Things uses OAuth for device authentication and to authorize the access of Smart Things Platform to the capabilities of devices. Authorization and access control use policies governed by a capability model. Secure communication of the devices with the gateway (hub) is encrypted using 128-bit AES using the Z-Wave or the ZigBee protocol.

Compared to our proposal, we do not need X.509 certificates and the policy rules are managed by A3Cs, which act as device owners. Finally, we do not need TLS, namely its handshake protocol, for implementing secure communication (but its secure transport protocol can be used in our system, though). Smart Things' capabilities are similar to our tickets. Finally, in our system the security of the link between a Gateway and a DH is defined by the DH's DHM and is independent of the end-to-end security that we can have between DDs and Clients. In other words, we do not associate end-to-end security with link security; they are independent.

8. Conclusions and Future Work

We described an architecture for managing and accessing IoT devices with a strong focus on privacy and security.

Regarding privacy, our proposal prevents IoT devices from sending data to arbitrary locations; they can only send it to authorized Clients and through authorized Gateways. Furthermore, IoT devices cannot leak identity-related information about who access their information or controls their activity.

Regarding the definition and enforcement of access control policies relatively to IoT devices, a major advantage of our proposal is a clear division of roles: A3Cs deal with policies, while the components that provide connectivity (Gateways and DHs) and access to IoT devices (DDs) deal with enforcement. Furthermore, the number of A3Cs is flexible and can be adjusted for many different scenarios. For instance, we can have a single A3C for an entire network, or we can have one for each component that requires an A3C.

Another advantage of our system is the use of asymmetric cryptography for authenticating components. Furthermore, we do not need certificates nor a PKI; a simple pre-distribution of identifiers derived from public keys is enough to recognize authorized credentials.

In terms of scalability and fault tolerance, the system is very much distributed, but also capable of automatically manage the connectivity of the necessary components. There are no central services, which could represent a threat in terms of fault tolerance, and the system is capable of bootstrapping itself from a very simple and reduced initial configuration.

As future work, we plan to fully implement and test this architecture, in particular in terms of performance. For that, we plan to use CoAP for supporting lightweight interactions between components and elliptic curves' for implementing lightweight asymmetric cryptographic transformations.

Author Contributions: Conceptualization, A.Z., H.G., J.A. and C.O.; validation, A.Z., H.G., J.A. and C.O.; investigation, A.Z., H.G., J.A. and C.O.; writing–original draft preparation, A.Z.; writing–review and editing, A.Z., H.G., J.A. and C.O.; supervision, A.Z. and H.G.; project administration, A.Z.; funding acquisition, A.Z.

Funding: This research was funded by Portuguese National Funds through the FCT—Foundation for Science and Technology, in the context of the projects CMU/CS/0042/2017 and UID/CEC/00127/2019.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Whitmore, A.; Agarwal, A.; Da Xu, L. The Internet of Things—A survey of topics and trends. *Inf. Syst. Front.* **2015**, *17*, 261–274. [CrossRef]
2. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
3. Ibarra-Esquer, J.; González-Navarro, F.; Flores-Rios, B.; Burtseva, L.; Astorga-Vargas, M. Tracking the Evolution of the Internet of Things Concept across Different Application Domains. *Sensors* **2017**, *17*, 1379. [CrossRef] [PubMed]
4. Russell, B.; Van Duren, D. *Practical Internet of Things Security*; Packt Publishing Ltd.: Birmingham, UK, 2016.
5. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
6. Zhang, Z.; Cho, M.C.Y.; Wang, C.; Hsu, C.; Chen, C.; Shieh, S. IoT Security: Ongoing Challenges and Research Opportunities. In Proceedings of the IEEE 7th International Conference on Service-Oriented Computing and Applications (SOCA 2014), Matsue, Japan, 17–19 November 2014; pp. 230–234.
7. Paul, A.; Jeyaraj, R. Internet of Things: A primer. *Hum. Behav. Emerg. Technol.* **2019**, *1*, 37–47. [CrossRef]

8. Sikder, A.K.; Petracca, G.; Aksu, H.; Jaeger, T.; Uluagac, A.S. A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications. *arXiv* **2018**, arXiv:1802.02041.
9. Alrawi, O.; Lever, C.; Antonakakis, M.; Monrose, F. SoK: Security Evaluation of Home-Based IoT Deployments. In Proceedings of the IEEE Symposium on Security and Privacy (SP 2019), San Francisco, CA, USA, 20–22 May 2019.
10. Nassiri, A. IoT and DDoS Attacks: A Match Made in Heaven. A10 Networks, Inc., 2019. Available online: https://www.a10networks.com/blog/iot-and-ddos-attacks-a-match-made-in-heaven (accessed on 14 September 2019).
11. Kambourakis, G.; Kolias, C.; Stavrou, A. The Mirai botnet and the IoT Zombie Armies. In Proceedings of the IEEE Military Communications Conference (MILCOM 2017), Baltimore, MD, USA, 23–25 October 2017; pp. 267–272.
12. Staff, A. IoT and DDoS: Cyberattacks on the Rise. A10 Networks, Inc., 2018. Available online: https://www.a10networks.com/blog/iot-and-ddos-cyberattacks-rise (accessed on 14 September 2019).
13. Hardt, D. *The OAuth 2.0 Authorization Framework*; RFC 6749; Internet Engineering Task Force: Fremont, CA, USA, 2012.
14. Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3*; RFC 8446; Internet Engineering Task Force: Fremont, CA, USA, 2018.
15. Rescorla, E. *HTTP over TLS*; RFC 2818; Internet Engineering Task Force: Fremont, CA, USA, 2000.
16. Tschofenig, H.; Fossati, T. *Transport Layer Security (TLS)/Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things*; RFC 7925; Internet Engineering Task Force: Fremont, CA, USA, 2016.
17. Ylonen, T.; Lonvick, C. *The Secure Shell (SSH) Protocol Architecture*; RFC 4251; Internet Engineering Task Force: Fremont, CA, USA, 2006.
18. Neuman, C.; Yu, T.; Hartman, S.; Raeburn, K. *The Kerberos Network Authentication Service (V5)*; RFC 4120; Internet Engineering Task Force: Fremont, CA, USA, 2005.
19. Aumasson, J.P.; Neves, S.; Wilcox-O'Hearn, Z.; Winnerlein, C. BLAKE2: Simpler, Smaller, Fast as MD5. In Proceedings of the 11th International Conference on Applied Cryptography and Network Security (ACNS 2013), Banff, AB, Canada, 25–28 June 2013; LNCS 7954, pp. 119–135.
20. Turner, S.; Chen, L. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*; RFC 6151; Internet Engineering Task Force: Fremont, CA, USA, 2011.
21. Shelby, Z.; Hartke, K.; Bormann, C. *The Constrained Application Protocol (CoAP)*; RFC 7252; Internet Engineering Task Force: Fremont, CA, USA, 2014.
22. Coelho, P.; Zúquete, A.; Gomes, H. Federation of Attribute Providers for User Self-Sovereign Identity. *J. Inf. Syst. Eng. Manag.* **2018**, *3*, 32. [CrossRef]
23. Cirani, S.; Picone, M.; Gonizzi, P.; Veltri, L.; Ferrari, G. IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sens. J.* **2015**, *15*, 1224–1234. [CrossRef]
24. Emerson, S.; Choi, Y.K.; Hwang, D.Y.; Kim, K.S.; Kim, K.H. An OAuth based Authentication Mechanism for IOT Networks. In Proceedings of the 2015 International Conference on Information and Communication Technology Convergence (ICTC 2015), Jeju, South Korea, 28–30 October 2015; pp. 1072–1074.
25. Sciancalepore, S.; Piro, G.; Caldarola, D.; Boggia, G.; Bianchi, G. OAuth-IoT: An access control framework for the Internet of Things based on open standards. In Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC 2017), Heraklion, Greece, 3–6 July 2017; pp. 676–681.
26. Choi, J.; In, Y.; Park, C.; Seok, S.; Seo, H.; Kim, H. Secure IoT framework and 2D architecture for End-To-End security. *J. Supercomput.* **2018**, *74*, 3521–3535. [CrossRef]
27. Sridhar, S.; Smys, S. Intelligent Security Framework for IoT Devices: Cryptography based End-To-End security Architecture. In Proceedings of the International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 19–20 January 2017; pp. 1–5.
28. Hsieh, G.; Foster, K.; Emamali, G.; Patrick, G.; Marvel, L. Using XACML for embedded and fine-grained access control policy. In Proceedings of the IEEE International Conference on Availability, Reliability and Security (ARES 2009), Fukuoka, Japan, 16–19 March 2009; pp. 462–468.
29. Seitz, L.; Selander, G.; Gehrmann, C. Authorization framework for the internet-of-things. In Proceedings of the IEEE 14th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM 2013), Madrid, Spain, 4–7 June 2013; pp. 1–6.

30. Atlam, H.F.; Alassafi, M.O.; Alenezi, A.; Walters, R.J.; Wills, G.B. XACML for Building Access Control Policies in Internet of Things. In Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018), Funchal, Portugal, 19–21 March 2018; pp. 253–260.
31. Crabtree, A.; Lodge, T.; Colley, J.; Greenhalgh, C.; Glover, K.; Haddadi, H.; Amar, Y.; Mortier, R.; Li, Q.; Moore, J.; et al. Building accountability into the Internet of Things: The IoT Databox model. *J. Reliab. Intell. Environ.* **2018**, *4*, 39–55. [CrossRef] [PubMed]
32. Hai, L.; Chunxiao, F.; Yuexin, W.; Jie, L.; Lilin, R. LDAP-based IOT Object Information Management Scheme. *J. Logist. Inform. Serv. Sci.* **2014**, *1*, 11–22.
33. Sermersheim, J. *Lightweight Directory Access Protocol (LDAP): The Protocol*; RFC 4511; Internet Engineering Task Force: Fremont, CA, USA, 2006.
34. Paganelli, F.; Parlanti, D. A DHT-Based Discovery Service for the Internet of Things. *J. Comput. Netw. Commun.* **2012**, *2012*, 1–11. [CrossRef]
35. Swetina, J.; Lu, G.; Jacobs, P.; Ennesser, F.; Song, J. Toward a standardized common M2M service layer platform: Introduction to oneM2M. *IEEE Wirel. Commun.* **2014**, *21*, 20–26. [CrossRef]
36. García Vázquez, A.; Soria-Rodriguez, P.; Bisson, P.; Gidoin, D.; Trabelsi, S.; Serme, G. FI-WARE Security: Future Internet Security Core. In *Towards a Service-Based Internet, Proceedings of the 4th European Conference (ServiceWave 2011), Poznan, Poland, 26–28 October 2011*; Abramowicz, W., Llorente, I.M., Surridge, M., Zisman, A., Vayssière, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; LNCS 6994, pp. 144–152.
37. Ammar, M.; Russello, G.; Crispo, B. Internet of Things: A survey on the security of IoT frameworks. *J. Inf. Secur. Appl.* **2018**, *38*, 8–27. [CrossRef]