LIAM: An Architectural Framework for Decentralized IoT Networks

Piet De Vaere
Department of Computer Science
ETH Zürich
Zürich, Switzerland
piet.de.vaere@inf.ethz.ch

Adrian Perrig
Department of Computer Science
ETH Zürich
Zürich, Switzerland
adrian.perrig@inf.ethz.ch

***Abstract*—Today's IoT deployments commonly resemble walled gardens: they are closed ecosystems in which manufacturers maintain significant control over devices after they have been deployed. This is typically the result of a centralized design approach where devices heavily rely on a monolithic, vendor-operated cloud service. We propose a distributed architecture that liberates these devices—and their data—by considering IoT devices as first-class network citizens and by grouping them in trusted network zones. These network zones support the devices contained in them by allowing tasks to be delegated from the device to the zone. However, devices are considered to be independent by default, and a task is only delegated when it is impossible or undesirable for the device to perform this task itself. We demonstrate how our architecture allows for novel access-control methods and context-dependent network views.**

I. INTRODUCTION

Recently, the Internet of Things (IoT) has been gaining significant traction. That is, more and more devices—informally called *things*—are being connected to the Internet, providing new functionalities and promising enhanced convenience. However, today's IoT platforms still face many challenges and are by no means a mature technology. In fact, Gartner states that IoT platforms are currently at their "peak of increased expectations" and expects 5 to 10 years before the technology matures [1].

One of the main shortcomings of today's IoT platforms is their lack of decentralization. Current market forces have led to the development of centralized, cloud-supported IoT architectures. While these architectures are easy to design and manage, their design comes with three inherent shortcomings: First, it leads to devices and data being locked in vertical silos. These silos can typically only be connected using non-standard interfaces at the application layer. Second, supporting cloud services usually require privileged access to the devices they assist. As these devices often reside in private locations, this requires trusting the cloud operator (typically the device vendor). Third, when cloud services cease to offer their services (e.g., because of liquidation or planned obsolescence), the functionality of the devices they assist decreases—in the worst case to that of a paperweight.

In this paper, we propose LIAM[1]: an architectural framework for decentralized IoT networks. LIAM avoids the shortcomings outlined above by considering IoT devices as first-class network citizens that engage in direct end-to-end communication with others. This is a vision that is shared by both the World Wide Web Consortium (W3C) in the Web of Things (WoT) working group [2] and the Internet Engineering Task Force (IETF) in the Constrained RESTful Environments (CoRE) working group [3]. In fact, supporting such an architecture was one of the reasons for IPv6's vast address space [4].

Concretely, LIAM considers devices to be independent by default, and to only delegate the specific functions for which it is impossible or undesirable for the device to perform these functions itself. Further, when functions are delegated, they are not delegated to a centralized cloud service but to a trusted LIAM zone. These zones group devices which are under the same administrative control and replace the centralized cloud services seen in current architectures. Similarly to cloud services, LIAM zones do not suffer the same resource constraints as IoT devices. In order to construct larger IoT networks, logical links between LIAM zones can be created.

LIAM is explicitly designed as a framework; that is, many aspects of its operation are left open to enable flexibility and adaptability to different application scenarios. This allows LIAM to maximally profit from current and future efforts to standardize IoT protocols. Moreover, it makes LIAM independent from the—often tedious—semantic standardization process [5], [6].

In this work, we define zone services for two device functions: device announcement and access control. We choose these services because they are essential in almost all IoT environments. Specifically, device announcement is required because the quantity and limited interfaces of IoT devices make manual device identification impractical. Likewise, access control is indispensible because of the sensitive nature of most IoT devices: not only do they expose private data, but they may also have to ability to perform security-sensitive actuation.

Because of their importance, we consider device announcement and access control to be core services of the LIAM architecture. Each LIAM-enabled device is able, but not obligated,

[1] LIAM stands for "Liam Includes Access Management". We discuss the rationale behind this name in Section IX.

to delegate these functions to its LIAM zone. However, as LIAM is designed as an open system, additional zone services can be added in the future.

The main contributions of this paper are:

- We present LIAM, a novel IoT architecture that considers IoT devices as first-class network citizens and eliminates the need for centralized cloud services.
- We present LIAM-Auth, an access-control mechanism that leverages the LIAM architecture and simplifies access-control management in IoT environments.
- We demonstrate how LIAM-Auth can be applied to a directory service to create context-dependent network views.
- We evaluate LIAM-Auth by implementing *permission tokens*, an extension to OAuth, and evaluating this implementation.

II. BACKGROUND

Being a framework, LIAM can make optimal use of existing protocols and standards. In this section we first introduce OAuth, which will serve as the base for the access-control zone service. Next, we look at the most prominent IoT directory proposals, which we will leverage to create a directory service.

A. OAuth and ACE-OAuth

OAuth is a widely deployed web authentication framework. It allows the owner of a web resource to grant access to this resource to a third party without requiring the resource owner to share its credentials with this third party. Moreover, OAuth allows the resource owner to limit the access rights of the third party to a subset of its own rights. Two versions of OAuth have been standardized by the IETF: OAuth 1.0 [7] and OAuth 2.0 [8]. This document considers only the latter.

Figure 1 displays an abstract OAuth protocol flow. In Steps 1 and 2 the resource owner issues an access grant to the client (i.e., the third party who wants to access the resources). This process and grant can take multiple forms, as is discussed below. Next, in Steps 3 and 4 the client exchanges this authorization grant for an access token at the authorization server. Finally, in Steps 5 and 6 the client uses this access token to obtain access to the protected resource from the resource sever.

The base specification of OAuth defines four types of authorization grants. The most common of these is the "authorization code" grant. When using this grant type, Step 1 consists of the client requesting the grant by redirecting the resource owner (typically through browser redirection) to the authorization server, where the resource owner is prompted to approve the grant (e.g., "Do you want to grant Hooly access to your Friendface pictures?") If the grant is approved, the authorization server redirects the resource owner back to the client and includes an authorization code which functions as the authorization grant. Besides the four standard grant types, the OAuth specification also specifies a mechanism to support additional grant types.

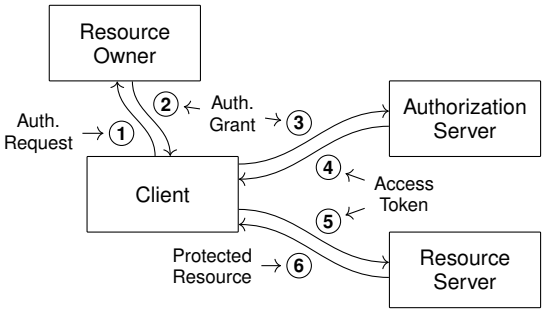

Fig. 1: The abstracted OAuth 2.0 flow. 1) The client performs an authorization request. 2) The resource owner issues an authorization grant. 3) The client presents the authorization grant and requests an access token. 4) The authorization server issues an access token. 5) The client presents the access token and requests access to the protected resource. 6) The protected resource is served to the client.

The IETF's Authentication and Authorization for Constrained Environments (ACE) working group is currently standardizing ACE using OAuth (ACE-OAuth) [9], an adaptation of OAuth for constrained environments such as IoT networks. ACE-OAuth adheres to existing specifications where possible, while specifying extensions where needed. For example, OAuth handles token revocation by issuing both short-lived access tokens and long-lived refresh tokens. Once an access token has expired, the client can obtain a new access token by presenting its refresh token to the authorization server. If the client's access is revoked, the authorization server will not issue a new access token. While this approach is suitable for typical web environments, it requires the authorization and resource server to maintain a synchronized clock, which may not be possible in constrained environments. Therefore ACE-OAuth specifies a nonce-based mechanism to ensure the freshness of access tokens.

B. W3C WoT and IETF CoRE Directories

Both the IETF (in the CoRE working group) and the W3C (in the WoT working group) are currently standardizing resource description and discovery in IoT environments. Concretely, the IETF proposes the CoRE Link Format [3] while the W3C proposes the Thing Description [10] format. Both of these standards make it possible to describe the hosted resources and interfaces of a server, but while the CoRE Link Format has a low-level focus, Thing Descriptions focus on providing rich device metadata. An example Thing Description is shown in Figure 2.

Furthermore, both resource description standards also specify a directory service: the IETF specifies the CoRE Resource Directory [11] and the W3C the Thing Directory [2]. These directory services host multiple device descriptions in a central location, rather than on each device individually.

```
{ "@context": "https://www.w3.org/2019/wot/td/v1",
  "id": "urn:dev:ops:34532-LiamBulb-3445",
  "title": "Liam Demo Bulb",
  "securityDefinitions": {
    "basic_sc": {"scheme": "basic", "in":"header"}
  },
  "security": ["basic_sc"],
  "properties": {
  "status" : {
    "type": "string",
    "forms": [{"href": "https://2001::42/status"}]
    }
  },
  "actions": {
  "toggle" : {
    "forms": [{"href": "https://2001::42/toggle"}]
} } }
```

Fig. 2: An example Thing Description for a lamp which exposes a "status" and "toggle" interface [10].

III. LIAM ARCHITECTURE

A. Overview

The main building blocks of the LIAM architecture are virtual network zones. Devices are grouped in these zones based on shared properties such as location, owner, function, or subnet. Zone operators are free to use any set of properties they desire.

Each network zone is identified by a zone URL pointing to a zone manager service. Additional URLs using different schemes may be assigned to the zone as well. When visiting a zone's URL, a zone description file is returned. This file is similar in nature to the Thing Description in the W3C WoT standard, and contains information about the LIAM zone and the services it provides. The zone description file is not a static file. Rather, its content and format will vary depending on the relationship between the zone manager and the requesting client.

Devices enter a zone by registering with the zone manager, resulting in the establishment of a trust relationship (see Section III-C). In order to construct larger networks, zone operators can establish associations between zones (see Section III-D). An example of such a network is shown in Figure 3. When two devices are in different zones but a path of associations exists between their zones, this path can be used to create a trust relationship between these devices. This relationship can then be leveraged for authentication and authorization (see Section IV-A).

Besides grouping devices, the main function of LIAM zones is to assist the devices they contain. Zones do this by allowing their member devices to delegate tasks to the zone. Contrary to today's IoT architectures, these delegations only take place when it is impossible or undesirable for the device to perform these tasks itself. It is important to note that, by default, devices are fully independent and that they only depend on the network zone for the specific tasks they delegate.

In this paper, we propose two zone services which are part of the core LIAM design: one for device announcements and one for access control. Moreover, LIAM is explicitly designed to allow additional zone services to be added in the future.

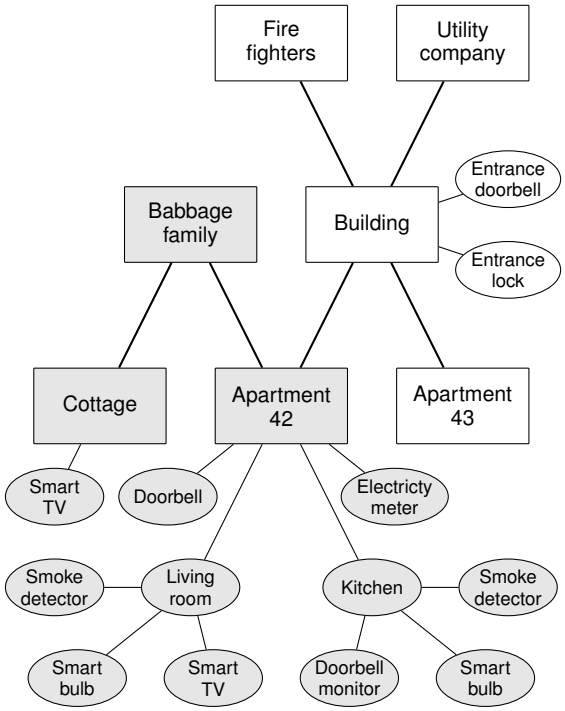

Fig. 3: A schematic example of a LIAM network. Rectangles represent LIAM zones, ovals represent directory entries. Lines indicate links between LIAM and the structure of the zone directories.

The use of network zones provides the following advantages over centralized architectures:

Efficiency: Network queries that only concern the local zone can remain within that zone. This reduces network load and latency.

Scalability: As multiple IoT networks can be interconnected, networks can grow very large. Zoning allows for network functions to scale horizontally rather than vertically.

Federation: Each zone can be under a different authority and can be administered independently.

Grouping: Zones create implicit logical groups of devices and other zones. These groups can be used when configuring access-control policies.

Flexibility: Devices are not bound to one specific service instance. Rather, they are free to join any network zone that offers the services they require.

B. Working Example

Figure 3 shows the LIAM network we will use as an example throughout this paper. It displays the private IoT network of the Babbage family in light grey. The Babbage family lives in apartment number 42. They run a LIAM zone on their home router to which they connect all the IoT devices in their home. The main URL of this zone

is `https://babbage.name/liam/apartment`, and a corresponding URL using the `coaps://` scheme[2] is defined. For convenience, they create logical living room and kitchen groups in the zone directory, in which they combine the devices in each of these rooms. During the weekend, Mr. and Ms. Babbage often spend time in their mountain cottage where they have an Internet-connected television. This television is a member of the LIAM zone managed by the router in the cottage. Because they want to be able to access their archive of recorded movies, both of their televisions should be in the same LIAM network. Therefore, they create the Babbage family zone which unites their cottage and apartment zones.

Some of the devices in the Babbages' apartment are general building utility devices (e.g., smoke detectors, doorbell monitor, electricity meter). In order to grant access to these devices to the building management, they have connected their apartment 42 zone to the building zone. The building zone is in turn connected to the fire-fighters and utitily-company zones to grant them access to the smoke detectors and electricity meter, respectively.

C. Device-Zone Interaction

Devices can join a LIAM zone by registering with the respective zone manager. This registration consists of the exchange of keying material between the device and the zone manager. Because this keying material will only be used for communication between the device and the zone manager, both symmetric and asymmetric keys can be used and a plethora of pairing techniques is available [13]–[17].

The LIAM zone maintains a set of attributes for each device. These attributes can range from the device's name and location to a detailed description of the device's interfaces or an internal trust level. Because these attributes are, in principle, only intended for zone-internal operations, they can be locally specified.

Once the device is registered, it will have a more privileged view of the zone description file discussed in Section III-A. This view will include information about the services offered by the zone and how to register for them. The format of the zone description can also be modified to best suit the client device. For example a resource-rich smart-TV may be served a rich JSON document while a constrained temperature sensor is served a file in CoRE Link Format.

In order to authenticate to the zone, standard TLS authentication is used. In the case of asymmetric keying material, authentication is performed through client certificates [18] or raw public keys [19]. In the case of symmetric keying material, the pre-shared key functionality of TLS is used [18]. The latter approach allows even constrained devices to use standard TLS without requiring them to perform asymmetric cryptographic operations.

[2]The Constrained Application Protocol (CoAP) is a protocol similar to HTTP optimized for constrained devices and networks. `coaps` indicates the use of CoAP over Datagram TLS (DTLS) [12].

D. Zone-Zone Interaction

Larger IoT networks can be created by combining multiple LIAM zones. This is done by creating point-to-point trust relationships between zones, which results in a graph-like network structure. An example of such a network can be seen in Figure 3. The graph-like structure of LIAM networks is one of its key strengths: whenever two nodes are a member of the same (connected) LIAM graph, an implicit trust relationship exists between them. Thus, when one explicit trust relationship is created, potentially many more implicit trust relationships are induced. These can then be leveraged to provide authentication as discussed in Section IV-A. Note that the strength of implicit trust relationships is dependent on the strength of the explicit trust relationships that induced it.

Zone linking is a process very similar to device registration. It consists of the exchange of keying material and the establishment of internal attributes for the remote zone. We explicitly do not specify a zone-linking protocol, as we expect domain-specific mechanisms to emerge. For example, the relationship between the building zone and apartment zones in Figure 3 could be based on the physical security of the cable connecting them. That is, an apartment building could provide one network socket in each residence for which it guarantees that the communicating party on the other end is the building's zone manager.

Once two zones are linked, they will provide each other privileged access to their zone description files. For example, a zone might provide a remote zone with permission tokens granting access to devices it manages. We discuss permission tokens in detail in Section IV-A.

IV. ZONE SERVICES

A LIAM zone provides zone services to its members. When it is impossible or undesirable for a device to perform a certain task and its LIAM zone operates a suitable zone service, the device can delegate this task to the zone manager.

Tasks might be more suitable to be executed by the zone manager rather than the IoT device for several reasons. For example, zone managers can run on corded devices with high-throughput network connectivity. This means that they do not suffer from the resource constraints which are common in IoT devices. Other notable reasons to delegate tasks include that it allows device owners to aggregate administrative tasks at the zone manager and that zone managers can leverage the trust relationships between zones to perform their functions more efficiently.

We demonstrate the utility of delegations by describing two LIAM zone services: an access-control and a directory service.

A. Access Control

Motivation. Access control is a prime candidate for delegation because IoT devices often do not have sufficient situational awareness for access-control decisions. For example, a low-power light sensor cannot be expected to have a full list of all authorized actors in its environment. Additionally,

managing and maintaining access-control lists on each individual device would be time consuming and cumbersome. Some devices might not even have sufficient memory to store their access policy. Moreover, access evaluation can be a complex and resource-intensive task. For example, it might require asymmetric cryptographic operations or the consultation of remote revocation lists.

In contrast, zone managers do not suffer from the same resource constraints as IoT devices. Moreover, they can leverage the trust relationships induced by the LIAM graph for access-control decisions. In order to demonstrate the power of LIAM-based access control, we design LIAM-Auth: a distributed authentication mechanism that leverages the LIAM graph to simplify access control.

Overview. LIAM-Auth is designed as an extension to OAuth that allows IoT devices to delegate access control to their zone manager. Similar to standard OAuth, this is accomplished through access tokens issued by the zone manager and accepted by the device. Additionally, LIAM-Auth allows zones to further delegate these access-control rights to other zones. This is accomplished through permission tokens (Ptokens), a new and delegable[3] type of OAuth authorization grant.

Ptokens are similar to access tokens in that they are associated with an access scope, but represent an authorization grant rather than an access capability. This is a fundamental difference and means that they must be used during communication with the authorization server rather than with the resource server. During this communication, they will be exchanged for an access token, which can then be used to access the protected resource.

A second fundamental difference between access tokens and Ptokens is that the holder of a Ptoken can generate a derived Ptoken with a subscope of the original Ptoken and can forward that token to another entity.

For example, consider that the electricity meter of apartment 42 in Figure 3 uses LIAM-Auth. The apartment 42 zone may then issue a Ptoken to the building zone, which may in turn issue a derived Ptoken to the utility company zone. The utility company can then use this Ptoken to derive Ptokens for its various services and employees. At each point in this process the delegating zone can narrow the access scope as appropriate. Assuming the final derived Ptokens contain the correct scope, they can then be used to obtain an access token to read out the electricity meter from apartment 42's authorization server. Finally, these access tokens can be used to obtain access to the electricity-meter readings. An abstract version of this flow is illustrated in Figure 4.

Delegation of Ptokens. The main difference between LIAM-Auth and OAuth is the ability to chain delegations. This ability enables three advantages: First, it improves the scalability of access control by allowing zone managers to perform coarse-grained access control (through access-control delegation). Nevertheless, zone managers retain the ability to perform fine-grained access control by issuing narrow-scoped tokens to local clients, and by performing additional checks when exchanging Ptokens for access tokens. For example, the apartment 42 zone in Figure 3 only needs to issue one Ptoken to the building zone which contains all access scopes related to building utilities. The building zone is then responsible for delegating the correct access scopes to the fire fighters, utility company, and its local clients. Nevertheless, the apartment 42 zone can still refuse to provide access tokens to clients, even if they are able to present a valid Ptoken.

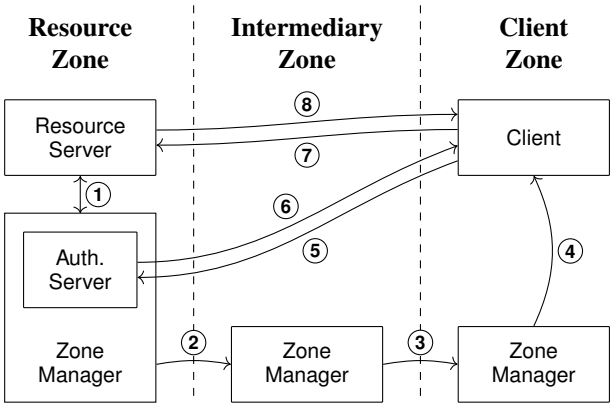

Fig. 4: The LIAM-auth flow: 1) The resource server negotiates access delegation with its zone manager. 2) The resource's zone manager issues a Ptoken to the intermediary zone. 3) The intermediary issues a delegated Ptoken to the client zone. 4) The client zone issues a delegated Ptoken to the client. 5) The client uses the Ptoken to request an access token from the authorization server in the resource zone. 6) An access token is issued to the client. 7) The client uses the access token to request the resource. 8) The resource is sent to the client.

Second, LIAM-Auth simplifies access-policy management by moving access-control decisions to administrators close to the client. These administrators have a better understanding of the client's role and its access requirements. For example, in the setting of Figure 3, the utility company is best suited to decide which of its employees or services should be granted access to the electricity-meter readings.

Third, LIAM-Auth makes the delegation of access rights explicit. When an entity is granted an access permission, it is effectively always able to delegate this permission (e.g., by operating a proxy service or by sharing its secret values). Making this ability explicit does not only lead to more transparency, but it also forces administrators to consider this behavior when managing access rules.

Relation with OAuth. LIAM-Auth is implemented by defining a new grant type for OAuth. This has a number of advantages. First, implementing a new OAuth grant type allows the resource server to be agnostic about the use of

[3]Although somewhat odd looking, all significant dictionaries list "delegable" as the derived adjective of "to delegate". Not "delegatable".

LIAM-Auth: it still verifies access tokens in the same way as before. This means that LIAM-Auth can be deployed without any changes to the resource servers. As resource servers are typically the most constrained nodes in IoT environments, and might be hard to update, this backwards compatibility significantly simplifies the deployment of LIAM-Auth for devices already supporting (ACE-)OAuth.

Second, because Ptokens represent an authorization grant rather than an access token, they do not give direct access to protected resources. This means that the authorization server can still perform additional checks before granting an access token and, if necessary, refuse to do so.

Third, using OAuth allows LIAM-Auth to inherit OAuth's solutions to common access control problems such as access revocation.

Implementation of Ptokens. We implement Ptokens as cryptographic proof-of-possession tokens. Further implementation information is discussed in Section V.

B. Directory Service with Context-Dependent Network Views

Motivation. Advertising their presence and functionality can be problematic for IoT devices because communication constraints might make direct peer-to-peer discovery impractical. For example, devices might be sleeping, or they might be connected to networks with high latency,[4] or they might lack support for efficient multicast communication.

Moreover, device-discovery mechanisms are usually limited to a single layer 2 or 3 network scope, which does not necessarily correspond to the set of entities with access rights to a device. Ideally, devices should be visible to any entity with access rights for it and invisible to all others.[5] Both of these difficulties can be overcome by operating a directory as a LIAM zone service.

Overview. The LIAM zone operates a directory server and allows zone members to insert records into this directory. LIAM is agnostic as to which directory standard is used, but we recommend the use of either the IETF's Resource Directory (RD) or the W3C's Thing Directory (TD). Besides listing descriptions of the interfaces to the various devices in the zone, zone administrators can also add auxiliary information to the directory, such as:

- a hierarchical directory structure;
- device annotations added by the zone based on the stored device attributes;
- references to directories of linked LIAM zones; or
- information about the environment in which the devices are placed and how the devices interact with that environment.

Because this directory server runs on zone infrastructure rather than the potentially constrained zone member devices, it does not suffer from the communication constraints outlined above. Moreover, as this directory service is operated by the zone manager, it can leverage the LIAM graph to perform access control to the directory by using LIAM-Auth. That is, it can distribute Ptokens to its neighbors granting them access to a subset of the directory. These neighbors can then create derived Ptokens—potentially with a narrower scope—and forward them to their neighbors, propagating access to the directory through the network.

[4]On low-power mesh networks with heavy duty cycling, RTTs in the order of tens of seconds are not unusual

[5]Also the permission to discover a device can be seen as a minimal access right.

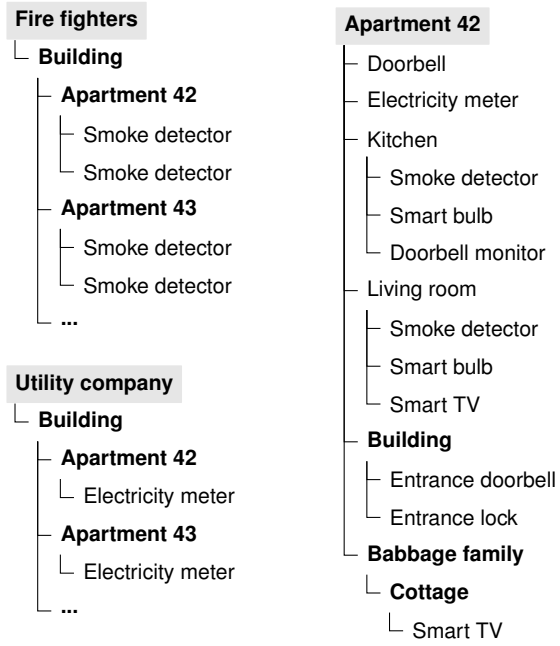

Fig. 5: Example of how the location in the LIAM graph determines the view of the network. The root of each tree corresponds to the vantage point of the client.

The result of this is that depending on the network position of a directory client, this client will receive a different set of Ptokens, resulting in a different view of the network. We call this a *context-dependent network view*, and demonstrate its effect in the example below.

Consider the LIAM network shown in Figure 3, in which all LIAM zones are operating a directory service. Within the Babbages' private network, the LIAM zones are fully trusted. Therefore, Ptokens which grant access to the full directory content are distributed between the grey nodes in the network. Conversely, the building zone is only minimally trusted and is given a Ptoken which only grants access to the building utilities in the directory, without displaying the directory's internal structure.

Similarly, the building zone issues a Ptoken to each apartment zone which allows listing of the building infrastructure devices (e.g., the entrance doorbell and lock). Moreover, from each Ptoken received from the apartment zones, the building zone creates a derived Ptoken granting access to the fire-prevention systems and forwards this token to the fire department. Analogously, the building zone also creates

Ptokens granting access to the electricity meters and forwards these tokens to the utility company.

Once all of these tokens have been distributed, each of the zones will possess Ptokens granting them access to different subsets of the directories. The effect of this is that when querying the directory service, each of them will receive a different set of answers, resulting in a different, context-specific, view of the network. As an illustration, the directory views of the fire fighters, utility company and apartment 42 are shown in Figure 5.

Note that, for simplicity, we have only discussed Ptokens granting access to specific parts of the device directory. In practice these Ptokens would also contain scopes allowing clients to interact with the devices themselves.

V. OAuth Permission Token Implementation

We now discuss the implementation of the permission tokens (Ptokens) introduced in Section IV. Ptokens represent a delegable access permission to a set of protected resources. When a token contains a delegated permission, we refer to it as a derived Ptoken. We refer to a token that is not derived from any other token as a base Ptoken.

A Ptoken implementation should have the following properties:

Verifiable: Given a Ptoken, it should be possible for a LIAM zone to efficiently verify if this token is, or is derived from, a base Ptoken issued by that zone.

Delegable: It should be possible for the holder of a Ptoken to delegate the rights granted by the Ptoken to a third party.

Scope refinement: It should be possible to narrow the scope of a Ptoken during the delegation process. It should never be possible to broaden the scope of a Ptoken.

Transparent: The issuer of a base token should be able to inspect the delegation chain when it receives a token derived from that base token.

Robust against disclosure: Leakage of a Ptoken should have minimal impact.

The last requirement is necessary because Ptokens are exposed to potential leakage as they move through the network (even when encrypted) and because a Ptoken—in contrast to an access token—can serve as the basis for additional tokens. Thus, the revocation of a single Ptoken can lead to entire sections of a LIAM network having reduced access rights.

Based on these requirements, we explore two Ptoken implementations: one based on asymmetric cryptography and one based on symmetric cryptography. We refer to these as asymmetric and symmetric Ptokens, respectively.

A. Notation

We use the following notation: K and K^{-1} denote a public and private asymmetric key, respectively; k denotes a symmetric key; $E_K(x)$ denotes encryption of x under K; $H(x)$ denotes the result of applying a one-way hash function to x; $\text{MAC}_k(x)$ denotes the Message Authentication Code of x using key k; and $a|b$ denotes a concatenated with b.

B. Asymmetric Ptoken Design

The asymmetric Ptoken is implemented as a cryptographically secured proof-of-possession token. That is, it is verified by a cryptographic operation rather than a database lookup and requires the token presenter to demonstrate knowledge of an asymmetric private key before it can use the token.

Cryptographically securing a token makes it possible for the receiver to verify the token with only local knowledge, which is required for the verifiability requirement. Using proof-of-possession tokens significantly reduces the effect of token disclosure as the Ptoken cannot be used without knowledge of the corresponding private key.

An asymmetric Ptoken consists of a sequence of token segments interleaved with signatures validating these segments. A base token (i.e., a token without delegations) has the structure:

$$\textbf{Token}_0 = \text{segment}_0 \mid E_{K_0^{-1}}(H(\text{segment}_0)),$$

where the issuing entity has the key pair (K_0, K_0^{-1}) and the token segment contains the following fields:

TokenID: an identifier unique to each token.

Delegation counter: a positive integer indicating the maximum allowed length of the delegation chain based on this token segment.

Name: a unique identifier of the token receiver.

Pubkey: the asymmetric public key (K_1) of the receiver. The token will be bound to the private key corresponding to this public key.

Scope: a set of access scopes to which this token grants access.

Time range: a time range during which the token should be accepted by the authorization server.

The validity of this token can be verified using the public key of the issuer, K_0.

In order to create a delegation, the token receiver can extend the base token by appending an additional segment and signing the resulting token with its private key. Assuming the receiver of the base token has the key pair (K_1, K_1^{-1}), the resulting token then has the following structure:

$$\textbf{Token}_1 = \text{token}_0 \mid \text{segment}_1 \mid E_{K_1^{-1}}(H(\text{token}_0 \mid \text{segment}_1)).$$

Additional delegations can be created by analogous extension of this token resulting in:

$$\textbf{Token}_i = \text{token}_{i-1} \mid \text{segment}_i \mid E_{K_i^{-1}}(H(\text{token}_{i-1} \mid \text{segment}_i)).$$

A token is considered valid if the following conditions hold:

- The signature chain is valid. That is, each segment_i is followed by a signature using the private key corresponding to the public key listed in segment_{i-1}. Or, if $i = 0$, by the private key of the base token issuer.
- For each $i > 0$ it holds that $\text{scope}_i \subseteq \text{scope}_{i-1}$, where scope_i is the scope listed in the i-th segment.

- For each $i > 0$ it holds that $0 \leq \text{counter}_i < \text{counter}_{i-1}$, where counter_i is the delegation counter value of the i-th segment.
- For each $i > 1$ it holds that time_i is contained in time_{i-1} where time_i is the time range of the i-th segment and that the current time is within the most constraining time range.

C. Symmetric Ptoken Design

Symmetric Ptokens have a similar structure as asymmetric Ptokens but are bound to a symmetric key rather than to an asymmetric key. The entity issuing a symmetric Ptoken maintains a master key k_M from which it derives a token-specific key k_0 as described below.

A symmetric Ptoken consists of a sequence of token segments terminated by an authentication tag. A base token (i.e., a token without delegations) has the following structure:

$$\textbf{Token}_0 = \text{segment}_0 \mid \text{MAC}_{k_0}(\text{segment}_0),$$

where k_0 is obtained as $H(\text{segment}_0|k_M)$, with the hash function being used as a forward-secure pseudorandom bit generator [20]. The token segment contains the following fields:

TokenID: an identifier unique to each token.
Delegation counter: a positive integer indicating the maximum allowed length of the delegation chain based on this token segment.
Name: a unique identifier of the token receiver.
Scope: a set of access scopes to which this token grants access.
Time range: a time range during which the token should be accepted by the authorization server.

When issuing this token, the issuing entity sends the token receiver both token_0 and $k_1 = H(k_0)$. In order to use token_0, the token presenter needs to cryptographically demonstrate knowledge of k_1. Note that because k_1 needs to be transported over the network, using symmetric proof-of-possession tokens provides less robustness against disclosure than using the asymmetric approach. However, securing a short key during transmission and storage is still easier than securing the entire token. Moreover, the key must not be transmitted when exchanging a Ptoken for an access token (Steps 5 and 6 in Figure 4), which is the only transmission in the life cycle of a Ptoken during which no pre-established keying material can be used.

In order to create a delegation, the token holder can extend the base token by removing the authentication tag, adding an additional segment, and reauthenticating the token as

$$\textbf{Token}_1 = \text{segment}_0 \mid \text{segment}_1 \mid \text{MAC}_{k_1}(\text{MAC}_{k_0}(\text{segment}_0) \mid \text{segment}_1).$$

It would then issue this token together with $k_2 = H(k_1)$ to the token receiver.

Additional delegations can be created through analogous extension of this token resulting in:

$$\textbf{Token}_i = \text{segment}_0 \mid \ldots \mid \text{segment}_{i-1} \mid \text{segment}_i \mid \text{MAC}_{k_i}(\text{tag}_{i-1} \mid \text{segment}_i),$$

where tag_{i-1} is the authentication tag of token_{i-1}. Intermediate authentication tags do not need to be included in the token, as they can be calculated by the verifier.

Token_i is issued together with (and bound to) key $k_{i+1} = H(k_i)$. Because of the structure of the keys, it is possible for the holder of k_i to calculate k_j for any $j \geq i$. However, as with every token derivation the included scope can be only narrowed, this is not an issue: knowledge of k_j never grants more privileges than knowledge of k_i.

A token is considered valid if the following conditions hold:

- The authentication tag of the token is valid. This implies that all intermediate authentication tags were also valid.
- For each $i > 0$ it holds that $\text{scope}_i \subseteq \text{scope}_{i-1}$, where scope_i is the scope listed in the i-th segment.
- For each $i > 0$ it holds that $0 \leq \text{counter}_i < \text{counter}_{i-1}$, where counter_i is the delegation counter value of the i-th segment.
- For each $i > 1$ it holds that time_i is contained in time_{i-1} where time_i is the time range of the i-th segment and that the current time is within the most constraining time range.

VI. Evaluation

We implement both the asymmetric Ptoken and symmetric Ptoken in Python and provide two sets of evaluations. In the first set, we benchmark the Ptokens in isolation (Section VI-A). For the second set we implement an HTTP server which performs Ptoken creation and verification in response to incoming requests (Section VI-B).

Our implementation uses SHA-256 as the hash algorithm, HMAC with 256-bit keys to generate authentication tags, and Ed25519 as the signature algorithm. All evaluations are performed using a single thread on an eight-core Intel i7-7820X 3.6 GHz processor with 32 GB of DDR4 2666 MHz memory.

A. Token Performance

We first discuss the performance of our Python Ptoken implementation in isolation. We benchmark the creation, derivation, and verification of Ptokens by measuring the time required to perform each of these operations 10^6 times. For each operation we execute the steps described below.

Token creation: A new Ptoken is created, serialized, and authenticated.
Token derivation: A base Ptoken is deserialized. Next, a derived token is generated, serialized, and authenticated.
Token verification: A base Ptoken is deserialized and its validity is verified.

Figure 6 displays the results of the benchmarks. In order to evaluate the effect of the token length on the verification times, we additionally perform 10^4 token validations

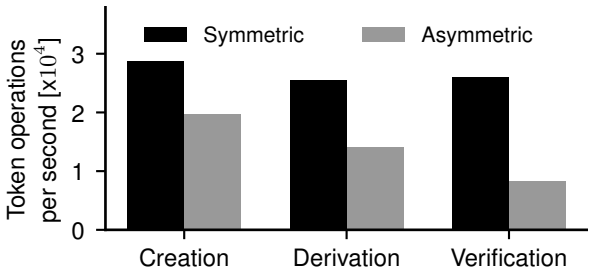

Fig. 6: Performance of the Ptoken implementations for the three main Ptoken operations.

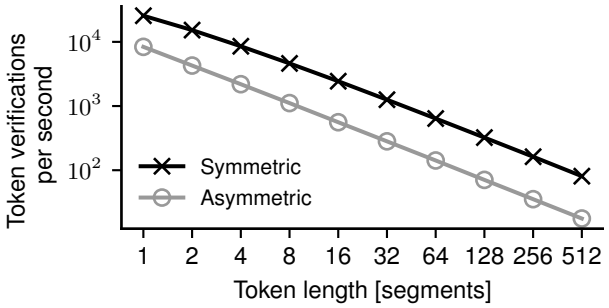

Fig. 7: The influence of the length of a Ptoken on its validation speed.

for tokens with length (expressed in number of segments) $2^n, n = \{0, 1, \ldots, 9\}$. The results are shown in Figure 7.

From Figure 6 we see that the processing speeds for symmetric and asymmetric Ptokens are within the same order of magnitude. However, tokens using symmetric cryptography outperform their asymmetric counterparts by a factor of 1.5 to 3, depending on the operation. These numbers fall in line with the expectations given the speed of HMAC and Ed25519 operations on the benchmark hardware. Figure 7 shows that token-verification times increase linearly with the token length (note the double-logarithmic axis).

B. HTTP Endpoint Performance

In order to gauge the performance impact of using Ptokens at an authentication endpoint, we have implemented an HTTP server using the Python Flask framework,[6] which issues and verifies Ptokens in response to incoming requests. We issued 10^5 token creation and verification requests for both Ptoken types. Furthermore, to obtain a baseline measurement we created a dummy endpoint which uses static tokens. When a client requests a token from this endpoint, a static string is returned. Similarly, when a client presents a token to this endpoint, the validity is verified by comparing the presented token to a static string.

As Ptokens act as proof-of-possession tokens, the authentication endpoint must verify ownership of the key bound to the presented token. Our HTTP server performs this verification

[6] `https://palletsprojects.com/p/flask/`

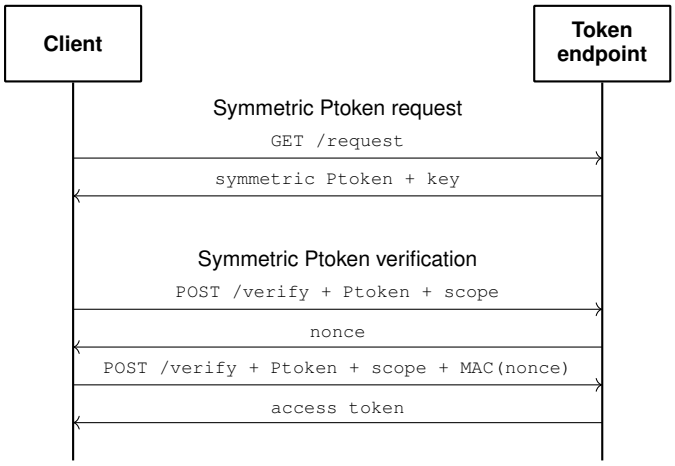

Fig. 8: The protocol flow of the HTTP symmetric Ptoken endpoint.

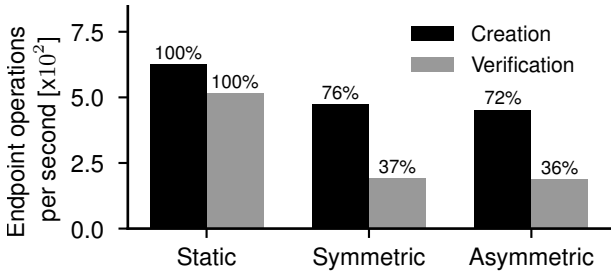

Fig. 9: Ptoken issuance and validation speeds of a Ptoken enabled HTTP endpoint.

using a nonce-based challenge–response protocol. Figure 8 illustrates this for symmetric Ptokens. The protocol for asymmetric Ptokens is similar but requires the client to attach its public key to the token request and uses a signature rather than an authentication tag in the verification phase. The results of the performance benchmark of the HTTP endpoint are show in Figure 9.

Figure 9 shows that the overhead of Ptoken creation during a token request is limited: only $24\,\%$ (symmetric) to $28\,\%$ (asymmetric) fewer tokens can be issued than when the server simply returns a static string. The performance impact for token verification is more significant, but can largely be attributed to the additional roundtrip required to verify knowledge of the key bound to the Ptoken. Note that we did not use TLS during these benchmarks. If we did, these performance penalties would likely be less significant, as additional round trips to establish the TLS connection would be required for all interactions. Moreover, when using asymmetric tokens, the challenge–response protocol can be eliminated by authenticating the client during TLS session establishment.

VII. Discussion

We designed LIAM as a framework for decentralized IoT networks that overcomes the inherent shortcomings of centralized IoT architectures: vertical silos are prevented by encouraging direct device-to-device communication; cloud services are replaced by trusted network zones; and rather than being bound to a specific cloud service, devices can join any zone offering the services they require. Additionally, LIAM allows for new access-control methods and introduces context-dependent network views.

Both these access-control methods and context-dependent network views rely on a new OAuth authorization grant type using permission tokens (Ptokens). We implemented these Ptokens as proof-of-possession tokens which provide verifiability through cryptographic operations; allow for delegation and scope refinement with a transparent delegation chain; and provide robustness against disclosure by requiring the token presenter to demonstrate knowledge of a cryptographic key.

Evaluating our Ptoken implementation shows that their use results in only a small performance reduction, and is likely to be faster than even basic database lookups. Given that asymmetric Ptokens do not require the transmission of private keying material over the network during the issuing process (in contrary to symmetric Ptokens), but result in only a moderate performance decrease over symmetric Ptoken, the use of asymmetric Ptokens is generally preferable. Moreover, LIAM-Auth is designed not to require constrained devices to perform operations on Ptokens. This considerably reduces the significance of the performance decrease introduced when using asymmetric Ptokens, further making the case for their use.

We also find that—unsurprisingly—Ptoken verification times increase linearly with the token length. However, as we expect delegation chains to be short in practice, this is acceptable. This expectation comes from the observation that long delegation chains dilute the strength of trust relationships: you trust your daughter more than you trust the husband of the cousin of the banker of your sister. Therefore, we expect long delegation chains to be treated similarly to unprivileged access in practice.

This trust dilution is exacerbated by the fact that LIAM does not use a global root of trust for authentication: that is, linking cryptographic keys to identities is done through claims made in Ptokens by remote zones. Providing global identities in LIAM, for example, through the use of a public key infrastructure, is considered as a future research direction.

Other future work includes creating additional zone services such as centralized device management or the offloading of resource intensive (cryptographic) operations. Finally, we see research opportunities in exploring how LIAM interacts with the underlying network. For example, by provisioning devices with network configuration, or by automatically creating communication channels between devices (e.g., by opening up firewalls or creating network tunnels).

VIII. Related Work

A. Access Management

Even though the need for access control and data privacy has been clear from the very beginning of ubiquitous computing [21], today's IoT deployments are plagued by access-control and over-privilege issues [22]–[26]. There is a significant body of recent work that discusses access control for cloud-based IoT platforms [27]–[30].

The IETF ACE working group is currently standardizing ACE-OAuth, an adaptation of classic OAuth to constrained environments [9]. Other mechanisms to use OAuth in IoT environments have also been proposed [31]–[33]. However, similarly to standard OAuth, these do not allow delegation chaining.

Most other proposals for capability-based access control in IoT networks either do not discuss delegation chaining [34]–[36] or require the involvement of the authorization server for each delegation [37], [38].

Gusmeroli et al. propose a capability token that allows delegation chaining without the involvement of an authentication server [39]. However, their mechanism is unsuited for constrained environments. For example, it requires token revocations to be distributed to resource servers which must cache them.

B. IoT Architectures

A number of IoT architectures have been proposed. Many authors propose the use of an "IoT Hub". This hub isolates the IoT device from the Internet, and aims to provide manageability, connectivity, and security [40]–[47]. However, these proposals retain the use of centralized architectures, and therefore do not solve their fundamental problems.

The IETF's CoRE working group is designing an IoT architecture that follows the Representational State Transfer (REST) model. In their architecture, devices expose a RESTfull interface that provides direct access to the resources they represent (e.g., a light switch, or a temperature sensor). Resource discovery can be performed by visiting a well-known URL on each network device [3]. Alternatively, a resource directory can collect information about resources held on other devices, and allow other hosts to query for this information [11].

The WoT working group of the W3C envisions an architecture similar to that of the IETF's CoRE working group [2], but at a higher level. The main element of the WoT architecture is the Thing Description format [10], which can be used to describe the network interfaces of IoT devices. Thing descriptions use the Resource Description Format (RDF) data model [48], which can be used to model rich metadata about devices and their surroundings. In order to ensure backwards compatibility, Thing Descriptions do not need to be hosted on the IoT devices they describe.

What all of these proposals lack, is clear support for interconnected IoT networks. That is, they are designed for single network deployments, without a clear vision of how to scale to distributed deployments at Internet scale.

C. Function offloading

There have been multiple proposals to offload functionality from constrained devices. In fact, the current common practice in IoT deployments is to offload as much functionality as possible from the IoT devices to a centralized cloud. The rising need for low-latency data processing led to the development of fog computing, an architecture in which the tasks typically executed on a centralized cloud are executed on network edge devices instead [49].

Other forms of function offloading include (D)TLS handshake offloading [50], and the offloading of digital signatures [51].

IX. Conclusion

Today's IoT platforms typically resemble centralized silos. While easy to develop and maintain, these architectures lead to vendor dependence. In this paper we present LIAM, a decentralized IoT architecture that groups devices in zones, allows devices to offload tasks to their zone, and allows logical linking between zones to construct larger networks.

Because zone links imply a trust relationship, LIAM is intrinsically suited to support authentication and access management. This is an important feature not shared by other distributed IoT architectures, which typically mimic the Web.

In contrast to access control on the Web, where data is public-by-default and add-hoc security mechanisms are used to provide access control when needed, access management is needed in almost every IoT architecture, as data is private-by-default and should only be accessible to entities who have been explicitly granted access. LIAM recognizes this difference, and requires entities to have explicit consent for every operation, including device discovery. We reflect this in LIAM's name: Liam Includes Access Management.

References

[1] Gartner, Inc., "Gartner hype cycle for emerging technologies," Aug. 2018. [Online]. Available: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/

[2] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of things (wot) architecture," W3C, W3C Candidate Recommendation, May 2019. [Online]. Available: https://www.w3.org/TR/2019/CR-wot-architecture-20190516/

[3] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format," RFC 6690, Aug. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6690.txt

[4] F. Kastenholz and D. C. Partridge, "Technical Criteria for Choosing IP The Next Generation (IPng)," RFC 1726, Dec. 1994. [Online]. Available: https://rfc-editor.org/rfc/rfc1726.txt

[5] J. Jimenez, H. Tschofenig, and D. Thaler, "Report from the Internet of Things (IoT) Semantic Interoperability (IOTSI) Workshop 2016," RFC 8477, Oct. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8477.txt

[6] C. Groves, L. Yan, and Y. Weiwei, "Overview of IoT semantics landscape," Huawei Technologies, techreport, 2016. [Online]. Available: https://www.iab.org/wp-content/IAB-uploads/2016/03/IoTSemanticLandscape_HW_v2.pdf

[7] E. Hammer-Lahav, "The OAuth 1.0 Protocol," RFC 5849, Apr. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5849.txt

[8] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749, Oct. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6749.txt

[9] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)," Internet Engineering Task Force, Internet-Draft draft-ietf-ace-oauth-authz-24, Mar. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ace-oauth-authz-24

[10] S. Käbisch, T. Kamiya, M. McCool, and V. Charpenay, "Web of things (wot) thing description," W3C, W3C Candidate Recommendation, May 2019. [Online]. Available: https://www.w3.org/TR/2019/CR-wot-thing-description-20190516/

[11] Z. Shelby, M. Koster, C. Bormann, P. V. der Stok, and C. Amsüss, "CoRE Resource Directory," Internet Engineering Task Force, Internet-Draft draft-ietf-core-resource-directory-23, Jul. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory-23

[12] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[13] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks," in *International workshop on security protocols*. Springer, 1999, pp. 172–182.

[14] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong, "Talking to strangers: Authentication in ad-hoc wireless networks." in *NDSS*, 2002.

[15] M. Cagalj, S. Capkun, and J. . Hubaux, "Key agreement in peer-to-peer wireless networks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 467–478, Feb 2006.

[16] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: using camera phones for human-verifiable authentication," in *2005 IEEE Symposium on Security and Privacy (S P'05)*, May 2005, pp. 110–124.

[17] C. Castelluccia and P. Mutaf, "Shake them up!: A movement-based pairing protocol for cpu-constrained devices," in *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '05. New York, NY, USA: ACM, 2005, pp. 51–64. [Online]. Available: http://doi.acm.org/10.1145/1067170.1067177

[18] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[19] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," RFC 7250, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7250.txt

[20] M. Bellare and B. Yee, "Forward-security in private-key cryptography," in *Topics in Cryptology — CT-RSA 2003*. Springer Berlin Heidelberg, 2003, pp. 1–18.

[21] M. Weiser, R. Gold, and J. S. Brown, "The origins of ubiquitous computing research at PARC in the late 1980s," *IBM Systems Journal*, vol. 38, no. 4, pp. 693–696, 1999.

[22] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Security implications of permission models in smart-home application frameworks," *IEEE Security & Privacy*, vol. 15, no. 2, pp. 24–30, mar 2017.

[23] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[24] C. Cimpanu, "Vulnerabilities found in GE anesthesia machines," ZDNet.com, Jul. 2019. [Online]. Available: https://www.zdnet.com/article/vulnerabilities-found-in-ge-anesthesia-machines/

[25] Z. Whittaker, "Flaws in a popular GPS tracker leak real-time locations and can remotely activate its microphone," May 2019. [Online]. Available: https://techcrunch.com/2019/05/10/gps-trackers-flaw/

[26] L. O'Donnell, "2 million IoT devices vulnerable to complete takeover," Apr. 2019. [Online]. Available: https://threatpost.com/iot-devices-vulnerable-takeover/144167/

[27] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized action integrity for trigger-action IoT platforms," in *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, 2018.

[28] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards providing contextual integrity to appified IoT platforms," in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017.

[29] R. Schuster, V. Shmatikov, and E. Tromer, "Situational access control in the internet of things," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*. ACM Press, 2018.

[30] S. Ravidas, A. Lekidis, F. Paci, and N. Zannone, "Access control in internet-of-things: A survey," *Journal of Network and Computer Applications*, vol. 144, pp. 79–101, oct 2019.

[31] P. Fremantle, B. Aziz, J. Kopecký, and P. Scott, "Federated identity and access management for the internet of things," in *2014 International Workshop on Secure Internet of Things*, Sep. 2014, pp. 10–17.

[32] D. Rivera, L. Cruz-Piris, G. Lopez-Civera, E. de la Hoz, and I. Marsa-Maestre, "Applying an unified access control for iot-based intelligent agent systems," in *2015 IEEE 8th international conference on service-oriented computing and applications (SOCA)*. IEEE, 2015, pp. 247–251.

[33] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari, "Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios," *IEEE sensors journal*, vol. 15, no. 2, pp. 1224–1234, 2014.

[34] P. N. Mahalle, B. Anggorojati, N. R. Prasad, R. Prasad *et al.*, "Identity authentication and capability based access control (iacac) for the internet of things," *Journal of Cyber Security and Mobility*, vol. 1, no. 4, pp. 309–348, 2013.

[35] J. L. Hernández-Ramos, A. J. Jara, L. Marin, and A. F. Skarmeta, "Distributed capability-based access control for the internet of things," *Journal of Internet Services and Information Security (JISIS)*, vol. 3, no. 3/4, pp. 1–16, 2013.

[36] L. Seitz, G. Selander, and C. Gehrmann, "Authorization framework for the internet-of-things," in *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2013, pp. 1–6.

[37] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad, "Capability-based access control delegation model on the federated iot network," in *The 15th International Symposium on Wireless Personal Multimedia Communications*, Sep. 2012, pp. 604–608.

[38] R. Xu, Y. Chen, E. Blasch, and G. Chen, "A federated capability-based access control mechanism for internet of things (iots)," 2018. [Online]. Available: https://doi.org/10.1117/12.2305619

[39] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the internet of things," *Mathematical and Computer Modelling*, vol. 58, no. 5-6, pp. 1189–1205, 2013.

[40] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "Iot gateway: Bridgingwireless sensor networks into internet of things," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Dec 2010, pp. 347–352.

[41] S. Cirani, G. Ferrari, N. Iotti, and M. Picone, "The IoT hub: a fog node for seamless management of heterogeneous connected smart objects," in *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*. IEEE, jun 2015.

[42] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy mediators: Helping iot cross the chasm," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications - HotMobile '16*, ACM. ACM Press, 2016, pp. 39–44.

[43] A. K. Simpson, F. Roesner, and T. Kohno, "Securing vulnerable home IoT devices with an in-hub security manager," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, mar 2017.

[44] M. Miettinen, P. C. van Oorschot, and A. Sadeghi, "Baseline functionality for security and control of commodity iot devices and domain-controlled device lifecycle management," *CoRR*, vol. abs/1808.03071, 2018. [Online]. Available: http://arxiv.org/abs/1808.03071

[45] R. Ko and J. Mickens, "DeadBolt: Securing iot deployments," in *Proceedings of the Applied Networking Research Workshop on - ANRW '18*. ACM Press, 2018.

[46] M. Leo, F. Battisti, M. Carli, and A. Neri, "A federated architecture approach for internet of things security," in *2014 Euro Med Telco Conference (EMTC)*. IEEE, 2014, pp. 1–5.

[47] S. M. R. Islam, M. Hossain, R. Hasan, and T. Q. Duong, "A conceptual framework for an iot-based health assistant and its authorization model," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2018, pp. 616–621.

[48] G. Schreiber and Y. Raimond, "RDF 1.1 primer," W3C, W3C Note, Jun. 2014, http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[49] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[50] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards viable certificate-based authentication for the internet of things," in *Proceedings of the 2nd ACM workshop on Hot topics on wireless network security and privacy - HotWiSec '13*. ACM Press, 2013.

[51] R.-H. Hsu, J. Lee, T. Q. S. Quek, and J.-C. Chen, "Reconfigurable security: Edge-computing-based framework for IoT," *IEEE Network*, 2018.