# DBAC: Directory-Based Access Control for Geographically Distributed IoT Systems

Luoyao Hao, Vibhas Naik, Henning Schulzrinne

Department of Computer Science, Columbia University, New York, NY, USA
Email: {lyhao, hgs}@cs.columbia.edu, vn2302@columbia.edu

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796804

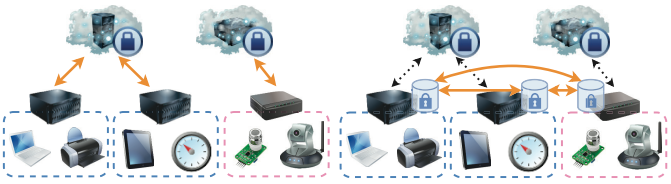
***Abstract*—We propose and implement Directory-Based Access Control (DBAC), a flexible and systematic access control approach for geographically distributed multi-administration IoT systems. DBAC designs and relies on a particular module, IoT directory, to store device metadata, manage federated identities, and assist with cross-domain authorization. The directory service decouples IoT access into two phases: discover device information from directories and operate devices through discovered interfaces. DBAC extends attribute-based authorization and retrieves diverse attributes of users, devices, and environments from multi-faceted sources via standard methods, while user privacy is protected. To support resource-constrained devices, DBAC assigns a capability token to each authorized user, and devices only validate tokens to process a request.**

## I. Introduction

With the goal of scaling IoT systems beyond a single home or enterprise, large-scale infrastructures incorporating heterogeneous IoT devices are proposed to bridge IoT platforms and facilitate cross-domain services [1]–[3]. With such systems, applications are able to run on top of a variety of devices from different manufacturers and platforms.

Indeed, heterogeneous scenarios in smart cities require ever-increasing interactions across multiple administrations [4], [5]. Owing to edge computing, operating systems and computing resources are geographically distributed near end users, through which applications can communicate, retrieve and aggregate data from multiple distributed sources [6]–[8]. As an example, a Building Automation System (BAS) should be operated both locally and remotely by various users (residents, superintendents, maintenance staff, visitors, etc.) and interact with multiple external systems. Home assistant devices can be connected to the BAS by the residents, sensor data are visible to the superintendents through local monitoring system, and each type of sensors should be monitored by the maintenance personnel of corresponding service providers through their platforms.

Despite the tremendous convenience and efficiency for data access, such systems still rely on centralized identity-based access control, which lacks efficiency and neglects the semantic-rich nature of IoT devices. For access control in large-scale IoT systems, identities may not matter all that much, as low-level identities are usually not interpreted in interactive rules. IoT access policy will mainly rely on subject roles and multi-faceted descriptors, e.g., custodians in charge can access the video surveillance in the warehouse during their shifts.



(a) Traditional architecture. (b) DBAC architecture.

Fig. 1. DBAC attaches and federates distributed directories for access control, unlike traditional solutions that divide systems and heavily rely on the cloud.

Besides, the centralized access control likely becomes the bottleneck, as cloud-based authorization sacrifices the system responsiveness and user experience [9].

Commercial access control solutions for IoT are mostly at the per-device granularity, and users are granted all-or-nothing access through identity-based authorization [10]. As devices often provide more than one function, the coarse-grained strategies are ill-suited in many multi-user scenarios. Cloud-supported solutions (e.g., AWS IoT [11], Apple HomeKit [12], Samsung SmartThings [13]) allow role-based or tag-based policies. Attribute-Based Access Control (ABAC) [14]–[17] and Capability-Based Access Control (CapBAC) [9], [18], [19] are favored by the research community for their higher flexibility and finer granularity. However, most mechanisms either assume an omnipotent centralized cloud, or conversely, make the system completely decentralized and hard to track permissions. Major challenges such as where and how to retrieve diverse attributes and how to accommodate resource-constrained devices, are not properly addressed in distributed scenarios, given that existing solutions focus on a single administration. Thus, generalizing existing mechanisms to distributed IoT systems is still an open problem.

Rather than treating access control as a standalone function, we explore a fundamental question: *what can IoT systems provide for access control?* Our answer is therefore a systematic approach called DBAC (for Directory-Based Access Control, or an interesting recursive acronym "DBAC Beyond ABAC or CapBAC"). In DBAC, we attach a well-designed directory module to each autonomous system and federate directories to support attribute-based authorization, as shown in Fig. 1. Access control mechanisms are no longer the servants for the systems. Instead, they build on and benefit each other.

Directory services are widely adopted by many large-scale systems to identify and look up digital resources effi-

ciently [20], [21]. Naturally, directories are brought to IoT systems to facilitate device management, service discovery, and data sharing [3], [22]–[24]. Unlike traditional directories that are entirely accessible or protected behind a single enterprise, IoT directories require more complicated visibility restrictions [3], [24]. DBAC also fills up this gap.

DBAC follows four fundamental design principles: (1) users do not have to trust or provide credentials to all of the systems; (2) user privacy should be well-protected, and private information should not be disclosed without permission; (3) the mechanism should be fine-grained and flexible, yet privileges should remain traceable; (4) subjects and objects can be centrally managed, and attribute provision should be in a distributed way.

To achieve these goals, DBAC applies a hybrid approach integrating both ABAC and CapBAC. Inspired by the rules of the physical world, access policies can be specified by a variety of attributes at the per-capability granularity. Attributes are categorized and retrieved from multiple sources including subjects, objects, third parties, contexts, and history records. Federated identity management and retrieval of subject attributes are supported by OpenID Connect [25], and third-party attribute providers are incorporated through OAuth2 [26]. Metadata (e.g., location, properties, functions) of devices are managed and discovered through distributed directories. Therefore, the overall access control is decoupled to two phases: access to device information stored in directories and further access to interfaces of devices, and both are subject to attribute-based policies. The authorization process burdening resource-constraint devices can be delegated to DBAC servers, and then a capability token will be assigned to authorized users for further interacting with devices. We prototype DBAC, deploy it on AWS EC2 servers, and demonstrate the feasibility by evaluating it with common attribute-based policies.

We make the following contributions:

- We identify the access control needs for large-scale distributed IoT systems and propose a solution from a systematic perspective.
- We design and implement DBAC to address two interrelated problems: access control for IoT directories and directory-base access control for IoT devices.
- We integrate flexible solutions, ABAC and CapBAC, with guaranteed granularity and manageability. Distributed attribute retrieval and federated identity management are addressed using standardized approaches.

The rest of this paper is organized as follows. Section II introduces the background and identifies some open issues. Section III overviews the design and structure of the system. Section IV describes the topology and functionalities of the federated directory systems. Section V presents the details of the two-phase approach. Section VI evaluates and demonstrates the efficiency of our prototype. Section VII discusses potential concerns. Finally, Section VIII concludes the paper.

## II. Background

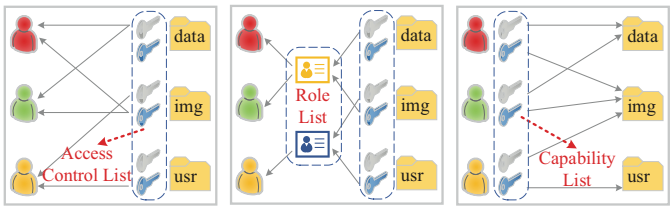
This section reviews access control mechanisms and directory services for IoT.

### A. Access Control Mechanisms

Traditional computer systems maintain an Access Control List (ACL) that associates user identities with privileges. With more entities joining the system, the size of ACL explodes, which takes up resources and makes the systems less efficient. To cope with it, Role-Based Access Control (RBAC) is widely adopted in today's cloud systems. RBAC groups user identities into a manageable number of roles and enforces policies per role instead of per user. However, to meet the flexible access control needs of IoT scenarios, a plethora of roles have to be created, making RBAC face the same issue as ACL [27].

For large-scale IoT systems, however, the identity of individual device may not matter, and the role of users may not be sufficient. Access control can be based on other criteria, such as location, proximity, date and time. To implement even a simple access policy resembling the physical world, e.g., a light switch can be turned on or off when a subject is in close proximity, we need a more generic and flexible access control solution. Two fine-grained access control methods, Attribute-Based Access Control (ABAC) [14]–[17] and Capability-Based Access Control (CapBAC) [9], [18], [19] have the potentials to create flexible access rules for IoT.

ABAC takes multi-dimensional characteristics into consideration. It collects attributes to determine an access request. In principle, the attributes can be user information, any types of records, location, device information, environment information, logs, etc. The attributes can be provided by users and internal or external systems. ABAC is considered a superset of RBAC, as RBAC creates roles taking attributes of subjects.




(a) ACL (b) RBAC (c) CapBAC

Fig. 2. Comparison of ACL, RBAC, and CapBAC. Traditional ACL resides on the administration, which maps privileges to each user. RBAC groups users to a limited number of roles and bundles policies to different roles. CapBAC assigns capability tokens that demonstrate the ownership of privileges to users, who then present correct tokens to access resources.

Fig. 2 summarizes how ACL, RBAC, and CapBAC work in a multi-user example. In CapBAC, a capability is referred as a communicable, unforgeable token assigned by an authority [18], which points to an object along with a set of associated access rights, e.g., a key is a capability to enter a house. For digital services, a capability is a token that contains resource identifiers and the access rights. Unlike ACL, capabilities are possessed by users. The system only validates the capability and decides to accept or deny a request.

### B. IoT Discovery and Directory

Unfortunately, unlike web pages, the so-called "Internet of Things" is difficult to discover through the Internet. IoT devices usually reside in closed application-specific ecosystems provided by their manufacturers.

Network resources on the Internet are reachable through content-based searching, and the metadata on an HTTP server can be requested through a set of "/.well-known" URLs. This mechanism does not work for IoT discovery for a host of reasons, including but not limited to visibility scope, protocol support, and resource constraints of devices [22]–[24], [28]. To enable efficient discovery for heterogeneous IoT devices, a promising method is to host a resource directory on a reliable server, capable of maintaining information and processing requests for a very large number of devices.

Unlike conventional network resources that are mostly reached by domain names or IP addresses, IoT devices are better described by semantic-rich and multi-dimensional characteristics [29]. IoT applications should not rely on hard-coded names to gather data from sensors and actuators, e.g., an application tracking assets in an enterprise targets devices frequently joining or leaving, so devices should be identified semantically by properties. For this purpose, global metadata directories store device properties, such as device type, capabilities, location, and enable resource discovery by these elements. Meanwhile, such directories offer better programmability and understandability for IoT systems, decoupling applications from network changes. As an example, controlling programs may lose the connection to the device (or worse, point to a wrong device) due to device replacement or a restart of the DHCP server, if devices are specified by addresses rather than properties.

Traditional network directories based on the X.500 standard and LDAP [30], [31] are ill-suited for IoT due to the lack of support for frequent updates, location-based features, and fine-grained access control. For IoT, directories are designed to facilitate discovery and data sharing, while the directories themselves are reachable through the Internet or a third-party service [3], [23], [32]. There are application-specific directories [3] providing fast look-up for mobile services, and protocol-specific directories [22], [23] assisting resource discovery in resource-constraint or low-power environments where devices might have sleep cycles or limited network protocol support and multicast-based or HTTP-based service discovery is infeasible. Metadata directories [24], [33], [34] store properties, capabilities, and logs of IoT devices to promote intelligent and automated services for smart cities.

### C. Open Issues

Existing access control solutions for IoT pursue finer granularity as the major goal. However, extending them to distributed IoT systems faces significant challenges.

*ABAC:* In principle, an access policy can specify attributes from subjects, objects, external environments, and runtime status. Current studies assume attributes are either centrally stored in the cloud or completely provided by the users [14]–[16], [27] for a single autonomous system, and therefore do not achieve the maximum comprehensiveness as the concept promise. However, geo-distributed systems have multi-faceted and multi-sourced attributes, which makes the retrieval of attributes from various sources across administrations a thorny issue.

*CapBAC:* CapBAC achieves the theoretically finest-grained granularity by delegating capabilities [18], but its drawbacks are also evident. The capabilities assigned to users are hard to track down or take back. Existing designs [9], [18] validate whether a capability has been revoked against permanent revocation records, which incurs additional storage overhead. Furthermore, it's hard to reliably walk through a long delegation chain to locate and remove capabilities of a laid-off person or a malicious user.

*IoT Directory:* Flexible access control is not needed for traditional directory services, and solutions are generally based on ACL [35]. Restrictions on data access to a directory are largely done by hiding the server, e.g., from queries beyond the enterprise network [24], while the directory itself is widely used for identity management [15], [36], [37]. Thus, even if directories are proposed to store information of IoT devices, their access control mechanisms falls short, which prevents the safe functioning of IoT directories [3], [24].

We investigate how distributed directories can offer access control for IoT. We address two interrelated problems: access control for directories and directory-based access control for IoT. To the best of our knowledge, it is the first work that jointly studies access control and IoT directories.

## III. Overview of DBAC

### A. System Architecture

We use the following terms and concepts:

**Thing Profile (TP):** A Thing Profile describes an IoT device using metadata including properties, functionalities, relationship with other devices.

**Distributed Autonomous System (DAS):** A DAS is a geographically distributed autonomous system operated by an organization, an enterprise, or any single administrative entity. A DAS owns and manages a number of IoT devices and their TPs.

**Administrator:** An administrator is able to manage one or more devices (and the TPs) in a DAS. The administrators can be designated by the DAS or the owners of devices.

**Local directory:** Each DAS operates at least one local directory, a minimum functional unit to store TPs and respond to requests from users or applications.

**Global directory:** A global directory is a logical union of all the local directories of the target geographical distributed systems.

We assume each device is associated with a TP describing its properties and functionalities. A device can be either a single physical entity (i.e., a sensor or actuator) or a grouped entity (e.g., an assembly line or a smart home). As the boundary between metadata and data is not clear, and making

this distinction is not of our interest, any descriptive data that can be stored in a short profile is viewed as metadata in this paper, including potential measurements. As an example, a temperature sensor may update current reading value to the TP, while time-series measurements are stored in a different external database. Therefore, administrators maintain descriptive data and lists of potentially accessible interfaces in the TPs and specify access control rules.

Note that DASes may not be independent of each other. A DAS may be divided to multiple DASes or overlaps with other DASes, e.g., an enterprise can have multiple departments, and local directories therefore may connect with each other as the same administrative hierarchy. Local directories inside the same DAS are organized as a directory information tree [20] and can appear as a single endpoint to outside users.

### B. Design Principles

We aim for an approach that can be generalized to a variety of IoT scenarios. While we focus on geographically distributed systems, our design principles listed below do incorporate those goals in [9], [10].

*1) Fine-Grained Access Support:* We do not intend to achieve finer granularity than existing ABAC and CapBAC frameworks, but provide sufficient system-wide supports to ensure that the access control remains fine-grained in practice. The system should provide detailed descriptions specifying the attributes and capabilities of each device for access control.

*2) Least-Attribute Principle:* The commonly referred "least-privilege principle" requires each privileged entity of the system to use the least set of privileges necessary to complete the job [38]. With this principle, the affected elements are minimal when accidents occur, and the interactions among entities are also reduced. In many systems, this principle is interpreted as granting least privileges to users. However, for ABAC across domains, the system should retrieve the least set of attributes needed when validating access requests. We call this requirement as "least-attribute principle".

*3) Privacy Preserving:* Private attributes should not be abused or transferred between systems without constraints. Many attributes are considered private or sensitive to users, which are stored in the trusted servers. Users must explicitly control when and what private attributes (e.g., location, medical records) are used for authorization, while public attributes can be retrieved automatically.

*4) Local Maintenance:* Profiles of devices and credentials of users should be managed by each local system, rather than being disseminated or replicated everywhere. The maintenance of subject and object information or access control policies is still in the hands of administrators of each local system. It does not introduce new burden or changes to each system and guarantees that device information and identity management are secure and scalable.

*5) Resource-constrained Device Support:* The system should accommodate devices with limited capabilities or resources. Flexible access control policies usually require significant efforts to validate and process requests, and it seems challenging to enforce such mechanisms to many resource-constrained devices. Access control solutions should be generalized to such devices, rather than only for particular devices.

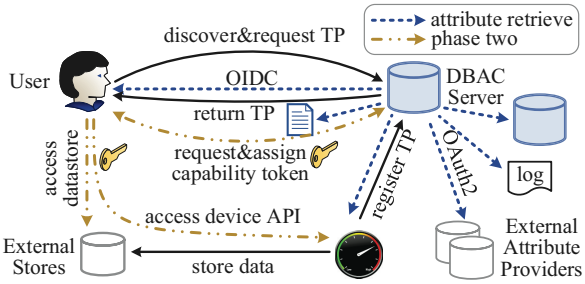



Fig. 3. Overview of the two-phase DBAC.

### C. System Workflow

DBAC applies a two-phase approach, as shown in Fig. 3. The DBAC server in the diagram includes the TP directory and the authorization server of the DAS[1]. In the first phase, the user[2] discovers and requests TPs stored in the directory through available query interfaces. The request is then processed with attribute-based policies. According to the corresponding policies, the DBAC server retrieves attributes from multiple sources including the user, stored objects, federated directories, external attribute providers, history logs.

Once a user is authorized, the requested TPs are returned to the user. The second phase is to access device interfaces or external data stores described in the TPs. This phase is optional as devices may have exclusive platform-specific access rules. Given the complexity of attribute retrieval and validation, devices or external data stores could delegate the authorization process to the DBAC server. The DBAC server assigns a capability token to each authorized user, indicating that the required attributes have been verified. The device only needs to check the token validity to process the request. In other words, DBAC servers are under the obligation to conduct the complicated authorization for resource-constrained devices.

## IV. IoT Directory in DBAC

Directories are commonly used for storing metadata of network objects. In DBAC, we attach a directory module to each DAS, while each DAS is able to establish its own directory and merge it into the whole structure for a global perspective. As the implementation details and evaluations of major directory functionalities, excluding access control solutions, have been mostly present in our previous work [24], we review the necessary components and functionalities of the directory system, and introduce some updates.

### A. Global and Local Directory

IoT directories decouple access to TPs from access to real devices and enable resource discovery by properties and functionalities rather than fixed identities. The global directory

[1]For convenience, we use "DBAC server" and "directory" interchangeably.
[2]The user is likely accessing the data mediated by an individualized or shared application.

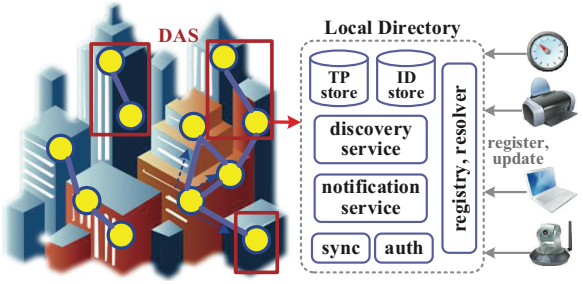



Fig. 4. Global directory architecture. The global system consists of multiple DASes, and each DAS can have multiple local directories.

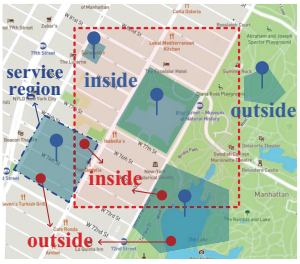



Fig. 5. Query multiple directories.

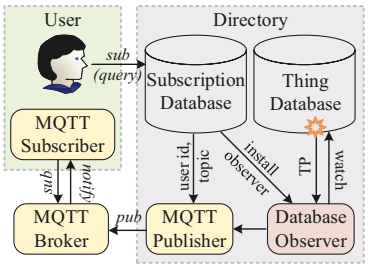



Fig. 6. Subscription model.

is distributed and federated, and provides standard interfaces for both administrators and clients.

As each DAS may or may not trust others in data or service provision, we do not restrict the relationships among DASes. Each DAS volunteers to participate in the overall system by constructing a local directory with simple configurations to specify which DASes it trusts. Furthermore, since a DAS might govern multiple affiliated DASes according to the administrative relationships, a local directory can be therefore associated with subordinate local directories. In this way, each DAS might own a set of local directories, and DASes are loosely federated as a global directory consisting of all the local directories.

This generic topology is representative and flexible, allowing each DAS to structure its own directories, design access rules, manage devices, and volunteer to participate in DBAC with simple configurations. As each local directory usually serves for a specific geographic area according to the administrative hierarchy, this structure also facilitates data aggregation based on service area of directories.

Fig. 4 depicts the topological architecture. The metadata of devices are collected by directories or updated by devices, and are then indexed and stored as TPs in the database. Information in a directory might need to be synchronized with other directories, in terms of the trust and administrative relationships. A local directory can be physically deployed on an edge computing node or a cloud server, depending on the scale of the system. A set of local directories within the same DAS may or may not be deployed together on the same physical server. They then expose data collaboratively or independently as different directories to outside applications, users, and directories. In practice, local directories are preferably hosted by different physical servers to avoid a single point of failure and improve the availability.

### B. Directory Implementation and Functionalities

We implement RESTful APIs of directory systems using the Python Flask framework, and each local directory is able to run independently as a web application. We use MongoDB to store TPs for each directory, and we index the device type by B-Tree and geographical coordinates by 2dsphere. The TPs are specified and validated according to W3C Thing Description [39], an on-going standardization work that manages metadata of devices as JSON objects.

In addition to basic Create, Read, Update, and Delete (CRUD) operations, we implement two types of IoT-oriented queries to make the system suitable for IoT management. *Aggregation query* refers to requesting data from multiple objects or directories, e.g., the mean value of thermometers placed in a vegetable greenhouse. *Location-based query* requests data by a geographic region represented as a polygon with longitude and latitude coordinates, or the administrative hierarchy of DASes, or both. Furthermore, we allow users to customize JSON scripts to request data through any combination of device properties and supported queries as a single query. To avoid troubling users to create credentials for each single directory, federated identity management is configured through OpenID Connect [25].

Depending on its position in the network, a directory can have superior or subordinate directories within the same DAS, trusted directories from different DASes, and unacquainted directories potentially reachable through the Internet. To efficiently reach out to connected directories, while not traversing the entire structure of directories, each local directory maintains distributed knowledge of the network and aggregated information from connected directories. Specifically, each local directory keeps the up-to-date information of certain types of devices owned by the neighbor directories, further reachable directories of neighbor directories for efficient search, addresses of neighbor directories and the logical root. This information is automatically propagated to affected directories when a TP is registered or deleted. Thus, a local directory only has a general vision of which connected directories to consult, without knowing any details about the stored objects. This design preserves privacy, as the direct connection between two directories implies some sense of trust, but the trust may not be transferable to further connected directories.

To enable queries across a large geographic area covering multiple DASes, we assume each directory has a service region, a bounding box covering the geographic coordinates of the registered devices, as shown in Fig. 5. When processing a query searching for a geographic polygon, it first finds the directories with covered or intersected bounding boxes and then extracts TPs from those directories. Note that whether a TP in an intersected bounding box is requested or not needs to be further determined by a geographic query, as the device might be inside or outside the searching area.

Another IoT-oriented function that makes our directory stand out from existing ones is the subscription feature. We integrate MQTT [40], a light-weight publish/subscribe protocol,

into directory query functions to enable interest-based subscription. The MQTT broker is implemented both locally using Eclipse Mosquitto [41] and remotely using HiveMQ [42]. The reasons for integrating MQTT come from its prevalence of IoT and the need for a notification system, while other IoT protocols might also be supported in the future. The workflow is depicted in Fig. 6. Users are able to subscribe the changes of TPs along with their queries sent to the IoT directory. Knowing when these changes occur enables users to keep track of pertinent devices or aggregated data without sending duplicate requests. When a TP is inserted or deleted, a database observer checks the properties in the TP against installed subscriptions, and notifications are then sent accordingly to the users who created the matching subscriptions.

## V. Two-Phase Access Control

We design and implement our prototype including the directory system and the two-phase DBAC. In phase one, clients discover and access TPs in the directories, and the authorization is based on multi-faceted and multi-sourced attributes. In phase two, clients access to further resources, which extends the attribute-based policies and allows capability-based access.

### A. Reasons for Decoupling

DBAC explicitly decouples the access to metadata from the access to device interfaces or data stores, with the help of directories. The reasons are twofold: first, directories provide metadata and aggregated information, and users may not need or have permissions to further operate devices or access detailed data; second, unifying the specific access control solutions for heterogeneous devices is a long-term process that is impractical at the current stage.

It is natural for cross-domain applications or users to discover available devices before accessing to them, especially when devices are frequently offline due to sleep cycles or mobility. The global directory is an intermediary between devices and clients, and it offers an efficient and reliable discovery approach by exposing diverse query interfaces. Besides, obtaining information in TPs is sufficient for many applications, rather than operating devices that might be tightly controlled by administrators or controlling programs.

Unlike an enterprise-scale or a smaller-size network that is relatively easier to apply uniform access control mechanism for all devices, incorporating the heterogeneity of IoT devices across multiple administrations has a long way to go, or nearly unrealistic for access control given that many devices may not support complicated authorization mechanism. A directory can be a plug-in service to realize access control mechanisms for stored objects, which does not conflict with existing access rules to real devices. For example, a smart lock taking correct fingerprints to unlock can also display its status as metadata visible to specific staff through the directory. Therefore, it provides an additional layer to enhance the interactivity of the IoT ecosystem and the programmability of IoT applications.

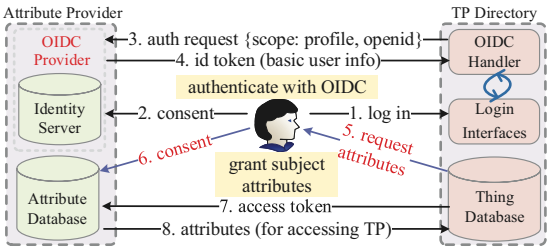



Fig. 7. Subject attribute retrieval with OIDC. The attribute provider and the TP directory belong to different DASes.

### B. Phase One: Access to TPs

In large-scale IoT scenarios, the completion of a function usually requires the cooperation of a set of devices, and designing access policies for each device can be cumbersome, needless to say identity-based policies cannot represent complicated access rules. In this case, a single policy should be able to control the assembly of devices, rather than being replicated for each individual identity.

Although ABAC conceptually incorporates all kinds of attributes into the access control policy, a major problem that has not been well-addressed is how to retrieve attributes from these sources, which did not gain enough attention by current systems where the attributes are centrally managed and provided. In distributed settings, however, we cannot assume users to provide a complete credential of attributes or a single system to store all needed attributes. To cope with this problem, we classify attributes into four types according to the sources of attributes: subject attribute, object attribute, environment attribute, and runtime status attribute. A DBAC server actively retrieves attributes from these sources. Among them object attributes are mostly self-included in the requested TPs maintained by the same DAS and therefore certainly retrievable. Subject attributes and environment attributes can be stored by other DASes or third-party servers, so we integrate two standard protocols, namely OpenID Connect (OIDC) [25] and OAuth2 [26] to retrieve such attributes.

When a user requests the TPs in a directory, the subject attributes (i.e., user's information) are stored in the identity server that belongs to either the same DAS or a different but federated DAS. In the former case, the user's information can be easily retrieved. In the latter case, subject attributes need to be retrieved across domains, since the user logged into a different DAS with OIDC. As using OIDC means that the user has less trust on the directory or does not want to create duplicate credentials, for either factor, the system should try to retrieve attributes in a federated manner. To keep the system pithy, we extend OIDC to support federated retrieval of subject attributes, instead of introducing a brand new service that makes users perplexed. The procedure is illustrated in Fig. 7. When a user initially logins to the TP directory with the credential from a different DAS through OIDC, only the least attributes necessary for login should be provided, rather than all the attributes the OIDC server maintains. Then, every time the user attempts to access a TP that requires additional subject attributes for authorization, the directory asks the user's

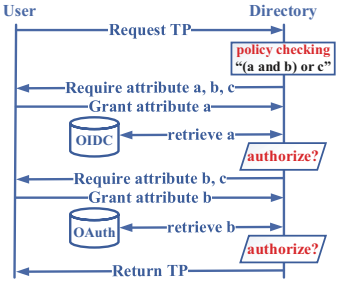



Fig. 8. Iterative attribute retrieval.

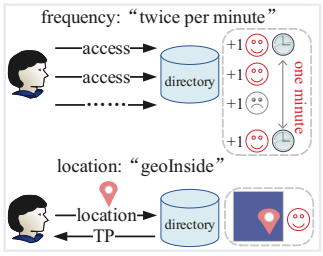



Fig. 9. Frequency and location.

consent once again to retrieve them from the federated identity server. This subtle change compared to pure OIDC is that we explicitly require the consent when needed for accessing to TPs, which prevents the unconscious leak of subject attributes. Meanwhile, this additional consent process can be combined with environment attribute retrieval, as described below.

Environment attributes refer to the attributes describing natural environment or sensed by surrounding services. They are either locally managed or provided by third-party services, and some of them can be considered private. For example, today's rainfall might be publicly retrievable, while the body temperature detected by a thermal imaging system might be sensitive and private. We implement the OAuth2 interfaces to retrieve those sensitive attributes under the user's permission. In other words, private attributes should be well-protected from being exposed without permission from the owner.

An access policy can be complicated, requiring both subject attributes and environment attributes from different servers. In this case, it needs the user's consents through both OIDC and OAuth2. To improve user experience, we combine the consent process, and the underlying process is parallel and unknown to users. Furthermore, given that there might be multiple combinations of needed attributes that are enough to get approved for an authorization request, we make the consent process iterative, allowing a user to grant a subset of needed attributes at a time, which provides an additional layer of privacy protection. An example is shown in Fig. 8. Note that this procedure is based on the assumption that a directory asks the user for the permission of retrieving attributes without disclosing any detail of access policies.

Most attributes are stateless, but some runtime status can be stateful, which requires the directory to maintain some history records to process an upcoming authorization request. As a most common need, we implement the extension to support access frequency as a runtime attribute, in the form of a threshold and a time duration (e.g., 5 times per minute). The relevant information including thing id, user id, and access time is captured in the database at the time of every resource access request, and outdated records are removed meanwhile. As shown in Fig. 9, the authorization decision is based on whether the current attempt does not exceed the access limit within the time duration specified in the policy.

DBAC supports location-based attributes in three ways: the directory name represents the service region, location descriptors (e.g., porch, garage) as regular attributes, and geographic

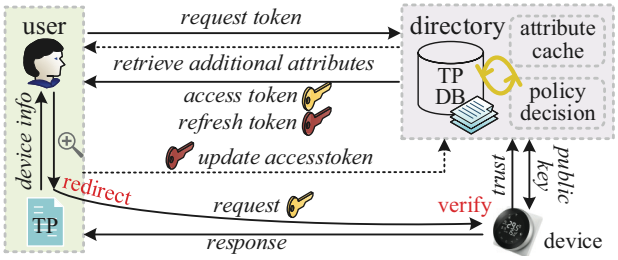



Fig. 10. Phase two of DBAC. Devices delegate authorization to directories.

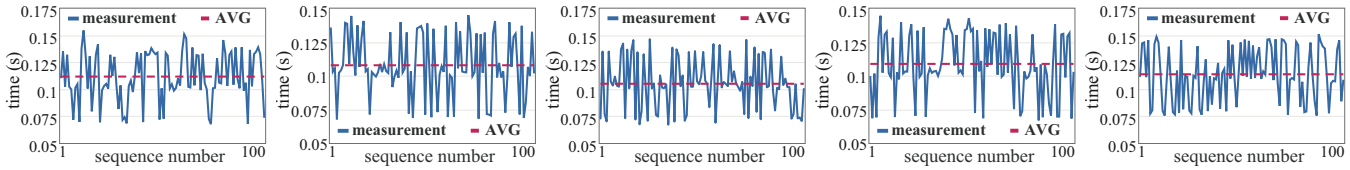
coordinates referenced by longitude and latitude. Handling the first two types is not beyond trivial attribute matching. For the location represented by geographic coordinates, we implement a "geoInside" rule to check whether a particular location is contained by a geographic polygon. It resembles the physical world where the access control rule is based on proximity (e.g., anybody inside a room can toggle the light switch). While the current system obtains the user's GPS location through the browser, the interface of providing coordinates remains open for other trustworthy approaches [43].

### C. Phase Two: Access to Device Interfaces or Databases

In this phase, we manage the access to the linked objects provided in the TPs, including device APIs and external databases storing produced data. As they may have distinct authorization mechanisms and TPs have been accessed in the first phase, participating in this phase is optional for both the users and devices.

Upon receiving the TPs, a user may want to access the resources (APIs, external databases) specified in the TPs. The user can then be redirected to the related endpoints and go through their authorization processes. However, as the directory has already validated some needed attributes, there is no reason to go through this process all over again, as long as the directory is trusted by those resources. Besides, many devices are resource-constraint, cannot afford to enforce complicated policies and retrieve or maintain all-faceted attributes.

DBAC shifts such burden from resources to directories, with the help of CapBAC. The workflow is shown in Fig. 10. Once a user requests further access to the resource specified in the TP, the directory retrieves and examines needed attributes on behalf of the resource. In other words, the resource delegates the attribute-based authorization process to the directory. Some of these attributes have already been obtained and cached for accessing to the TP in the first phase, and other attributes are freshly retrieved for the current access. If the attributes match those required conditions in the access policy, the directory issues a capability token (i.e., a signed ticket) to the user. The token encapsulates necessary information of the authorization, including identities of subjects and objects, approved privileges, an optional attribute list, the token validity period. Afterwards, the user can present this token along with an access request to the resource. With the verified signature in the token, the resource trusts that the directory has checked the attribute-based policy and the access request can be safely approved.




(a) all can access (b) date time comparison (c) email matching (d) device type matching (e) inside polygon

Fig. 11. Authorization time for five common access control rules.

To revoke a capability token, many systems require devices to maintain a revocation list [9], [18], and the token must be verified against the revocation list for every access. We suggest to use two kinds of tokens: a short-period access token and a long-period refresh token. The refresh token can be a proof of satisfying the policies at an earlier time and is used to request a new access token when a previous one is expired. By tuning the validity periods of the tokens, the DBAC server controls the life cycles of tokens and updates attributes flexibly when needed, and devices do not maintain the revocation list.

Therefore, as long as an entity is able to verify the signature in the token, it can enjoy the great profit brought by the DBAC system. This significantly lowers the bar of enforcing complicated access control rules.

### D. Access Control Dilemma

*Discover first or access first:* The last thing we want to see is that users provide a large number of sensitive attributes but do not get the expected results. However, it is also disastrous to disclose results before validating needed attributes. This leads to a dilemma, namely providing attributes before discovery or discovery before providing attributes. DBAC returns tailored results to users before asking for private attributes. The tailored response conveys a rough idea to the users, informing them what and about how many TPs they will see once authorized. Afterwards, if it is deemed worthwhile, the users can consent to providing private attributes to obtain detailed information. This design improves both efficiency of discovery and accuracy of service.

*Policy conflicts in DBAC:* Potential policy conflicts may pose another problem. It may happen more often compared with traditional identity-based access control frameworks. Even in a well-managed system, the administrators might unintentionally inject different policies that yield contradictory decisions on an access request, or intentionally overwrite a previous policy. To solve this problem, we associate each policy with a priority level. When a conflict occurs, the policy with higher priority is adopted. Ties should be avoided, otherwise reported instead of being randomly broken.

## VI. Performance Evaluation

This section evaluates the performance of our prototype system, with a focus on access control mechanisms.

### A. Evaluation Setup

To setup a testbed and evaluate DBAC, we host each directory system as a Flask web application in an Amazon AWS EC2 t2.micro instance. The access control language is implemented following the XACML standard [27]. MongoDB [44] is set up to store TPs and access policies in a separate amazon t2.micro instance. Each web application instance is configured to point to an individual MongoDB database in the database server. Python unit test framework is used as the entry point to run our measurements. Tab. I summarizes the configurations of utilized hardwares.

TABLE I
Configurations of the Server and the Client

| Category | Specification |
|---|---|
| CPU (server) | Intel(R) Xeon(R) E5-2676 v3 @ 2.40GHz, 1 core |
| CPU (client) | Intel(R) Core(TM) i7-7700HQ, 4 cores, 2.80GHz |
| System | Ubuntu 20.04.2 LTS (server), Windows 10 (client) |
| Storage | 1GiB RAM (server), 16GB RAM (client) |
| Database (server) | MongoDB 4.4.6 [44] |

### B. Evaluation Results

We begin our evaluation by comparing the authorization time affected by processing different attribute-based policies. Five most common policies with different data types are selected. The first policy is a simple one that allows access to everything. The other policies match attributes based on the numeric, string, and geographic polygon. Based on the results shown in Fig. 11 and Fig. 12, we can infer that all the data types take a similar time to perform authorization. From our experiments, we observe that all the internal policy decisions against different data types take negligible part of the overall runtime. The policy retrieval from the database dominates the processing time, and the propagation delay (the one-way latency is around 37 ms in our measurement) dominates the network latency. Hence, all the requests take roughly the same time. The CPU usage also follows a similar trend as that of runtime. The geographic attribute-based authorization uses slightly more CPU to perform the more complex operation of determining whether the location is inside the geographic polygon.

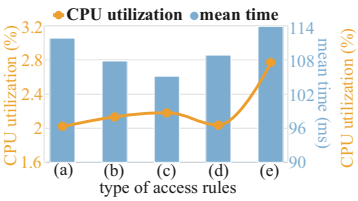



Fig. 12. Mean time and CPU utilization for the five common policies.

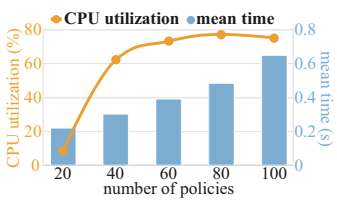



Fig. 13. Mean time and CPU usage for varying number of policies.

We proceed to test how the system scales by enforcing the system to validate multiple policies for a single request. We create a large number of mock policies varying from 20 to 100. Each policy contains a unique attribute-based rule.

The request is set up such that the authorization criteria fail in all the policies. This is to ensure that all the policies are evaluated and mitigate early return on the successful authorization. From Fig. 13, it is evident that the authorization time increases linearly as the validated number of policies increase. These results match our expectations because the policies are evaluated one after another. The CPU usage also increases but tapers off after a certain point.

Then, we evaluate our system by performing tests on basic operations of the creation and deletion of a policy. Fig. 14 and Fig. 15 measure the runtime of registering and deleting 100 mock policies each for the five common attribute-based categories. As expected, both operations have a very acceptable mean runtime close to 0.1 second (i.e., 30 ms if the propagation delay is deducted).

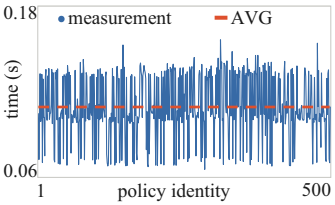



Fig. 14. Policy registration time.

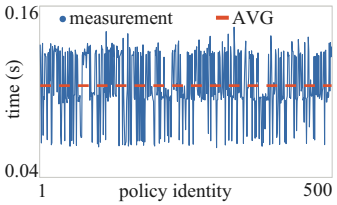



Fig. 15. Policy deletion time.

The second phase of DBAC is mainly loaded by the creation of capability tokens. Token creation is a cryptographic process including encoding and hashing. This is yet another integral operation in our application, as it is the first step to get access to external resources and objects. We apply JSON Web Token (JWT) [45] as the implementation of capability tokens. As shown in Fig. 16, the capability creation takes more than 200 ms on average which is higher than the other operations we have seen so far. We omit the client-side processing time, as all it needs is to validate the signed capability token, without any actions on attributes or policies.

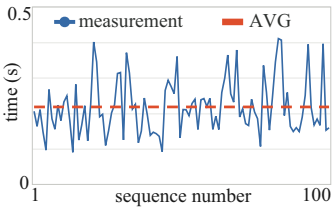



Fig. 16. Capability creation time.

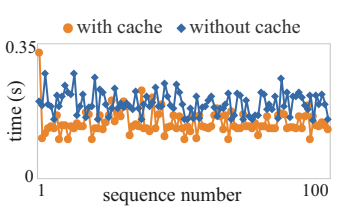



Fig. 17. Cache evaluation.

Among all of the directory functionalities, aggregation query over a large geographical region might be the most fascinating but time-consuming function. We perform a test that leverages the distributed caches with our system. Totally seven directories are hosted and an aggregation query is conducted to get all the TPs within a geographical region. A request is sent to all the directories whose bounding boxes are configured to intersect with the input polygon. This operation is optimized by implementing a simple in-process cache mechanism, for the evaluation purpose. The bounding box for each directory is retrieved and stored in the cache of the requested directory, so it does not need to traverse all over other directories to get their service region. Based on the intersection of the polygon with the bounding box, the directories are filtered locally, and the query is sent to all the filtered directories to obtain the TPs. As shown in Fig. 17, by incorporating the simple cache mechanism, the runtime of geographic aggregate query decreases 26% compared to that without caches.

## VII. Discussion

*Scalability:* We would like to point out that DBAC unlikely raises new scalability issues. Scalability concerns are mostly for the distributed directories, rather than the access control mechanisms. The distributed directories are associated with the IoT systems, logically centralized with multiple global endpoints. Directories only maintain distributed knowledge and either directly respond to the queries or simply forward messages. Intensive aggregation and authorization work is all done locally. Thus, no new scalability challenges are introduced beyond those in existing systems [3], [9], [23], [46].

*Encryption:* There are multiple ways to encrypt messages in ABAC and CapBAC systems [47]–[49] with guaranteed confidentiality, robustness, and scalability. DBAC is a system-level approach to addresses challenges of typical access control mechanisms for IoT, which is largely orthogonal to particular encryption techniques. Enforcing specific encryption schemes is an interesting topic but beyond the scope of this paper.

*Location Representation:* In DBAC, geographic locations are primarily represented by geographic points or polygons with vertices of longitude and latitude. Other common location descriptors can be a civic address (e.g., “90 Cohoes Ave, Green Island, NY”) or a location property (e.g., “hallway” or “backyard”). Translating these descriptors to geographic vertices can be done through standard protocols or available APIs [32], [50], and we indeed configure an extension of the DBAC system to support geographic objects of OpenStreetMap [50].

## VIII. Conclusion

In this paper, we present DBAC, a systematic access control mechanism for IoT directories and devices. DBAC is a two-phase mechanism that decouples access to metadata from access to device APIs or data stores. It combines fine-grained solutions, ABAC and CapBAC, to describe access policies resembling the attribute-based rules of the physical world, as well as supports resource-constraint devices. Standard solutions, including W3C TD, OIDC, and OAuth2, are integrated or modified to achieve federation, retrieve attributes from a variety of sources, and preserve privacy. The authors have provided public access to their code at https://github.com/Halleloya/DBAC.

## Acknowledgement

This work is supported by the National Science Foundation under grant CNS 19-32418. The authors would like to thank Andrea Huang, Ryan Liang, and Hongfei Chen for their help in developing the prototype.

## REFERENCES


[1] A. Pintus, D. Carboni, and A. Piras, "Paraimpu: a platform for a social Web of Things," in *ACM International Conference on World Wide Web (WWW)*, 2012, pp. 401–404.

[2] Z. Wen, T. Lin, R. Yang, S. Ji, R. Ranjan, A. Romanovsky, C. Lin, and J. Xu, "GA-Par: Dependable microservice orchestration framework for geo-distributed clouds," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 31, no. 1, pp. 129–143, 2019.

[3] V. P. Kafle, Y. Fukushima, P. Martinez-Julia, and H. Harai, "Directory service for mobile IoT applications," in *IEEE Conference on Computer Communications Workshops*, 2017, pp. 24–29.

[4] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review (SIGCOMM CCR)*, vol. 44, no. 5, pp. 27–32, 2014.

[5] P. De Vaere and A. Perrig, "Liam: An architectural framework for decentralized IoT networks," in *IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2019, pp. 416–427.

[6] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 337–352.

[7] J. Cao, L. Xu, R. Abdallah, and W. Shi, "EdgeOS_H: a home operating system for internet of everything," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1756–1764.

[8] V. Issarny, B. Billet, G. Bouloukakis, D. Florescu, and C. Toma, "LATTICE: A framework for optimizing IoT system configurations at the edge," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 1797–1805.

[9] Q. Zhou, M. Elbadry, F. Ye, and Y. Yang, "Heracles: Scalable, fine-grained access control for Internet-of-Things in enterprise environments," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2018, pp. 1772–1780.

[10] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, "Rethinking access control and authentication for the home Internet of Things (IoT)," in *USENIX Security Symposium (USENIX Security)*, 2018, pp. 255–272.

[11] "AWS IoT," https://aws.amazon.com/iot/.

[12] "Apple HomeKit," https://developer.apple.com/homekit/.

[13] "Samsung SmartThings," https://www.smartthings.com/.

[14] M. A. Al-Kahtani and R. Sandhu, "A model for attribute-based user-role assignment," in *IEEE Annual Computer Security Applications Conference (ACSAC)*, 2002, pp. 353–362.

[15] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services," in *IEEE International Conference on Web Services (ICWS)*, 2005.

[16] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2010.

[17] S. Bhatt and R. Sandhu, "ABAC-CC: Attribute-based access control and communication control for Internet of Things," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2020, pp. 203–212.

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

[19] Q. Zhou, M. Elbadry, F. Ye, and Y. Yang, "Towards fine-grained access control in enterprise-scale Internet-of-Things," *IEEE Transactions on Mobile Computing (TMC)*, vol. 20, no. 8, pp. 2701–2714, 2020.

[20] D. Comer and R. E. Droms, "Uniform access to internet directory services," *ACM SIGCOMM Computer Communication Review (SIGCOMM CCR)*, vol. 20, no. 4, pp. 50–59, 1990.

[21] R. E. Droms, "Access to heterogeneous directory services," in *IEEE International Conference on Computer Communications (INFOCOM)*, 1990.

[22] M. Stolikj, R. Verhoeven, P. J. Cuijpers, and J. J. Lukkien, "Proxy support for service discovery using mDNS/DNS-SD in low power networks," in *IEEE international symposium on a world of wireless, mobile and multimedia networks (WoWMoM)*, 2014.

[23] Z. Shelby, M. Koster, C. Bormann, and P. van der Stok, "Core resource directory," IETF, Internet-Draft 6749, 2020. [Online]. Available: https://www.ietf.org/id/draft-ietf-core-resource-directory-24.html

[24] L. Hao and H. Schulzrinne, "Goldie: Harmonization and orchestration towards a global directory for IoT," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2021.

[25] D. N. Sakimura, J. Bradley, and M. Jones, "Openid connect standard 1.0-draft 21," 2012.

[26] D. Hardt, "The OAuth 2.0 Authorization Framework," IETF, RFC 6749, 2012. [Online]. Available: http://tools.ietf.org/rfc/rfc6749.txt

[27] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, and K. Scarfone, "Guide to Attribute Based Access Control (ABAC) definition and considerations," *NIST Special Publication*, vol. 800, no. 162, pp. 1–54, 2014.

[28] H. Cai and T. Wolf, "Self-adapting quorum-based neighbor discovery in wireless sensor networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2018, pp. 324–332.

[29] Y. Fathy, P. Barnaghi, and R. Tafazolli, "Large-scale indexing, discovery, and ranking for the Internet of Things (IoT)," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, p. 29, 2018.

[30] W. Yeong, T. Howes, and S. Kille, "X.500 Lightweight Directory Access Protocol," IETF, RFC 1487, 1993. [Online]. Available: http://tools.ietf.org/rfc/rfc1487.txt

[31] P. Barker, "Providing the X.500 directory user with QoS information," *ACM SIGCOMM Computer Communication Review (SIGCOMM CCR)*, vol. 24, no. 3, pp. 28–37, 1994.

[32] T. Hardie, A. Newton, H. Schulzrinne, and H. Tschofenig, "LoST: A Location-to-Service Translation Protocol," IETF, RFC 5222, Aug. 2008. [Online]. Available: http://tools.ietf.org/rfc/rfc5222.txt

[33] M. Alkalbani, B. Hamdaoui, N. Zorba, and A. Rayes, "A blockchain-based IoT networks-on-demand protocol for responsive smart city applications," in *IEEE Global Communications Conference (GLOBECOM)*, 2019.

[34] L. Hao and H. Schulzrinne, "When directory design meets data explosion: Rethinking query performance for IoT," in *IEEE International Symposium on Networks, Computers and Communications (ISNCC)*, 2020.

[35] Y. Yagi, N. Kitsunezaki, H. Saito, and Y. Tobe, "Rwfs: Design and implementation of file system executing access control based on user's location," in *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014, pp. 805–810.

[36] Y.-S. Yeh, W.-S. Lai, and C.-J. Cheng, "Applying lightweight directory access protocol service on session certification authority," *Computer Networks (CN)*, vol. 38, no. 5, pp. 675–692, 2002.

[37] "Microsoft Azure active directory," https://azure.microsoft.com/en-us/services/active-directory/.

[38] J. H. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[39] "W3C Web of Things (WoT) Thing Description," https://www.w3.org/TR/wot-thing-description/, 2020.

[40] "OASIS MQTT Version 5.0 Standard," https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[41] "Eclipse Mosquitto," https://mosquitto.org/.

[42] "HiveMQ," https://www.hivemq.com/.

[43] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang, "Towards street-level client-independent IP geolocation." in *USENIX Symposium on Network System Design and Implementation (NSDI)*, vol. 11, 2011, pp. 27–27.

[44] "MongoDB," https://www.mongodb.com/.

[45] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, May 2015. [Online]. Available: http://tools.ietf.org/rfc/rfc7519.txt

[46] A. Nußbaum, J. Schütte, L. Hao, H. Schulzrinne, and F. Alt, "Tremble: Transparent emission monitoring with blockchain endorsement," in *IEEE International Conference on Internet of Things (iThings)*, 2021.

[47] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM Conference on Computer and Communications Security (CCS)*, 2006, pp. 89–98.

[48] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE Symposium on Security and Privacy (S&P)*, 2007, pp. 321–334.

[49] B. Wang, W. Song, W. Lou, and Y. T. Hou, "Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2015, pp. 2092–2100.

[50] "OpenStreetMap," https://www.openstreetmap.org/.