EAI Endorsed Transactions
on Industrial Networks and Intelligent Systems
Research Article EAI.EU

Fine-Grained Access Control for Smart Healthcare Systems in the Internet of Things★

Shantanu Pal[1,*], Michael Hitchens[1], Vijay Varadharajan[2], Tahiry Rabehaja[1]

[1]Department of Computing, Macquarie University, NSW 2019, Sydney, Australia
[2]Advanced Cyber Security Engineering Research Centre, University of Newcastle, NSW 2308, Australia

Abstract

There has been tremendous growth in the application of the Internet of Things (IoT) in our daily lives. Yet with this growth has come numerous security concerns and privacy challenges for both the users and the systems. Smart devices have many uses in a healthcare system, e.g. collecting and reporting patient data and controlling the administration of treatment. In this paper, we address the specific security issue of access control for smart healthcare systems and the protection of smart *things* from unauthorised access in such large scale systems. Commonly used access control approaches e.g. Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC) and Capability-Based Access Control (CapBAC) do not, in isolation, provide a complete solution for securing access to IoT-enabled smart healthcare devices. They may, for example, require an overly-centralised solution or an unmanageably large policy base. We propose a novel access control architecture which improves policy management by reducing the required number of authentication policies in a large-scale healthcare system while providing fine-grained access control. The devised access control model employs attributes, roles and capabilities. We apply attributes for role membership assignment and in permission evaluation. Membership of roles grants capabilities. The capabilities which are issued may be parameterised based on attributes of the user and are then used to access specific services provided by *things*. We also provide a formal specification of the model and a description of its implementation and demonstrate its application through different use-case scenarios. The evaluation results of core functionality of our architecture are provided with the practical testbed experiments.

Received on 22 December 2017; accepted on 28 February 2018; published on 20 March 2018

Keywords: Internet of Things, Healthcare Systems, Access Control, Policy Management, Security

doi:10.4108/eai.20-3-2018.154370

1. Introduction

With the rapid improvement in the Internet of Things (IoT) [2], healthcare systems are becoming faster, wearable and more easily accessible as well as remotely available. Previously healthcare systems were designed as dedicated environments but now must function in an open-systems context [3]. Such ubiquitous and heterogeneous healthcare systems can now include a range of wearable devices and smart sensors for automatically collecting, storing and reporting health-related information and assisting in diagnosis and treatment [23]. It is predicted that, by 2020, there will be 50 Billion connected devices in the Internet [5]. This will increase the number of connected devices per person dramatically. Healthcare is one of the key sectors where this change will make a significt impact. With the growing number of smart mobile devices (e.g. smartphones, tablets, PDAs, etc.) there is a similarly expanding range of compatible healthcare apps. for such devices, with nearly 165,000+ currently available [18]. Demand for the use of IoT-based solutions in the healthcare context is only likely to grow, with medical professionals relying on this technology for efficient and secure access to patient data.

In the IoT context a specialist doctor visiting a patient in a hospital may bring their own smart device on which they wish to view the output of the sensors attached to the patient, while a doctor employed by the hospital or

★This paper includes work previously published by the authors [39].
*Corresponding author. Email: shantanu.pal@hdr.mq.edu.au

other staff member may have a hospital issued-device. A nurse in that same hospital may wish to view the output of, and control, patient-attached sensors, but will likely have a different level of access to that of a specialist or attending doctor (cf. Fig. 1). The healthcare system will need to include policies and access control mechanisms which will support the variety of access required. When the range of staff, patients and data in a modern healthcare setting is considered the myriad possible levels of access become clear. In a smart healthcare system it is important to manage and track who (e.g. user and device) is connecting to and accessing what (e.g. system and resource). Thus, there is a need for robust, scalable and secure healthcare systems [41]. In particular unauthorised access to these wearable devices (and connected medical equipment) can breach a patient's privacy and can generate potential threats to an organisation's resources. For example, a patient's pacemaker can be used to generate a fatal shock or a drug infusion pump (e.g. insulin or antibiotics) can be controlled by an attacker to change the drug dosage [44].

The demand for IoT-enabled healthcare devices is increasing day by day [34]. These devices are connected to a patient's body, periodically monitoring their health status using various healthcare apps. (e.g. Cue [4] and Medtronic [21]) and communicate via wireless medium (e.g. IEEE 802.15.4 or Bluetooth Low Energy (BLE) wireless personal area network technology). Such a convergence of the digital and physical world promises improved healthcare but also poses numerous security issues and privacy challenges [19]. IoT devices may be low-powered, memory constrained and possess limited processing power. This means it is impractical to enforce heavy-weight security mechanisms via these devices. From the communication point of view, heterogeneous network environments, wireless mediums, high mobility of *things*, dynamic network topology and availability of infrastructure for communication are major barriers to deploying secure solutions [40]. Therefore, managing the vast amount of heterogeneous devices, applications and their associated services is a challenging task, especially to different requirements and resources. Note, in this paper we introduce *things* as a collection of services, application and potentially their associations that are coming from one or more IoT devices and users.

2. Background and Problem Statement

Several proposals have examined the suitability of applying commonly used access control measures for the IoT [30] [31], e.g. using Role-Based Access Control (RBAC) [6], Attribute-Based Access Control (ABAC) [47] and Capability-Based Access Control (CapBAC) [10]. RBAC uses *roles* to manage the relationship between

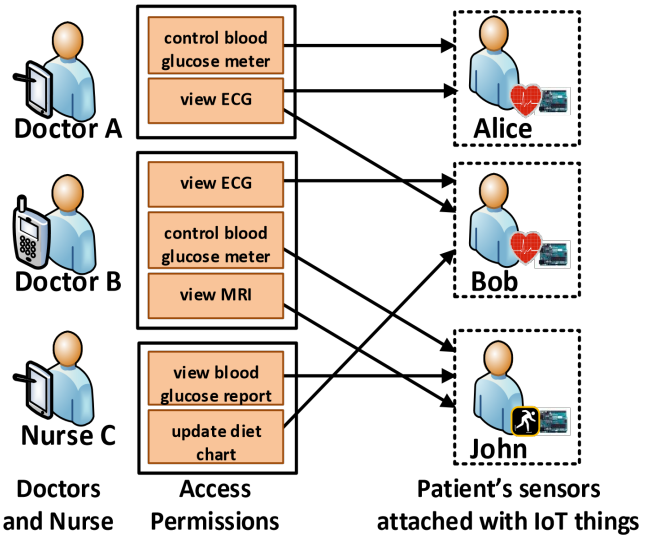

Figure 1. A fine-gained access control scenario for different actors in a healthcare system. Where a doctor/nurse with an appropriate permision can only view/control a subset of a patient's IoT-enabled healthcare 'things'. For instance, a specialist Doctor A (e.g. cardiologist) can view a particular patient's (e.g. Bob) heart sensor related data.

users and policies. It provides rights to the specific roles and users are made members of appropriate roles, rather than granting permissions directly to the users. RBAC introduces an extra layer (i.e. the *role*) between the users and the resources and permissions do not need to be assigned to each individual, rather permissions are assigned to the roles. The users associated with that particular role can then access the resources. However, to explicitly identify and assign users to roles is difficult in a dynamic and large-scale system e.g. the IoT. Moreover, in a smart healthcare system, there may be vast numbers of users and short-term interactions, which argue against such heavy-weight access control mechanisms due to the difficulty in managing and updating the resulting policy base. Therefore, it maybe inefficient to implement access control policies simply using RBAC in such systems.

Unlike RBAC, ABAC makes access control decisions based on the 'attributes' of system entities (e.g. users, resources, etc). An attribute defines the characteristics of an entity. For example, for subjects (users or applications) attributes can include name, organisation, title, for resources, size, date, performance and for environments, date, time and physical location [37]. In ABAC, policies are formulated as attribute expressions and access is granted if the requirements of a policy is satisfied. This allows significant flexibility in specifying fine-grained policies. It is difficult, however, to achieve this in ABAC without deploying a large policy base. Without some means of grouping together policies with the same required attributes, but which provide

access to different resources, the number of policies can multiply. In the IoT context, this means, we may have a difficulty with the scale of the number of *things*. ABAC can be used to provide a context-aware approach for user-role assignment by using user's attributes (e.g. location, designation, etc.) to assess role membership. This allows more flexibility and conciseness in specifying the access of subjects to resources by avoiding the need to specify individual relationships on a per 'subject-object' basis. CapBAC approaches (e.g. [8], [12]) can be used to provide fine-grained access control and have been suggested for use in IoT systems due to the low requirements they place on *things*. A *capability* can be defined as a communicable, unforgeable token of authority [10]. It is associated with an object and a set of access rights to that object, allowing the user that holds the capability to access the resources. While capabilities provide a fine-grained approach to access control, providing a capability to each user for every resource they may access is not scalable without some additional approach to policy management. Also all these systems (with the exception of some recent capability-based proposals e.g. [12]) tend to be heavily centralised, which is not an ideal model for an IoT-enabled smart healthcare system.

One common problem with the aforementioned access control approaches is that, individually, they are not explicitly designed or suitable for managing scalability, whether it is devices, users or policies, in practical IoT scenarios. Handling policy management on an individual basis, with an assumption that the identities of all users who will need access are known beforehand and that all policy rules should be individually recorded in the system on an a priori basis (e.g. the identities of the specialist doctors responsible for each patient) is not realistic in the case of IoT. While ABAC can be employed in dynamic scenarios to handle user scalability, the complexity of policy management, especially on a fine-grained level, can be daunting. RBAC can be used to manage the scale of the number of users by employing role-relationships, but typically needs a priori identification of the authorised users and is infliexble in the face of dynamic user assignment. Capabilities can be used to generate unique permissions for each of the users and can reduce the centralisation of the system but capabilities come with well-known management issues [16]. Thus, there is a need for alternative approaches to access control for IoT-enabled smart healthcare systems.

3. Contributions and Paper Structure

- We propose an access control architecture that allows authorised users to access services provided by resource-constrained IoT devices in a smart healthcare system.
- Our model combines attributes, roles and capabilities to provide a fine-grained and flexible approach while minimising the number of required authorisation policies that need to be created and maintained.
- We apply attributes for role membership assignment and in permission evaluation. Membership of roles grants capabilities. The capabilities which are issued may be parameterised based on further attributes of the user and are then used to access specific services provided by *things*.
- We provide a formal specification of the model and a description of its implementation and demonstrate its application through different use-case scenarios.
- We detail the evaluation results of core functionality of our architecture.

The rest of the paper is organised as follows. First we present an outline use-case in section 4. Then, in section 5, we discuss the solution overview. We briefly discuss the system challenges and requirements in section 6. In section 7, we explore our proposed access control architecture. In section 8, we discuss the model design with various access scenarios. We describe a detailed implementation in section 9 and give some performance evaluation results in section 10. In section 11, we present related works followed by a discussion in section 12. Finally, we conclude the paper and outline the future work in section 13.

4. Use-Case Example

An IoT-enabled smart healthcare system is composed of smart sensors, wearable devices, intelligent network infrastructure and the actors who use the system, all governed by the appropriate policies [34]. There will be various actors within the system, and these actors will require various levels of access to the *things*. Access will be controlled by policies expressed and managed within the system. Some actors will be employees of the organisation controlling the system, for example doctors and nurses. Other actors will be outside the organisation but have a formal relationship to it, e.g. visiting medical specialists, employees of subcontracting firms (e.g. for building maintenance) and patients. Finally, there will be actors who have a much less formal relationship with the organisation, e.g. visiting friends and relatives of patients.

4.1. A Practical Scenario

Now, consider a healthcare facility where patient monitoring and treatment delivery is controlled via smart *things*. Various devices and sensors will be

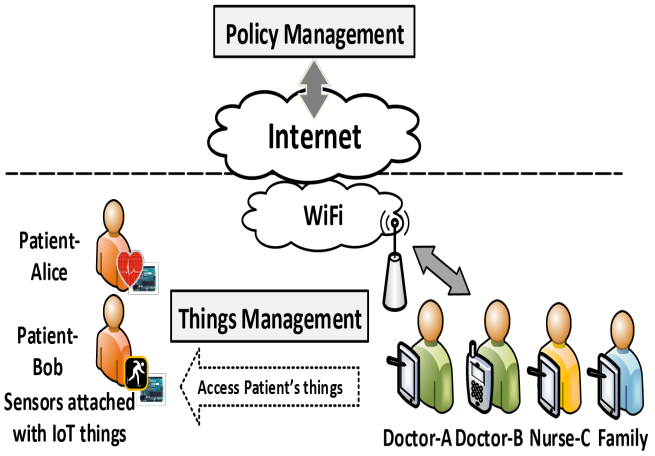

Figure 2. Our use-case scenario The 'Policy Management' moduleholds policies that allow a user (e.g. doctor nurse or family) access to the IoT 'things' associated with a particular patient. The 'Things Management' modulemanages the access controlof each IoT 'thing'.

attached to the patients (i.e. wearable) and accessed by healthcare professionals (and possibly others). In Fig. 2 we depict our scenario. The devices and sensors may be allocated to a patient on admission or at any stage during their stay at the facility. When a device or sensor is allocated to a patient this will be registered in the system, including a notation as to which patient they are assigned. Access to the sensors will depend upon the policies of the facility and this may include different users having different levels of access to the same device. For example, a nurse may have read-only access to a drug delivery device whereas a doctor may be able to alter the dosage. The following actors are involved in our scenario, and the healthcare processes are based on a medical care regime in the Australia [27].

- Patient: People receiving medical treatment.
- Doctor: Primary care physicians. They may provide appropriate advice for the first instance when a patient comes for medical treatment.
- Specialist: Doctors who specialise in a particular area of medicine e.g. cardiologist, neuro-surgeon, etc.
- Nurse: Clinical staff who regularly check the patient's health and provide medical support services.
- Family and friends: Individuals who visit the patient.

4.2. Granting Di erent Level of Acces

In our scenario, we want to provide different actors with different levels of access to the patient's medical sensors and the data they provide while protecting the patient's privacy. For simplicity, we consider that the *things* for different patients may have different associated access control policies and that these policies are stored inside a centralised system. However, enforcement is handled locally. This may either be within the *things* themselves or a local management device if the *things* have insufficient functionality for the task. This both takes advantage of the edge-intelligence of such systems and avoids performance bottlenecks in the central system. In a real-life hospital environment there may be hundreds of doctors, nurses and other staff and possibly thousands of patients with each patient having multiple sensors attached. No one medical professional would have access to all sensors on all patients. Conceivably, no medical professional may be able to access all the sensors on a single patient and even if they could the levels of access may vary from medical professional to medical professional. Consider two examples of the complexities:

- Medical specialists, e.g. cardiologists, should only be able to access the relevant information for patients under their care. If Doctor A is treating patient Alice and Doctor B is treating patient Bob then A should only be able to access the readout from Alice's heart sensor and B should only be able to access the readout from Bob's heart sensor.
- Nurses may be assigned to a particular ward. Each patient in the ward may have a standard set of sensors attached, in addition to ones that may be particular to their condition. Nurses should be able to access the standard sensors for all patients in their ward, but not for patients in other wards.

5. Solution Overview

An access control solution is required that can efficiently provide the required policy management for situations e.g. those described above. We propose a solution combining RBAC, ABAC and CapBAC. Role membership is specified using attributes, not explicit user to role assignment. This provides a greater level of flexibility and conciseness in role management as updates to individual role membership do not have to be noted. It also allows a consistent and easily managed approach to which users will be members of which roles. By employing roles, rather than a direct mapping between attribute expressions and permissions, we do not have to needlessly duplicate attribute expressions. It also allows us to take advantage of the power of the role-hierarchies of RBAC. Adopting an approach similar to [10] we use capabilities as credentials to govern access to *things*. Capabilities are distributed to users and presented to the edge devices (i.e. *things*) for access. The *things* can check the validity of capabilities

without consulting the central systems. This reduces the centralisation of the system and eliminates a potential performance bottleneck.

5.1. Policy Management

Within this general system description there remains a question as to how to formulate and distribute capabilities while maintaining the minimum number of access control policies. Consider how policies may be written, and permissions granted (i.e. capabilities distributed), in the first example of the previous section. A number of alternatives exist:

1. A simplistic solution would be to create a role 'cardiologist' which provides its members with a capability that grants access to all sensors of type 'heart monitor'. This would allow every cardiologist to access every patient's heart monitor, violating patient privacy.

2. Each capability could have associated with it a test, evaluated on access, that ensures the patient is under the care of the specialist presenting the capability. This would increase the processing and bandwidth requirements on the *things*. The relevant credential, proving the relationship between specialist and patient, would have to be provided to the *thing* and any signature on it checked. Signature checking is the most time-consuming activity involved so any such extra requirements on the *things* should be avoided, especially as the check would need to occur on every access.

3. Patient-specific roles, e.g. 'cardiologist of Alice' and 'cardiologist of Bob', could be created, which would only confer access to the sensor(s) of the particular patient. This would fulfil the requirements for specialists to only be granted access to the sensors of their patients. However, it would be difficult and time-consuming to manage and produce a large number of similar policies. It may also be difficult to assemble this information a priori (given both the large number and dynamic nature of doctor-patient combinations).

None of these alternatives provides us with the required level of flexibility and fine-grained access without creating a needlessly high number of policies to be authored and maintained.

A preferable alternative is to take the first option above, but provide additional information, e.g. a credential proving the relationship between specialist and patient, to the capability issuing system. The capability generated and returned to the user would then only grant access to the nominated patient. In effect, the capabilities are *parameterised* by the additional information provided, in a manner analagous to the role parameterisation of [20]. While the signature on the attribute credential may still have to be checked, this is superior to option two as the signature is only checked once (on capability issuance, not on every use) and by the central policy management system, which will have superior processing power compared to the individual *things*. The *things* would only have to check the signature on the capability, not on both the capability and the attribute credential.

The solution also fulfills the requirements of the second example given, although here the information provided for parameterisation would be a credential affirming assignment of a nurse to a particular ward. In both cases the *things* would need to be registered with the central system, along with such attributes as the patient they are assigned to and that patient's current ward.

Note that in effect the capability issuing system stores *capability templates* as defined by the relevant policies. Most CapBAC systems for IoT access control would effectively store capability templates. For example, the capabilities of [10] include expiry time and identity of the user to whom the capability is issued. These would be placeholder values in capability template stored by the policy manager and filled in when the actual capability is instantiated for distribution. We extend this idea to the identity of the resource to which the capability allows access. For example, the template might note that the capability allows to devices of class 'heart_sensor' but that the issued capability can only give access to the heart sensor of a patient under the care of the requesting actor. Proof of this relationship between specialist and patient would be an attribute credential demonstrating its existence. The heart sensors would be registered by the system as being assigned to the patient. From this information and the capability template an appropriate capability can be generated and issued.

The effect of this approach is analagous to that of Schwartmann [36], who was also concerned with providing fine-grained access within a healthcare system. However, Schwartmann's approach involved predefining activation contents for *each* patient on a per-role basis. This does not scale well as the number of patients increases, as the policy expression for each patient must be handled independently.

As we demonstrate in section 8, by relying on attributes attesting to the relationship we can abstract such policy settings into a minimal number of actual policies. Further differences between our system and that of [36] include the reliance of the later on explicit user assignment to roles and its centralised approach to permission checking (i.e. the central system is consulted on every access) rather than employing capabilities.

5.2. Capability Management

Details on the components of our proposed system and their functions are discussed in section 7.1. Here we provide a discussion on capability instantiation.

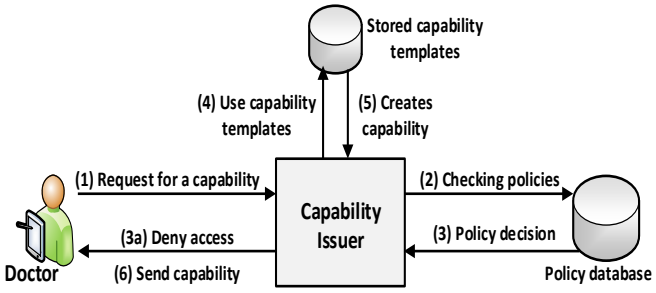

Figure 3. Issuing of a capability from a capability template.

Fig. 3 presents a simple outline of a capability issuing process using a capability template. When a request for a capability reaches the capability issuer (step 1), it checks the corresponding access policies from the policy database (step 2) and if satisfied (step 3) it contacts the capability template database (step 4) to instantiate a capability from the appropriate capability template (step 5). Finally, it sends the issued capability to the user (step 6). If the corresponding policies do not match, then the request terminates at the first instance and a response sends back to the user (step 3a). Fig. 4, illustrates the use attributes in role membership and capability instantiation.

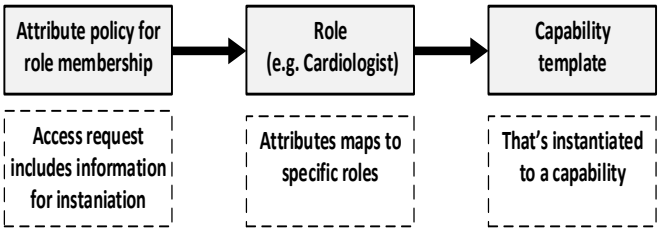

Figure 4. Use of attributes in capability instantiation.

A capability template is composed of fields needed to generate an actual capability. In section 8.1, we discuss capability structure and its different fields in detail. Some fields, e.g. the operations the capability provides access to (e.g. read, write, etc.) and conditions to be evaluated on capability use, may be pre-defined. Others, e.g. capability and user identity, expiry time and the exact devices the issued capability will allow access to, will be specified based on policies and other information stored in the system and attributes supplied by the requesting user. Capability templates may differ in how much variance they allow in instantiation. For example, a capability template may simply provide users with a capability for all doors of class 'Public_Access'. All capabilities instantiated from such a template will grant access to the same set of resources. In other cases, a capability will provide more fine-grained access. For example, referring to Fig. 1, a doctor with a role 'cardiologist' (e.g. Doctor A) wants to access the heart sensor of a patient (e.g. Bob). In this case Doctor A sends a request along with attributes satisfying role membership and attesting to their status as the cardiologist of Bob. The capability template will have state that it allows access to devices of class 'heart_sensor' but that instantiated capabilities will only include the identity of heart sensors registered to patients for whom the requestor has provided attributes attesting that they are the cardiologist of that patient. Fig. 5, illustrates the use of a capability in obtaining access.

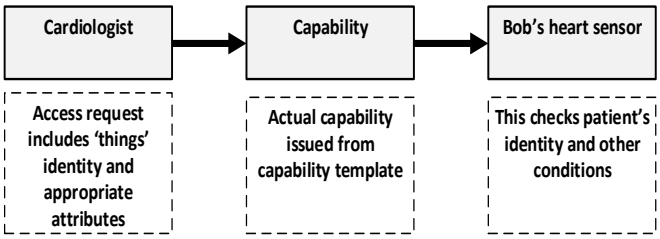

Figure 5. Using a capability for accessing a 'thing'

In the policy database, the policies can be stored based on standard XACML (eXtensible Access Control Markup Language [26] structure, which is a XML-based general purpose access control policy decision language for managing access to resources. The policy database can store these policies in the form of serialised XACML and the associated metadata.

6. Challenges and Requirements

The IoT poses its own particular security needs beyond the essential requirements for all computing systems [38] [50]. Several authors have examined the challenges and requirements for IoT and smart healthcare systems [35] [13]. Specific challenges associated with the IoT scale, heterogeneity and the need for light-weight solutions. Particularly important security challenges facing healthcare systems include:

- Data confidentiality: for example a nurse may not be allowed to see all the patient records that a specialist doctor can view.
- Privacy: healthcare records are a particularly sensitive example of user records.
- Availability: healthcare records need to be available to enable prompt patient treatment.
- Authentication: users must be properly identified to securely manage access to the sensitive data and resources of healthcare systems.

Appropriately designed access control for healthcare systems are required to help meet these challenges. Access control for the IoT is significant due to the massive scale of the number of *things* and their associated services and applications. Within this context, the following requirements can be placed on the design of a secure access control architecture [29] [46].

- Fine-grained access control: access must be properly tailored to the range of actors within the system and the roles that they fulfill.
- Scale: any proposal must scale with the number of *things* and users that may appear in IoT-enabled healthcare systems.
- Heterogeneity: accommodation must be made for the heterogeneous nature of such systems in terms of devices, communication technologies, etc.
- Light-weight: the resource-constrained nature of *things* must be recognised and light-weight solutions supported.
- Appropriate decentralisation: given the edge intelligence present in many IoT systems and their potentially large scale, security provisioning should be placed as close as possible to the point of need, while recognising both the need for data confidentiality and ease of management.

7. Proposed Access Control Architecture

In an IoT system, communications between *things* can involve various device types (e.g. smart phones and Internet-aware devices), routing protocols (e.g. Routing Protocol for Low power and Lossy Networks) and interaction patterns. We assume that the *things* broadcast the service that they have within a short distance to the other devices that are physically present typically in the form of BLE beacons in real-time e.g. Google Beacons [9].

7.1. Architectural Components

In Fig. 6 we depict our proposed access control architecture. It is composed of the following components: User Device (UD), Things (TH), Central Management System (CMS) and Things Registration Repository (TRR). The CMS consists of Role Manager (RM), Capability Database (CD) and Policy Management Unit (PMU). The PMU consists of an Evaluation Engine (EE) and Policy Database (PD).

TH's register in the system via the TRR. TH's will advertise their services. When a UD first detects a desired service (from the TH's API) it contacts the CMS to obtain the required credentials (i.e. capabilities).

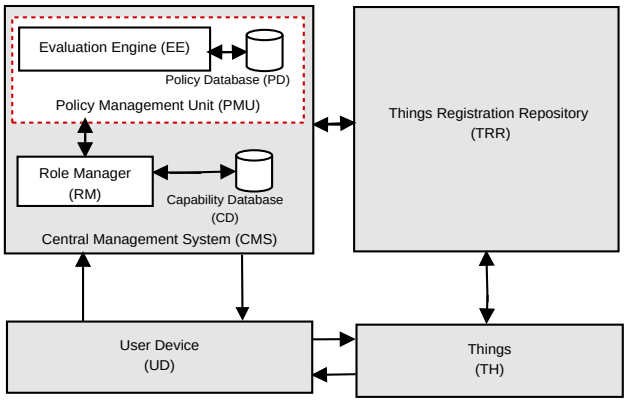

Figure 6. The proposed access control architecture.

The CMS will check the TH's registration in the TRR and supply the required credentials according to the policies maintained by its components.

- **User Device (UD):** A smart mobile device (e.g. smartphone, tablet, PDA, etc.) that belongs to a particular human user (e.g. doctor, staff, family member, etc). A UD is capable of interacting with a TH. The UD stores a user's attributes (e.g. name, ID, etc.) and issued capabilities. Attributes that the users hold are issued by a trusted authority.
- **Things (TH):** Smart IoT devices, for example, attached to the patient's sensors. A TH is capable of validating a capability. Once the authorisation decision is made, a response message (i.e. *allow* or *deny*) is sent back to the UD. In our architecture, these devices are highly resource-constrained in memory, battery and computational power. To reduce the proliferation of policy information and to improve user's privacy, a TH does not know the roles of the users or the attributes that they possess.
- **Central Management System (CMS):** The CMS is the core component of our architecture. It communicates with the UD, receiving attributes and issuing capabilities, which it also signs. The CMS is composed of the RM, CD and PMU.

1. Role Manager (RM): Stores the roles, role hierarchy and the mapping from roles to permissions (capabilities). The RM is a centralised (e.g. cloud-based) server, as it is infeasible to implement its activities within a resource-constrained IoT device. Unlike most RBAC systems an explicit role to user mapping is not maintained. Instead, attribute rules are associated with each role and stored in the PD. Users who can satisfy the rules are granted the permissions associated with the role.

2. Capability Database (CD): Stores the capability templates, which are used to create actual capabilities as defined by the parameterisation rules (cf. section 8.1). A list of revoked capabilities are also stored inside the CD.

3. Policy Management Unit (PMU): Verifies the associated polices (e.g. role membership/inheritance or capability parameterisation). It consists of the following two components:

-**Evaluation Engine (EE):** Evaluates a user request for a capability by locating the attribute rules that must be satisfied for role membership, checking user provided attributes against those rules and creating the requested capability.

-**Policy Database (PD):** Holds the attribute rules which grant role membership and define capability parameterisation.

• **Things Registration Repository (TRR):** Holds the identities and attributes of each TH in a particular network domain. The TRR is dynamically updated when a TH joins or leaves the network.

7.2. Communication Protocol

In Fig. 7 we illustrate the protocol for satisfying a user request. The TH broadcasts the services it provides to UDs located in proximity using IEEE 804.15.4 Bluetooth Low Energy (BLE) beacon or a similar protocol (step 1). If the UD possesses an appropriate capability, then the communication proceeds to step 6. Otherwise, the UD communicates with the CMS, specifying the TH and service it wishes to access, in order to obtain an appropriate capability (step 2). The CMS uses the RM and CD to locate the appropriate capability template and extracts the necessary rules from the PD. This is used to inform the UD of the attributes that must be presented (step 3).

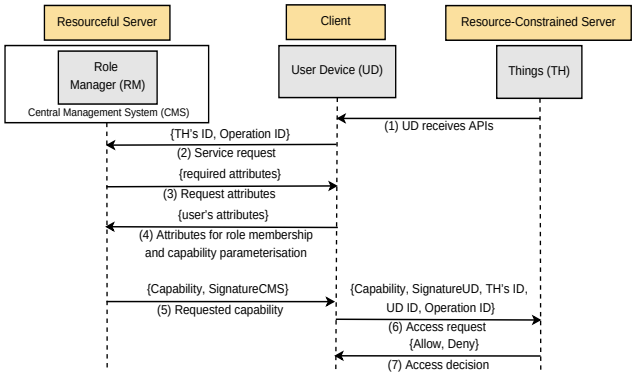

Figure 7. Protocol for service access.

These attributes are those required to obtain role memberships and (if specified) further attributes required for capability parameterisation. Assuming that the UD holds, on behalf of the user, attributes that will satisfy the requirements it sends them, and the user identity, to the CMS (step 4). The CMS checks that the supplied attributes satisfy the requirements for role membership. It then creates a capability for the requested *thing*/operation pair, by filling in the capability template with the user's identity, any necessary time stamps (e.g. beginning and end times for capability lifespan) and any parameterisation information. The user's identity is required to ensure that the capability cannot be employed by an unauthorised user to gain access. The capability, and a signature from the CMS, is then sent to the UD (step 5). The UD may now present the capability, with a signed request, to the TH (step 6). The TH will check the capability, as outlined in Algorithm 1, including checks on the CMS's signature on the capability and the UD's signature on the request and reply to the UD (step 7).

Algorithm 1 takes the capability supplied by the user, the operation requested, the user's identity, the identity of the TH and the signatures on the request and capability. It checks that the current time is within the period defined by the issued and expiry fields of the capability, that the user making the request was the one to whom the capability was granted, the capability allows access to the requested TH and operation, that any condition rules contained within the capability are satisfied and that the signatures are valid. Signature checks are left to last as they are the most-consuming operation. Conditions can involve context e.g. correct date and time, the location, etc. or properties of the TH itself, e.g. available storage, remaining battery power and any other conditions related to the state of the TH itself. These conditions are listed in a capability and are evaluated locally within the THs. The algorithm returns a decision on whether the requested operation is allowed or denied.

Algorithm 1 Capability authorisation process.

Input: Capability, TH_{ID}, $User_{ID}$, $OpReq$, $UD_{Sig}(Req)$, $CMS_{Sig}(Capability)$
Output: PermissionDecision
$PermissionDecision = \text{“}Deny\text{”}$
if *Capability is not NULL* **then**
  if $Valid(TimeStamp)$ **then**
    if $User_{ID} = Capability(User)$ **then**
      if TH_{ID} *in* $Capability(things)$ **then**
        if $OpReq$ *in* $Capability(ops)$ **then**
          if $ConditionRules = TRUE$ **then**
            if $Valid(UD_{Sig})$ & $Valid(CMS_{Sig})$ **then**
              $PermissionDecision = \text{“}Allow\text{”}$
            end
          end
        end
      end
    end
  end
end

8. Model Design and Access Scenarios

We first give a formal specification of our design. We then follow this by outlining various access control scenarios for our design based on the interactions between the UD and the TH.

8.1. A Formal Specification

Basic Concepts. Our proposed model has the following components: R, A, $Capt$, C, U, T, O, Cla and E (*roles*, *attributes*, *capability templates*, *capabilities*, *users*, *things*, *operations*, *classes* and *environment* respectively). We represent a Cla as an extensible programme that creates objects for designing and building applications. We require the following mappings:

- $RCapt : R \times Capt$, a many-to-many role to capability template assignment relation.
- $ClaO : Cla \times O$, a many-to-many class to operation assignment relation.
- $ClaT : Cla \times T$, a one-to-many class to *things* assignment relation.
- $CaptT : Capt \times T$, a many-to-many capability template to *things* assignment relation.
- $CaptO : Capt \times O$, a many-to-many capability template to operation assignment relation.
- $CaptC : CaptC$, a one-to-many capability template to capability assignment relation.
- $UC : U \times C$, a one-to-many user to capability assignment relation.

Note that equivalents to $CaptT$ and $CaptO$, CT and CO exist, mapping capabilities to *things* and operations. CO inherits directly from $CaptO$, with capabilities mapping to the operations defined by $CaptO$ for the capability template from which they were derived. The *things* that a capability maps to via CT is a subset of the corresponding mapping in $CaptT$, as defined by the paramterisation rules and supplied attributes.

- $UA_k (1 \leq k \leq K)$, $TA_m (1 \leq m \leq M)$ and $EA_n (1 \leq n \leq N)$ are the pre-defined attributes for users, *things* and environments, respectively. Where K is the number of user attributes, M is the number of *things* attributes and N is the number of environment attributes. We follow the approach of [47].
- The attribute assignment relations ($ATTR$) for user u, *things* t and environment e are $ATTR(u)$, $ATTR(t)$ and $ATTR(e)$ respectively. Where,

  $ATTR(u) \subseteq UA_1 \times UA_2 \times \cdots \times UA_K$

  $ATTR(t) \subseteq TA_1 \times TA_2 \times \cdots \times TA_M$

  $ATTR(e) \subseteq EA_1 \times EA_2 \times \cdots \times EA_N$

- We use four attribute-based *Policy Rules* for our model.

  - *Role – Membership Rule*: A boolean function of the user and environment attributes $f(ATTR(u), ATTR(e))$. This exists for each role for *role – to – user* mapping, and specifies what attributes a user (u) must possess to become a member of the role in a specific environment (e). This can be denoted as follows:

  $Policy\ Rule : Role\_Membership\ (u, e) \leftarrow f(ATTR(u), ATTR(e))$

  - *Capability – Parameterisation Rule*: A rule which specifies which *things* can be accessed using a capability generated from a capability template ($Capt$) given provided user attributes. This can be denoted as follows:

  $Policy\ Rule : Capability\_Parametisation\ (u, Capt) \leftarrow f(ATTR(u), Capt)$

  - *Condition Rule*: This is a boolean function of the *things* and environment attributes $f(ATTR(t), ATTR(e))$. It decides whether a user (u) can access a *thing* (t) in a specific environment (e). This can be denoted as follows:

  $Policy\ Rule : Condition\ (t, e) \leftarrow f(ATTR(t), ATTR(e))$

  - *Delegation Rule*: A boolean function of the user and environment attributes $f(ATTR(u), ATTR(e))$. This attribute rule specifies what attributes a user (u) must possess if that user is to be eligible to employ a delegated capability. This can be denoted as follows:

  $Policy\ Rule : Delegation\ (u, e) \leftarrow f(ATTR(u), ATTR(e))$

Capability Structure. In our model, we use the following capability structure. A capability template simple consists of the t, o and CoR fields and associated parameterisation rule, the other fields being created when a capability is created. The o and CoR fields are copied into the new capability, the t field of the capability may be a subset of that of the template, as specified by the capability parameterisation rules.
$\{Cap_{id}, U_{id}, Iss_{id}, Iss_{time}, Exp_{time}, t, o, Del, Sig, CoR, DelR\}$
where:
• Cap_{id} : Capability ID is the unique identity of each capability.
• U_{id} : User ID is the unique identity of the specific user to which the capability has been granted.
• Iss_{id} : Issuer ID is the unique identity of the entity issuing the capability.

• Iss_{time} : The time at which the capability was issued to the user.
• Exp_{time} : The time at which the issued capability will expire.
• t : This identifies either a class ID ($cl \in Cla$) or a set of related *things* ID, where all the *things* are instances of the same class. This is to note that, $t \subseteq T$ and,

$$t = \begin{cases} cl_{id} \mid cl_{id} \in Cla \\ \{t_{id_1}, t_{id_2}, t_{id_3}, \ldots, t_{id_n}\} \mid \exists\ cl_{id} \in Cla \rightarrow \\ \forall t_{id_i} \in \{t_{id_1}, t_{id_2}, t_{id_3}, \ldots, t_{id_n}\}\ t_{id_i} \in ClaT(cl_{id}) \end{cases}$$

• o : This identifies a set of operations that can be performed on the *thing*(s) to which the capability allows access. This is to note that, $o \subseteq O$ and,

$o = \{o_{id_1}, o_{id_2}, o_{id_3}, \ldots, o_{id_n}\} \mid \exists\ cl_{id} \in Cla \rightarrow o \subseteq ClaO(cl_{id})$

Note that the class ID (cl_{id}) here is the same class ID which is discussed for t.
• Del : The delegation right. This is a binary value that specifies whether or not the capability can be delegated to others.
• Sig : This is the digital-signature of the issuer of the capability. This protects the integrity of the capability from being forged or tampered with.
• CoR : A set of *Condition Rules*. It is at the discretion of the *things* how to interpret multiple rules (e.g. whether all must be satisfied or only one). Importantly, the CoR references local contexts. For example, the TH's location e.g. a particular room in a building or the date and time.
• $DelR$: A set of *Delegation Rules*. These are the attribute rules that must be satisfied before the capability can be delegated.

Compared to the other capability structure e.g. [10], which follow a heavy-weight XML structure, we use JSON (JavaScript Object Notation). Below we show a sample capability:

```
{
  "Cap_id" : "jXEPy0UFLzC4oa4ROYTCRTS39",
  "U_id" : "SN#12348484",
  "Iss_id" : "medical#organisationA",
  "Iss_time" : "0506171200",
  "Exp_time" : "0506181200",
  "t" : "heartsensor#patient1",
  "o" : "read",
  "Sig" : "jJhbGciECEF0OSQVMiLC0eXApS",
  "CoR" : [{
    "date" : "06062017",
    "loc" : "e6a360"
  }]
}
```

Capability Revocation. In our model, a capability can be revoked in two different ways. Firstly, implicitly by the capability reaching its expiry time and, secondly, explicitly by maintaining a capability revocation list. They are as follows:

• **Validity Expiration:** In this case the issued capability automatically expires after a certain period of time that is explicitly stated during the generation of the capability (i.e. Exp_{time}). In the normal case, a user could simply replace an expired capability by requesting a new one from the RM. To prevent, if revocation is required, steps should be taken to prevent the user obtaining a replacement capability. This can be achieved in a number of ways. Firstly, the user's attributes could be updated so that they no longer qualify for membership of the role that provides the capability. Secondly, the attribute rule of the role could be updated so that the user again no longer qualifies for membership. Finally, the capability template could be removed from the role. Note that implicit revocation has the advantage that the TH does not have to check on each use to determine whether a capability has been revoked. The disadvantage is that capabilities are only actually revoked once they have expired.
• **Maintain Capability Revocation Lists:** It is possible to maintain a list for revoked capabilities in the CD. On an access request, the TH provides the capabilities identity to the CD, which then informs the TH as to whether the capability has been revoked or not. This approach does not require waiting for capability expiration, but does require communication with the CD on each access. In the IoT, this extra centralisation and processing requirement on the TH may be undesirable.

Capability Delegation. In our model, capability delegation can be done followed by two possible techniques. Both rely on the delegated user satisfying the delegation rules specified in the capability. For example, consider a situation where Doctor A wishes to delegate a capability to Doctor B. In the first case, Doctor A provides Doctor B with the delegated capability, along with a signed credential stating that they authorise the delegation. When Doctor B wishes to employ the capability then he or she must, when they present the capability to the TH provide additional information. This additional information is the signed credential from Doctor A and any attributes required to satisfy the delegation attribute rule held in the capability. Assuming all checks are valid, access is allowed.

For the second case, the same information is provided by Doctor A to Doctor B, but Doctor B then access the CMS to obtain a new capability for the TH and resource. This new capability will contain Doctor B's identity in the U_{id} field. Doctor B can then present this new capability to the TH and access will proceed as normal.

Note the differences between the two cases, despite the same information is being given to Doctor B. In the first case, the CMS is not consulted. This assists in decentralising the system. However, the CMS is then unaware of the delegation. Revocation of Doctor A's

capability will also result in immediate cancellation of Doctor B's access, as it is Doctor A's capability that is provided by Doctor B to the TH. If that capability has been revoked, Doctor B's access will be denied. This alternative places more requirements on the TH as it must carry out the checking of the delegation rule of the capability.

In the second case, there are not additional requirements on the TH. It will not be aware that a delegated capability is being used. As the CMS is responsible for issuing the delegated capability it may place additional rules on whether the delegation is allowed. Revocation of access of Doctors A and B can be separately enforced. This allows for a more fine-grained management, but at the cost of additional complexity. This option also increases the centralisation of the system, due to the increased involvement of the CMS.

8.2. Different Access Scenarios

We return to the use-case example that we discussed in section 4, and discuss different access scenarios based on the issued capability, different access operations on THs and various conditions.

Scenario 1: First access. In this scenario, a user (i.e. the UD in our architecture) receives APIs from a TH. The UD communicates with the CMS requesting a specific service from a specific TH. The UD needs a capability to perform an operation and we assume that the UD does not have an appropriate capability. The UD sends the appropriate attributes to the CMS to satisfy role membership and, if required, capability parameterisation. If satisfied, the CMS issues the capability. The UD requests access to the operation from the TH and presents the capability. The TH checks that the capability authorises the requested access, via Algorithm 1. If the algorithm returns 'Allow' the UD is granted access. For example, nurse C can access Bob's clinical sensors with a valid capability.

Scenario 2: Subsequent accesses, same 'thing', same operation. In this scenario, a UD wishes to repeat an operation on a TH for which the user has already obtained an appropriate capability. As the UD already has an appropriate capability it makes the access request directly to the TH, presenting the capability. The TH again checks that the capability authorises the requested access, via Algorithm 1. If the algorithm returns 'Allow' the UD is granted access. Note that CMS is not involved in this scenario and that the UD did not need to obtain a new capability. For example, doctor A can access Bob's cardiac sensors several times after obtaining a capability without consulting the CMS after the first access.

Scenario 3: Subsequent accesses, same 'thing', different operation. Capabilities may allow access to multiple operations, and such capabilities can be used to access operations other than that for which the capability was initially requested. If a capability that the UD holds allows the access, refer to scenario 2.

Scenario 4: Access to multiple 'things' with a single capability. Capabilities may allow access to multiple THs. With the first access, the capability is obtained as in scenario 1. For subsequent accesses the UD contacts the new TH, identifies that it already holds an appropriate capability by searching the database of capabilities stored on it. It then presents the capability along with the access request as in scenario 2. For example, nurse C is allowed to access the body temperature and blood pressure sensors of multiple patients (e.g. Bob and John) using a single capability. Note that if the capability allows access to multiple THs and multiple operations on those THs, then access to a different TH may involve a different operation to the initial access.

Scenario 5: Invalid issuer of the capability and/or signature on request. A UD has a capability and wants to perform a desired operation. However the capability has not been provided by an issuer (i.e. a CMS) that the TH recognises. The UD presents the capability to the TH along with the access request. When the TH checks the signatures on the capability and the request it will reject the request (Algorithm 1 returns 'Deny') and the access will not be allowed.

Scenario 6: Capability has expired. A UD has a capability that allows the access but the capability has expired (its end time is exceeded). If the UD detects this, then refer to scenario 1. If the UD presents it to the TH anyway then TH checks the capability and discovers that the time of expiration of the capability has been reached. Algorithm 1 returns 'Deny'. If the UD wishes to obtain access they need to request a new capability for the particular access required, which may be obtained as per scenario 1.

Scenario 7: Validating local conditions. A capability may contain condition rules which must be validated by the TH before access is granted. Conditions can involve context e.g. correct date and time, location, etc. or properties of the TH itself, e.g. available storage, remaining battery power and any other conditions related to the state of the TH itself. Thus, when the UD sends a capability to the TH, along with all the checks mentioned above (see scenario 1) the TH checks the condition rules in the capability. If the condition rules are successfully validated, the access is allowed (Algorithm 1 returns 'Allow') otherwise access is denied.

9. Implementation

We have constructed a preliminary implementation of our design. As the smart THs, and communication

with them, represent the most resource-constrained aspects of the system we have concentrated on the communication between the UD and the TH and the checking of capabilities within the smart THs. We use an Android smart phone (HTC Desire 626G+) as the UD (i.e. the client). We use the ESP8266-12E microcontroller for the implementation of a TH (i.e. the resource-constrained device) because these devices are highly optimised to guarantee a high level of performance with low power and low memory consumption. The ESP8266-12E has a ready to use WiFi connection and fully supports the TCP/IP stack. It has a 32-bit RISC CPU with scalable speed, 60-160MHz, 42KB of RAM and 4MB of flash memory. The CMS is currently being implemented on a HP laptop with the following features: 2.5GHz Intel Core i7-6500U (dual-core, 4MB cache, up to 3.1GHz with Turbo Boost), Intel HD Graphics 520, 8GB LPDDR3 SDRAM (1,866MHz) and a WiFi access module. It is running Ubuntu Server 12.04.5 LTS-32 bits. However, the other parts of the architecture (e.g. the RM, PMU and TRR) could easily be built using resourceful server side application, for instance, Firebase [7] cloud hosting can be used. The implementation includes a physical testbed to measure the access control performance between the UD and the TH.

The code running in the TH has been developed using nodeMCU [25], a 'Lua'-based firmware SDK. Lua is a high level language which greatly facilitates design and implementation but the code it produces will not be optimised, so the results obtained are conservative. We have implemented the policies for capability checking inside the EPS8266-12E (cf. Algorithm 1). Our main target is to achieve a decentralised authorisation mechanism that is able to take decisions based on the capability inside the resource-constrained THs. We used the AES algorithm (with 128 bit key size) for secure communication, we use the RSA algorithm for validating the signature of the issuer (i.e. CMS) and the requester (i.e. UD) of the capability. We used MD5 message digest for authentication purposes. We chose JSON as the format for the capability token as stated above. The UD and CMS are implemented using RESTful (Representational State Transfer) client server technology.

10. Performance Evaluation

In this section we present the results of our performance evaluation of the core of our system - the communication between the UD and smart TH - and the evaluation of a capability by a TH. As the THs are the most resource-constrained elements of the system it is important to ensure that the requirements of our architecture do not pose an unmanageable load upon them. We examine a number of scenarios, from those described in section 8.2, including both when access is allowed and when it is denied and involving the TH checking a varying number of conditions. In order to demonstrate the feasibility of our model, each test was run 100 times to ensure reliable data. To demonstrate the time taken by our proposal, all considerations which are extraneous are excluded, e.g. delays due to other network traffic. This allows us to demonstrate the load/delay created by our proposal. For every success and fail, an appropriate message (e.g. allowed or denied) is sent to the UD. Note that all times are shown in milli-seconds (ms).

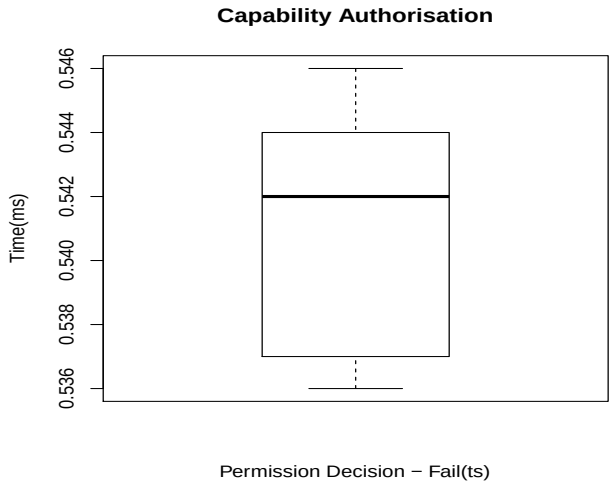

Figure 8. The capability authorisation time for an invalid capability. In this case the timestamp(ts) is not valid, therefore, the capability is rejected at the very firs instance without checkingthe other fields

Fig. 8 shows the case where the capability fails on the initial test of Algorithm 1, whether the current time falls within the valid period defined by the issued and expiry time fields of the capability. This represents the minimum time for a response by a TH once a UD presents it with a capability. The results show that, the median time taken for this stage is 0.542ms ($\mu = 0.540$, $\sigma = 0.003$).

Fig. 9 shows two results (i) when the time stamp is valid but the capability does not apply to the TH. The median time taken for this stage is 0.564ms ($\mu = 0.564$, $\sigma = 0.003$). And (ii) when the time stamp is valid, the capability applies to the TH but the capability does not allow a valid operation on the TH. The median time taken for this stage is 0.577ms ($\mu = 0.578$, $\sigma = 0.003$). These results are only slightly longer than those in Fig. 8. The second result is longer than the first as the test for the capability applying to TH must be carried out and passed before the test on whether the requested operation is allowed by the capability is applied.

Fig. 10 shows the results of checking a single condition. The right-hand result is when the condition

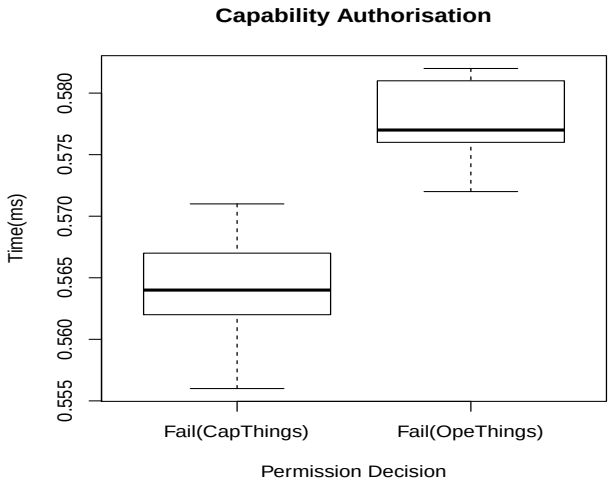

Figure 9. The capability authorisation time for an invalid capability. 'Fail(CapThings)' denotes that the capability does not apply to the *thing*. 'Fail(OpeThings)' denotes that the capability does not allow a valid operation on the *thing*. Note, in both the cases the timestamp is valid.

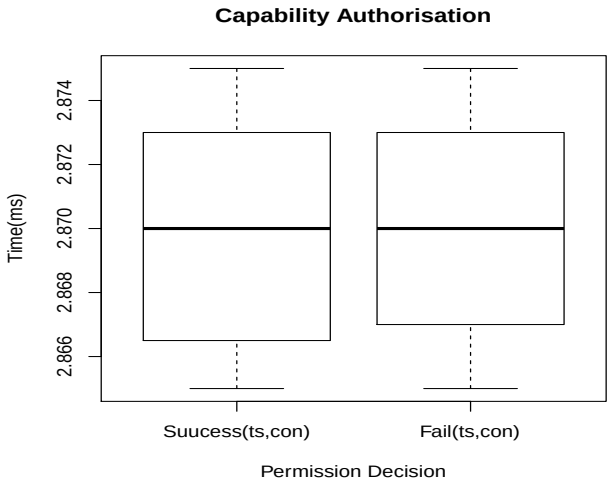

Figure 10. The capability authorisation time for condition success and failure. 'Success(ts,con)' represents the time taken when condition returns true after all previous checks have succeeded. Similarly 'Fail(ts,con)' represents the time taken when the condition is not valid but all previous checks succeeded.

is failed, the left hand result is when the condition is passed. For comparison purposes signature checking was excluded. Note that this time here is considerably longer than the previous cases, although still only a handful of milliseconds, as there is some setup involved in condition checking. The condition checked here was whether the location of the TH, represented as a string stored in the TH, satisfied the requirement expressed in the 'condition rule'. The median time required when the condition evaluates to true is 2.87ms (μ = 2.861, σ = 0.003) and when the condition evaluates to false is 2.87ms (μ = 2.861, σ = 0.003). That the results are the same (at least to the precision shown here) is to be expected, as all other fields must be checked before this test and whether a condition succeeds or fails will require much the same operations. The median RAM and ROM required for the both cases are 15KB and 9.2KB respectively.

Fig. 11 shows the results when the number of conditions to be checked are varied between one and four. In all cases, all conditions returned true to enable checking to complete. The first condition is the same as in Fig. 10 for comparison purposes. The second condition checked whether the current time was within a period specified in the condition rule, the third condition checked the date and the fourth condition involved checking the remaining battery power in the TH. Note that checking of extra conditions after the first added very little time, the set up being the same in all cases. The median time required for the one condition checking is 2.87ms (μ = 2.873, σ = 0.003), two conditions is 2.87ms (μ = 2.877, σ = 0.003), three conditions is 2.88ms (μ = 2.885, σ = 0.003) and for the four conditions is 2.89ms (μ = 2.894, σ = 0.003). Again we have excluded signature checking. Total time for a response to the UD is less then three milliseconds. Similar systems, e.g. [12] will involve equivalent signature checks. That is, two checks, one for the UD's signature on the request and another for the CMS (or equivalent) signature on the capability. The best times in the literature for these signature checks, using Elliptic Curve Cryptography (ECC) are on the order of 300ms ([12] gives a figure of 288.18ms). Including the time period for two such checks in our results would mean the signature checks, which are not the focus of our research, would completely overshadow the time taken by the functions of our proposal. This does show that our proposal adds no significant time to the basic signature checks required by all capability-based access control proposals for the IoT. In Table 1, we summarise the major performance results.

Table 1. Performance Details

Description	Median Time
Fail(CapThings)	0.564ms
Fail(OpeThings)	0.577ms
Suucess(ts,con)	2.87ms

11. Related Work

Moosavi et al. [22] present a mobility-enabled secure healthcare scheme for the IoT. The proposed scheme uses DTLS (Datagram Transport Layer Security) for

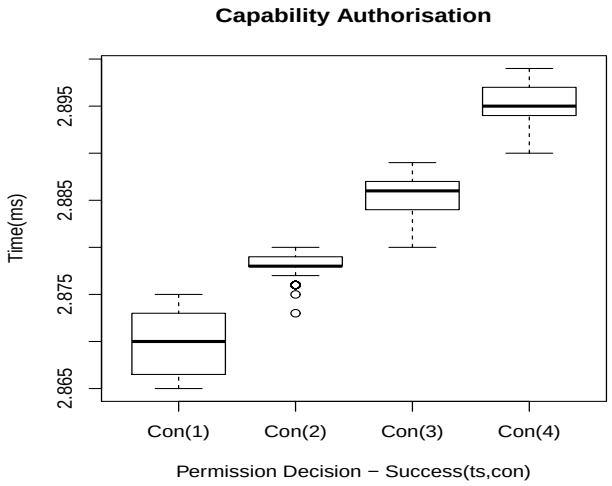

Figure 11. The capability authorisation time for a valid capability i.e. the time stamp and condition(s) are valid. In this case we vary the number of conditions from one to four

end-user authentication and authorisation. However, policy management is outside the scope of that work, instead simply assuming users have valid credentials. While their approach is distinctly different to that presented here, the RAM and ROM overheads are of the same order of magnitude. Tarouco et al. [43] explore the interoperability issues between various users and devices in an IoT-enabled healthcare system through WiFi/Bluetooth gateway supported by the Simple Network Management Protocol (SNMP) or Web services. Their proposal is based on an SNMP agent acting as a proxy between the user device and smart sensor. Access control will be implemented through the agent, but only a high-level description is given.

Ren et al. [33] and Lee et al. [14], amongst others, employ ECC for securing healthcare systems. Using ECC, a high level of encryption becomes more practical and feasible as it reduces the key size and computational costs of public key cryptography, however these proposals do not examine wider issues of access control, e.g. policy specification and management. Liu et al. [15] present an access control model for an IoT system combining ECC with RBAC. ECC is used for key establishment during entity authentication and RBAC is used for managing access control policies. Their approach to using RBAC is highly centralised, with all decisions being made in a central server, and all information being required beforehand. There is no equivalent in their proposal to our approach of allowing a previously unknown user to gain access by providing the appropriate attributes.

Various approaches e.g. [1], [42], [30] have used RBAC, sometimes in combination with other access control mechanisms, in the context of healthcare systems. For example, [1] combines RBAC with Discretionary Access Control (DAC) and Mandatory Access Control (MAC). RBAC is used to assign roles to actors within the system, with a MAC approach to overall security labels. Patients manage DAC-style access control lists to control access to their data. [30] combines RBAC with activity-based access control in a Kerberos-based ticket granting system to provide access to patient medical records. None of these proposals address access to IoT-based sensors in the healthcare context. Zhang and Tian [49] present an access control model using RBAC and security-relevant contextual information (e.g. time, location, environment, etc.) for an IoT system. Unlike our proposal, they do not give a system architecture or discuss in detail where access decisions are made. The implication is that the system is highly centralised, which is not an ideal solution for the IoT. Xu et al. present an IoT-based data accessing design for emergency medical services [45]. While this model shows how to collect, integrate and interoperate IoT data flexibly in order to provide support to emergency medical services it simply abstracts access control into the business activities layer of their model.

Mukherjee et al. [24] and Ray et al. [32] present an ABAC-based approach to medical healthcare data, but this addresses central data stores, not IoT-enabled sensors which may be the source of such data. Zhang and Liu [48] present an ABAC model to provide a fine-grained access control for IoT systems. The proposed model allows permissions to be assigned to a user for accessing resources based on user attributes, resource attributes, environment attributes and current tasks. While their framework includes policy decision and enforcement points, unlike our system it is not clear where in the system these are implemented. Further, in their design policies also have to be written in advance, based on explicit user identity.

Several proposals discuss capability-based approch for IoT and healthcare. These approaches enable access control to be tailored to the requirements of individual users and avoid fully centralised systems by providing users with access tokens (i.e. capabilities) which smart IoT devices can validate for access. Gusmeroli et al. [10] present a CapBAC approach for IoT. A similar approach is proposed by Hernandez-Ramos et al. [11] which describes a distributed CapBAC framework for IoT devices. A highly optimised version of Elliptic Curve Digital Signature Algorithm (ECDSA) is implemented inside these devices ensuring end-to-end authentication, integrity and non-repudiation. However, while offering significant detail on capability structure and processing neither of these proposals address capability distribution or questions of how the information defining which users have access to which capabilities is stored. They simply assume that a capability issuer exists. Mahalle et al. [17] and Ondiege

et al. [28] present essentially similar capability-based proposal for access control in the IoT. It is worth noting that the first three (i.e. [10], [11] and [17]) include the subject identity in the capability, to allow verification of the user making an access request.

As noted earlier, Schwartmann [36] proposed using attributes to extend RBAC to cater for one of the situations we address, allowing doctors to only access resources associated with their particular patients. As also noted, Schwartmann's approach required per-patient policy specification and did not use attributes for role membership. It was also heavily centralised. Mao et al. [20] had a similar proposal, where a language-based approach was taken to role parameterisation. While this was more concise than Schwartmann's proposal, requiring only a single rule per role (not target patient), the parameterisation was at the role level, meaning that all permissions of the role were parameterised under the same rule. This does not provide fine-grained access unless multiple roles are created. This minimises the policy reduction advantages. Unlike our proposal, the system is again centralised and no implementation details are given.

Our system extends upon the previous capability-based access approaches. We reduce the number of capabilities required in the system by allowing capabilities to grant access to more than one *thing*. In previous proposals capabilities were device-specific. More importantly, we discuss how users obtain capabilities. Role membership via presentation of attributes simplifies role specification and avoids the need to have a priori knowledge of the identity of every possible user. Capability parameterisation, allowing different capabilities to be created from the same capability template, again simplifies the creation of the policy base. Given the vast number of users and *things* that may exist in an IoT system, and the number of relationships between them, this reduction is necessary in creating manageable policy bases. Given the sensitivity of patient data in the healthcare context management of the policy base, to ensure only intended access is allowed, is a crucial consideration.

12. Discussions

Access control, policy enforcement and identity management are amongst the most important issues for the development of secure IoT-enabled smart healthcare systems. We have proposed an access control architecture combining the strengths of RBAC, ABAC and capabilities. Attributes are used for both role membership decisions and for parameterising capabilities, allowing fine-grained access control with a minimum of policy specification. The architecture is flexible, as role membership is based on attributes, not an a priori knowledge of which roles users are assigned to. This allows a degree of flexibility and conciseness in policy specification unachievable in most other proposed systems. The use of attributes in the condition rules provides additional fine-grained control. The parameterisation of capabilities allows policy specifications to apply to multiple user-*thing* relationships without providing unnecessarily widespread access or requiring extensive policy development. The architecture is partially decentralised, making effective use of the resource-constrained *things*, by tasking them with validating the capability. On the other hand, the management of roles is carried out by the central management system (of which there may be many in a practical system), as the information managed by it would overburden the *things'* storage and processing capacity.

Our results suggest that this style of decentralised authorisation systems based on resource-constrained *things* can be an alternative to fully centralised authorisation systems for a large scale IoT system. The time required for decentralised authorisation is practical for a wider IoT-enabled system deployments and within the capacity of resource-constrained devices. The limiting factor remains signature checking, with the time requirements of the unique features of our system being two orders of magnitude less than state of the art in signature checking employing ECC. While our system still requires such checks, we are not adding significant extra resource requirements in providing flexible and fine-grained access control. It is also worth noting that communication between the user device (i.e. the UD) and central management system (i.e. the CMS) will not be necessary in many cases, as our capabilities can provide access to more than one smart *thing* (i.e. the TH) without loss of security.

13. Conclusion and Future Work

With the advances in IoT-enabled smart healthcare technologies and its applications, there is a need for security and privacy to protect the system against unauthorised access. Significantly, access control, policy management and enforcement and identity management are crucial when considering the massive scale of the numbers of *things* and related applications. By combining RBAC, ABAC and CapBAC we have leveraged the advantages of all these approached, particularly in providing fine-grained access and ease of management. Attributes were employed for role membership, capability paramterisation and evaluating conditions in permissions. Capability parameterisation allows us to provide fine-grained access while minimising the number of policies required. We do not record explicit role membership, removing the need to manage the role-user mapping that is standard in RBAC systems. However, by employing RBAC, we simplify the policy

management for capabilities, as it can be specified on per-role basis, rather than per-user basis.

One limitation from a privacy point of view of our design is that while the role manager does not need to maintain a list of users who are entitled to membership of each role, it can build up such a list as users request capabilities. Similarly, as user identities are recorded in capabilities, edge devices can build up lists of users that access them. In our future work, we will examine questions of identity, e.g. the use of pseudonyms, to better preserve user privacy. In this work we used HTTP, however we currently are redeveloping the architecture using CoAP (Constrained Application Protocol). We also plan to study trust management in the context of our architecture.

Acknowledgement. The research is supported by the International Macquarie University Research Excellence Scholarship.

References

[1] Alhaqbani, B. and Fidge, C. (2008) Access Control Requirements for Processing Electronic Health Records. In Hofstede, A., Benatallah, B. and Paik, Y. [eds.] *Business Process Management Workshops* (Springer Berlin Heidelberg), *Lecture Notes in Computer Science* **4928**, 371–382. doi:10.1007/978-3-540-78238-4_38, URL http://dx.doi.org/10.1007/978-3-540-78238-4_38.

[2] Ashton, K. (2009) That 'Internet of Things' Thing. *RFID*, URL http://www.rfidjournal.com/articles/view?4986.

[3] Bhatt, Y. and Bhatt, C. (2017) Internet of Things in HealthCare. In Bhatt, C., Dey, N. and Ashour, S. [eds.] *Internet of Things and Big Data Technologies for Next Generation Healthcare* (Springer International Publishing), *Studies in Big Data* **23**, 13–33. doi:10.1007/978-3-319-49736-5_2, URL http://dx.doi.org/10.1007/978-3-319-49736-5_2.

[4] Cue (2017) Cue App., URL https://cue.me/#inflammation. [Online: accessed 05-June-2017].

[5] Evans, D. (2011) The Internet of Things: How the Next Evolution of the Internet Is Changing Everything, URL https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. [Online: accessed 02-Nov-2017].

[6] Ferraiolo, F., Sandhu, R., Gavrila, S., Kuhn, R. and Chandramouli, R. (2001) Proposed NIST Standard for Role-based Access Control. *ACM Trans. Inf. Syst. Secur.* **4**(3): 224–274. doi:10.1145/501978.501980, URL http://dx.doi.org/10.1145/501978.501980.

[7] Firebase (2017) Mobile and web application development platform, URL https://firebase.google.com/. [Online: accessed 07-June-2017].

[8] Gong, L. (1989) A Secure Identity-Based Capability System. In *Proceedings of the IEEE Symposium on Security and Privacy*: 56–63. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.1785.

[9] Google (2015) Google Beacon Platform, URL https://developers.google.com/beacons/. [Online: accessed 10-Oct-2017].

[10] Gusmeroli, S., Piccione, S. and Rotondi, D. (2013) A capability-based security approach to manage access control in the Internet of Things. *Mathematical and Computer Modelling* **58**(5-6): 1189–1205. doi:10.1016/j.mcm.2013.02.006, URL http://dx.doi.org/10.1016/j.mcm.2013.02.006.

[11] Hernandez-Ramos, J., Jara, A., Marin, L. and Skarmeta, A. (2013) Distributed Capability-based Access Control for the Internet of Things. *Journal of Internet Services and Information Security* **3**(3/4): 1–16. URL http://isyou.info/jisis/vol3/no34/jisis-2013-vol3-no34-01.pdf.

[12] Hernández-Ramos, J., Jara, A., Marín, L. and Skarmeta A. (2016) DCapBAC: embedding authorization logic into smart things through ECC optimizations. *International Journal of Computer Mathematics* **93**(2): 345–366. doi:10.1080/00207160.2014.915316, URL http://dx.doi.org/10.1080/00207160.2014.915316.

[13] Jaiswal, S. and Gupta, D. (2017) Security Requirements for Internet of Things (IoT). In Modi, N., Verma, P. and Trivedi, B. [eds.] *Proceedings of International Conference on Communication and Networks* (Springer Singapore), *Advances in Intelligent Systems and Computing* **508**, 419–427. doi:10.1007/978-981-10-2750-5_44, URL http://dx.doi.org/10.1007/978-981-10-2750-5_44.

[14] Lee, S., Alasaarela, E. and Lee, H. (2014) Secure key management scheme based on ECC algorithm for patient's medical information in healthcare system. In *The International Conference on Information Networking (ICOIN)* (IEEE): 453–457. doi:10.1109/icoin.2014.6799723, URL http://dx.doi.org/10.1109/icoin.2014.6799723.

[15] Liu, J., Xiao, Y. and Chen, P. (2012) Authentication and Access Control in the Internet of Things. In *The 32nd International Conference on Distributed Computing Systems Workshops* (IEEE): 588–592. doi:10.1109/icdcsw.2012.23, URL http://dx.doi.org/10.1109/icdcsw.2012.23.

[16] Liu, Y., Zhang, Y., Ling, J. and Liu, Z. (2017) Secure and fine-grained access control on e-healthcare records in mobile cloud computing. *Future Generation Computer Systems* doi:10.1016/j.future.2016.12.027, URL http://dx.doi.org/10.1016/j.future.2016.12.027.

[17] Mahalle, P., Anggorojati, B., Prasad, N. and Prasad, R. (2013) Identity Authentication and Capability Based Access Control (IACAC) for the Internet of Things. *Journal of Cyber Security and Mobility* **1**(4): 309–348. URL http://riverpublishers.com/journal_article.php?j=JCSM/1/4/2.

[18] Maheu, M., Nicolucci, V., Pulier, M., Wall, K., Frye, T. and Hudlicka, E. (2017) The Interactive Mobile App Review Toolkit (IMART): a Clinical Practice-Oriented System: 1–13. doi:10.1007/s41347-016-0005-z, URL http://dx.doi.org/10.1007/s41347-016-0005-z.

[19] Malina, L., Hajny, J., Fujdiak, R. and Hosek, J. (2016) On perspective of security and privacy-preserving solutions in the internet of things. *Computer Networks* **102**: 83–95. doi:10.1016/j.comnet.2016.03.011, URL http://dx.doi.org/10.1016/j.comnet.2016.03.011.

[20] Mao, Z., Li, N. and Winsborough, W. (2006) Distributed Credential Chain Discovery in Trust Management with

Parameterized Roles and Constraints (Short Paper). In Ning, P., Qing, S. and Li, N. [eds.] *Information and Communications Security* (Springer Berlin Heidelberg), *Lecture Notes in Computer Science* **4307**, 159–173. doi:10.1007/11935308_12, URL http://dx.doi.org/10.1007/11935308_12.

[21] MiniMed (2017) Medtronic, URL http://www.medtronicdiabetes.com/home. [Online: accessed 05-June-2017].

[22] Moosavi, R., Gia, N., Nigussie, E., Rahmani, M., Virtanen, S., Tenhunen, H. and Isoaho, J. (2016) End-to-end security scheme for mobility enabled healthcare Internet of Things. *Future Generation Computer Systems* **64**: 108–124. doi:10.1016/j.future.2016.02.020, URL http://dx.doi.org/10.1016/j.future.2016.02.020.

[23] Mujica, G., Portilla, J. and Riesgo, T. (2017) Deployment Strategies of Wireless Sensor Networks for IoT: Challenges, Trends, and Solutions Based on Novel Tools and HW/SW Platforms. In Keramidas, G., Voros, N. and Hübner, M. [eds.] *Components and Services for IoT Platforms* (Springer International Publishing), 133–154. doi:10.1007/978-3-319-42304-3_8, URL http://dx.doi.org/10.1007/978-3-319-42304-3_8.

[24] Mukherjee, S., Ray, I., Ray, I., Shirazi, H., Ong, T. and Kahn, G. (2017) Attribute Based Access Control for Healthcare Resources. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, ABAC'17 (New York, USA: ACM): 29–40. doi:10.1145/3041048.3041055, URL http://dx.doi.org/10.1145/3041048.3041055.

[25] nodeMCU, URL http://www.nodemcu.com/index_en.html. [Online: accessed 20-Apr-2017].

[26] OASIS (2013) extensible access control markup language (xacml), 3.0, URL http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. [Online: accessed 02-Nov-2017].

[27] Australia's healthcare system (2014) URL https://www.healthdirect.gov.au/australias-healthcare-system. [Online: accessed 05-May-2017].

[28] Ondiege, B., Clarke, M. and Mapp, G. (2017) Exploring a new security framework for remote patient monitoring devices. *MDPI Computers* **6**(1). 1–12. doi:10.3390/computers6010011, URL http://www.mdpi.com/2073-431X/6/1/11.

[29] Park, K. and Shin, H. (2017) Security assessment framework for IoT service, Telecommun Syst, (Springer US), **64**(1): 193–209. doi:10.1007/s11235-016-0168-0, URL http://dx.doi.org/10.1007/s11235-016-0168-0.

[30] Pulur, N., Altop, D. and Levi, A. (2016) A Role and Activity Based Access Control for Secure Healthcare Systems. In Abdelrahman, H., Gelenbe, E., Gorbil, G. and Lent, R. [eds.] *Information Sciences and Systems 2015* (Springer International Publishing), *Lecture Notes in Electrical Engineering* **363**, 93–103. doi:10.1007/978-3-319-22635-4_8, URL http://dx.doi.org/10.1007/978-3-319-22635-4_8.

[31] Ranjan, A. and Somani, G. (2016) Access Control and Authentication in the Internet of Things Environment. In Mahmood, Z. [ed.] *Connectivity Frameworks for Smart Devices*, Computer Communications and Networks (Springer International Publishing), 283–305. doi:10.1007/978-3-319-33124-9_12, URL http://dx.doi.org/10.1007/978-3-319-33124-9_12.

[32] Ray, I., Ong, C., Ray, I. and Kahn, G. (2016) Applying attribute based access control for privacy preserving health data disclosure. In *The IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*: 1–4. doi:10.1109/BHI.2016.7455820, URL http://dx.doi.org/10.1109/BHI.2016.7455820.

[33] Ren, Y., Werner, R., Pazzi, N. and Boukerche, A. (2010) Monitoring patients via a secure and mobile healthcare system. *IEEE Wireless Communications* **17**(1): 59–65. doi:10.1109/mwc.2010.5416351, URL http://dx.doi.org/10.1109/mwc.2010.5416351.

[34] Riazul Islam, M., Kwak, D., Humaun Kabir, M., Hossain, M. and Kwak, S. (2015) The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access* **3**: 678–708. doi:10.1109/access.2015.2437951, URL http://dx.doi.org/10.1109/access.2015.2437951.

[35] Samaila, G., Neto, M., Fernandes, B., Freire, M. and Inácio, M. (2017) *Security Challenges of the Internet of Things* (Springer International Publishing), 53–82. doi:10.1007/978-3-319-50758-3_3, URL https://doi.org/10.1007/978-3-319-50758-3_3.

[36] Schwartmann, D. (2004) An Attributable Role-Based Access Control for Healthcare. In Bubak, M., Albada, G., Sloot, P. and Dongarra, J. [eds.] *Computational Science - ICCS* (Springer Berlin Heidelberg), *Lecture Notes in Computer Science* **3039**, 1148–1155. doi:10.1007/978-3-540-25944-2_149, URL http://dx.doi.org/10.1007/978-3-540-25944-2_149.

[37] Servos, D. and Osborn, L. (2017) Current Research and Open Problems in Attribute-Based Access Control. *ACM Comput. Surv.* **49**(4). doi:10.1145/3007204, URL http://dx.doi.org/10.1145/3007204.

[38] Sfar, R., Natalizio, E., Challal, Y. and Chtourou, Z. (2017) A roadmap for security challenges in the internet of things. *Digital Communications and Networks* doi:https://doi.org/10.1016/j.dcan.2017.04.003, URL http://www.sciencedirect.com/science/article/pii/S2352864817300214.

[39] S. Pal, M. Hitchens, V. Varadharajan and T. Rabehaja (2017) On design of a fine-grained access control architecture for securing iot-enabled smart healthcare systems. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MobiQuitous, (Melbourne, Australia). doi:10.1145/3144457.3144485, URL https://doi.org/10.1145/3144457.3144485.

[40] Sicari, S., Rizzardi, A., Grieco, A. and Coen-Porisini, A. (2015) Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks* **76**: 146–164. doi:10.1016/j.comnet.2014.11.008, URL http://dx.doi.org/10.1016/j.comnet.2014.11.008.

[41] Suhardi and Ramadhan, A. (2016) A Survey of Security Aspects for Internet of Things in Healthcare. In Kim, J. and Joukov, N. [eds.] *Information Science and Applications (ICISA)* (Springer Singapore), *Lecture Notes in Electrical Engineering* **376**, 1237–1247. doi:10.1007/978-981-10-0557-2_117, URL http://dx.doi.org/10.1007/978-981-10-0557-2_117.

[42] Sun, L., Wang, H., Yong, J. and Wu, G. (2012) Semantic access control for cloud computing based on e-Healthcare. In *Proceedings of the 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (IEEE): 512–518. doi:10.1109/cscwd.2012.6221866, URL http://dx.doi.org/10.1109/cscwd.2012.6221866.

[43] Tarouco, M., Bertholdo, M., Granville, Z., Arbiza, M., Carbone, F., Marotta, M. and de Santanna, J. (2012) Internet of Things in healthcare: Interoperatibility and security issues. In *The International Conference on Communications (ICC)* (IEEE): 6121–6125. doi:10.1109/icc.2012.6364830, URL http://dx.doi.org/10.1109/icc.2012.6364830.

[44] WIRED (2015) How the Internet of Things got Hacked, URL https://www.wired.com/2015/12/2015-the-year-the-internet-of-things-got-hacked/. [Online: accessed 01-Oct-2017].

[45] Xu, B., Da Xu, L., Cai, H., Xie, C., Hu, J. and Bu, F. (2014) Ubiquitous Data Accessing Method in IoT-Based Information System for Emergency Medical Services. *IEEE Transactions on Industrial Informatics* **10**(2): 1578–1586. doi:10.1109/tii.2014.2306382, URL http://dx.doi.org/10.1109/tii.2014.2306382.

[46] Yaqoob, I., Ahmed, E., Hashem, A., Ahmed, I., Gani, A., Imran, M. and Guizani, M. (2017) Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges. *IEEE Wireless Communications* **24**(3): 10–16. doi:10.1109/mwc.2017.1600421, URL http://dx.doi.org/10.1109/mwc.2017.1600421.

[47] Yuan, E. and Tong, J. (2005) Attributed Based Access Control (ABAC) for Web Services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS'05 (Washington, USA: IEEE Computer Society): 561–569. doi:10.1109/icws.2005.25, URL http://dx.doi.org/10.1109/icws.2005.25.

[48] Zhang, G. and Liu, J. (2011) A Model of Workflow-oriented Attributed Based Access Control. *International Journal of Computer Network and Information Security* **3**(1): 47–53. URL http://www.mecs-press.org/ijcnis/ijcnis-v3-n1/IJCNIS-V3-N1-7.pdf.

[49] Zhang, G. and Tian, J. (2010) An extended role based access control model for the Internet of Things. In *The International Conference on Information, Networking and Automation (ICINA)* (IEEE), **1**: 319–323. doi:10.1109/icina.2010.5636381, URL http://dx.doi.org/10.1109/icina.2010.5636381.

[50] Zhang, N., Demetriou, S., Mi, X., Diao, W., Yuan, K., Zong, P., Qian, F. *et al.* (2017) Understanding IoT Security Through the Data Crystal Ball: Where We Are Now and Where We Are Going to Be. URL http://arxiv.org/abs/1703.09809.pdf. 1703.09809.pdf.