On Design of A Fine-Grained Access Control Architecture for Securing IoT-Enabled Smart Healthcare Systems

Shantanu Pal*, Michael Hitchens*, Vijay Varadharajan†, Tahiry Rabehaja*
*Department of Computing, Macquarie University, Sydney, NSW 2109, Australia
†Advanced Cyber Security Engineering Research Centre, University of Newcastle, NSW 2308, Australia
shantanu.pal@hdr.mq.edu.au,{michael.hitchens,tahiry.rabehaja}@mq.edu.au, vijay.varadharajan@newcastle.edu.au

ABSTRACT

The Internet of Things (IoT) is facilitating the development of novel and cost-effective applications that promise to deliver efficient and improved medical facilities to patients and health organisations. This includes the use of smart 'things' as medical sensors attached to patients to deliver real-time data. However, the security of patient data is an ever-present concern in the healthcare arena. In the wider deployment of IoT-enabled smart healthcare systems one particular issue is the need to protect smart 'things' from unauthorised access. Commonly used access control approaches e.g. Attribute Based Access Control (ABAC), Role Based Access Control (RBAC) and capability based access control 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. To address these issues 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. We devise a hybrid access control model employing 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 further attributes of the user and are then used to access specific services provided by IoT '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. Evaluation results of core functionality of our architecture are provided.

CCS CONCEPTS

• **Security and privacy → Security requirements**; **Access control**; **Mobile and wireless security**; *Distributed systems security*; • **Networks → Network security**;

KEYWORDS

Internet of things, healthcare systems, access control, policy management, security,

MobiQuitous, November 7-10, 2017, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5368-7/17/11…$15.00
https://doi.org/10.1145/3144457.3144485

ACM Reference format:
Shantanu Pal*, Michael Hitchens*, Vijay Varadharajan†, Tahiry Rabehaja*. 2017. On Design of A Fine-Grained Access Control Architecture for Securing IoT-Enabled Smart Healthcare Systems. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Melbourne, VIC, Australia, November 7-10, 2017 (MobiQuitous),* 10 pages.
https://doi.org/10.1145/3144457.3144485

1 INTRODUCTION AND BACKGROUND

With the rapid improvement in the Internet of Things (IoT) [2], healthcare systems are becoming faster, wearable and 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]. 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 [19]. With the growing number of smart mobile devices (e.g. smartphones, PDAs, etc.) there is a range of healthcare apps. compatible with wearable devices, with nearly 165,000+ healthcare apps. are available at present [14]. Demand for the use of IoT 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 or staff may have a hospital issued-device. Similarly 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 [29]. In particular unauthorised access to these wearable devices (and connected medical equipments) can breach a patient's privacy and can generate potential threats to an organisation's resources. The demand for IoT-enabled healthcare devices is increasing day by day [27]. These devices are connected to a patient's body, periodically monitoring their health status using various healthcare apps. (e.g. Cue [4] and Medtronic [17]) and communicate via wireless medium (e.g. IEEE 802.15.4). Such a

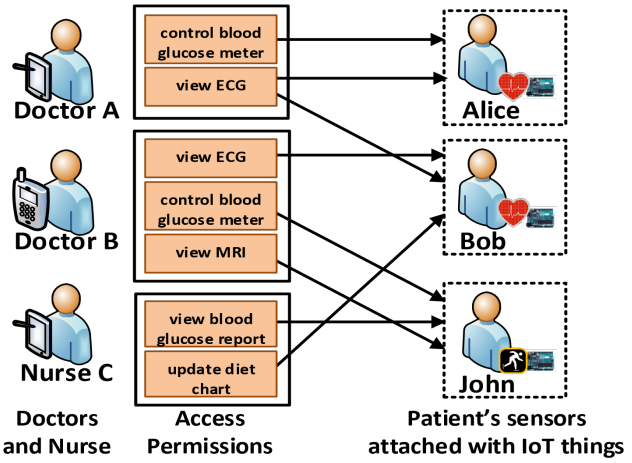

Figure 1: A fine-grained access control scenario for different actors in a healthcare system. Where a doctor/nurse with an appropriate permission can only view/control a subset of a patient's IoT-enabled healthcare 'things'. For instance, a specialist Doctor A (e.g. cardiologist) can view a particular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

convergence of the digital and physical world promises improved healthcare but also poses numerous security issues and privacy challenges [15]. IoT devices are low-powered, memory constrained and have 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 [28].

Several proposals have examined the suitability of applying commonly used access control measures for the IoT [23] [24], e.g. using Role-Based Access Control (RBAC) [5], Attribute-Based Access Control (ABAC) [33] and capability-based access control [8]. RBAC uses *roles* to manage the relationship between 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. However, to explicitly identify and assign users to roles is difficult in a dynamic and large-scale system e.g. 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. Therefore, it is ineffective 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). It can also be used to provide a context-aware based approach for user-role assignment by using user's properties (e.g. location, designation, etc). It is difficult, however, in ABAC to achieve both fine-grained access control without deploying a large policy base. In the IoT context, this means, we may still have a difficulty with the scale of the number of *things*.

Capability-based access control approaches (e.g. identity-based capability [7], distributed capability-based access control mechanism (DCapBAC) [10]) 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 [8]. It is associated with an object (uniquely identified) 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. [10]) tend to be heavily centralised, which is not necessarily 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 scale the users in a role-relationship, but typically needs a priori identification of the authorised users and capabilities that can be used to generate unique permissions for each of the users; capabilities come with well-known management issues.

In this paper we propose an access control approach for IoT-enabled smart healthcare that combines RBAC, ABAC and capabilities to produce a fine-grained and flexible approach while minimising the number of policies that need to be created and maintained. First we present an outline use-case in section 2. Then, in section 3 we explore our proposed access control architecture. In section 4 we discuss the model design with various access scenarios. We describe a detailed implementation in section 5 and give some performance evaluation results in section 6. In section 7, we present related works, and conclude the paper in section 8.

2 USE-CASE EXAMPLE AND SOLUTION OUTLINE

Consider a healthcare facility where patient monitoring and treatment delivery is controlled via smart *things*. Various devices and sensors will be 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 they are allocated to a patient they will be registered in the system, including noting 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 [21].

- 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.

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 IoT *things* for different patients will have different associated access control policies and that these policies are stored inside a centralised system. However, enforcement is handled locally within the *things*, both to take advantage of the edge-intelligence of such systems and to avoid 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 on 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.

Adopting an approach similar to [8] we propose to use capabilities as credentials to govern access to *things*. The question is how to formulate and distribute those 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. 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 fulfill 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).

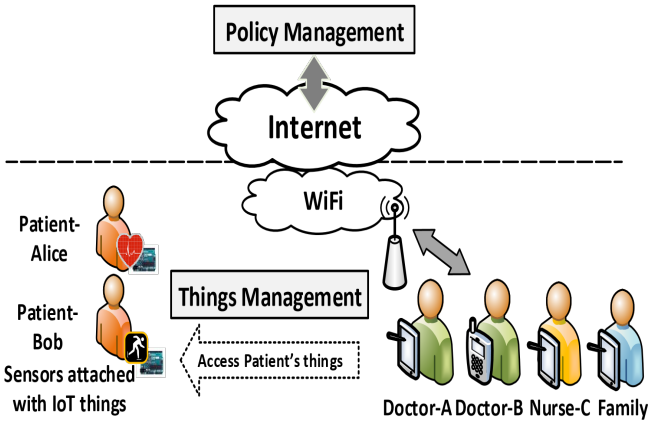

Figure 2: Our use-case scenario. The 'Policy Management' module holds policies that allow a user (e.g. doctor, nurse or family) access to the IoT 'things' associated with a particular patient. The 'Things Management' module manages the access control of each IoT 'thing'.

A preferable alternative is to take the first option, but provide additional information, a credential proving the relationship between specialist and patient, to the capability issuing system. The capability 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 [16]. While the signature on the 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 credential. The solution also fulfils the requirements of the second example, 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. On issuance, the templates are filled in with relevant information, e.g. validity time, and the appropriate parameterisation information.

3 PROPOSED ACCESS CONTROL ARCHITECTURE

3.1 Architectural Components

In Fig. 3 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 it contacts the CMS to obtain the required credentials (capabilities). 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, 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 Policy Database (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. 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.

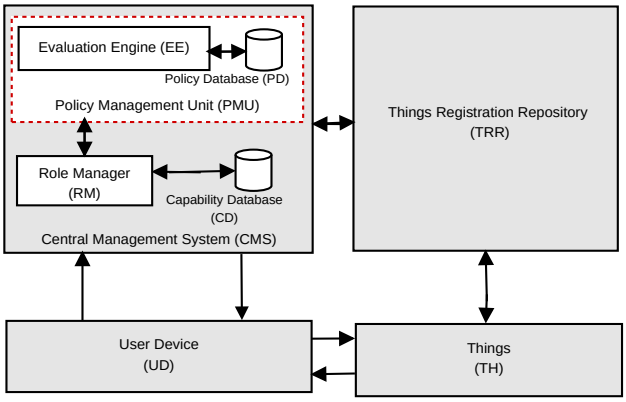

Figure 3: The proposed access control architecture.

3.2 Communication Protocol

In Fig. 4 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).

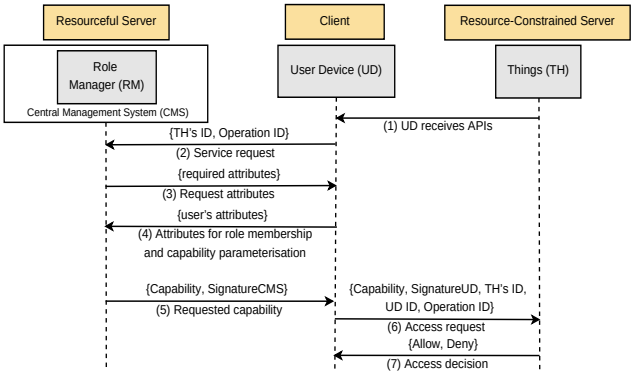

Figure 4: Protocol for service access.

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 is not passed to an unauthorised user. 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.

```
Input: Capability, TH_ID, User_ID, OpReq, UD_Sig(Request),
 CMS_Sig(Capability);
Output: PermissionDecision
PermissionDecision = "Deny";
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) AND Valid(CMS_Sig)
                         then
                            PermissionDecision = "Allow";
                        end
                    end
                end
            end
        end
    end
end
```

Algorithm 1: Capability authorisation process.

4 MODEL DESIGN

4.1 Different Access Scenarios

We return to the use-case example that we discussed in section 2, and discuss different access scenarios based on the issued capability, different access operations on THs and various conditions.

4.1.1 ***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 the role-membership. 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.

4.1.2 ***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.

4.1.3 ***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.

4.1.4 ***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.

4.1.5 ***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.

4.1.6 ***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.

4.1.7 ***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.

4.2 A Formal Specification

4.2.1 ***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 [33].
- 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 three 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:
    $Policy\ Rule : Condition\ (t, e) \leftarrow$
    $f(ATTR(t), ATTR(e))$

4.2.2 ***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.
$< Cap_{id}, U_{id}, Iss_{id}, Iss_{time}, Exp_{time}, t, o, Sig, CoR >$, 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.
- 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.

Compared to the other capability structure e.g. [8], which follow a heavy-weight XML structure, we use JSON (JavaScript Object Notation). Below we show a sample capability:

```
{
  "Cap_id" : "jXEPy0UFLzC4oa4ROYTCRTS39",
  "U_id" : "SN12348484",
```

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


5 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 a HTC Desire 626G+ smart phone 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 [6] 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, 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.

6 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 4, 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).

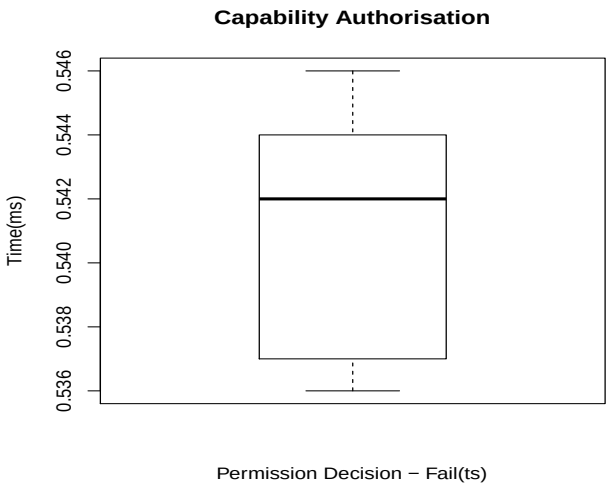

Figure 5: The capability authorisation time for an invalid capability. In this case the time stamp (ts) is not valid, therefore, the capability is rejected at the very first instance without checking the other fields.

Fig. 5 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 (μ = 0.540, σ = 0.003).

Fig. 6 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 (μ = 0.564, σ = 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 (μ = 0.578, σ = 0.003). These results are only slightly longer than those in Fig. 5. 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. 7 shows the results of checking a single condition. The right-hand result is when the condition 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

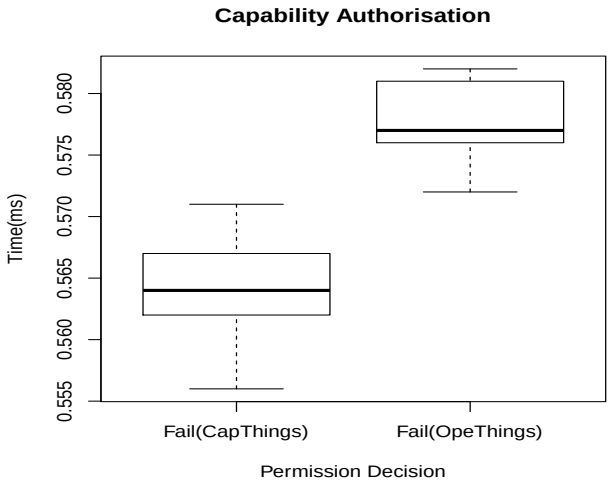

Figure 6: The capability authorisation time for an invalid capability. 'Fail(CapThings)' denotes that the capability does not apply to the *thing*. 'Fail(OpeThings)' denotes that the capability does not allow a valid operation on the *thing*. Note, in both the cases the time stamp is valid, i.e. the capability is valid for a certain period of time.

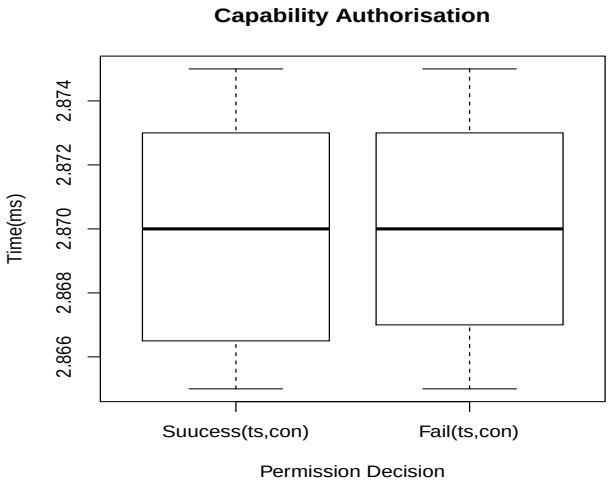

Figure 7: The capability authorisation time for condition success and failure. 'Success(ts,con)' represents the time taken when condition returns true after all previous checks have succeeded. Similarly 'Fail(ts,con)' represents the time taken when the condition is not valid but all previous checks succeeded.

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. 8 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. 7 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. [10] 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 ([10] 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.

7 RELATED WORK

Moosavi et al. [18] present a mobility enabled secure healthcare scheme for the IoT. The proposed scheme uses DTLS (Datagram Transport Layer Security) for 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. [31] 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. [26] and Lee et al. [11], 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. [12] present an

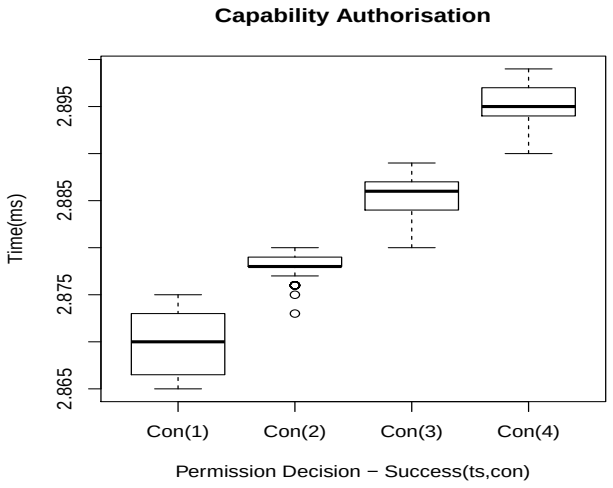

Figure 8: The capability authorisation time for a valid capability i.e. the time stamp and condition(s) are valid. In this case we vary the number of conditions from one to four.

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], [30], [23] 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. [23] 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 [35] 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 [32]. 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. [20] and Ray et al. [25] 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 [34] 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 access control 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. [8] present a capability-based access control approach for IoT. A similar approach is proposed by Hernandez-Ramos et al. [9] which describes a distributed capability-based access control 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. [13] and Ondiege et al. [22] present essentially similar capability-based proposal for access control in the IoT. It is worth noting that the first three include the subject identity in the capability, to allow verification of the user making an access request.

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 a priori know 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.

8 CONCLUSION AND FUTURE WORK

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 IoT *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 IoT *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 and central management system will not be necessary in many cases, as our capabilities can provide access to more than one smart *thing* without loss of security.

In our future work we will extend the implementation of our system, for example by including flexibility in condition specification and optimise the code, moving from Lua to a more efficient language. In this work we used HTTP, however we currently are redeveloping the architecture using CoAP (Constrained Application Protocol). We will also study trust and identity management in the context of our proposed architecture.

ACKNOWLEDGMENTS

The research is supported by the International Macquarie University Research Excellence Scholarship. We thank Richard Miller from the Faculty of Science and Engineering for his insightful comments on the paper. We also thank the anonymous reviewers for their useful comments and suggestions.

REFERENCES

[1] B. Alhaqbani and C. Fidge. 2008. Access Control Requirements for Processing Electronic Health Records. In *Business Process Management Workshops*, A. Hofstede et al. (Eds.). LNCS, Vol. 4928. Springer, 371–382.
[2] K. Ashton. 2009. That 'Internet of Things' Thing. *RFID Journal* (2009).
[3] Y. Bhatt and C. Bhatt. 2017. Internet of Things in HealthCare. In *Internet of Things and Big Data Technologies for Next Generation Healthcare*, C. Bhatt et al. (Eds.). Studies in Big Data, Vol. 23. Springer, 13–33.
[4] Cue. 2017. https://cue.me/#inflammation. (2017). [Accessed 05-June-2017].
[5] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. 2001. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security.* 4, 3 (Aug. 2001), 224–274.
[6] Firebase. 2017. https://firebase.google.com/. (2017). [Accessed 07-June-2017].
[7] L. Gong. 1989. A Secure Identity-Based Capability System. In *Procedings of the IEEE Symposium on Security and Privacy.* 56–63.
[8] S. Gusmeroli, S. Piccione, and D. Rotondi. 2013. A capability-based security approach to manage access control in the Internet of Things. *Mathematical and Computer Modelling* 58, 5-6 (Sept. 2013), 1189–1205.
[9] J. Hernández-Ramos, A. Jara, L. Marín, and A. Skarmeta. 2013. Distributed Capability-based Access Control for the Internet of Things. *Journal of Internet Services and Information Security* 3, 3/4 (Nov. 2013), 1–16.
[10] J. Hernández-Ramos, A. Jara, L. Marín, and A. Skarmeta Gómez. 2016. DCapBAC: embedding authorization logic into smart things through ECC optimizations. *International Journal of Computer Mathematics* 93, 2 (1 Feb. 2016), 345–366.
[11] Y. Lee, E. Alasaarela, and H. Lee. 2014. Secure key management scheme based on ECC algorithm for patient's medical information in healthcare system. In *Procedings of the International Conference on Information Networking 2014 (ICOIN).* IEEE, 453–457.
[12] J. Liu, Y. Xiao, and C. Chen. 2012. Authentication and Access Control in the Internet of Things. In *Procedings of the 32nd International Conference on Distributed Computing Systems Workshops.* IEEE, 588–592.
[13] P. Mahalle, B. Anggorojati, N. Prasad, and R. Prasad. 2013. Identity Authentication and Capability Based Access Control (IACAC) for the Internet of Things. *Journal of Cyber Security and Mobility* 1, 4 (10 Mar. 2013), 309–348.
[14] M. Maheu, V. Nicolucci, M. Pulier, K. Wall, T. Frye, and E. Hudlicka. 2017. The Interactive Mobile App Review Toolkit (IMART): a Clinical Practice-Oriented System. *Journal of Technology in Behavioral Science* (2017), Springer, 3–15.
[15] L. Malina, J. Hajny, R. Fujdiak, and J. Hosek. 2016. On perspective of security and privacy-preserving solutions in the internet of things. *Computer Networks* 102 (June 2016), 83–95.
[16] Z. Mao, N. Li, and W. Winsborough. 2006. Distributed Credential Chain Discovery in Trust Management with Parameterized Roles and Constraints. In *Information and Communi. Security*, P. Ning et al. (Eds.). LNCS, Vol. 4307. Springer, 159–173.
[17] Medtronic. 2017. http://www.medtronicdiabetes.com/home. (2017). [Accessed 05-June-2017].
[18] S. Moosavi, T. Gia, E. Nigussie, A. Rahmani, S. Virtanen, H. Tenhunen, and J. Isoaho. 2016. End-to-end security scheme for mobility enabled healthcare Internet of Things. *Elsevier Future Generation Computer Systems*, 64 (Nov. 2016), 108–124.
[19] G. Mujica, J. Portilla, and T. Riesgo. 2017. Deployment Strategies of Wireless Sensor Networks for IoT: Challenges, Trends, and Solutions Based on Novel Tools and HW/SW Platforms. In *Components and Services for IoT Platforms*, G. Keramidas et al. (Eds.). Springer, 133–154.
[20] S. Mukherjee, I. Ray, I. Ray, H. Shirazi, T. Ong, and M. Kahn. 2017. Attribute Based Access Control for Healthcare Resources. In *Procedings of the 2nd ACM Workshop on Attribute-Based Access Control (ABAC).* ACM, NY, USA, 29–40.
[21] Australian Institute of Health and Welfare. 2014. http://www.aihw.gov.au/australias-health/2014/health-system/. (2014). [Accessed 05-May-2017].
[22] B. Ondiege, M. Clarke, and G. Mapp. 2017. Exploring a New Security Framework for Remote Patient Monitoring Devices. *Computers* 6, 1 (2017).
[23] N. Pulur, D. Altop, and A. Levi. 2016. A Role and Activity Based Access Control for Secure Healthcare Systems. In *Information Sciences and Systems 2015*, O. Abdelrahman et al. (Eds.). LNEE, Vol. 363. Springer, 93–103.
[24] A. Ranjan and G. Somani. 2016. Access Control and Authentication in the Internet of Things Environment. In *Connectivity Frameworks for Smart Devices*, Zaigham Mahmood (Ed.). Springer, 283–305.
[25] I. Ray, T. Ong, I. Ray, and M. Kahn. 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.
[26] Y. Ren, R. Werner, N. Pazzi, and A. Boukerche. 2010. Monitoring patients via a secure and mobile healthcare system. *IEEE Wireless Communications* 17, 1 (Feb. 2010), 59–65.
[27] S. Islam, D. Kwak, H. Kabir, M. Hossain, and K. Kwak. 2015. The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access Journal* 3 (2015), 678–708.
[28] S. Sicari, A. Rizzardi, L. Grieco, and A. Porisini. 2015. Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks* 76 (Jan. 2015), 146–164.
[29] A. Ramadhan. 2016. A Survey of Security Aspects for Internet of Things in Healthcare. In *Information Science and Applications (ICISA) 2016*, K. Kim et al. (Eds.). LNEE, Vol. 376. Springer, 1237–1247.
[30] L. Sun, H. Wang, J. Yong, and G. Wu. 2012. Semantic access control for cloud computing based on e-Healthcare. In *Procedings of the 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD).* IEEE, 512–518.
[31] L. Tarouco, L. Bertholdo, L. Granville, L. Arbiza, F. Carbone, Marcelo Marotta, and Jose J. de Santanna. 2012. Internet of Things in healthcare: Interoperatibility and security issues. In *Procedings of the International Conference on Communications (ICC).* IEEE, 6121–6125.
[32] B. Xu, L. Xu, H. Cai, C. Xie, J. Hu, and F. Bu. 2014. Ubiquitous Data Accessing Method in IoT-Based Information System for Emergency Medical Services. *IEEE Transactions on Industrial Informatics* 10, 2 (May 2014), 1578–1586.
[33] E. Yuan and J. Tong. 2005. Attributed Based Access Control (ABAC) for Web Services, In Procedings of the IEEE International Conference on Web Services. 561–569.
[34] G. Zhang and J. Liu. 2011. A Model of Workflow-oriented Attributed Based Access Control. *International Journal of Computer Network and Information Security* 3, 1 (2011), 47–53.
[35] G. Zhang and J. Tian. 2010. An extended role based access control model for the Internet of Things. In *The International Conference on Information, Networking and Automation (ICINA)*, Vol. 1. IEEE, 319–323.