Securing Smart Home IoT Systems with Attribute-Based Access Control

Gaurav Goyal
Indian Institute of Technology
Kharagpur, West Bengal, India
puravgoyal@iitkgp.ac.in

Peng Liu
Pennsylvania State University
University Park, Pennsylvania, USA
pxl20@psu.edu

Shamik Sural
Indian Institute of Technology
Kharagpur, West Bengal, India
shamik@cse.iitkgp.ac.in

ABSTRACT

Over the last few years, there has been an increased proliferation of IoT systems for smart homes, enabling owners to remotely manage a variety of devices and gadgets installed on their properties. This growth was made possible due to several innovative contributions from the industry in device & sensor technology, efficient networking protocols, as well as extensive deployment of cloud infrastructure, and the development of user-friendly smartphone applications. However, the security of such systems, especially controlled access to the devices and their functionality, is still lagging. There were some recent attempts to develop access control methods for smart home IoTs. While the solutions appear to be interesting, they either ignore the practical issues faced during real-world deployment in IoT systems or do not support fine-grained access control as required by such applications. In this paper, we show how the security of smart home IoT systems can be strengthened through the use of attribute-based access control, which has been considered due to its several distinct advantages including the ability to specify fine-grained security policies and consideration of environmental conditions for making access decisions. A prototype implementation of the proposed framework has been done in the SmartThings IoT platform. An extensive set of experiments show that the approach is quite promising.

CCS CONCEPTS

• **Security and privacy** → *Domain-specific security and privacy architectures*; **Access control**.

KEYWORDS

Attribute-Based Access Control, Home IoT, Security Policy, SmartThnigs, IoT Cloud

ACM Reference Format:
Gaurav Goyal, Peng Liu, and Shamik Sural. 2022. Securing Smart Home IoT Systems with Attribute-Based Access Control. In *Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems (SaT-CPS '22), April 27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510547.3517920

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SaT-CPS '22, April 27, 2022, Baltimore, MD, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9229-7/22/04…$15.00
https://doi.org/10.1145/3510547.3517920

1 INTRODUCTION AND MOTIVATION

Smart Home IoT Devices are becoming all-pervasive both due to the level of comfort they bring to the owners as well as the capability they provide in managing one's home from anywhere in the world. Several innovative business models can be built around such smart homes. In next-generation rental applications, for instance, one can implement fine-grained service-based models where the tenant dues are calculated based on the facilities used out of a comprehensive list of available services. Although applicable for long-term leases also, it becomes particularly attractive for short rental applications. For example, a potential guest might be ready to pay for the TV, refrigerator and WiFi connection in the apartment, but would like to be excused from the cost of using the microwave or the dishwasher.

While smart IoT device firmware already supports individual device control, such use cases call for the setting up of an appropriate application-level access control - a term that denotes specification of security policies along with enforcement mechanisms that determine who can access which resource in a connected system. The goal is to enable the owner/manager of a smart home (possibly sitting in another part of the globe) to set up fine-grained policies for the tenants. In a more simplistic scenario in which the smart home is not rented to anyone, this kind of capability ensures selective availability of resources to different members of a household, which can be extended to controlling the granting of access to friends and neighbors. For example, if the family is on vacation (i.e., the value of the attribute "out of town" is True) in Hawaii, a neighbor may be automatically allowed by the policy to access certain IoT devices during emergency (e.g., when a "house caught fire" event happens).

Despite the compelling prospects, there is a stark dichotomy in the current state of research in the field of access control for IoT systems. While the device manufacturers are keen on providing increasingly rich functionality through new and upgraded hardware, firmware and cloud-based services, they ignore the requirements for supporting application-level security of these devices [1]. On the other hand, researchers working in the field of computer security, more specifically in access control, have recently put forward interesting access control models for IoT [2] [3] [4]. However, these models tend to overlook some of the practical constraints of the IoT stack, like limited access to user and device information and difficulty in modifying the main flow of program execution.

In this paper, we attempt to bridge this gap and show to what extent the capabilities of an existing IoT stack can be used to build an effective access control system that supports some of the desired features of next-generation smart home applications. It may be noted that the size of a home IoT system can vary from as small as a single house to really large like a multi-storied apartment building. The corresponding number of users also varies over a

wide range. Alongside, it needs to be kept in mind that any access control mechanism for IoT will have a set of rules, which potentially need to be updated quite frequently.

Among the various types of available access control models, Discretionary Access Control (DAC) [5], cannot be used in this context as it would involve a significant amount of administrative overhead. For each individual user and IoT device, the owner would be required to specify the type of access that is allowed. Role-Based Access Control (RBAC) [6], which is arguably the most popular access control model for commercial database applications, works well with respect to the organizational hierarchy of roles. However, they are unsuitable in the application under consideration since roles are not center stage in home IoT systems. Instead, events, contexts and attributes are the typical determining factors in making an access decision. Specifically, events and contexts cannot be handled by core RBAC. While there are certain temporal and spatial RBAC extensions [7][8], these are not sufficiently standardized and hence, it is not prudent to attempt to use them with IoT systems, a sector that is heavily standard-driven. The scheme proposed in [9] attempts to bridge the gap between what the users think they are controlling (the so-called mental model) and what is actually authorized in the IoT system. However, this approach is also owner-centric and does not easily extend to restricting access under various environmental conditions or for a diverse group of users (e.g., device owner, other family members, neighbors and friends) and devices.

Considering the above facts and also that IoT is an event-driven system, the attributes of a system (comprised of users, devices and operating environment) will be best suited for constructing security policies. Attribute-Based Access Control (ABAC), a recently proposed model [10] [11], is thus considered to be appropriate for hardening the security of home IoT systems. An added advantage of ABAC is that it can support both coarse as well as fine-grained policies depending on the need of the system, thus providing ample flexibility to the home owner. We also identify its limitations and delineate the kind of additional support needed from IoT vendors to allow fully capable ABAC models to be implemented. In summary, this paper makes the following unique contributions:

i The proposed authorization framework overcomes the lack of access control features in industry standard smart home IoT systems.
ii It lets the owners specify their policies independent of the IoT cloud and then integrate these two for controlling access to their devices. Owners' privacy is thus also preserved.
iii Through the use of ABAC, it allows authorization decision making based on environmental attributes like time of day, weekday/weekend, etc., and also on the location from which the attempt to access an IoT device has been generated – requirements that are especially pertinent for home IoT systems.

The rest of the paper is organized as follows. In Section 2, we introduce some of the preliminary concepts required for understanding the proposed framework. Section 3 identifies the challenges in practical deployment of access control in home IoT systems as well as the design principles to be followed in arriving at an effective solution. In Section 4, we describe in detail the proposed framework for ABAC-based authorization in home IoT systems. A prototype implementation of the framework is presented in Section 5 and

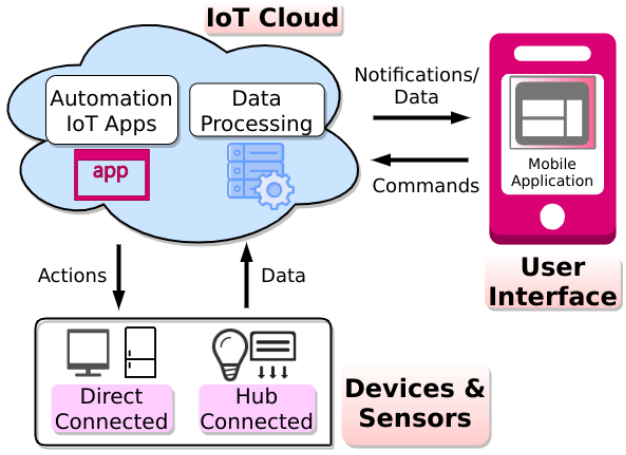

Figure 1: IoT System

results of a detailed set of experiments in Section 6. Section 7 gives a brief survey of related work in this field. We conclude the paper and suggest new directions for research in Section 8.

2 PRELIMINARIES

In this section, we first briefly introduce the basic working principle of home IoT systems. We also describe the main components of attribute-based access control.

2.1 Working of Home IoT Systems

As shown in Figure 1, a home IoT system is the integration of three different components [12], which are described below.

i. **Devices and Sensors** - These are the components that collect data from the environment in which they operate and send them to a back-end cloud infrastructure (usually referred to as the IoT Cloud). There can be a single sensor or several sensors bundled together as a group. Devices also typically have sensors embedded in them with the added capability of performing some actions. They can either directly communicate with the cloud or use a hub for connection.

ii. **IoT Cloud** - It is the component where most of the processing is done. The data passed on by IoT devices to the cloud is integrated, analyzed and finally used for the intended purpose. Such processing can be fairly simple, such as checking whether the temperature of a room is within an acceptable range, or quite complex, like using computer vision algorithms on video streams to identify objects, e.g., intruders on the premises of a property. In some cases, a piece of application code running on the cloud can send commands back to the devices when certain "if this, then that" rules are triggered. It is the IoT Cloud that is responsible for authorizing the user and the corresponding devices for granting access. Whenever a user sends a command remotely to the devices, the IoT Cloud effectively acts as a proxy between a smart home user and the supported devices.

iii. **User Interface** - The data sent by the smart home devices and possibly processed by the IoT cloud is notified to the user over email, text, or some other notification mechanism. Generally, there is a smartphone application (popularly called an app) through which

the users interact with the home IoT devices. The user can use this app either to proactively check in on the system or to be alerted about events that need an appropriate response. For instance, a user might want to check the video feeds on various rooms of one or more of her house properties. She can also send commands via this interface to her home IoT devices, for example, to change the target temperature setting of the air conditioner in her living room.

2.2 Components of ABAC

In Attribute-Based Access control, the decision on granting of requested access is taken based on the concept of attributes, which essentially are characteristics of users (interchangeably called subjects), resources (interchangeably called objects), and environmental conditions. When an access request is originated, it has four associated components, namely, the user from whom the request comes, the resource on which the access request is made, the environmental factors that can potentially influence the access decision, and the operation to be performed or permission to be granted.

While the term *subject* in ABAC typically refers to an active entity like a human user or an agent acting on her behalf, an *object* is commonly considered to be passive and is to be protected from unauthorized access by subjects. The notion of *environment* helps in capturing various operational conditions including location and time of access request. The inclusion of environmental conditions helps in supporting the specification of dynamic access rules, one of the hallmarks of ABAC that makes it well-suited for access control in home IoT systems.

As mentioned above, each of the three types of entity, i.e., user, object and environmental conditions, are associated with a prespecified set of attributes. In turn, each such attribute can take one or more possible values. After all the users, objects and environmental conditions have been assigned their appropriate attribute-value pairs, the desired set of access control rules (together called an ABAC policy) is specified. Each rule effectively states which set of users can have access to which set of objects under what environmental conditions. Together, the rules define the correct set of accesses that are authorized in the system.

With this background, we now proceed to describe the challenges specific to home IoT systems and also the principles behind the design of our proposed system architecture.

3 RESEARCH CHALLENGES AND DESIGN PRINCIPLES

In this section, we first identify some of the technical challenges in implementing effective access control for home IoT. Based on such challenges, several design principles have been arrived at that guide our proposed architecture.

3.1 Challenges in Enforcing Access Control for Home IoT Systems

An in-depth study of the existing home IoT systems reveals the following research challenges when it comes to developing a robust yet flexible authorization scheme.

• **Accessing IoT Cloud Code:** IoT cloud is the component where most of the processing is done. It stores all the relevant data about the system. However, in general, the IoT cloud acts as a black box to the application programmers including security developers. It is therefore difficult to identify the internal data representations and work with them. Like a black box, IoT cloud prohibits getting the details of user-cloud and cloud-device interactions which are useful in building policy and ensuring access control.

• **Acquiring User Details:** An important aspect of access control in IoT is getting the details (e.g., attributes) about the user. There are often several users (e.g., device owners, family members, friends, neighbors) along with many automated apps. This renders it almost impossible to acquire the unique identity of the user (UserID) initiating an access request.

• **Acquiring Object Details:** The internal representation of IoT devices is not made available to the programmer unless there is an API (Application Programming Interface) that allows access to the requisite information. Hence, getting details about the devices like the channel (peripheral) involved in the current access request is not particularly straightforward.

• **Design Simplicity:** The authorization scheme must be understandable even to an ordinary user and not just to IoT experts. Otherwise, any layer of security is eventually rendered unusable.

• **Feasibility:** Access control should be economically viable. Without incurring any additional cost, an owner should be able to update the security policy readily in a well-defined schema according to the needs of the system.

• **Security:** The authorization scheme will maintain a lot of sensitive data about the system, which should not get exposed to those who are not supposed to see them. Otherwise, such data may be exploited for criminal activities. For example, the knowledge that a home owner is not available every evening for two hours can be made use of for burglary.

The above-mentioned research challenges had to be overcome for establishing effective access control in-home IoT systems.

3.2 System Design Principles

In order to achieve the desired goals of securing home IoT systems, the following fundamental design principles were adhered to.

• **Privacy Retention:** The authorization system should only handle the access requests and grant or restrict access to various resources in an IoT system It also should not make a user's private data available to the outside world. This principle addresses the security challenges discussed in the previous subsection.

• **Privileged Owner:** The designated owner (or her delegated entity) of a home IoT system is solely authorized to set the security policy. Further, it is only she who gets to see the details of its implementation while for the rest of the users, the authorization scheme acts more like a black box.

• **Minimum Modification:** The existing IoT apps should be modified only to the bare minimum needed and it must not change the event handler logic of such apps. The additional piece of code for authorization is to be invoked just before command execution.

• **Usability:** The authorization system should be easy to use and not difficult to understand. Any change in authorization has to be made with minimal effort. The solution is intended to reduce the burden of access management on users rather than increasing it. The Design Simplicity challenge identified in the last sub-section is addressed by this principle

• **Lightweight:** The authorization approach should not inhibit performance through additional overhead. Also, the setup and continued usage of the authorization scheme have to be feasible for winning user acceptability. Being lightweight, it meets the feasibility challenge detailed in last subsection.

With these stated principles, we next describe how we design and implement a practically deployable solution for access control in home IoT systems that can address the identified design challenges.

4 ABAC FOR HOME IOT SYSTEMS

In the context of a home IoT system, the property owner along with other authorized users are the subjects, the IoT sensors and devices are the objects and the current operating condition is the environment. Their attribute-value pairs are used to create the ABAC policy. Whenever there is a change in the state of the system, an event is triggered, which is handled by the corresponding IoT app. These apps are meant to take actions corresponding to the event. However, when these apps are augmented with ABAC functionality, the access control policy is first checked to determine if the access should indeed be granted. The IoT app sends an authorization request to a Policy Enforcer, which in turn performs rule matching with respect to the given policy. The Policy Enforcer returns the result of rule matching to the IoT app, and the app executes the intended action only if the response is true, i.e., the ABAC policy set up by the owner does allow the access.

The high-level architecture of such a system as shown in Figure 2 has three major components, namely, Policy Store, Enhanced IoT App, and Policy Enforcer. In this figure, we depict how the basic IoT system architecture of Figure 1 is enhanced to include the additional functionality of the ABAC layer. It may be noted that the IoT App block has been moved outside the IoT Cloud block to denote that some of the additional features can be implemented separately.

4.1 Policy Store

The owner of a home IoT system first identifies her access control needs based on the kinds of devices she owns and the types of users to whom she wishes to give access. She might typically prefer to give a higher degree of access to her immediate family members and possibly more restrictive access to others like friends and neighbors. Also, the devices on which these users can have access are likely to vary. Even for the same device, the users may be given permissions selectively. A naïve though representative example would be to allow a certain group of users (for example, kids in the family) to check the current room temperature as sensed by an air conditioner while another group (for example, adults in the family) may be authorized to set the target temperature. On the other hand, neighbors may not be allowed any access to the air conditioner although they can turn on the sprinkler when the home IoT system owner is vacationing. The manager of a rental company as discussed in the introductory section (like Airbnb) similarly can have varying access control requirements along with their differential pricing strategies.

Once the access control requirements have been identified, the same is maintained in a Policy Store as shown in Figure 2. Keeping in mind the design challenges discussed in the last section, we consider this to be an external facility so that there is no need to access the IoT cloud, which might be prohibited. Further, keeping the policy store separate preserves the privacy of the owner's access policies and also makes it portable in the event of a need to change the IoT service provider.

The Policy Store contains the attributes of users, resources, and environment as discussed in the last section. It also maintains the ABAC policy defined in terms of the attribute-value pairs of the three entities. Such types of data may either be stored in relational databases or even in NoSQL databases if there are too many unstructured components of the attribute.

It may be noted that adding or modifying the content of Policy Store by the owner is to be done through a user-friendly GUI (ABAC Owner Interface) so that the user is not bogged down by technological challenges of the implementation and can instead focus on deciding the requirements.

4.2 Enhanced IoT App

Since the IoT devices and sensors installed in a smart home send data to and receive commands from the IoT cloud (either directly or through a hub), by default the operation bypasses any ABAC policy even if it is meticulously specified in the Policy Store. Thus, we need to enhance the code of such an IoT app so that the control gets regulated by the authorization policy - a step that can never be bypassed.

In order to achieve the desired level of authorization through effective use of policies specified by the owner and maintained in the Policy Store, the IoT apps need to be augmented. This implies a careful study of the IoT app source code and identification of the nodules that mediate access. The next step is to introduce appropriate calls to the Policy Enforcer (described in the next sub-section) so that the control transfers there. At the same time, user and device information is passed as parameters so that their attributes can be determined and rules used effectively to decide whether the access should be granted or not.

4.3 Policy Enforcer

In our architecture, policy enforcement takes place at the server-side of the authorization process. An IoT app as a client sends an authorization request to a server. This server, which we call the Policy Enforcer as seen in Figure 2, is comprised of a Request Handler and a Rule Matcher. The Request Handler receives incoming authorization requests from the IoT app, parses the information contained in the request, and fetches subject, object, and environment attribute details from the Policy Store. These are fed to the Rule Matcher for deciding whether the access should be granted or denied. The Request Handler also returns the response received from the Rule Matcher back to the client, i.e., the IoT App.

The Rule Matcher attempts to match the attribute information and the access request passed by the Request Handler with the rules listed in the ABAC Policy Store. This rule matching process can be brute force or index supported depending on how the rules are maintained in the Policy Store (which could be relational or NoSQL as mentioned before). The result of rule matching is *True* if there exists at least one rule in the Policy Store that is satisfied for the requested access. Otherwise, it returns *False*. The Rule Matcher sends this decision to the Request Handler, which in turn conveys

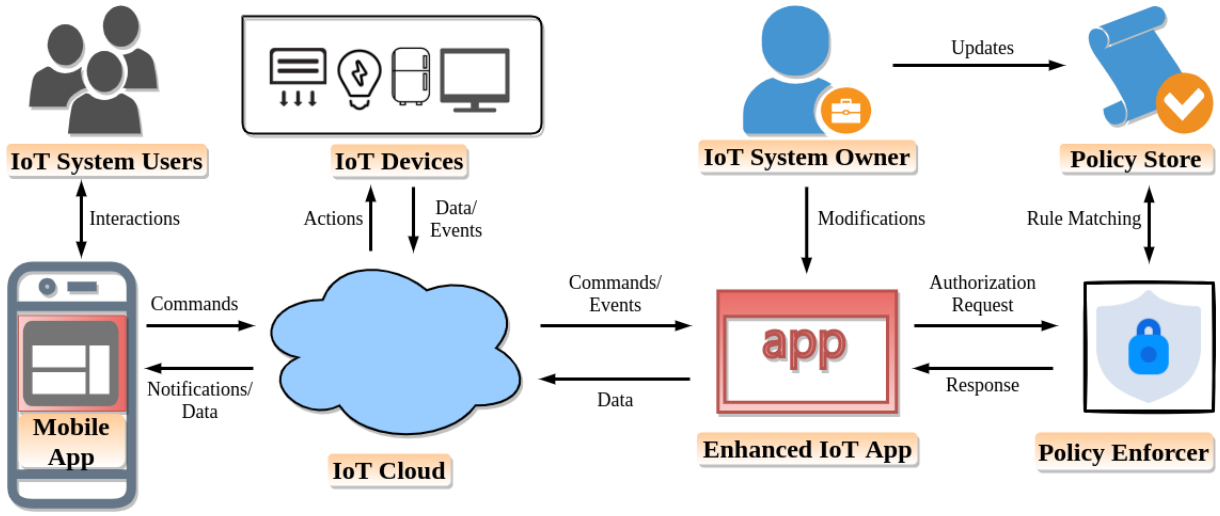

Figure 2: System Architecture

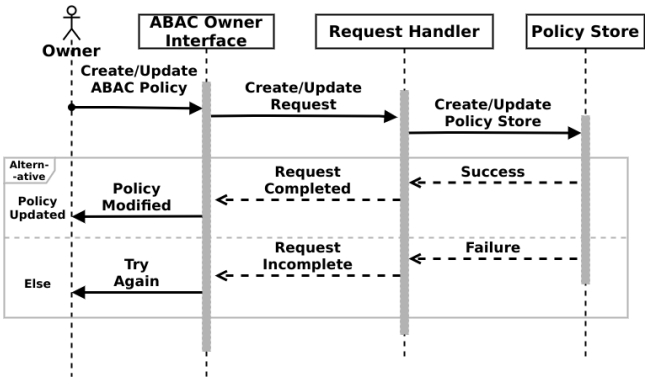

Figure 3: Creating or Updating Policy Store

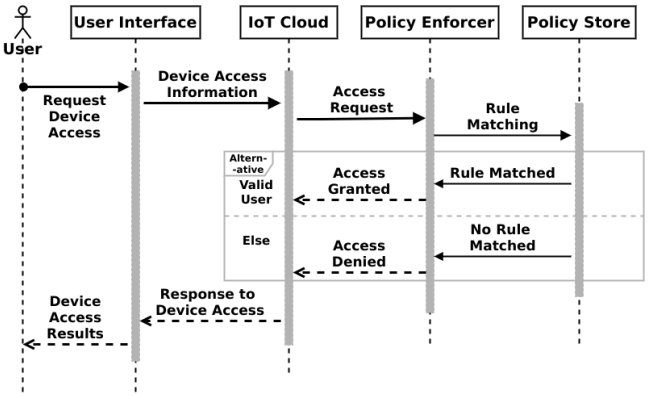

Figure 4: Processing Access Requests

it to the IoT App. Once the event handler in the IoT app receives a True response, the user is allowed to perform the desired operation. Else, the access is blocked.

4.4 Operational Steps

In this sub-section, we describe the detailed working steps of our proposed ABAC-based access control for any given home IoT system. Broadly, it is comprised of two use cases, namely, Creation & updating of policies (We refer to it as Policy Update) and Processing of access requests.

4.4.1 Policy Update. The pre-requisite for updating the security policy is to first identify the access control requirements of the concerned IoT system and then determine the attribute values for its various entities. These attribute-value pairs of the entities are used for creation of rules in the Policy Store. The ABAC model discussed here does not support administrative access control for authorizing users to create or update in policy store. The assumption here is that only authorized users can create or update any data in the policy store. As shown in the sequence diagram of Figure 3, the smart home owner uses a GUI interface (ABAC Owner Interface) to create and update the Policy Store. This front-end generates a Create/Update request depending on the desired action.

The request is sent to a Request Handler server hosted on the cloud. It processes the incoming request and appropriately creates new entries or updates existing entries in the Policy Store. On successful creation/updating, the Request Handler returns a success response to the GUI, which in turn is conveyed to the owner. A meaningful message is similarly sent to the owner in the event of a failure so that she can attempt to retry.

4.4.2 Processing of Access Requests. In this use case, a user (not necessarily the smart home owner) wants to access a device in a home IoT system. As shown in the sequence diagram of Figure 4, the user initiates an access request via a different GUI (Smartphone application in most cases). A device access request is generated and sent to the IoT cloud. It consists of information about the user, the device, and the desired operation to be performed on the device. The IoT cloud parses the request and is then handled by event

handlers of the particular device. These event handlers are within the SmartApp whose code has been enhanced by the owner.

The enhanced SmartApp generates an access request before executing the operation initiated by the user. This request consists of the UserID, ObjectID and operation to be performed on the device. It is sent to the Policy Enforcer being hosted on the web. The Request Handler component of the Policy Enforcer processes the access request. The Rule Matcher within the Policy Enforcer uses the result of the parsed request to check for the existence of matching rules in the Policy Store.

The owner can use any appropriate logic depending on the system's need for parsing requests and performing rule matching while setting up the Policy Enforcer. If the user is found to be legitimate and a matching rule is found, the Policy Enforcer returns with an access grant response to the IoT Cloud and access denial otherwise. At this stage, the SmartApp can continue its execution if access is granted; otherwise, no operation is performed. IoT Cloud returns the response to the device access request back to the User Interface. The user finally receives the device access result via the interface.

From the detailed operational steps, it can be observed that we apply the principles introduced in the last section to design the various components of the proposed architecture. We assure privacy, minimality of modification to the existing code base, retaining of owner privileges, usability, and low footprint implementation in this architecture. In the next section, we present the details of a prototype implementation on an industry-standard IoT platform.

5 PROTOTYPE IMPLEMENTATION

The architecture described in the last section takes into account the practical constraints of home IoT systems. At the same time, it is generic enough to be implemented in almost all IoT platforms. We have made a prototype system implementation on the SmartThings platform[2]. SmartThings supports numerous devices and IoT systems and provides a platform for developers to work on and create custom applications called SmartApps. We use the authorization mechanism discussed in the last section to implement access control for SmartThings compatible home IoT systems. SmartThings was chosen as the target platform since its architecture is modular and also well-documented. More importantly, it has a very large user base with thousands of IoT systems supported by the platform. These systems, however, currently lack any support for attribute-based access control. Thus, demonstrating a proof of concept on the SmartThings platform makes for a convincing case. Once fully deployed, it is also expected to benefit the highest number of home IoT users.

5.1 SmartThings Components

The key components of SmartThings are described below:

• **Mobile app:** It is the mobile application of the SmartThings platform. A user can send commands to make changes in the state of the devices via this app. The user also views all the data about devices and systems through this app. It offers several functions to the owner to control her IoT system.

• **SmartThings Cloud:** All the data from the devices are processed and commands from the SmartThings app to change the state of a device are executed in the SmartThings cloud. SmartApps code as described next is also stored in SmartThings cloud along with other information about the users.

• **SmartApps:** These are the application codes that facilitate home automation. When there is a state change in any device, an event is created. The event is passed on to the event handlers, which in turn call SmartApps to execute needed actions for that event.

• **SmartThings API:** This API serves several purposes in the general setting. We use it to send HTTP GET requests from SmartApps for authorization checks. The API also receives a response for each request and delivers it back to SmartApps.

Now, we discuss the implementation of ABAC-based authorization on the SmartThings platform.

5.2 Policy Creation

One of the practical difficulties identified during our prototype implementation is that the SmartThings platform does not let the programmer know the unique identity of the user making an access request. Owing to this, the individual users cannot be distinctly identified in the IoT system on SmartThings platform. Thus, all the rules have to be made applicable to all the users. However, it is possible to access the object (device) identifiers. Hence, the owner can create ABAC rules based on the attribute-value pairs of the devices and environmental conditions.

Some examples of device attributes that have been used in our prototype are deviceID, the sensitivity level of the device, etc. On the other hand, attributes for the environmental condition are the time of day, month, season, etc.

In order to store the ABAC policies, we have designed a set of relational database tables. The database schema is comprised of tables for Users, Objects and Policy, which store User ID User description, Object ID Object description and the Rule ID-operation pair. Two tables, namely, ObjectAtributes and UserAttributes contain details about the attribute-value pairs of Objects and Users, respectively. A set of three tables RuleObjectAttributes, RuleUserAttributes, and RuleEnvironmentAttributes maintain rule IDs and the corresponding attribute-value pairs for each of Object, User and Environment. A RequestLog table maintains a record of access requests to the authorization system.

The objectID (deviceID) for a SmartThings device and its associated information are obtained by executing appropriate SmartApp code. Listing 1 shows the code used to get the deviceID of a switch *Switch1*. Upon executing the code in a certain instance, we get *94d6f0bb-356b-4668-8303-c8fca831a9e3* as the deviceID in the console log for an example switch.

5.3 SmartApp Code Modification

We edited the sample source code of several SmartApps in order to send authorization requests to the Policy Enforcer before execution of any command via those SmartApps. For every listed action command, we append an appropriate piece of code (similar to Listing 1) to obtain the objectID of the device(deviceID) on which the desired operation is to be performed. A suitable data structure is defined which is populated with the URL of the Policy Enforcer (Server), path and header information. The header contains objectID and the action to be performed. The code for carrying out an operation on

[2]www.smartthings.com

```
// Getting Device ID for the switch Switch1
def deviceID = Switch1.id
log.debug"Device ID of switch Switch1: ${deviceID}"
// Defining data structure which contains details of access
      request
def parameters = [
    uri: "http://abac-env.eba-e2cpsej4.us-west-2.
         elasticbeanstalk.com/",
    path: "/checkRequest",
    headers: [
                objectId: deviceID,
                 operation: 'on'
               ]
]
// try-catch block to send HTTP GET request and to process its
       response
try {
    httpGet(parameters) { response ->
        response.headers.each {
        log.debug "${it.name} : ${it.value}"
    }
    if (response.data == "True") // Rule Matches
         Switch1.on() // Access granted
    else
         Switch1.off() // Access denied
    }
} catch (e) {
    log.error "something went wrong: $e"
}
```

Listing 1: Additional Code for SmartApp: Gets the device ID of the concerned IoT device. Sends the access request to Policy Enforcer, handles the response of request, and grants or denies access to the device.

a device is kept in an *if* clause. The httpGet method supported by SmartThings API is used to send a GET request to the server. This request is an authorization request and the response to this request determines whether the action will be allowed to be performed or not. Both the code for the *if* clause and the HTTP request are enclosed in a *try-catch* block to handle any error, thus eliminating the possibility of occurrence of extraneous situations.

5.4 Amazon AWS-Based Policy Enforcement

The Policy Enforcer is a server that needs to be hosted on any compatible platform and made available 24×7. In our prototype implementation, we host it in Amazon AWS[3]. Flask is used to build the server and MySQL is the deployed database. The server has two components: Request Handler and Rule Matcher as described in the last section.

The Request handler traps the incoming request, which is parsed, and the objectID as well as the operation details are obtained from its header. This data is provided as input to the Rule Matcher, which fetches the attribute-value pairs corresponding to the objectID by running a SQL query on the MySQL database It also gets the rules that allow the particular operation to be performed. Next, for each rule, the attribute-value pairs supported by that rule are obtained and matched against the attribute-value pairs corresponding to the objectID. If there is at least one matching rule, the Policy Enforcer grants access. Otherwise, access is denied.

The prototype implementation was used to test the functioning of ABAC-based authorization through simulation studies. Different devices and their events (e.g., turning a light ON or OFF, opening or closing a valve, etc.) were simulated in the SmartThings platform with appropriate ABAC policies defined in MySQL database tables hosted on AWS. It was observed that the system was correctly functioning by trapping the event calls and sending appropriate authorization requests to the Policy Enforcer. Based on the response, the operations were either granted or denied as expected.

[3]https://aws.amazon.com/

5.5 Illustrative Example

We now present a sample execution cycle for our ABAC-based authorization implementation on the SmartThings platform. To keep this illustrative example easily understandable, we use a simple IoT system. The Groovy source code of this SmartApp shows that there are four devices in the system, all being switches, named S1, S2, S3 and S4. For each switch, there are two states: *on* and *off*. Triggering of a switch refers to changing the state of the switch from *on* to *off* or vice versa. In this SmartApp, there are two event handlers. The event handler E3 handles the event of triggering of S3 and another event handler E4 handles triggering of S4. When the triggering of S3 takes place, E3 executes a command to turn *on* S1. On the other hand, in case of triggering of S4, E4 executes a command to turn *on* S2.

To augment this IoT system with appropriate ABAC based authorization, we set up a Policy Store with tables mentioned in Subsection 5.2. The policies are at the device level since in SmartThings, the user attributes are transparent. In the Policy Store, we create rules based on the various device and environmental attributes. The rules have been hand-crafted in such a manner that they do not grant access to turn *on* S2 when S4 is triggered but they do allow to turn *on* S1 upon triggering S3. The source code in Groovy SmartApp was modified in order to send authorization requests to the Policy Enforcer before executing commands. The additional code was included in both the event handlers E3 and E4. For the device S1, the object attribute-value pairs are (Load, High) and (Privacy, Public) while for S2, these are (Load, Low) and (Privacy, Public).

Initially, all the four switches are in *off* state. When we turn *on* S4, in the absence of any access control system, S2 should be turned *on* through command execution from the event handler E4. But, we observe that S2 remains in the *off* state. This is because the ABAC rules (See Table 1) do not grant access to turn *on* S2 and hence, it remains in the *off* state. When we check the response from the Policy Enforcer in the Console Log, for S2, we see the response as *False*.

Next, if we turn *on* S3, the final state of S1 is changed from *off* to *on*. This change of state is ideally expected due to command execution in Event Handler E3. Morever, the ABAC policy also has to allow the state change for S1 to *on*. It is observed from the table that in the case of S1, the access is granted by *rule-1* in the ABAC policy and hence S1 is turned *on*. In the console log also, we find that for S1, the response is *True*. Hence, it is seen that for both the devices, the ABAC system is functioning correctly and is able to control access to devices as desired by the owner of the home IoT system.

6 EXPERIMENTAL RESULTS

For evaluating the proposed framework, we carried out several experiments on our prototype implementation. Three different sets of studies were conducted to see the impact of various factors on the

Table 1: Sample Policy for the Illustrative Example

Rule ID	Device Attribute Name	Device Attribute Value	Environment Attribute Name	Environment Attribute Value	Operation
rule-1	Load	High	Time	Day	on
	Privacy	Public	Month	September	on
rule-2	Load	High	Time	Day	off
	Privacy	Private	Month	September	off
rule-3	Load	Low	Time	Night	on
	Privacy	Private	Month	September	on
rule-4	Load	Low	Time	Night	off
	Privacy	Public	Month	September	off

response time. Since there are several parameters, in each experiment, we vary one and keep the values of the rest as fixed. However, for the fixed set of parameters, we consider three different configurations (Configurations 1, 2 and 3). These are chosen in such a way that Configuration 1 denotes a low configuration, Configuration 2 a medium, and Configuration 3 a high end configuration. While the data set combinations were synthetically generated, they were loaded in the Policy Store database hosted on Amazon AWS and the access requests were generated through the SmartThings programming platform running on the Samsung IoT cloud.

In the first experiment, we study the impact of the number of ABAC rules on the overall response time using Data Set 1. There are three configurations in this data set as shown in Table 2, which represent the combinations of the values of the parameters that were fixed in the experiments. The evaluation results are plotted in Figure 5. As seen in the figure, there is some increase in response time with the increasing number of rules. However, the change is only minimal. It also needs to be noted here that the overall response time as depicted in this figure (as well as in subsequent figures for the remaining two experiments) includes the time to process the requests and responses between SmartThings and Amazon AWS (together requiring about one second). We use a free subscription of AWS with limited resource configuration. A higher configuration is expected to reduce the overall response time further.

In the second experiment, we study the impact of the number of devices in the IoT system on the response time. Data Set 2 was used for this study. The corresponding three configurations are shown in Table 3. The results of this study shown in Figure 6 depict that there is hardly any variation of response time with the number of devices. Thus, any owner of a home IoT system should not be worried about using the system even if she has several devices to control through ABAC policy specification.

In the final experiment, we study the impact of the number of attributes for each device in the IoT system. Data Set 3 was used for this study. The corresponding three configurations are shown in Table 4 and results in Figure 7. Similar to the second experiment, for this study also, it is observed that there is hardly any impact of the number of attributes on the response time. Thus, the owner can add various device attributes for implementing fine-grained access

Table 2: Fixed Parameter Values for Three Configurations when Number of Rules is varied in Figure 5

	Configuration Number		
	1	2	3
Number of Devices	10	50	100
Number of Device Attributes	10	20	30
Number of Values for each Device Attribute	20	40	60
Number of Environment Attributes	5	10	20
Number of Values for each Environment Attribute	10	20	40

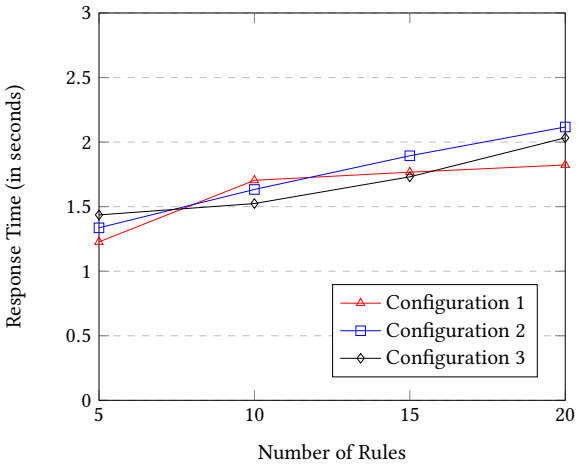

Figure 5: Response Time with Number of Rules

Table 3: Fixed Parameter Values for Three Configurations when Number of Devices is varied in Figure 6

	Configuration Number		
	1	**2**	**3**
Number of Rules	5	10	20
Number of Device Attributes	10	20	30
Number of Values for each Device Attribute	20	40	60
Number of Environment Attributes	5	10	20
Number of Values for each Environment Attribute	10	20	40

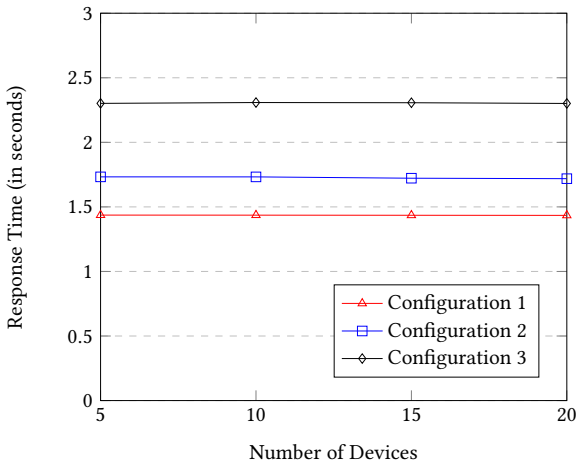

Figure 6: Response Time with Number of Devices

Table 4: Fixed Parameter Values for Three Configurations when Number of Device Attributes is varied in Figure 7

	Configuration Number		
	1	**2**	**3**
Number of Rules	5	10	20
Number of Devices	10	50	100
Number of Values for each Device Attribute	20	40	60
Number of Environment Attributes	5	10	20
Number of Values for each Environment Attribute	10	20	40

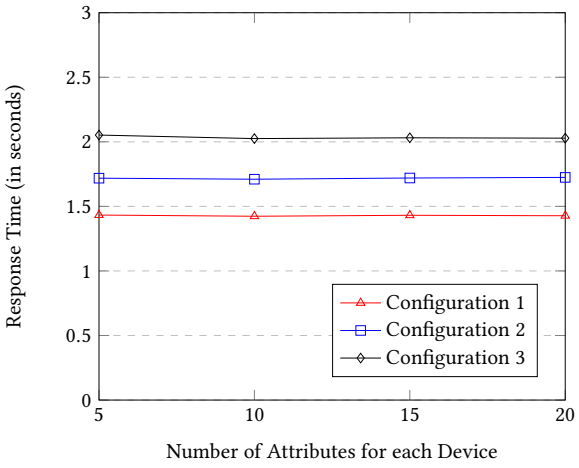

Figure 7: Response Time with Number of Device Attributes

control without adversely affecting the performance. It may be noted that we did not experiment with the environmental attributes since the results are expected to be similar to that of varying the number of device attributes.

Overall, the experimental results indicate that the proposed access control framework works well for different types of system settings. The constraints in data sets used are quite realistic with respect to a Smart Home and it essentially makes the experiments really close to real-world scenarios. After evaluating our authorization framework on the above three data sets and observing the results, wit may be concluded that the proposed ABAC-based solution is quite efficient and accommodates significant variations in Smart Home IoT systems with ease.

7 RELATED WORK

Existing literature on access control for IoT systems includes work that is practically implementable under the specific constraints of such systems as well as attempts that extend some of the traditional models to this domain. However, many of the latter ones make certain assumptions that do not always hold in the context of home IoT systems. In addition to the methods discussed in the introductory section (Section 1), we review some such approaches.

In [13], the notion of access control contract is introduced. Both static as well as dynamic access right validation are done based on a predefined set of policies and the behavior of the subject, respectively. There is a judge contract to block users who misbehave in the system and a register contract to manage access control and misbehavior. However, the system uses a different contract for every access, which makes its practical deployment infeasible. In contrast, an access control model for resource sharing based on RBAC is considered in [14]. Although it provides optimal authorization routes for received authorization requests, in certain cases, it does not achieve the desired level of performance. Also, as it considers RBAC, fine-grained access control cannot be easily established without causing role explosion. A role-based disclosure control technique suitable for IoT applications is suggested in [27]. However, dissemination of IoT data might violate an individual's privacy whose IoT devices contribute to the data. While the data can be obfuscated and the privilege to de-obfuscate it is granted based

on an RBAC policy, it is too complex for practical implementation and is also not fine-grained.

Unlike the above approaches, the work proposed in [15] uses an ABAC policy, but it only considers the user attributes. Also, as identified by us earlier in this paper, in real-life home IoT systems like SmartThings, user information is often not available for policy specification. In another competing approach [16], the Google Cloud Platform Access Control (GCPAC) model is extended to make it feasible for Google cloud platform IoT systems. However, the solution is not quite efficient and the use of ABAC for fine-grained access control is left as future work.

Among the approaches that consider fine-grained access control, [17] uses an ABAC model for a cloud-enabled AWS IoT platform. It extends the AWSIoTAC model with ABAC functionality including granular access control in the AWS IoT platform. However, the solution has a limited focus on the capabilities of IoT devices. In [18], the authors propose to use capability-based access control which grants access based on the capabilities of the user. While several interesting policies can be defined, it cannot be scaled to large IoT systems and also ignores the object or environment attributes. In contrast to these models, [31] shows how RBAC itself can be used to administer role based smart home IoT systems. Other related work that identify access control requirements in IoT include [28], [29] and [30]. However, none of these allow the IoT device owners to easily specify access control policies for a group of devices using their attributes and that too under different environmental conditions like time of day, etc.

In a completely different approach, [19] utilizes a community-based structure for defining access rights in distributed IoT environments. For a particular community of smart objects, evaluation of access is done by the resource rich objects on behalf of the constrained ones. An IoT entity with certain access rights in this context denotes that it has to act as an entitled party for an obliged party in a relationship established through a system of norms in a particular community.

In the context of such existing work, our approach towards security home IoT systems stands out due to its ability to specify fine-grained access control through ABAC, practical considerations like lack of user information in the IoT cloud, and handling of other challenges as identified in Section 3. Thus, the proposed work on ABAC based access control of smart home IoT devices can be used to complement some of the above-mentioned existing techniques.

8 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have demonstrated how attribute-based access control can be built as an added layer of security in smart home IoT systems. As a proof of concept, the code of an existing IoT application, specifically the SmartThings platform, is suitably modified for trapping access requests. These requests are then redirected to an external server for ABAC policy rule matching. Depending on the presence of relevant rules, an access decision is taken.

A shortcoming of the current implementation is that the subjectID (a unique identifier for subjects) cannot be captured in the SmartThings platform. As a result, subject-specific ABAC rules could not be defined in the prototype implementation although the generic architecture does support that. We recommend that provision be made in SmartThings for providing such information through an appropriate API. This would enable defining access rules for groups of users having common attribute values, making the system more versatile. We plan to incorporate cloud edge into our solution architecture. Next, we will build an interactive interface for creation and updating of rules in the policy store even by users not having all the necessary technical expertise.

ACKNOWLEDGMENTS

The work of Shamik Sural is supported by CISCO University Research Program Fund, Silicon Valley Community Foundation under award number 2020-220329 (3696).

REFERENCES

[1] Xu, T., Wendt, J. & Potkonjak, M. Security of IoT Systems: Design Challenges and Opportunities. *IEEE/ACM International Conference On Computer-Aided Design (ICCAD)*. pp. 417-423 (2014)
[2] Ameer, S., Benson, J. & Sandhu, R. The EGRBAC Model for Smart Home IoT. *IEEE International Conference On Information Reuse And Integration For Data Science*. pp. 457-462 (2020)
[3] Ameer, S. & Sandhu, R. The HABAC Model for Smart Home IoT and Comparison to EGRBAC. *ACM Workshop On Secure And Trustworthy Cyber-Physical Systems*. pp. 39-48 (2021)
[4] Bezawada, B., Haefner, K. & Ray, I. Securing Home IoT Environments with Attribute-Based Access Control. *Third ACM Workshop On Attribute-Based Access Control*. pp. 43-53 (2018)
[5] Sandhu, R. & Samarati, P. Access Control: Principle and Practice. *IEEE Communications Magazine*. **32**, 40-48 (1994)
[6] Sandhu, R., Coyne, E., Feinstein, H. & Youman, C. Role-based Access Control Models. *Computer*. **29** (1996)
[7] Bertino, E., Bonatti, P.A., & Ferrari, E. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):191–233 (2001).
[8] Ray, I. & Toahchoodee, M. A Spatio-Temporal Role-Based Access Control Model. *Proc. of 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security)*. 211–226 (2007).
[9] Tian, Y., Zhang, N., Lin, Y., Wang, X., Ur, B., Guo, X. & Tague, P. SmartAuth: User-Centered Authorization for the Internet of Things. *26th USENIX Security Symposium*. pp. 361-378 (2017)
[10] Hu, V., Kuhn, D., Ferraiolo, D. & Voas, J. Attribute-Based Access Control. *Computer*. **48** (2015)
[11] Xin, J., Krishnan, R. & Sandhu, R. A unified attribute-based access control model covering DAC, MAC and RBAC *IFIP Annual Conference on Data and Applications Security and Privacy*. pp. 41-55 (2012)
[12] Lv, W., Meng, F., Zhang, C., Lv, Y., Cao, N. & Jiang, J. A General Architecture of IoT System. *IEEE International Conference On Computational Science And Engineering*. **1** pp. 659-664 (2017)
[13] Zhang, Y., Kasahara, S., Shen, Y., Jiang, X. & Wan, J. Smart Contract-Based Access Control for the Internet of Things. *IEEE Internet Of Things Journal*. **6**, 1594-1605 (2019)
[14] Liu, Q., Zhang, H., Wan, J. & Chen, X. An Access Control Model for Resource Sharing Based on the Role-Based Access Control Intended for Multi-Domain Manufacturing Internet of Things. *IEEE Access*. **5** pp. 7001-7011 (2017)
[15] Ye, N. et al. An Efficient Authentication and Access Control Scheme for Perception Layer of Internet of Things. *Applied Mathematics Information Sciences*. **8** (2014)
[16] Gupta, D., Bhatt, S., Gupta, M., Kayode, O. & Tosun, A. Access Control Model for Google Cloud IoT. *IEEE 6th Intl Conference On Big Data Security On Cloud*. pp. 198-208 (2020)
[17] Bhatt, S. et al. Attribute-Based Access Control for AWS Internet of Things and Secure Industries of the Future. *IEEE Access*. **9** pp. 107200-107223 (2021)
[18] Gusmeroli, S., Piccione, S. & Rotondi, D. A capability-based security approach to manage access control in the Internet of Things. *Mathematical And Computer Modelling*. **58** pp. 1189-1205 (2013)
[19] Hussein, D., Bertino, E. & Frey, V. A Community-Driven Access Control Approach in Distributed IoT Environments. *IEEE Communications Magazine*. **55**, 146-153 (2017)
[20] Zhou, W. et al. Reviewing IoT Security via Logic Bugs in IoT Platforms and Systems. *IEEE Internet Of Things Journal*. pp. 1-1 (2021)
[21] Radovici, A., Rusu, C. & Serban, R. A Survey of IoT Security Threats and Solutions. *2018 17th RoEduNet Conference: Networking In Education And Research (RoEduNet)*. pp. 1-5 (2018)
[22] Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G. & Ghani, N. Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations. *IEEE Communications Surveys Tutorials*. **21**, 2702-2733 (2019)
[23] Aman, M., Basheer, M. & Sikdar, B. Two-Factor Authentication for IoT With Location Information. *IEEE Internet Of Things Journal*. **PP** pp. 1-1 (2018)
[24] Wang, Y., Jin, J., Li, Y. & Choi, C. A Reliable Physical Layer Authentication Algorithm for Massive IoT Systems. *IEEE Access*. **PP** pp. 1-1 (2020)
[25] Hu, V. et al. Guide to Attribute Based Access Control (ABAC) Definition and Considerations [includes updates as of 02-25-2019]. *NIST Special Publication, 800(162)*.
[26] Aman, M., Chua, K. & Sikdar, B. Mutual Authentication in IoT Systems Using Physical Unclonable Functions. *IEEE Internet Of Things Journal*. **4**, 1327-1340 (2017)
[27] Yavari, A. et al. Scalable Role-Based Data Disclosure Control for the Internet of Things. *IEEE 37th International Conference On Distributed Computing Systems (ICDCS)*. pp. 2226-2233 (2017)
[28] Yunhan J. et al. A. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. *24th Annual Network and Distributed System Security Symposium, NDSS*. pp. 1-15 (2017)
[29] Celik, B. Z. et al. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. *26th Annual Network and Distributed System Security Symposium, NDSS*. pp. 1-15 (2019)
[30] Ding, W. , Hu, H. & Cheng, L. IOTSafe: Enforcing Safety and Security Policy with Real IoT Physical Interaction Discovery. *28th Annual Network and Distributed System Security Symposium, NDSS*. pp. 1-15 (2021)
[31] Shakarami, M. & Sandhu, R. Role-Based Administration of Role-Based Smart Home IoT. *2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*. pp. 49-58 (2021)